Data-Driven One-to-One Web Site Generation for Data-Intensive Applications*

Stefano Ceri Piero Fraternali Stefano Paraboschi

Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci 32, I-20133 Milano, Italy
{ceri,fraterna,parabosc}@elet.polimi.it

Abstract

A data-driven approach can be fruitfully used in the specification and automatic generation of data-intensive Web applications, i.e., applications which make large amounts of data available on the Web. We present a multi-level architecture based on orthogonal abstractions for the definition of the structure, derivation, navigation, composition, and presentation of Web sites; then we show how these ingredients are used in *Torii*, a tool environment for the specification and automatic generation of Web sites, currently developed in the context of a large Esprit project.

By means of design tools, specifications are collected in a design repository, which is next used for Web page generation. This dynamic, data-centered approach opens up opportunities for personalizations: each user can be mapped to an individual hypertextual view of the Web site (called *site view*), and business rules may be used to change site views, both statically and dynamically. We argue that personalization of Web access (also called *one-to-one Web delivery*) is naturally supported by the proposed data-driven approach, and is at the same time a key ingredient of the Web applications of the near future.

We acknowledge the support of ESPRIT Project 28771 W3I3, MURST Project Interdata, CNR-CESTIA, and the HP Internet Philanthropic Initiative.

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

1 Introduction

The integration between Web applications and DBMS technology is subject to continuous and fast evolution; a variety of new technologies are being developed and brought to the market. Tools like Microsoft's Active Server Pages, Allaire's Cold Fusion, and many others, greatly simplify the implementation of integrated Web-DBMS sites. However, these technological advances are not matched by parallel efforts in data abstraction and modeling. Although in the community of database research it has been widely accepted that the "Web changes everything" [4], little effort has been devoted so far to adapt data design methods to the use of the Web as the fundamental data interface.

This paper is concerned with extending the classical data-centric approach to database application design by incorporating Web-related concepts into it. This effort can also be seen specularly from the viewpoint of Web design methods and tools, as an attempt at making them aware of data design aspects.

This methodological approach is well suited to the category of *data-intensive Web applications*, i.e., those Web sites whose primary purpose is to present a large amount of content to a variety of possible users. With respect to traditional database applications, data-intensive Web sites, have the following differences:

- Simpler functional requirements: typically a data-intensive Web site must offer a generic user the possibility to browse a large collection of data, in a way that fulfills some application-specific goal (e.g., in electronic commerce, showing to each user exactly those goods that he will most likely purchase). Sophisticated interaction control is not normally necessary, because the flow of activities is determined by the user via browsing.
- Simpler transactional requirements: in most

cases, it is sufficient to offer high-performance read-only access, and write access is reduced to a few standard operations on a well-delimited fraction of the data (e.g., in electronic commerce, the addition of items to the users' shopping cart).

- Focus on interface organization and ease of navigation: users must immediately grasp the way in which the site is structured, be offered a rich variety of navigation options, and be always confronted with a carefully crafted and appealing presentation of the information.

- Support of *one-to-one* Web delivery [11]: each user must have the impression of interacting with the application by means of a dedicated interface, specifically tailored to his needs and preferences. One-to-one delivery not only requires identifying users and their preferences, but also tracking their interaction with the application and updating the interface dynamically to reflect any improved understanding of the user's needs, typically by means of reactive mechanisms (such as the so-called business rules).

- Support of *multi-device output generation*: with the availability of the Internet on such diverse devices as cellular phones and digital television, content delivery must be automatically tuned to different output languages (e.g., HTML, HDML [13], ATVEF [1]) and rendition capabilities.

We claim that current database and Web design methods and tools are insufficient for coping with the development of highly personalized and dynamic data-intensive Web sites, and that new design abstractions are in order, supported by adequate tools. This paper is concerned with Torii, a tool environment for the specification and automatic generation of Web sites [1]. Torii is developed as part of the W3I3 (Web-based Intelligent Information Infrastructure) Project[2], whose goal is to develop the abstractions, technologies, methods, and tools to support one-to-one data-intensive Web applications.

[1] Torii, an ancient Finnish word, is the signpost that in medieval Scandinavia was used to indicate the Market Place, i.e. the place in the village were people met to chat and exchange goods.

[2] W3I3 is a 3 Million Euro Project of the Esprit IV Framework, sponsored at 50% by the EU. The W3I3 Consortium includes Politecnico di Milano as technology provider, TXT Ingegneria Informatica from Italy as software integrator, Digia (Digital Information Architects) from Finland as responsible of interface design and exploitation. The W3I3 Consortium also includes two pilot users with huge Web applications: OTTO Versand from Germany, the world's largest mail order company, and KPN Research from the Netherlands, the research branch of the major telecom company of Holland.

1.1 Related Work

The Torii proposal builds on the vast body of research that has been devoted to data design techniques, tools, and methods [3], and to the similar approaches developed for hypermedia applications [10, 12]. In fact, in the software industry most of the major data development projects are conducted by first giving an abstract representation of the data content (also called a "conceptual schema"), and then mapping such content to a logical and physical representation [3]. The availability of an abstract, implementation-independent schema is useful not only during design, but also for maintenance, quality assessment, and reverse engineering of data-intensive applications. We expect that a similar, high-level approach to data design for industrial applications will soon characterize the development of data-intensive Web sites. Indeed, several recent projects, namely Araneus [2], Autoweb [9], WebArchitect [12], and Strudel [7], have proceeded along this direction.

In particular, Araneus starts from the description of the content and then derives and/or integrates Web sites; with respect to Araneus, Torii adds to Web modeling the dimensions of presentation design, user modeling, customization and business rules. Strudel proposes a novel way of developing Web sites based on the declarative specification of the site's structure and content by means of queries. In Strudel, the specification of navigation is not orthogonal to structure and presentation, because navigable links and index collections are specified together. An in-depth comparison of Autoweb, Araneus, and Strudel appears in [5].

1.2 Comparison with Commercial Tools

The Torii system fits into the broad market of tools for designing a Web application. We give a concise classification of them, and indicate their main merits and limitations.

- *Visual HTML editors* and *site managers* (like, e.g., NetObject's Fusion, Macromedia's Dreamweaver and Microsoft's FrontPage) concentrate on HTML production and do not support the integration of large amounts of data.

- *HTML-DB integrators* (like, e.g., Microsoft's Active Server Pages (ASP) and Cold Fusion) provide a way to scale the dimension of a site by producing HTML pages dynamically from database content, but are implementation-level tools and do not address conceptual modeling and site customization.

- *Web application generators* (like, e.g., Oracle Designer 2000 Web Generator and Hyperwave) start from conceptual modeling and produce the Web site automatically, but have limitations in the expressiveness of the concepts available to the designer for structuring a Web site. They do not

provide a description of user-oriented site views, nor business rules.

- *Rule-driven Web site generators* use rules extensively for matching content to users. The most relevant product in this class is Broadvision (http://www.broadvision.com), which is very powerful in matching content to users based on profile information, transaction history, session behavior, and other data. Broadvision does not support the separation between structural, composition, navigation, and presentation aspects, therefore Broadvision rules apply mostly to content delivery and not to the other aspects of a Web site's delivery.

For a thorough review of the state of the practice of commercial tools for Web design, we refer the reader to [8].

1.3 Outline of the Paper

The main research objective of W3I3 is to rise the level of abstraction of a site's specification, by enriching and refocusing the classical models for database and hypertext design; in Section 2 we will show how Torii supports the five orthogonal dimensions of structure, derivation, navigation, page composition, and presentation. We will discuss in Section 3 how Torii supports personalization, by introducing both a declarative approach, based on data derivation, and a reactive approach, based on business rules. Then, in Section 4 we will describe the implementation of the Torii system, and show that Torii introduces important novelties with respect to the predecessor Autoweb. Finally, we will discuss in Section 5 some preliminary experiences of use of Autoweb, Torii's predecessor, so as to justify the most important decisions that were taken in Torii.

2 Support of Orthogonal Abstractions in Torii

Modeling Web sites according to the Torii approach and tools consists of specifying five orthogonal perspectives: structure, derivation, navigation, page composition, and presentation. This multi-level organization has been motivated in [5]. We here give emphasis to the innovative features of the Torii system.

2.1 A Multi-Level Architecture

The Torii architecture is an extension of the classical multi-level architecture of database systems.

- The *Physical Level* describes the low-level organization of data. Torii will use standard relational technology for implementing this layer, for reasons of availability, standardization and efficiency.
- The *Structural Level* describes the high-level organization of data using the Entity-Relationship model, with the additional notion of "target", which describes how concepts should be aggregated within applications.
- The *Derivation Level* denotes how new concepts can be derived from the concepts of the structural schema. It is analogous to the definition of views commonly found in database applications.
- The *Composition Level* describes how the concepts of the structural schema are mapped to Web pages. Composition-level constructs, called *site views*, group related pages, which fulfill common user requirements; they are analogous to different external schemas of the same database.
- The *Navigation Level* describes the way in which relationships among data should be translated into hypertextual links.
- The *Presentation Level* is responsible of defining the appearance of Web pages, independently of the language used for page construction (HTML, XML, etc.).

All the above levels are associated both with graphic and textual descriptions: each concept is first provided graphically by using the Torii WYSIWYG interface and then mapped to its textual form. The complete XML syntax of Torii models can be found in [6], while a preliminary description of the user interface is given in [14]. We next describe each level separately, except the physical level, which is automatically generated by the Torii system.

2.2 Structural Model

The structural model of Torii defines the organization of the structured data which is used by the application; it is independent from the data models used by the data sources, where such content is stored. The model is based on the classical Entity-Relationship model, and uses the concepts of *entity*, *attribute*, and *relationship*. All relationships are binary and have no attributes. The model also supports classical *generalization hierarchies*. Properties with multiple or structured values constitute *components* and correspond to the classical part-of relationship. Each entity instance, or object, has an *object identifier*, and each relationship and component has *cardinality constraints* (min and max values). In addition, concepts of the structural model are clustered into *targets*; each target represents an application object of the real world, that may be represented by means of several concepts of the structural schema.

Torii offers two important predefined targets, *Profile* and *Metadata*, whose structure is described by means of the Torii structural model. The *Profile* target

contains information regarding users and groups. For each user, we collect identification information (identifier, email address), login history, group membership, and trace information (the URLs of pages accessed at given times). Groups are associated to site views, i.e., to collections of pages which constitute a Web site.

```
<TARGET id="Profile" entryEntity="User">
 <ENTITY id="User">
  <ATTRIBUTE id="UID" type="Number"/>
  <COMPONENT id="Trace" minCard="0" maxCard="N">
    <ATTRIBUTE id="URL" type="Url"/>
    <ATTRIBUTE id="TraceTime" type="Time"/>
  </COMPONENT>
  ...
 </ENTITY>

 <ENTITY id="Group">
  ...
 </ENTITY>
</TARGET>
```

Profiles can be specialized for each application context. For instance, a specialization of the entity *User* in the context of electronic commerce is indicated below. Customers provide their name, age, sex, birthday, and purchase preferences, and are classified by the system (e.g., according to the wealth of their district). The system tracks automatically their last purchases (at most ten).

```
<ENTITY id="Customer" superEntity="User">
  <ATTRIBUTE id="Name" type="String"/>
  <ATTRIBUTE id="BirthDate" type="Date"/>
  <COMPONENT id="PurchasePreferences" minCard="0"
             maxCard="10">
    <ATTRIBUTE id="ProductType" type="String"/>
    <ATTRIBUTE id="Rank" type="Number"/>
  </COMPONENT>
  ...
</ENTITY>
```

In a similar manner, the *Metadata* target enriches objects with information concerning their creator, the modification and usage pattern, their validity, and possibly some user or group-specific access right specification. Also in the case of metadata, it is possible to introduce specializations to reflect the needs of a specific application. Metadata are reachable from the concepts of the structural schema by means of an implicit *Meta* relationship, that is assumed between each concept and the corresponding meta-information.

2.3 Derivation Model

Derivation is the process of defining as many conceptual views of the data as needed to support the various interfaces required by the different users of the Web application. Torii permits the definition of derived attributes, components, relationships, and entities.

The derivation is based on Torii-DL (Derivation Language), whose syntax and semantics are defined in [6]. The language supports path expressions, used for traversing relationships and for accessing components. A predefined *Self* variable represents the entity for which a derived element is being defined. An *ISA* predicate allows checking for membership of instances in a sub-entity.

Path expressions, variable declarations and the management of components are inspired by OQL, embedded within a restricted syntax that permits an efficient and simple translation to the underlying relational model. Each derived concept is translated into a view on the relational database managing the data.

In the example below, a derived attribute is added to entity *Book*, to add a "New" icon to the books which have been introduced into the system by less than 45 days.

```
<ATTRIBUTE id="NewIcon" type="Image"
   value="'icons/new.gif'
          where Self.Meta.Created > Sysdate-45"/>
```

Another derived attribute presents a *Special Offer* if the user accessing the data is member of the *Best Customers* group[3].

```
<ATTRIBUTE id="Special Offer" type="Number"
   value="Price*0.8
          where User.OfGroup.Name='Best Customers'/>
```

2.4 Composition Model

Composition is the process of specifying the content of each page; pages may be associated to an entire target, or to a specific entity within a target, or to a specific component within an entity. In order to preserve coherence between the structural and composition schemas, a page cannot arbitrarily intermix information coming from multiple targets; however, by means of derivation it is possible to extend the information content of a target with derived attributes, components, and relationships, computed on the basis of other related targets.

Conversely, each concept may be described by means of multiple pages; this feature allows the designer to represent the information on the same real-world object in different ways for serving the needs of different users. A complete personalized view of the application (called a *site view*) is constructed for a class of users, by defining a set of user-specific pages and the connections between them. The same concept of the structural model can be mapped to multiple pages belonging to the same site view.

The definition of a page for a target, entity, component, or collection of the structural schema requires:

[3] In both examples of derived attributes, the value is null if the predicate is false; we anticipate that Torii' style sheets, describing the presentation aspects of a given Web page, automatically omit to generate those regions corresponding to null attributes, and rearrange automatically the surrounding regions. Thus, derived attributes whose value is null are automatically "removed" from a given page.

- The mapping between structure and page content. By default, all attributes, components, and relationships of a given target, entity, or component are included into the page (including the derived ones), but the designer may explicitly exclude some of them to tailor the content of a page to the requirements of a class of users.
- The choice of the collections to be included in the page. This aspect is discussed in Section 2.5.2.
- The choice of the pages that are reachable from the page currently being defined, by means of relationships or components. This process is recursive, and is completed when all page references are resolved.

All of the above steps are optional and have default rules; in this way, composition modeling can be entirely skipped, e.g., for Web applications that present a single site view, offer a single page for each concept of the structural model, and do not need any form of personalization.

2.5 Navigation Model

Navigation is the fundamental access paradigm of Web applications: pages are visited by following links connecting anchors to destination pages. A fundamental assumption of Torii is that most navigable links between pages in data-intensive Web sites correspond to conceptual connections in the structural model, i.e., reflect the semantic organization of the site expressed in its structural schema; these are called *contextual navigations*, to denote the fact that the navigation occurs from a given object to a related object in the given semantic context. However, some other *non-contextual* navigations may occur, which enable the user to access other objects regardless of the current semantic context. Coherently with this assumption, Torii navigation modeling requires the specification of two complementary, but orthogonal, aspects: *object traversal* and *non-contextual data access*.

2.5.1 Object traversal

Object traversal specifies how relationships and components of the structural schema are used to navigate from one concept to a related concept. Navigation semantics is straightforward when moving from one page to exactly another page, e.g., following a one-to-one relationship. However, in many cases Torii applications need to support *one-to-many navigation*, e.g., from an entity to the elements of a component, or to a set of entities connected by an n-ary relationship.

For the sake of abstraction, in the following we call *navigable container* any set of related items that must be navigated as a result of a user request. The operational semantics of container navigation is defined by the designer, who establishes the *navigation mode* of the container by considering five independent dimensions:

- *Filtering*: enables the definition of filter predicates that select a subset of the objects of the container.
- *Indexing*: presents representative information of each selected element of the container within an index, enabling a further selection by the user.
- *Showing*: indicates how many selected objects are presented on the same page.
- *Sorting*: defines the order in which the selected objects of the container are presented.
- *Browsing*: indicates the possibility of scrolling from one element of the container to another one, e.g., to the next or previous one.

By assembling different values for the above dimensions, the designer customizes the navigation modes which are most suitable for the application. In addition, Torii supports a rich collection of *predefined navigation modes* which correspond to the choices along the above dimensions which are most commonly used together:

- *Index*, with two variants of location (*detached* and *embedded*); it offers no browsing. A *detached index* may have a filter, producing the *filtered index* mode, also without browsing. The filter is retained for repeated searches.
- *Guided tour*, which presents the objects in a sequence which can be browsed using all the available browsing commands. A variant, the *indexed guided tour*, adds a detached index, so as to enable an initial jump into an arbitrary position of the sequence.
- *Show all*, which presents the elements of a container contiguously, one after the other, either embedded in the same page (*embedded show all*) or detached in a separate page (*detached show all*).

The default modes are summarized in Table 1; when the designer does not provide a navigation mode for a container, the mode *detached index* is the default navigation mode, except for components, whose default is the *embedded show all* mode.

Figure 1 shows the pages that correspond to five different ways of navigating the relationship between an author and his books. Figure 1.a illustrates the *detached index* mode: by clicking on the *AuthorToBook* link, an index of books is presented and after selecting one entry, a specific book is displayed. Figure 1.b illustrates the *guided tour* mode: by clicking on the *AuthorToBook* link, the first book (in descending order of year) is shown and commands are available to

	Sorting	Filtering	Indexing	Showing	Browsing
Detached index	Yes	No	Detached	One	No
Filtered index	Yes	Detached	Detached	One	No
Embedded index	Yes	No	Embedded	One	No
Guided tour	Yes	No	No	One	Yes
Indexed guided tour	Yes	No	Detached	One	Yes
Show all	Yes	No	No	All	No
Show random	No	No	No	All	No

Table 1: Synopsis of the most common navigation modes

scroll the books of the same author. Figure 1.c illustrates the *filtered index* mode: by clicking on the *AuthorToBook* link, a form is presented in a separate page to filter the books to be seen (by publisher and year), then an index of the books that satisfy the filtering condition is presented from which a specific book can be accessed. Figure 1.d illustrates the *detached showall* mode: by clicking on the *AuthorToBook* link, all books are presented together in a new page. Finally, Figure 1.e illustrates the *embedded showall* mode: all books are presented together on the same page of their author.

2.5.2 Non-contextual Data Access

Collections are sets of objects which are meaningfully grouped together; they can be collectively accessed from a user regardless of the current context [4]. Collections can be used to access a Torii application from "outside" (e.g., from the home page); however, they can also be used to move freely within an application. For instance, when a page represents a specific course, the page can include the collections of all courses, of the offered degrees, or even of the services offered by the university. A collection could enclose advertising material reachable from a given page but not related to the page's content.

Enclosing a collection in a page means to add a navigation option anchored to the collection's name. If the collection has a singleton element, that element is immediately shown after activating the link. Otherwise, the collection acts in the same way as a container, and the same mechanisms described for object traversal are applicable for selecting and reaching individual elements of the collection from the collection's name. Torii collections can be hierarchically structured to form collections of collections, and their member objects may be defined *intensionally*, i.e., by means of Torii-DL expressions, or *extensionally*, i.e., by explicit enumeration.

2.6 Presentation Model

Presentation is concerned with the look and feel of Web pages, in particular with the design of the general page layout, with the placement of specific pieces of information on the page, and with the selection of graphical resources like backgrounds, icons and animations.

The basic unit of presentation is the page; each page is associated to one or more *style sheets*, each specifying a different way of presenting the page instances on the screen. Style sheets are formally expressed in XML and can be defined visually by means of suitable WYSIWYG tools.

The style sheet language contains the following elements:

- A sublanguage to define metric spaces, i.e., regions of the screen which can host the presentation of a page. Presently, such regions are bidimensional, but the style sheet language is open to the description of multidimensional metric spaces.
- A sublanguage to define the internal structure of screen regions. Such language is based on an extended notion of grid, enabling overlapping regions and flexibly defined multispan regions.
- A sublanguage to define presentation panels, i.e., the content associated to each screen region. Panels are recursively constructed from atomic elements, which permit the insertion into a page of either language-dependent pieces of content (e.g., HTML text), or of content extracted from the database (e.g., an attribute, a component, or an outgoing relationship). If the content of a given conceptual element is null, the corresponding region is not generated, and surrounding regions are automatically rearranged. The goal of this sublanguage is to allow the designer to define panels at a varying degree of granularity and completeness, to enable the construction of reusable style sheet libraries.

A *default* page style is generated for each page, based on a very simple layout. The screen is separated into five predefined regions by means of a grid. The upper region contains the page header; the left region contains icons enabling non-contextual navigation; the two regions immediately below the header contain icons enabling contextual navigation, respectively on components and relationships; finally, the

[4] In hypertexts and hypermedia, collections are also called "indexes"; we avoid this term which is already quite overloaded.

Figure 1: Examples of Navigation Modes

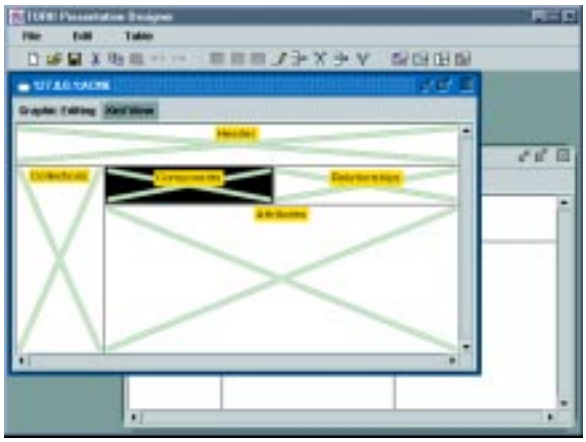

Figure 2: The Torii Presentation Designer

lower right region occupies most of the page and contains the actual content extracted from the database. Figure 2 shows the interface of the style sheet editor operating on the default style. The predefined panels are personalized on the specific entity by supplying entity-specific graphical resources (icons, banners).

Various languages (HTML 3.2, HTML 4, VRML, ATVEF [1], HDML [13], etc.) can be used to implement the presentation pages. Torii provides optional language-dependent attributes, which can be associated to the elements of a style sheet to improve the presentation in a specific language.

3 One-to-One Delivery

One-to-one delivery indicates the ability of personalizing a Web site so as to offer each user a "dedicated view" of the site. Personalization in Torii occurs at three levels.

- At the composition level: the designer builds site views suitable for given groups of users. The user, when connecting to the Web site, provides enough information in order to be classified as member of a given group, and consequently sees a given site view.

- At the derivation level: the designer defines intensional concepts (e.g., attributes, components, collections) whose definition may depend on user-specific data and metadata. In this way, customization is declaratively specified by the designer, and then the runtime page generator computes and presents the information specific to a given user.

- At the business rule level: the designer writes business rules to handle personalization. Typically, business rules manipulate user-specific information, and this results in a change of the site's organization or presentation.

As we have already discussed the possibility of building distinct site views for each group of users, we concentrate on the remaining two options, which are the most innovative.

3.1 Declarative Personalization

Profile data can be used to customize the site in a declarative way. To this end, the content of derived concepts can be defined in terms of the profile data of the specific user who is accessing the site. For example, in an electronic commerce site, the *SpecialOffers* collection can be tailored to the purchase preferences of the user, stored in the profile data of the entity *Purchaser*. To do so, it is sufficient to define a derivation query that looks up user profile data to select matching products among those currently discounted, as follows:

```
<COLLECTION id="SpecialOffers" range="Product"
     value="Product as P
            where P.Type in
              User.PurchasePreferences.ProductType
            and P.Discount > 0.2"/>
```

Customization takes place automatically by including the above collection in a page visible to users of type *Purchaser*. If the query fails (e.g., because the user has no known purchase preferences or because there is no special offer matching his preferences), then no collection is shown; if the derivation query succeeds, the right offers are automatically included in the pages seen by the individual user.

As another application, consider the addition of a collection of items based on the shopping history. Assume that certain products have high resale potential (e.g., books, magazines, CDs); then, when one such item is part of the purchase history of a given user, we collect in the *PotentialResales* collection all items of the same category which were added to the site in the last thirty days. The collection can then be highlighted with an attractive presentation that indicates the new releases of products belonging to the user's shopping history.

```
<COLLECTION id="PotentialResales"
      range="Product"
      value="Product as P
             where P.Category in
               User.LastPurchase.ProductCategory
             and P.ProductResalePotential = 'Y'
             and P.Meta.Created > Sysdate - 30"/>
```

Even more simply, personalization can take place by means of derived attributes whose value is linked to personal properties of the user; e.g., the following *WelcomeMessage* is displayed only on the user's birthday. Style sheets may associate to the welcome message any kind of visualization, generated only if the attribute is not null.

```
<ATTRIBUTE id="WelcomeMessage"
    value="'Special welcome in your birthday!'
           where User.Birthday = Sysdate"/>
```

3.2 Business Rules

Business rules are specified according to the well-known event-condition-action paradigm: they are triggered by a given event, once triggered the condition is considered, and if the condition is true then the action is executed. Rules have tuple-level granularity, therefore each elementary data change event is followed by the triggering of the rules related to it. Rules triggered by the same event may be prioritized; a priority is a positive number, and higher numbers denote higher priorities.

Events considered by rules are classified as *clickstream* and *data change* events. The former include the start and end of a session and each new page access. When the event is a new page being accessed, the event may be specified as relative to a specific entity, denoted by its name.

Conditions evaluate predicates or queries over the database. The condition is satisfied if the predicate is true or the query result is not empty; bindings produced in the evaluation of the condition are available for the execution of the action. Conditions may be missing, in which case the action is always executed. In the evaluation of conditions, special variables are associated to the triggering event. In particular:

- When the event is a clickstream event, the variable *User* denotes the user who performed the clickstream action.
- When the event is a page access, a variable associated to the event is bound to the given entity instance being accessed.
- When the event is a data change, variables *Old* and *New* denote, respectively, the values of the entity instance before or after the operation.

Actions performed by rules include adding or dropping elements from collections, executing simple data updates, sending mail to users or assigning them a given site view.

We next present several examples of business rules, classified according to the specific application being performed by them. A precise characterization of the syntax and semantics of rules can be found in [6].

3.2.1 Rules for User Classification

These rules are focused on assigning a given view to the current user. The condition is typically a predicate on the current user; the action is an assignment of the (current) site view. Rules are executed in priority order, but as soon as one of them assigns a site view to the current user, this value cannot be further updated within the session. The two rules below assign suitable site views to users who classify respectively as kids or as frequent purchasers; the former criterion is predominant, as indicated by the rules' priority.

```
<RULE id="ClassifyKid">
    <EVENT eventType="SessionStart"/>
    <CONDITION predicate="User.Age < 12"/>
    <ACTION action="Assign(SiteView,'SV-KIDS')"/>
    <PRIORITY value="100"/>
</RULE>

<RULE id="ClassifyFrequent">
    <EVENT eventType="SessionStart"/>
    <CONDITION
       predicate="User.LastLogin - Sysdate < 10"/>
    <ACTION action="Assign(SiteView,'SV-FREQ')"/>
    <PRIORITY value="50"/>
</RULE>
```

The advantage of using business rules instead of a static grouping of users and assignment of site views to groups lies in the higher dynamicity and ease of evolution.

3.2.2 Rules for Managing User-Defined Collections

Business rules may be used for managing user-specific information, so as to increase the information about the user and thereby enable current or future personalizations. The following rule adds to the user-defined collection *User.VisitedBook* the book currently being visited.

```
<RULE id="AddToVisitedBooks">
    <EVENT eventType="newpage(Book as B)"/>
    <ACTION action="add(User.VisitedBooks,B)"/>
</RULE>
```

The following two rules add to (drop from) the collection *Group.Chatline*, associated to each group, the user who is starting (ending) a session. On the basis of this information, newsgroup management software may be used to enable conversation among these users so as to facilitate their on-line exchange of information and mutual advice about possible purchases.

```
<RULE id="AddToActiveUsers">
    <EVENT eventType="SessionStart"/>
    <ACTION action="add(Group.Chatline,User)"/>
</RULE>

<RULE id="DropFromActiveUsers">
    <EVENT eventType="SessionEnd"/>
    <ACTION action="drop(Group.Chatline,User)"/>
</RULE>
```

3.2.3 Rules for Pushing Information to the Users

A typical example of push technology in electronic commerce is concerned with providing a user with information about new purchase opportunities. The following rule reacts to the insertion of a new entity *Book* by sending a message to all users who are interested in the book topics, provided that they accept e-mail notifications.

```
<RULE id="NewRelevantBook">
  <EVENT  eventType="Insert(Book)"/>
  <CONDITION
    query="user as U where U in new.Topics.InterestedIn
                      and U.Solicitable = 'Y'"/>
  <ACTION
    action="SendMsg(To:U.address,
                Text:'This book is now available in our
                      electronic store:'+new.Title)"/>
</RULE>
```

3.2.4 Rules for Metadata Management

Although less likely, rules may as well change meta-information [5]. For instance, the following rule increments the number of visitors of the pages of given books.

```
<RULE id="CheckDBTextbooks">
    <EVENT eventType="newpage(Book as B)"/>
    <CONDITION
       predicate="B.Gender = 'Database Textbooks'"/>
    <ACTION action="Increment(B.Meta.Visitors)"/>
</RULE>
```

4 Architecture

The architecture of the Torii system has been conceived to meet the following goals:

- Runtime performance: the system must be able to support high loads.
- Evolvability: due to the inherent dynamicity of the Web standards and technologies, the system architecture must facilitate product evolution. The basic requirements are the porting to different software platforms (Microsoft IIS/ASP, Java servlets), code generation in different Web languages (HTML 3 and 4, XML, ATVEF, HDML), and support for data feeds from heterogeneous legacy databases.
- Adherence to open and commercial standards: to capitalize on existing software and reduce development and maintenance efforts, the architecture is based on well established architectural, data storage, and content formatting standards (SQL, XML 1.0 [16], DOM [15], Java and Corba 2.0).

The experience with the Autoweb system proved inadequate with respect to most of the above objectives:

- Autoweb has a monolithic CGI-server structure, with an active process managing all the phases of page generation. The runtime engine does not benefit from data integration and parallelization capabilities provided by industrial platforms (e.g., application servers).

[5] In principle, business rules can change arbitrary information, including the data content, but this application is not envisioned by the users of the W3I3 Consortium.

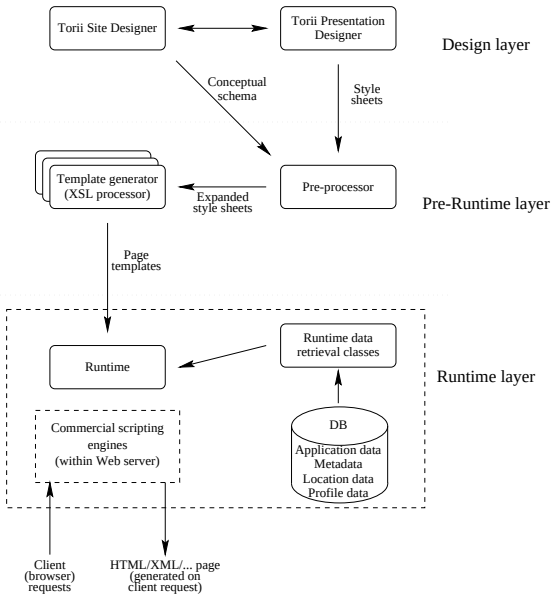

Figure 3: Architecture of the Torii system

- Service requests are serialized by the Autoweb server, which strongly limits performance for heavily loaded systems.
- The monolithic structure results in a tightly integrated piece of software, which proves difficult to manage in view of possible evolution of the design methodology, layout generation strategies, and underlying technologies.
- Autoweb caches output pages to enhance the runtime performance; this technique must be reconsidered in a scenario in which pages are personalized on a per-user or a per-group basis.

Due to these observations, the Torii system is based on a completely different architecture, illustrated in Figure 3. Generally speaking, the Torii architecture consists of three main layers:

- The design layer includes the tools for modeling Web applications; the output of the design layer is a Torii application, coded in the Torii modeling language.
- The pre-runtime layer comprises a chain of code generators that transform the Torii application into an intermediate representation suitable for processing on top of commercial Web-database systems.
- The runtime layer consists of an interpreter for converting page templates into actual application pages. The main functionality of the runtime layer is to merge application data into page templates to obtain the fully instantiated final application pages.

In addition, the Torii architecture includes a repository storing the following data:

- Application data: the actual content delivered by the application to the users. It is further subdivided into structured application data (e.g., database tuples, valid XML documents), and semi-structured and unstructured application data (e.g., image files, HTML pages).
- Metadata: information about the conceptual schema of the application, as defined by the conceptual design.
- Location data: information about the mapping between the Torii conceptual schema and the physical data structures that host the application data (e.g., external databases).
- Profile data: information about users and groups, used to customize a site. User data are managed according to privacy legislation and must be disclosed to owners on demand.

4.1 The Design Tools

The design layer consists of two major components:

- Torii Site Designer: permits the definition of a Torii application and manages a project repository containing the schemas of the defined Torii applications.
- Torii Presentation Designer: deals with the specific aspect of presentation specification, by letting the designer define in a visual way presentation models and style sheets to be used in a Torii application (see Figure 2).

¿From the architectural standpoint, the Torii Design Environment is a multi-user client server application, sitting on top of a project repository structured as a collection of DOM objects. The user interface permits the visual editing of Torii schemas, which are translated in real-time into a DOM representation, so that both the graphical and XML specifications are available to the designer. A dual-interface to persistence permits to store DOM objects either as XML files or database objects.

4.2 The Pre-Runtime and Runtime Environments

The output of the design tools is the input of a chain of transformations operated by the Torii Pre-Runtime and Runtime environments, which ends with the actual application page in the chosen delivery language.

The transformation process is clearly divided into two steps:

- The Pre-Runtime environment pre-processes the Torii presentation styles by unfolding all the implicit information (e.g., type information about the concept to which the style sheet is applied) to obtain a fully instantiated, language independent, specification of the layout, which is then turned into a language and platform dependent page template. Such template is complete, except for the actual data content, which is replaced by suitable calls to the runtime data retrieval classes.
- The Torii data retrieval classes are installed in a commercial engine, which is used to merge application data into templates to produce the actual page sent to the browser.

The Pre-Runtime processors are coded as a library of Java classes which permits the unfolding of presentation styles based on application metadata, plus a set of XSL programs (style sheets, in the XSL terminology) which map pre-processed XML presentation styles into code in the language of choice. With this architecture, adding a new output language does not require coding a new compiler, but simply writing a new XSL program that does the mapping.

The page generation process is illustrated in Figure 3. It is worth noticing that the architectural solutions adopted are flexible enough to guarantee the effortless reconfiguration of the page generation process, which is the base for tracking technology changes.

The requirement for performance clearly pushes in the direction of anticipating as much processing as possible before runtime, but also clashes with the need of serving content dynamically and on the base of the knowledge dynamically accumulated about the user. This tension is the major architectural challenge faced in Torii, and we have solved it by carefully designing the distribution of responsibilities among the different components of the model and of the system.

5 Experiences

Torii is at its infancy, but the predecessor Autoweb system is operational since the end of 1996 and has been used in several projects, both in the industry and in university.

These experiences have proven that a data-driven approach to Web Site design is very useful, offering the following advantages:

- The modeling concepts of the structural model constitute a design notation which can be used also by non-technical people, like graphic designers and content producers.
- Data quality improves, because the Web content needs to adapt to a rigid format; this is especially important when content is highly volatile, as in distance learning applications.

- The approach yields to a dramatic reduction of prototyping times; once the structure schema is in place, a default Web site is already ready to run.

- Maintenance and evolution are dramatically improved, because changes in the structural model can be automatically or semi-automatically propagated to the implementation. This benefit is fundamental, because even model-driven design does not always fully reflect requirements, and requirements change during or after application delivery.

- The decision to provide a high-level model of presentation, which is automatically translated into physical pages, enforces presentation consistency and coherence.

Further experiences of use will be collected through the development of two pilot applications of the industrial partners of the W3I3 Consortium. Otto Versand, which already makes a fraction of its revenues through a large Web site offering a selection of its mail-order catalogs, will experiment the use of Torii to add one-to-one functionality and more effective management and evolution facilities to its application. KPN Research, which manages HetNet, a Web-based network with about one million subscribers in the Netherlands, will use Torii to develop an application acting as an intermediary between service providers and customers, and to offer personalized access and facilitated service location to HetNet subscribers.

6 Conclusions

In this paper, we have argued that data-centric abstractions - already quite popular in database design - are applicable, with very strong potentiality, to data-intensive Web applications. The rationale of this argument is that data-intensive Web applications have a very simple control pattern (e.g., if compared to the nontrivial applications of databases); as such, a fully automatic development of the Web application is quite feasible, even from very rich specifications. We have also shown the very many orthogonal dimensions along which such specifications are needed in order to develop effective Web sites for a variety of users, with different needs.

This approach presents substantial advantages, because implementation and prototyping are immediate, and operation, maintenance and reengineering are greatly facilitated. In addition, the availability of a data-centric repository storing all information about Web site generation, together with the data content, opens up new opportunities to one-to-one Web generation, in the form of data derivations or of business rules operating on the repository itself. The power and novelty of applying both declarative queries and business rules for handling personalizations has been shown by means of several examples.

Presently, the design of the ToriiSoft Environment is completed, and the prototypes of Torii tools are at various stages of design and implementation; some of the innovative features of Torii are already supported by its predecessor project Autoweb, which is operational.

References

[1] Advanced television enhancement forum specification (ATVEF), Feb. 1999. available at http://www.atvef.com.

[2] P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. In *Proc. 23rd VLDB*, pages 206–215, Athens, Greece, Aug. 26-29, 1997.

[3] C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design*. Benjamin Cummings, Menlo Park CA, 1993.

[4] P. Bernstein et al. The Asilomar report on database research. *ACM Sigmod Record*, 27(4):74–80, Dec. 1998.

[5] S. Ceri, P. Fraternali, and S. Paraboschi. Design principles for data-intensive web sites. *ACM Sigmod Record*, 28(1):84–89, Mar. 1999.

[6] S. Ceri, P. Fraternali, and S. Paraboschi. Specification of W3I3 models. Technical Report W3I3PAP2, W3I3 Esprit Project n. 28771, Feb. 1999.

[7] M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. Catching the boat with Strudel: Experiences with a web-site management system. In *Proc. Sigmod'98*, pages 414–425, Seattle, June, 1998.

[8] P. Fraternali. Tools and approaches for developing data-intensive web applications: A survey. *ACM Comput. Surv.*, 1999. To appear.

[9] P. Fraternali and P. Paolini. A conceptual model and a tool environment for developing more scalable and dynamic Web applications. In *Proc. EDBT'98*, pages 421–435, Valencia, Spain, March, 1998.

[10] T. Isakowitz, E. A. Stohr, and P. Balasubramanian. RMM: a methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, 1995.

[11] D. Peppers and M. Rogers. *Enterprise One to One: Tools for Competing in the Interactive Age*. Currency-Doubleday, 1997.

[12] K. Takahashi and E. Liang. Analysis and Design of Web-based Informations Systems. In *Proc. Sixth Int. WWW Conf.*, Santa Clara, California, 1997.

[13] Unwired Planet Inc. Handheld device markup language (HDML) specification, Apr. 1997.

[14] M. K. Uusitalo. Specification of tools user interface. Technical report, W3I3 Esprit Project n. 28771, Feb. 1999.

[15] World Wide Web Consortium. Document Object Model (DOM) level 1 specification. Oct. 1998.

[16] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. Feb. 1998.