WPI-CS-TR-99-32 Feb 1999

The RE-WEB Approach towards Web View Generation and Restructuring: Re-usable ODMG-based Templates

by

Kajal T. Claypool
Elke A. Rundensteiner
Li Chen
Bhupesh Kothari

Computer Science
Technical Report
Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

The RE-WEB Approach towards Web View Generation and Restructuring: Re-usable ODMG-based Templates*

Kajal T. Claypool, Elke A. Rundensteiner, Li Chen, Bhupesh Kothari

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609–2280
{kajal|rundenst|lichen|bhupesh}@cs.wpi.edu

Abstract

In our emerging digital paper-less society, massive amounts 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 libaries are two representative sample applications with such needs. In this paper we have presented a database-centric approach called Re-WEB that addresses this need for flexible web site generation, restructuring, and maintenance simply by embracing object-oriented database technology. Namely, by associating web semantics with the modeling constructs of the ODMG object model, view schemata map to web site layouts and database objects map to actual web pages. By generating and restructuring objects views, a large class of web site structures (web views) can thus be supported using this Re-WEB approach. And, the DBMS in Re-WEB, having full knowledge of the logical structure of web views defined over the database, can thus bring standard database techniques to bear for efficiently maintaining the web views. To ease the process of web site specification and construction, we also propose the notion of generic web view transformations that can be encapsulated into reusable templates. Genericity and reusability of the templates is achieved due to the parameterization of the transformations on the one hand, and the query-based access to the system dictionary on the other hand, allowing the transformation to both inquire about as well as manipulate classes at the schema level at run time. We demonstrate in this paper that these generic web view transformations, if collected in a template library, have the potential to represent 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 system focussing on the issue of reusable view generation templates at the content and not at the presentation style level of abstraction.

*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.

Keywords: Web Views, ODMG, OQL, Object Views, Reusable Transformations, Restructuring.

1 Introduction

In this paper we present a database-centric approach for flexible web site generation and re-structuring called Re-WEB. Unlike other web site management approaches [AMM, FFLS97, AM98] that invent either specialized web query languages, web page scheme languages, or web hypergraph data models, the key principle underlying Re-WEB is that we embrace existing object-oriented database (OODB) technology [Cea97]. We demonstrate that OODB systems are suitable and in fact sufficient for supporting a wide variety of diverse web site views over the same underlying data simply by associating web semantics with the object model constructs. In this first version of Re-WEB, we focus on Java's binding of the ODMG model only and define web semantics for it. In Re-WEB, thus a view schema over an OODB system unambiguously represents the complete structure of a web site defined off that database as well as the content of the corresponding web page objects. As commonly done in other approaches [CL97], the addition of visual presentation styles to be applied to the web site structure specifying for example font sizes or list indent bullets is done in a separate stage of the web site generation process.

Our Re-WEB approach[CRCK98] being based on the standard ODMG object model, offers numerous advantages. First and foremost, we can bring standard database technology to bear on our problem. For instance, the OQL query language can be utilized to restructure the database and consequently to specify object views that effectively model restructured web sites. Query processing technology can be exploited to optimize the restructuring queries for view schemata and hence indirectly also the web site generation process. The DBMS, having full knowledge of the structure of web pages defined over the database, can thus be in control of efficiently keeping the web pages up-to-date with the underlying database.

However, although OODB systems provide powerful transformation capabilities in the form of a query language, such queries are typically quite complex to specify – especially for web site designers that may not be familiar with database technology in general and query languages in particular. We note that there are certain restructurings that are rather common, such as for example the combination of two types into one based on some condition or the flattening of one complex referenced type into its referring type possibly to several levels of nesting. It thus may be possible to capture some of the complex yet reoccuring restructurings in the form of transformations that can be made use of by web designers. However, the provision of any such fixed set of transformations would not be satisfactory, as it would be very difficult for any one user or system to pre-define all possible semantics and all possible transformations that could be required by a user in the future. In the context of Re-WEB, we address this by using the concept of *schema transformations* that use a database query language to combine schema manipulation, evolution primitives and object transformations [CJR98b]. A la SERF [CJR98a, Jin98, Nat98, CJR98c] we use a technique for generalizing these *schema transformations* and encapsulating them as *(transformation) templates* such that they are applicable to any schema and thus are re-usable for building new transformations.

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 focussing on the issue of reusable view generation templates. To recap, the notion of the reusable view generation templates [CJR98d] thus gives users:

- The *flexibility* to define the transformation semantics of their choice [1].

[1] Of course, it is always possible to write ad-hoc one-time-usage OQL queries as well, if and when needed.

- The *extensibility* of defining new complex transformations meeting user-specific requirements that can be added to the library.
- The *generalization* of these transformations to templates to make them applicable to any schema, and thus to be *re-usable* for different web view generation purposes.
- The *ease* of template specification by programmers and non-programmers alike due to the exclusive usage of standard ODMG constructs.
- The *speed* of allowing web-site designers to rapidly specify rather diverse sets of web sites due to the utilization of pre-defined transformations,
- The *soundness* of these user-defined transformations in terms of assuring schema consistency[CJR98d].
- The *portability* of these transformations across OODBs as libraries.

Re-WEB is fully compliant with the ODMG standard. 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'B97]) to other ODMG-compliant platforms. 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.

To summarize, the contributions of this Re-WEB project include:

- the identification of a novel approach for powerful web-site generation and restructuring based solely on standard database technology achieved by associating web semantics with different database constructs both at the schema and at the data level,
- a rich variety of diverse web views supported by the system due to the utilization of OQL as the underlying transformation specification language,
- the notion of reusable transformation templates for powerful web site restructuring, offering a wide array of advantages to both end-users and developers as detailed above,
- 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),
- the design and partial implementation of the Re-WEB system using standard ODMG-compliant technology as a proof of concept of the Re-WEB approach. The use of ODMG increases the portability of our tool to other OODB systems.

The rest of the paper is organized as follows. Section 2 describes the web semantics model we associate with ODMG model constructs. Section 3 represents the overall Re-WEB approach, including detailed examples demonstrating the utility of our approach. Section 4 covers work related, while Section 5 concludes this paper.

2 Web Semantics for the ODMG Object Model

The ODMG Object Model is based on the OMG Object Model for object request brokers, object databases and object programming languages [Cea97, Clu98]. For the purpose of the Re-WEB framework we limit our description of the ODMG Object Model to Java's binding of the object model and define web semantics for it, while extensions of our web semantics for other ODMG modelling constructs is possible in the future.

Types. The basic category for an ODMG compliant database is *types* (are also referred to as classes). The type definition gives the structure and the behavior specification for its instances. Correspondingly in our Re-WEB model, the structure of a type is interpreted as modelling the **web-page-structure** for a given web page. There is a one-to-one correspondence between a type in an OODB and a **web-page-structure** of a web page.

Objects. The basic modeling primitives for an ODMG compliant database are *objects* and *literals (or immutable objects)* both of which are categorized by their *types*. Thus an object in the OODB system is interpreted to represent an individual **web-page** object whose structure is defined by its **web-page-structure**, i.e., the type of the object. As per ODMG, each object has a unique object identifier that persists through the lifetime of the object and serves as a means of reference for other objects. We parallel this in our web semantics by mapping an object identifier to a **URL**. Thus each **web-page** object has a unique **URL**.

Literals. Literals, on the other hand, do not have object identifiers and a change to a literal results in a new literal. ODMG defines different types of literals such as atomic literals, structure literals and collection literals. A literal of any type is translated to a **web-item** on the **web-page** of its containment object. For example, an atomic literal such as `string` for a given database object is represented as a **web-item** which in this case is its value inside its **web-page** object.

Collection Objects. The ODMG object model defines *collection objects* to be composed of distinct elements, each of which can be an instance of a type, another collection or a literal type. A collection object is a special object that has no object identifier associated with it and hence in our web semantics the *collection object* corresponds to a **web-list** which is a special kind of a **web-item**. For example, a collection (list) of *literals* is mapped to an (ordered) list of structured **web-items** on a **web-page** and a list of *objects* is mapped to an ordered list of **URLs**.

Extent of Types. Although Java's binding of the ODMG model does not as yet support the notion of extents, we have found it to be a necessary extension to the binding when working with object databases. We define web semantics for an extent in terms of the elements of the extent, i.e., the number of elements in an extent is the equal to the number of **web-pages** being modelled.

Inheritance. Although ODMG defines multiple inheritance, Java's binding of ODMG Model supports only single inheritance [2]. For our web semantics we flatten the class hierarchy, i.e., if a type has three (3) inherited attributes and two (2) local attributes, then the **web-page-structure** for the type will contain all five (5) **web-items**. Thus while We exploit inheritance for the incremental specification of **web-page-structures**, there is no explicit mention of inheritance expressed between web-layouts.

Schema. An **ODMG schema** is composed of a set of object and literal type definitions and a class hierarchy. A database schema as per our web semantics hence models a **web-site** by defining its structure, called its **web-site-structure**. Given an ODMG schema corresponds to a set of type definitions, a web-site-structure for a given web-site corresponds to a set of web-page-structures. Lastly, a database associated with an ODMG schema corresponds to the set of all web pages generated using the above mapping.

[2] We deal with the *extends* relationship which does specialization of one class to another.

XML markup as a Metalanguage. In Re-Web project, we choose the uprising Extensible Markup Language (XML)[xml98a] as an intermedia to represent the ODMG object model. Later on it is associated with diverse stylesheets for multiple and flexible web pages output, thus the presentation is independent from the semantics contents and can free content authors from style issues. Basically, XML is a good candidate to be a universal data exchange format. It acts as a common syntax for expressing structure in data, which is tagged for its content or meaning. The hierarchy of tagsets is called Document Type Definition (DTD) of this XML, placing constraints and semantic interpretations on a data set, can be treated as one kind of schema or type of such set of XMLs. XML data model is a tree-structured nodes, each node is a XML element and can be mapped to the literal in ODMG data modle. Each XML file can be considered as a unique object to convey certain information. XML separates content from presentation, Web builders have a new way to control design, display, and output issues. Style sheets are the answer to be applied to the XML files to get the real web pages.

1 summarizes the web and XML semantics that we have defined for the ODMG object model.

ODMG Primitives	**XML Semantics**	**WEB Semantics**
Schema	**set of DTDs**	**web-site-structure**
Type	**DTD**	**web-page-structure**
Object	**XML file**	**web-page**
OID	**Name Space**	**URL**
Collection of objects	**Collection of XML files**	**web-list of URLs**
Atomic literal	**Atomic element**	**web-item**
Struct literal	**Structured element**	**structured web-item**
Collection of Literals	**Collection of elements**	**web-list of web-items**
Extent of a type	**set of XML files of one DTD**	**set of web-pages for a web-page-structure**

Table 1: Web Semantics for ODMG Object Model

Figure 1 shows an example of a set of web pages created from the depicted schema and the underlying database using the web semantics mapping shown in Table 1. Here for each object of the class `Course` and of the class `Professor`, we create a **web-page**. The structure of each web page, i.e., the **web-page-structure**, is given by the definition of the respective classes. The `Course` class has a collection property `taught-by` that contains instances of the `Professor` class who teach the particular course. This is reflected in the `Course` **web-page** as a list of **URLs** (using underlined names, for clarity) that point to the respective `Professor` **web-page**. Atomic literals are translated as **web-items**[3] such as `name, description,` etc.

3 The Re-WEB Framework

3.1 Features of the Re-WEB Framework

The WEB semantics for the ODMG object model as described in Section 2 translate the database semantics of types and objects to the WEB semantics of web-page layouts and web-pages themselves. Hence, for each view schema and associated database there is exactly one web site layout and a set of web pages that could be generated. Thus, alternative web sites with alternative layouts can be generated simply by describing diverse view schemas over the database. To arrive at the desired set of web-pages, we simply have to re-structure the database types and objects such that they accurately reflect the layout and content of the web-pages themselves.

[3] For clarity we have shown the property name and then the actual value of the property for the particular object.

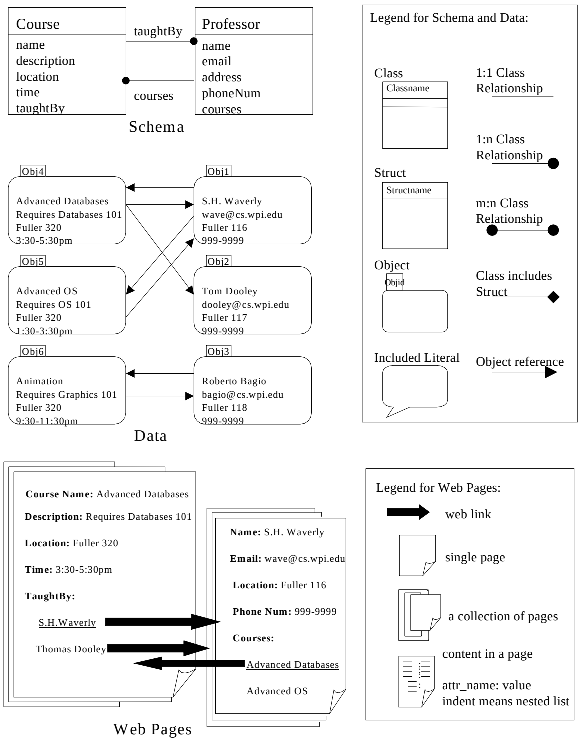

Figure 1: The Schema, the Database, and the Corresponding Web Pages.

The goal of the Re-WEB framework is to support *arbitrary user-customized* and possibly *very complex* database schema transformations thereby supporting a *flexible*, *powerful* and *customizable* way of generating web-pages. A declarative approach to writing these transformations is to use OQL together with a view mechanism. OQL, a declarative language, can express a large class of view derivations and also has the expressive power for realizing any arbitrary object manipulations to transform objects from one type to any other type. Moreover, OQL can invoke any required system methods for supporting the view mechanism. Sometimes, however, creating a new view schema may be an overkill. 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 hence propose the use of schema evolution primitives for in-place manipulation of the schema. Thus transformations using OQL, schema evolution primitives and a view mechanism can be utilized for the generation of new view schemata, i.e., new web sites, as well as for the re-structuring of existing view schemata, i.e., existing web sites.

However, writing these transformations for the re-structuring of the database is not a trivial task as the view definition queries can be very complex. Similar to schema evolution transformations, however, it is possible to identify a core set of commonly applied transformations [BKKK87, CJR98b]. For example, flattening a set of objects such that they appear as a list of web-items rather than a list of URLs is a common transformation that can be applied for different web layouts. Thus in our framework through the use of the Schema Repository we offer *re-use* of transformations by encapsulating and generalizing them and assigning a name and a set of parameters to them. From here on these are called *view transformation templates* or *templates* for short. Further we propose the development of a library of such templates. This is an open, extensible framework as developers can add new templates to the library, once they identify them as recurring. We envision that a template library could become an important resource in the web community much like the standard libraries in the programming environment.

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 *extensibilty* of defining new complex structures through new view transformations, and the *re-usability* of these structures through the notion of view transformation templates.

3.2 Re-WEB Architecture

Figure 2 gives the general architecture of the Re-WEB framework. The components listed on the top half of the figure make up the framework and thus are to be provided by any implementation realizing the Re-WEB framework. The components listed below the line represent system components that we expect any underlying OODB system to provide. These system components of an OODB include:

- **Schema repository.** For a view transformation to be generalizable to a template, we need to be able to query and access the metadata in some form (as will be explained below.). Most OODB systems indeed do provide access to the metadata.

- **Schema Manager.** View transformations, complex or simple, as defined by the Re-WEB framework rely on the underlying OODB to provide a view mechanism and also support for some in-place schema manipulations.

- **Query language.** 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. If needed, the query language must also be capable of invoking any view mechanism primitives and the schema evolution primitives. We note that a query language, like OQL, has the expressibility power to do all of the above.

- **Object manipulations.** Beyond selection, the query language should provide some support for creating, deleting and modifying objects. OQL provides this through the invocation of system-defined update methods.

- **Web Layout Generator.** In Re-Web project, we choose XML as a metalanguage to represent the ODMG data model. Here the *Web Layout Generator* can generate from a schema of ODMG model into a set of XML formatted files for the Web structure. For each class in OODB system, a XML DTD is generated and for all the extends of this class, correspondingly, the generator produces a set of XML files of the same DTD. According to the DTD, we select XML Stylesheet Language (XSL) to specify the association of presentation style with XML information. XSL is consist of a location mechanism (context, selector, pattern, query) capable of addressing into XML structure and an action performed on the located content.

- **Web Layout Manager.** As stated before, our Re-WEB system has the powerful re-structuring mechanism to allow the user the *flexibility* to define web-site structures, which represent the semantics content of web pages and the links between them, we also want to have web page layout styles separated from their contents to make reuse of data and get multiple output formats. Our *Web Layout Manager* serves for this aim. It associats style with XML data, allows interchange of data among different users and authoring, editing, browsing, viewing tools, enables both authors/publishers and users to determine presentation of marked up data and allows for the packaging of "document types" that can be shared and reused.

- **Web Page Generator.** Based on the DTDs generated from *Web Layout Generator*, we can apply diverse stylesheets onto them to get any effection we want for web pages. In our Re-Web system, The *Web Page Generator* is actually a XSL engine that performs a transformation process that can convert one XML document to a HTML document. This transformation process is running on a XML document with a XSL stylesheet to drive a style-attachment process to creates the formatted output. Arbortext company has just released its XSL tool called XML Styler[xml98b], many other companies or individuls also propelled such process by providing their tools[ie498, Tho98, Cla98].

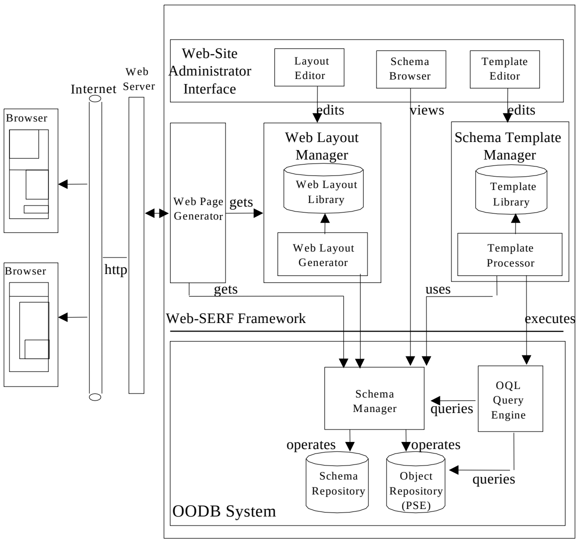

Figure 2: Architecture of the Re-WEB Framework.

Figure 3 also shows how the framework modules and the system modules interact with each other. Re-WEB targets users such as web-site administrators that need to generate or maintain web pages over the underlying database. The **Schema Browser** lets the user view and browse both view as well as base schemata. In order to construct a schema modeling a desired web site structure, the web administrator can either choose a suitable view transformation template from the template library, write a *re-usable* and *generalized* template using the **Template Editor**, or simply write an OQL query transformation. The **Template Processor** instantiates and executes a given view template using the parameters suppiled by the user [4]. In general, a template [5] uses a query language to query over the schema repository, i.e., the metadata and the application objects. The template also uses the query language to invoke both the view schema primitives and the schema evolution primitives for creating and storing views in the database and for modifying the schema types, and system-defined functions for updating the object instances.

With the underlying view schema in the right structural form, the **Web Layout Generator** can be invoked to generate a web layout using the web semantics (see Section 2) that are defined for the underlying object model [6]. Presentation styles can be applied to the web layout produced by the Web Layout Generator and this layout can then be used to generate the actual web pages by retrieving objects associated with the view schema from the database and populating the web pages.

[4] An instantiated template is a sequence of pure OQL statements.

[5] Although we distinguish between a transformation and a template, unless explicitly stated we use the term template to refer to both.

[6] Generating a schema implies generating an entire web site and hence a web-site layout, whereas generating only a type generates a web-page layout.

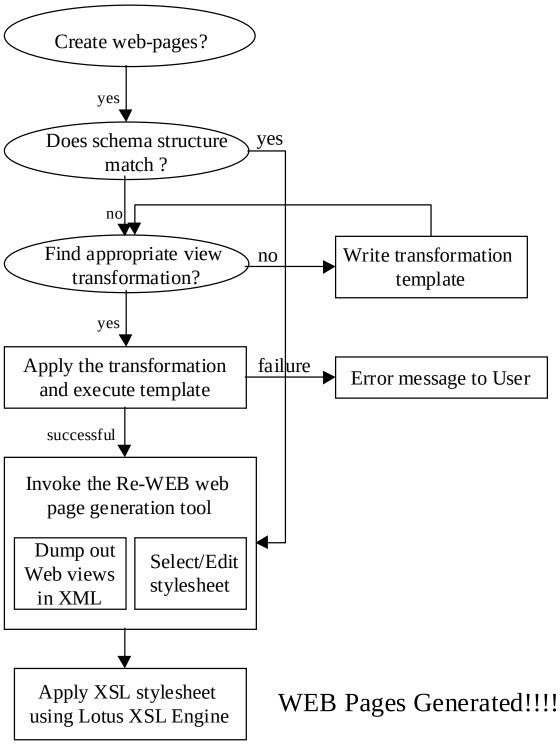

Figure 3: Interaction of the Different Modules in the Re-WEB Framework.

3.3 Re-WEB View Transformation Templates

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 layout of a web site. A view transformation combines a query language with a standard set of view and schema evolution primitives to produce other more complex view transformations.

This is an alternative view schema and web site view of the example in Figure 1. For example, in Figure 1, the layout of the web page reflects the structure of the given schema. However, this web-page layout may not meet the user's requirements and the user may instead desire a web view of the same data as depicted in Figure 4. The generation of the web view depicted in Figure 4 from the database schema given in Figure 1 first necessitates a re-structuring of the underlying database to correctly reflect the layout of the web-page. 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* 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 1 to the schema in Figure 4, the convert-to-literal of the attribute `taught_by` defined for the class Course from a collection of objects of type `Professor` to a collection of literals of `Struct-Professor` structure type. Figure 5 shows the *convert-to-literal* view transformation expressed in our framework using OQL, view creation

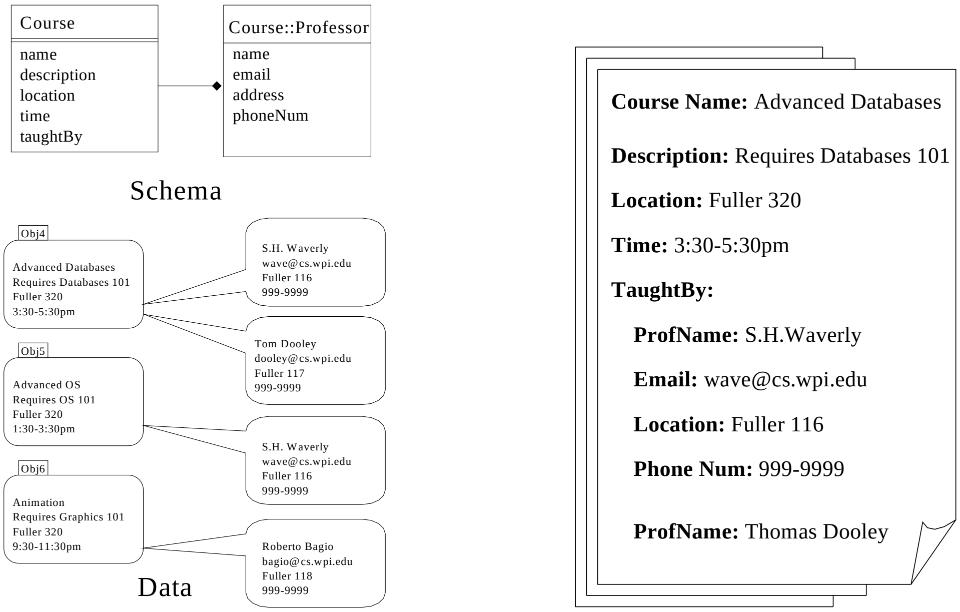

Figure 4: The DB Schema and the Matching Web Page Generated from them.

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 A: Query the MetaData.** To make a transformation general and re-usable for any possible schema in the form of a transformation template, it is necessary that a user be able to query the metadata using a query language. This information can then be used to make decisions about changes to the schema. In Figure 5 this step denoted by **Step A** retrieves the complex property type contained in the collection `taught_by`.

- **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 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 5 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 5 shows the primitive for creating the view class `CourseView` from the base class `Course` and 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 in guaranteeing the schema consistency after the application of a transformation [CJR98d]. The information gathered in *Step A* can provide the metadata to be changed as well as provide information needed for determining how to change the metadata, both serving as input to these schema evolution primitives. For example, **Step D** in Figure 5 shows the addition of an attribute to the view class `CourseView` through the *add_attribute* primitive.

```
// RefClass is the class that needs to be flattened.
   RefClass = element(
                  select a.AttrType
                  from MetaAttribute a                                              Step A
                  where a.AttrName = taughtBy
                  and a.ClassDefinedIn = Course) = Professor;

// a view definition.
   define ViewDef (ViewClass)
      select c                                                                      Step B
      from ViewClass c
      where c.department = "Computer Science";

// creating a view class for the main class, also
// invoke the view definition for getting objects for view class CourseView.
   create_view_class (Course, CourseView, ViewDef(Course));

// flatten the RefClass to a structure.                                            Step C
   create_view_struct (RefClass, Struct-Professor);

// add a new attribute Struct-Professor as a structure for CourseView.
   add_attribute (CourseView, Struct-Professor,
                  collection<Struct-Professor>, null);                              Step D

// Get all the objects of a class
   define Extents (ClassName)
      select c
      from ClassName c;                                                             Step E

//convert the collection of oids to a collection of structures.
   define flattenedCollection (Struct-Professor, Professor, Object, taughtBy)
      select (Struct-Professor *)p.*
      from Professor p
      where exists (p in object.taughtBy)

// for each object of CourseView, change it's taughtBy to Struct-Professor
// structure of its referring RefClass' object collection.                         Step F
   for all object in Extents(CourseView)
      object.set(object.Struct-Professor,
         flattenedCollection(Struct-Professor, Professor, object, taughtBy));

// remove the taughtBy from CourseView.
   delete_attribute (CourseView, taughtBy);
                                                                                    Step G
```

Figure 5: Convert-to-literal Transformation.

- **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, an identical view 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 5 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 schema structural changes and the system-defined functions for updating the objects can also be invoked from OQL, as in **D** and **Step F**.

A Re-WEB transformation as given in Figure 5 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 the notion from SERF [CJR98b] and introduces the notion of templates. A template is an arbitrarily complex transformation that has been encapsulated and generalized with a name and a set of parameters.

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 `taught_by` attribute in our example, and assigning a name to the transformation e.g., *convert-to-literal* in our example, a transformation becomes a *generalized reusable* transformation applicable to any application schema. Figure 6 shows the generalized *convert-to-literal* transformation of Figure 5 as a template. Vice versa, the *convert-to-literal* template shown in Figure 6 can be instantiated with the variables `Course` and `taught_by` and results in the Re-WEB transformation depicted in Figure 5. A Re-WEB template is thus a named sequence of OQL statements extended with parametrization that can be translated down to pure OQL statements during the process of instantiation.

```
Some named queries are pre-defined as follows:

// usign named query for view definition
define ViewDef ( ViewClass )
   select c
   from ViewClass c;

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

// convert the coll. of oids to a coll. of structs.
   define flattenedCollection( StructName, RefClass, Object, AttrToFlatten )
      select (StructName)p.*
      from RefClass p
      where exists( p in Object.AttrToFlatten )

begin template convert-to-literal (MainClassName, MainViewName,
                                    AttrToFlatten, StructName)
{

   // Define the class that needs to be flattened.
   refClass = element(
                 select a.attrType
                 from MetaAttribute a
                 where a.attrName = $AttrToFlatten
                 and a.classDefinedIn = $MainClassName );

   // Create a view class using the view primitive
   create_view_class ( $MainClassName, $MainViewName
                       ViewDef ( $MainClassName ));

   // Flatten refClass to a struct using the system method
   create_view_struct ( refClass, $StructName);

   // a new attribute to hold the structure in the
   add_attribute ( $MainViewName, $StructName,
                   collection<$StructName>, null);

   // convert the collection of oids to a collection of structures
   for all obj in Extents( $MainViewName )
      obj.set(obj.$StructName,
              flattenedCollection($StructName, refClass, obj, $AttrToFlatten));

   // remove the attributetoFlatten
   delete_attribute( $MainViewName, $AttrToFlatten);
}
```

Figure 6: Convert-to-literal Template.

Here we show an example of how to do such restructuring as multi-leveled nested inline by making use of the basic inline template, we could see that once we have some core templates for the basic transformation operations, we can flexibly construct arbitrary complex ones by easily reusing the core templates.

In this example of Figure7, we are assuming that we have a chain of classes, that is, we want to encapsulate all the information into the main class A1ClassName. These classes are chained together via the link A1AttrToFlatten in A1Classname to its referred class. In this example, we call such classes being referred to RefClass. Here we are given a chain of AttrToFlatten attributes through which we could follow

```
begin template Nested-convert-to-literal (A1ClassName, A1ViewName, A1AttrToFlatten,
               A1StructName, A2AttrToFlatten, A2StructName, .., AnAttrToFlatten, AnStructName)
{
      var A2RefClass, A2ViewName, .., AnRefClass, AnViewName;

      // find the class on the next nested level to be flattened.
      A2RefClass = element(
                 select a.AttrType
                 from MetaAttribute a
                 where a.AttrName = $A1AttrToFlatten and a.ClassDefinedIn = $A1ClassName );

      // define the view for the next nested level class.
      A2ViewName = (string *)A2RefClass + "View"; ..,

      // go deep down to the most nested level.
      A3RefClass = .., A3ViewName = ..,
      ..,
      AnRefClass = element(
                 select a.AttrType
                 from MetaAttribute a
                 where a.AttrName = $A<n-1>AttrToFlatten and a.ClassDefinedIn = $A<n-1>RefClass );

      AnViewName = (string *)AnRefClass + "View";

      // from the innerest level up to the top level, do convert.
      convert-to-literal ($AnRefClass, $AnViewName, $AnAttrToFlatten, $AnStructName)

      convert-to-literal ($A<n-1>RefClass, $A<n-1>ViewName,$A<n-1>AttrToFlatten, $A<n-1>StructName)

      ..,
      // finally, we get all levels of classes inlined in one class.
      convert-to-literal ($A1ClassName, $A1ViewName, $A1AttrToFlatten, $A1StructName)
}
```

Figure 7: Nested inline template by reusing basic inline template.

all the chained classes. Thus in the template we declare all these classes as A2RefClass, A3RefClass and so on through the last one that has an attribute to be flattened, also apply the same declaration processes for ViewName, i.e, A2ViewName, A3ViewName, etc. Then we could reuse the basic inline template we introduced before in Figure 6, inlining the neighbouring classes from the innermost cascadingly up to the top-most level. At the end, we will reach the main class A1ClassName now completely flattened out.

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 *reusability* of these transformations by parameterizing them. By the example of Figure 7, we could see that this thus makes the templates reusable and applicable to any application schema for the generation of a wide variety of web structures. Further we propose the development of a library of such templates. Such a template library could become an important resource in the web community much like the standard libraries in the programming environment.

4 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 [AM98] is one such example with its data model being based on extended OEM model–hypertrees, with abstractions for references, collections, nesting and ordering to model web structures. It's not as light as OEM [PGMW95] or similiar models and not as heavy-weight as the more traditional schema-based models. WebOQL synthesized ideas from diverse Web query languages, such as WebSQL [MMM96], W3QS [KS95] and UnQL [BDHS96], all of which share the notion of viewing the Web as a database that can be queried using a declarative language. WebOQL focusses on schema-free data modeling, thus supporting to capture web site structures and the restructuring of sites while querying the web. Our work now instead focusses on flexible and reusable mechanisms for web site construction and management exploiting standard schema knowledge and not on the issue of sprawling queries over the web.

Our Re-WEB approach has been inspired by web site restructuring systems like Araneus[AMM] and Strudel [FFLS97], and like 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 backend data repository. This thus forces them to flatten the structure of a web site down into relational tables when loading web data, and to again determine some means of adding web structure back to the tuples stored in the tables when presenting them back to the web. 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. For this reason, for the step of mapping from relational tables (or views) down to structured web pages, the Araneus project developed a separate web page layout language called Penelope that defines the web page structure by specifying nested page layout statements. This is not needed in our Re-WEB approach where the hyper-graph structure of a web site can be modeled by simply associating implicit web semantics with the standard ODMG object model.

Strudel [FFLS97] advocates the separation of a web site management system into three separate layers: the underlying data model, the logical site structure and the final visual presentation of web pages. Strudel uses a graph-based data model with nodes representing either documents or atomic values and with arcs with strings standing for attribute names. Being based on a proprietary hypergraph model, Strudel also proposes a new language for manipulating and querying this graph. Any such approach based on a new proprietary model and language has to re-investigate many database issues that otherwise could be addressed by already established technologies for commercial database systems. For example, to efficiently maintain the consistency between web site views and the database, new view maintenance techniques may have to be designed, even though such problems already have solutions in standard OODB technology. For this reason, our approach instead is firmly grounded on standard OO modeling and particular the ODMG object model, allowing us on the one hand to 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 easily transferrable to other systems that are based on the ODMG standard.

To summerize, here we give a comparison between our Re-WEB approach with other systems from such perspectives as the data sources they are exploiting, the data models they are based on, their respective query languages and querying capabilities, their restructuring methods and their web page rendering mechanisms.

	Strudel	**WebOQL**	**Araneus**	**ReWeb**
Data Source	**Semi-Structured Data**	**Semi-Structured Data**	**Data Stored in RDB**	**Data Stored in OODB**
Data Model	**OEM**	**Extended OEM**	**Relational Model**	**ODMG Model**
Query	**Gragh Traversal**	**Gragh 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**

Table 2: A Comparision Between Related Systems

From Table2, we observe that these different web site management systems make different assumptions to their systems. Both Strudel and WebOQL are targetting at semi-structured data available on the web, the data source has little restriction to its schema and thus is modeled as a light weighted object model – Object Exchangable Model (OEM), which is basically a graph of notes and thus the main query operation on the data model can be treated as graph traversal; Araneus project is using a relational model to manage their source data (or anyway, they need a strict way to convert the web based available data into the relational model). And they have their own ADM model as the page scheme onto which the source data can be mapped (basically, the mapping are embeded selection, projection and join operations) using their self-defined language Penelope; Similiar to Araneus system, our Re-Web project also relies on the mature database techniques but makes use of ODMG model to define web semantics. This approach is natural in the sense that there indeed exists much information somewhat restricted by structure and could be modeled, managed and queried by OODB.

For the task of restructuring the web site, Strudel system needs to have the knowledge of the data graph beforehand and then performs the restructuring instruction while WebOQL can do the restructuring dynamically while querying the gragh. Araneus uses their own page schema define language (i.e, Penelope) to do the constructing of the web page schemes from the data cells in the relational tables. In our Re-Web approach, OQL query language and SERF template can be utilized to restructure the database and consequently to specify object views that effectively model restructured web sites.

As for the final web page rendering phase, WebOQL has no separate such phase while Strudel and Araneus apply HTML template and self-defined stylesheet respectively, all of them are using procedural stylesheets to implement the formatting process. As a constract, Re-Web project adopt a declarative stylesheet – XSL to provide constraints to the formatting process, which is emprinted the describing characteristics and constraints versus saying what to do using programming (scripting) languages.

To date, there are already many commercial tools out there for having web-pages constructed almost automatically based on a well-structured database. It's true that a webmaster or system administrator can build dynamic pages much faster and easier using Active Server Pages by the tools of Microsoft, like InterDev. These approaches are based on the technologies for the postprocessing of the query results from the backend databases. Their methods start from the point that the results are already uploaded to the client sides from the databases. Whatever the web pages they restructure or the web applications they develope would have no effect at all on the underlying databases nor have any way being reflected there and thereby would not be reused.

As we mention in Table 2, Re-Web dumps the web views out in XML format and then flexibly applies diverse XSL style sheets to generate the most effective web pages. We are choosing XML since it is receiving tremendous praise for being the universal data format for the Web, and there exist OODB companies that have commercial XML tools for creating data-driven Web applications with high-performance and scalable data management services for server-side XML. For example, Object Design Inc. bases its many products on ObjectStore, a leading object database management system (ODBMS), providing full server-side XML data management.

Distinguished from many such approaches, which are based on new proprietary models and languages and have to re-investigate many database issues, Our approach instead 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 transferrable to other systems that are based on ODMG.

5 Conclusions

In this paper we have presented a database-centric approach for flexible web site generation, restructuring, and maintenance, called Re-WEB, that is completely based on object-oriented database technology. A large class of web site layouts (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 reusable templates. Genericity and reusability 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. 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 focussing on the issue of reusable view generation templates.

Acknowledgments. The authors would like to thank students at the Database Systems Research Group at WPI for their interactions and feedback on this research. In particular, we are grateful to Jin Jing, Chandrakant Natarajan, Xin Zhang, Anuja Gokhale, Parag Mahalley, Swathi Subramanian and Jayesh Govindrajan for their input on the implementation of the SERF and CHOP tools using PSE Pro2.0.

References

[AM98] G. Arocena and Alberto Mendelzon. WebOQL: Restructuring Documents, Databases, and Webs. In *In IEEE Int. Conf. on Data Eng.*, pages 24–33, 1998.

[AMM] P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. In *VLDB'97*, pages 206–215.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada*, pages 505–516, 1996.

[BKKK87] J. Banerjee, W. Kim, H. J. Kim, and H. F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD*, pages 311–322, 1987.

[Cea97] R.G.G Cattell and et al. *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann Publishers, Inc., 1997.

[CJR98a] K.T. Claypool, J. Jin, and E.A. Rundensteiner. OQL_SERF: An ODMG implementation of the template-based schema evolution framework. Technical Report WPI-CS-TR-98-14, Worcester Polytechnic Institute, July 1998.

[CJR98b] K.T. Claypool, J. Jin, and E.A. Rundensteiner. SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework. In *Int. Conf. on Information and Knowledge Management*, pages 314–321, November 1998.

[CJR98c] K.T. Claypool, J. Jin, and E.A. Rundensteiner. SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework. In *Int. Conference on Information and Knowledge Management*, pages 314–321, November 1998.

[CJR98d] K.T. Claypool, J. Jin, and E.A. Rundensteiner. SERF:Schema Evolution through an Extensible, Re-usable and Flexible Framework. Technical Report WPI-CS-TR-98-9, Worcester Polytechnic Institute, May 1998.

[CL97] I. F. Cruz and W. T. Lucas. Delaunay: a Visual Framework for Multimedia Presentation. In *IEEE Symposium on Visual Languages (VL '97)*, 1997.

[Cla98] James Clark. *Jade from James Clark: http://www.jclark.com/jade/.* 1998.

[Clu98] S. Cluet. Designing OQL: Allowing objects to be queried. *Journal of Information Systems*, 23(5):279–305, 1998.

[CRCK98] K.T. Claypool, E.A. Rundensteiner, L. Chen, and B. Kothari. Re-usable ODMG-based Templates for Web View Generation and Restructuring. In *CIKM'98 Workshop on Web Information and Data Management (WIDM'98), Washington, D.C., Nov.6*, 1998.

[FFLS97] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for a Web-Site Management System. *SIGMOD*, 26(3):4–11, September 1997.

[ie498] *XSL support in IE4 from Microsoft: http://www.microsoft.com/xml/xsl/.* Microsoft Inc., 1998.

[Jin98] J. Jin. An Extensible Schema Evolution Framework for Object-Oriented Databases using OQL. Master's thesis, Worcester Polytechnic Institute, May 1998.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the World Wide Web. In *Int. Conference on Very Large Data Bases*, pages 54–65, 1995.

[MMM96] A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. In *Conference on Parallel and Distributed Information Systems*, pages 80–91, 1996.

[Nat98] C. Natarajan. CHOP: An Optimizer for Schema Evolution Operation Sequences. Master's thesis, Worcester Polytechnic Institute, June 1998.

[O'B97] P. O'Brien. Making Java Objects Persistent. *Java Report*, 1(1):49–60, 1997.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *IEEE Int. Conf. on Data Engineering*, pages 251–260, 1995.

[Tho98] Henry Thompson. *XSLJ from Henry Thompson: http://www.ltg.ed.ac.uk/ ht/xslj.html.* 1998.

[xml98a] *Extensible Markup Language (XML^{TM}): http://www.w3.org/XML/.* World Wide Web Consortium, 1998.

[xml98b] *XML Styler from ArborText: http://www.arbortext.com/xmlstyler/.* Arbortext Technology, 1998.