Towards a Software Engineering Approach to Web Site Development

Francesco Coda **Carlo Ghezzi** **Giovanni Vigna** **Franca Garzotto**
Dipartimento di Elettronica
Politecnico di Milano
P.za Leonardo da Vinci, 32
20133 Milano, Italia
+39 2 2399 1
[coda|ghezzi|vigna|garzotto]@elet.polimi.it

ABSTRACT
The World Wide Web (WWW) has become "the" global infrastructure for delivering information and services. The demands and expectations of information providers and consumers are pushing WWW technology towards higher–level quality of presentation, including active contents and improved usability of the hypermedia distributed infrastructure. This technological evolution, however, is not supported by adequate Web design methodologies. Web site development is usually carried out without following a well-defined process and lacks suitable tool support. In addition, Web technologies are quite powerful but rather low-level and their semantics is often left largely unspecified. As a consequence, understanding the conceptual structure of a complex Web site and managing its evolution are complex and difficult tasks. The approach we advocate here is based on sound software engineering principles. The Web site development process goes through requirements analysis, design, and implementation in a high-level language. We define an object-oriented modeling framework, called WOOM, which provides constructs and abstractions for a high-level implementation of a Web site. An important feature of WOOM is that it clearly separates the data that are presented through the site from the context in which the user accesses such data. This feature not only enhances separation of concerns in the design stage, but also favors its subsequent evolution. The paper provides a view of the approach and of its current prototype implementation.

Keywords
World Wide Web, object-oriented model, authoring, hypermedia

1 INTRODUCTION

From its first introduction in 1990 [2], the World Wide Web (WWW) is evolving at a fast pace. The number of WWW sites is increasing as Internet users realize the benefits that stem from a globally interconnected hypermedia system. Companies, organizations, and academic institutions exploit the WWW infrastructure to provide customers and users with information and services. The expectations of both providers and consumers are driving a relevant research effort aimed at improving the WWW technology. Examples are represented by the introduction of active contents in static hypertext pages by means of languages and technologies like Java [9] and JavaScript [3] and by the use of the Servlet technology [15] to customize Web servers behavior. This technological evolution has promoted a shift in the WWW intended use. The Web infrastructure is going beyond the mere distribution of information and services towards the deployment of a platform for generic distributed applications in a worldwide setting.

This promising scenario is endangered by the weakness of the current methodologies that support the development of Web applications. Current WWW technologies provide low-level mechanisms that enable, for example, particular visual effects or application integration. The use of these mechanisms is not guided by a systematic methodology that provides the Web site developers with a higher-level view on the structure of the site hypermedia base and a well-defined development process supported by suitable tools. Most current WWW site development and management practices rely only on the developer's knowledge and expertise.

This situation reminds the childhood of software development when applications were developed without methodological support, without the right tools, simply on the basis of good common sense and individual skills. WWW site development suffers from a similar problem. Most WWW developers delve directly into the implementation phase, paying little or no attention to requirements acquisition and specification and going through a very informal design phase (if any). In most cases, implementation is performed by using a low-level technology, such as the Hypertext Markup Language (HTML) [13]. Using the analogy with conventional software development, this approach corresponds to implementing applications through direct mapping of very informal designs into an assembly-level language. Furthermore, the lack of suitable abstractions makes it difficult to reuse previously developed artifacts, or to develop frameworks that capture the common structure of classes of applications and allow fast development by customization. Finally, the management of the resulting Web site is diffi-

cult and error prone, because change tracking and structural evolution must be performed directly at the implementation level. This problem is particularly critical since WWW systems, by their very nature, are subject to frequent updates and even redesigns.

Software engineering research has delivered technologies (e.g., object-oriented programming languages), methodologies (e.g., design paradigms), and tools (e.g., integrated development environments) that support the software development process. Being effectively supported, software developers are able to deliver quality products in a timely and cost-effective manner. A similar approach has to be followed in order to bring WWW development out of its immaturity. The problem of WWW site development must be tackled by providing methodological and technological support for each phase of the development process. While the conceptual modeling of a WWW site, being independent of the implementation technology, can be effectively supported by notations developed for hypermedia systems, there is a strong need for an implementation methodology and a corresponding technological support that bridges the gap between high-level design and low-level implementation, by providing suitable high-level abstractions and constructs.

This paper addresses these problems by proposing a WWW object-oriented modeling framework, called WOOM – *Web Object Oriented Model*. WOOM provides concepts and abstractions that help in the mapping from high-level design of a Web site into an implementation that uses "standard" WWW technology. This model extends the basic Web model by providing richer semantics, new constructs, and support for complex management tasks. In addition, we built a prototype authoring tool that supports WWW site development using WOOM concepts and abstractions. The tool is able to translate automatically a WOOM model instance into a Web site implemented using the conventional WWW technologies. The resulting process is similar to the compilation of high-level languages to binary code. It is not a simple translation, though. Rather, it is based on a powerful transformation scheme that allows high-level object descriptions, given in WOOM, to be customized in a context-sensitive way as they are translated into the target Web site. As a result, this approach clearly separates the description of the data from the way the data are presented through the Web interface. The same data can be presented differently in different contexts. This separation not only helps in designing the application, but also provides support to changes and Web site evolution.

This paper is structured as follows. Section 2 proposes a development process for Web sites. Section 3 defines the requirements for a suitable implementation technology. Section 4 presents WOOM, a modeling framework for Web site implementation. Section 5 discusses the proposed model. Section 6 presents our authoring tool based on the model. Section 7 draws some conclusions and outlines future work.

2 A WORLD WIDE WEB SOFTWARE PROCESS

The development of a Web site that provides structured access to a large amount of information, possibly under different views and through different contexts, is a complex activity. In order to deliver high-quality Web applications within limited time and budget, developers should follow a well-defined development process, possibly supported by suitable tools and notations.

Web sites are structured sets of multimedia information woven with hypertext references. The process of developing the artifacts that compose the information system is similar to software development. In addition, since Web sites may include programs that produce contents dynamically (like CGI scripts, Java applets, or JavaScript code) the two development processes may even converge.

The benefits of a well-defined and supported software process are well known [8]. For these reasons, as for conventional software, we propose to break down the development of a Web site into a number of phases: requirements analysis and specification, design, implementation. After the site has been implemented and delivered, its structure and contents are maintained and evolved. In defining these phases for the development process we do not imply a specific development process structure. Different process models (waterfall, spiral, prototype based) can be accommodated in the framework. Sections 2.1 through 2.4 discuss how such phases can be structured, based on our experience. In particular, Sections 2.3 and 2.4 provide the background motivations for the work we describe here, which developed a modeling framework supporting the implementation and maintenance of Web sites.

2.1 Requirements analysis and specification

During requirements analysis, the developer collects the needs of the stakeholders, in terms of *contents*, *structuring*, *access*, and *layout*. Contents requirements define the domain-specific information that must be made available through the Web site. Structuring requirements specify how contents must be organized. This includes the definition of *relationships* and *views*. Relationships highlight semantic connections among contents. For example, relationships could model generalization (*is-a*), composition (*is-composed-of*), or domain-dependent relationships. Views are perspectives on information structures that "customize" contents and relationships according to different use situations. Different views of the same contents could be provided to different classes of user (e.g., an abstract of a document can be made accessible to "external" users, while the complete document can be made accessible to "internal" users). Access requirements define the style of information access that must be provided by the Web site. This includes priorities on information presentation, indexing of contents, query facilities, and support for guided tours over sets of related information. Layout requirements define the general appearance properties of the Web site, such as emphasis on graphic effects vs. text-based layout.

We argue that existing tools supporting requirements specification and traceability of requirements through all development artifacts can be used in this context too. Further research is needed both to extend the above framework and to identify the additional specific features that a tool supporting requirements for Web based applications should exhibit.

2.2 Design

Based on the requirements, the design phase defines the overall structure of a WWW site, describing how information can be organized and how users can navigate across it. A careful design activity should highlight the fundamental constituents of a site, abstracting away from low-level implementation details, and should allow the designer to identify recurring structures and navigation patterns to be reused [5]. As such, a good design can survive the frequent changes in the implementation, fostered by the appearance of new technologies.

2.2.1 Support to design

Being largely implementation-independent, the design activity can be carried out using notations and methodologies that are not explicitly Web-oriented. Any design methodology for hypermedia applications could be used; e.g., HDM [6], RMDM [1], or OOHDM [14]. Our experience is based on the adoption of the HDM (Hypertext Design Model) notation [4], which is shortly described in the rest of this subsection.

In designing a hypermedia application, HDM distinguishes between the *hyperbase layer* and the *access layer*. The hyperbase layer is the backbone of the application and models the information structures that represent the domain, while the access layer provides entry points to access the hyperbase constituents.

The hyperbase consists of *entities* connected by *application links*. Entities are "first-class" structured pieces of information. They are used to represent conceptual or physical objects of the application domain. An example of entity in a literature application is "Writer". *Application links* are used to describe domain-specific, non-structural relationships among different entities (e.g., an application link from a "writer" to the "novels" he wrote). Entities are structured into *components*, i.e., clusters of information that are perceived by the user as conceptual units (for example, a writer's "biography"). Complex components can be structured recursively in terms of other components. Information contained in components is modelled by means of *nodes*. Usually, components contain just one node, but more than one node can be used to give different or alternative views (*perspectives*, in HDM) of the component information (e.g., to describe a piece of contents in different languages, or to present it in a "short" vs. an "extended" version). Navigation paths inside an entity are defined by means of *structural links*, which represent structural relationships among *components*. Structural links may, for example, define a tree structure that allows the user to move from a root component (for example, the data-sheet for a novel) to any other component of the same entity (e.g., credits, summary, reviews, etc.)

Once entities and components are specified, as well as their internal and external relationships, the access layer defines a set of *collections* that provide users with the structures to access the hyperbase. A collection groups a number of "members", in order to make them accessible . Members can be either hyperbase elements or other collections (nested collections). Each collection owns a special component called *collection center* that represents the starting point of the collection. Examples of collections are *guided tours*, which support linear navigation across members (through next/previous, first/last links), or *indexes*, where the navigation pattern is from the center to the members and viceversa. For example, a guided tour can be defined to navigate across all horror novels, another one can represent a survey of 14th century European writers.

2.3 Implementation

The implementation phase creates an actual Web site from the site design. As a first step, the elements and relationships highlighted during design are mapped on the constructs provided by the chosen implementation technology. Entities, component, links, and access structures are associated with particular primitives provided by the hypermedia technology or composition thereof. As a second step, the site is *populated*. The actual information is inserted by instantiating the structures defined in the previous step and the cross-references representing structural and application links among the elements. Collections are then created to provide structured access to the hyperbase contents. The third step is *delivery*. The site implementation must be made accessible using standard WWW technologies, namely Web browser like Netscape' s Navigator or Microsoft' s Internet Explorer that interact with servers using the Hypertext Transfer Protocol (HTTP). This can be achieved by translating the site implementation into a set of files and directories that are served by a number of "standard" WWW servers (also called *http daemons* in the UNIX jargon).

2.3.1 Support to implementation

According to the current practices, the implementation of a Web site is carried out using standard WWW technologies and tools based on the constructs provided by these technologies, such as text or image editors and format converters [12]. This is the source of most of the difficulties of Web site development. In fact, the standard Web technology is very low-level and semantically poor. The basic abstractions available to Web developers are:

- *HTML pages*, i.e., text files formatted using a low-level markup language;
- *directories*, i.e., containers of pages; and
- *references*, i.e., strings of text embedded in HTML tags that denote a resource (e.g., an HTML page) using a common naming scheme.

There are no constructs to define complex information structures like sets of pages with particular navigational

patterns, such as lists of pages or indexes. Such structured sets of information must be realized manually by composing the existing constructs and primitives. In addition, there is no way to create document templates and mechanisms to extend existing structures by customization. The development of a set of documents exhibiting the same structure is carried out in an *ad hoc* manner by customizing manually sample prototypes. There are no constructs or mechanisms to specify different views of the same information and to present such views depending on the access context. This hampers effective reuse of information. The only form of reuse is *by copy*. Some authoring tools like Microsoft's FrontPage [10] and NetObject's Fusion [11] try to overcome some of these limitations by providing a site-level view on the information hyperbase. Nonetheless, these tools are strictly based on the low-level concepts of HTML pages and directories. As a consequence, the developer is faced with a gap between the high level concepts defined during design and the low-level constructs available for implementation.

2.4 Maintenance

Web sites have an inherently dynamic nature. Contents and their corresponding structural organization may be changed continuously. Therefore, maintenance is a crucial phase, even more than in the case of conventional software applications. As for conventional software, we can classify maintenance into three categories: *corrective, adaptive*, and *perfective maintenance* [8]. Corrective maintenance is the process of correcting errors that exist in the Web site implementation. Examples are represented by internal dangling references, errors in the indexing of resources, or access to outdated information (as in the case of published data with an expiration date). Adaptive maintenance involves adjusting the Web site to changes in the outside environment. A notable example is represented by verification of the references to documents and resources located at different sites. Outbound links become dangling as a consequence of events over which the site developer has no control. Thus, maintenance is a continuous process. Perfective maintenance involves changing the Web site in order to improve the way contents are structured or presented to the end user. Changes may be fostered by the introduction of new information or the availability of new technologies. Perfective maintenance should reflect updates to the requirements and design documents. Maintenance in general, and perfective maintenance in particular, is by far the activity that takes most of the development effort.

2.4.1 Support for maintenance

Presently, Web site maintenance is carried out using tools like link verifiers or syntax checkers that operate directly on the low-level Web site implementation. This approach may be suitable for some cases of corrective and adaptive maintenance, but does not provide effective support for tasks that involve knowledge of the high-level structure of the Web site. For example, since reuse is achieved by copy, modifying a reused component, like a recurring introduction paragraph for a number of documents, involves the identification of every use of the component and its consistent update. In a similar way, modification of the structure or style of a set of similar documents requires updates in all instances. For example, if we decide that the background color of the summary page of all "horror" novels must be changed to purple, this requires consistent change of all files representing such summaries. More generally, since perfective maintenance may require a modification of the structure and organization of information, it should be supported by a structural view of the site and of the relationships between design elements and their implementation constructs. These relationships are of paramount importance because they allow the developer to reflect design changes onto the implementation and viceversa. Standard Web technologies do not provide any means to represent these relationships and the high level organization of information. Another problem concerns maintenance of hypertext references. In the standard WWW technology, references are just strings embedded inside the HTML code of pages; they do not have the status of first-class objects. Therefore, their management and update is an error prone activity.

3 REQUIREMENTS FOR AN IMPLEMENTATION TECHNOLOGY

As we discussed in the previous section, current standard WWW technologies do not provide adequate constructs and tools to support Web site implementation and maintenance. The technology used to implement a Web site should provide high level abstractions that support the mapping of design entities into implementation constructs and the management of the high level structure of the site. At the same time, since the WWW is accessed by users using the standard WWW interface, an implementation technology must provide tools to automate such mapping into standard WWW elements like HTML pages, directories, and the like.

The requirements for a Web site implementation technology can be derived by examining the founding principles of software engineering [8].

Rigor and formality An implementation technology should provide a clear definition of the entities involved in the implementation process, their relationships, and their associated semantics. A precise definition of the characteristics and behavior of the different elements of a Web site supports developers' understanding of the application at hand.

Separation of concerns An implementation technology should clearly separate among *contents* and *structuring representation, navigational specification,* and *contents presentation.* It should be possible to define different types of entities with specific characteristics and behavior. In addition, the technology should allow the developer to extend the type hierarchy with application specific types. An implementation technology should provide support for structuring the site contents in a well-defined manner. The capability to define navigational patterns to access site resources separately from the mod-

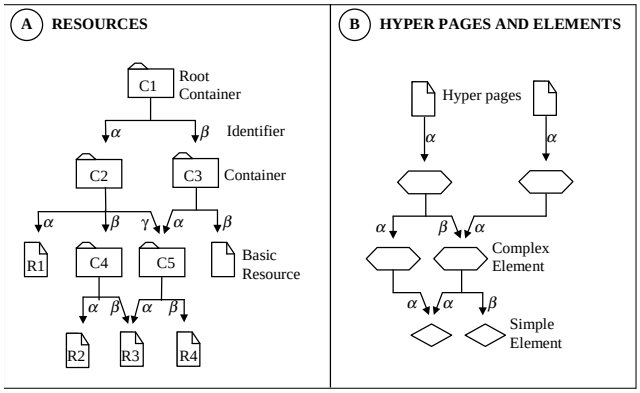

Figure 1: The containment relationship between information entities

eling of the site's contents should be provided. The process of extracting the view of the site contents to be delivered to the end user should be specified separately from the information sources and should be parameterized using the access context.

Modularity An implementation technology should provide constructs and abstractions that allow the developer to divide a complex problem in smaller simpler components. Modules should use information hiding to allow easy integration and management. Modularization mechanisms could then be used to support reuse.

Abstraction An implementation technology should abstract away from low level, unimportant details, identifying important concepts and relationships. In addition, abstraction should highlight and model concepts that are left implicit or hidden in the standard Web technology.

Anticipation of change Web sites, by their nature, undergo changes constantly. It is important to provide support for maintenance. Structuring techniques, requirements/design/implementation tracking, modularization, separation of concerns and abstractions, are the basis of effective Web evolution.

Generality An implementation technology should provide general mechanisms that support the development of implementation constructs and allow the developer to create *ad hoc* constructs and customize the existing ones.

4 A WEB SITE MODEL

Based on the requirements listed in Section 3, in this section we propose an object-oriented *modeling framework* [16] for the development of a World Wide Web site, called WOOM (Web Object Oriented Model). WOOM abstracts away from technological details and defines primitives and constructs that support effectively the developer during the implementation and maintenance phases of Web site development process. The model allows the developer to define the site contents, the navigational structure, and the service architecture. A site is implemented by creating a WOOM model instance, i.e. by instantiating the objects defined by the model. The WOOM model instance is then automatically translated into conventional WWW technologies.

According to WOOM, a Web site can be defined in terms of the following entities: *resources*, *elements*, *sites*, *servers*, *links*, and *transformers*.

Resources and *elements* are the fundamental entities of the model. They model the contents and the structure of a collection of information. Resources can be divided into *containers* and *basic resources.* A container is a collector of resources. The containment relationship among resources defines a DAG (Direct Acyclic Graph)[1], in which containers are intermediate nodes and basic resources are leaves. The root of the DAG is called the *root container* and encloses all the resources of the site. Each resource, with the notable exception of the root container, is contained at least in one container (see Figure 1-A). WOOM provides a number of predefined container types: *lists*, *trees*, *indexes*, and *sets*. Additional container types can be defined by the Web designer by extending the WOOM framework. Lists organize the enclosed resources in a linear fashion. They are used to represent a sequential relationship inside a group of resources (e.g., the pages that compose a guided tour through the novels of a given writer). Trees impose a hierarchical structure to the enclosed resources. For example, the novels of a given writer can be classified into genres: horror, science fiction, etc.; science fiction novels, in turn, can be classified into, say genetics, astronomy, etc. Indexes organize the contained resources in two-level trees. For example, an author's novels can be grouped into "youth", "maturity", and "late" novels. Sets are simply used to group r esources without any specific relationship among them, but characterized by some common visual or semantic property. Each container type exports an interface that allows other entities to access the enclosed resources without exposing the container's internal implementation details. For instance, one can address the first resource of a list or the root of a tree without specifying the target resource by name.

Basic resources are information repositories. They are distinguished into *opaque resources* and *hyper pages.* Opaque resources are unstructured resources. Subclasses of this class are *images*, i.e., graphic objects, *applets*, i.e., programs that are activated on the client side, *scripts*, i.e., applications that are activated on the server side, and *external*. External resources are those types of information that are not directly supported by the current Web technology and are managed by means of external helper applications. These resources include audio and video information, PostScript files, binaries, and other similar entities.

Hyper pages are hypertext pages, which may contain text, anchors, and references to pictures, sounds, and animations. The contents of a hyper page are modeled by a collection of *elements* (see Figure1-B). An *element* is an information unit, like a text paragraph, an anchor, or a dotted list. Elements can be *simple* or *complex*. Simple elements are atomic data containers, while complex ele-

[1] Circular containment relationships are forbidden.

ments contain an ordered list of other elements. For example, the image placeholder element (IMG) is a simple element, while the BODY element may be composed of some paragraphs, a table, etc. Each element belongs at least to one hyper page. As for resources, the containment relationship among elements defines a DAG.

Resources and elements may have some associated *attributes*. Attributes specify entity properties, such as: expiration date, version, relevance, graphic properties (e.g., the font size of a text string), etc.

Each container, hyper page, and complex element associates a unique identifier to each of the enclosed entities. The identifier can be used to denote an entity among others enclosed in the same container, complex element, or hyper page. The identifiers are represented in Figure 1 as labels associated to the containment relationship arcs.

The *context* of a resource (or element) is its position inside the resources/elements containment DAG. Contexts are specified through *pathnames*. The pathname of an entity is a sequence of identifiers that describes the path from the root container to the entity. This identification mechanism is similar to the well-known naming scheme based on pathnames adopted by file systems. An entity can be reached through many different paths. As a consequence, an entity can be in different contexts. For example, let us consider Figure 1-A. Resource R3 belongs to three contexts, because it is contained by both containers C4 and C5, and container C5 is contained by both containers C2 and C3. Therefore, resource R3 is accessible through paths "/ α/β/β", "/ α/γ/α" and "/ β/α/α".

Each resource and element has an associated *translate* operation. The invocation of the *translate* operation for an entity produces its implementation in the conventional WWW technology. That is, hyper pages are translated into HTML files, containers into directories, and elements into character strings representing HTML tags. The translation operation of entities composed by sub-entities (containers, hyper pages, and complex elements) is recursive. The translation of the composed entity is built using the results of the translation of its sub-entities. For instance, the translation operation of a hyper page invokes the translation operation on the enclosed elements and uses the resulting text strings to build an HTML file.

A site is composed of a root container and one or more *servers*. A site models a set of related information, represented by the set of resources contained (directly or indirectly) in the site's root container. The associated *servers* are used to define the network access points to the site contents. A server corresponds, at run-time, to an HTTP daemon process that answers to the end user requests for resources belonging to the site. Each server is characterized by a unique *address* and has an associated *container* and *context* that limit the scope of the resources that are accessible through the server. For instance, a server associated to the container C5 of Figure 1 with the context "/ α/γ" allows one to access resources R3 and R4 under the contexts "/ α/γ/α" and "/ α/γ/β" (see the translation process later in this section).

In WOOM, references are modeled via *links*. Links are first-class objects that associate a source element with a destination resource *within a particular context*. Context information must be taken into account when creating links because, as we will see later, a resource in different contexts may assume different forms. By keeping track of the context of the destination resource, WOOM links may lead to a particular version of the resource. Links are composed of an *element reference*, a *resource reference*, a *context specification*, a *server reference*, and some *script parameters*. The element reference identifies the element that assumes the role of anchor (the starting point of the link). The resource reference and context specification identify the destination of the link. The server reference specifies the server that must be used in order to retrieve the destination resource. Script parameters are used when the destination of the link is a *script* resource. They are the input parameters used in script invocation. Links are translated into URLs embedded in attributes of the *anchor* HTML tag during the translation process.

A WOOM model instance can be translated into a traditional file-based site implementation. The translation process is triggered by calling the *translate* operation on the root container. The operation propagates in a top-down fashion to the entire WOOM instance DAG. An entity positioned in the DAG in *n* different context is translated *n* times. This is the case, for example, of a *paragraph* element enclosed in two different hyper pages. The shared paragraph will correspond, after the translation, to two distinct strings in two different HTML files. This is necessary because conventional WWW technology does not allow information sharing at the element level (two files can not share a piece of text).

As we mentioned, in WOOM, site contents (resources and elements) can be put in context; that is, they can be modified on the basis of their current context during the translation process. For instance, consider a hyper page describing a novel (e.g., Primo Levi's *La Tregua*) that is placed in two containers. The first container is a list enclosing all the novels of a particular genre (e.g., "holocaust" novels). The second is a set collecting the novels of the same writer (e.g., Levi's books). In the first context the page can be modified to include information that highlight the relationship between the movie and the genre, and links to navigate inside the list. In the latter context, the resource can be modified to include references to the author's biography. The modifications that must be applied to the entities within a particular context are specified by means of *transformers*. Transformers are objects with an associated state and a *transform* operation that takes as parameter a resource or element object. The transform operation modifies the element or resource passed as parameter and eventually returns it. Transformers can be associated to containers to influence the translation of the enclosed entities.

The translation operation of an entity receives a list of transformer objects from one of its parents in the DAG structure, and returns the entity representation in the conventional WWW technology (i.e., strings, files, and directories). The translation operation follows a fixed schema. First, it creates a shallow copy of the entity (i.e., a resource or an element is created). Such a copy is passed to the *transform* method of the first transformer belonging to the list received during the translation invocation. The transformer modifies the copy depending on its own state and the values of the copy attributes. The modified object is then passed to the *transform* method of the next transformer, until the last transformation has been applied. At this step of our work, we do not constrain what transformers can do. Transformers may add, delete, and modify the object attributes; they may reconfigure the containment relationship; they may add new sub-entities, delete or reorder references to existing sub-entities, etc. In addition, the composition of transformers may eventually return a null object, meaning that the current resource or element must not be translated at all. If the returned object is non-null, the translation operation is invoked on every resource or element contained in the modified entity. The translation operation invoked on a descendant entity receives as a parameter a list of transformers obtained by appending the transformers associated with the entity being translated (if any) to the list of transformers received by the parent entity. The translation of descendant nodes returns their representation in the conventional WWW technology. These data are used by the entity to produce its own translation. Note that since transformers are applied to copies of resources and elements, the original entities defined by the site author are not modified.

A predefined set of transformers has been introduced in WOOM to implement navigational patterns that take advantage of the topological structure imposed by some containers (lists, indexes, and trees). For example, a *list transformer* has been associated to list containers to impose a sequential navigation among the enclosed resources. During the translation process the transformer is able to add proper navigational garnishment to the hyper pages contained in the list. Similarly, a navigational pattern has been predefined for the index container type to allow end users to navigate from the root resource to the leaves and back again. Other transformers have been predefined for the tree container in a similar way.

Transformers can be used to define properties common to a set of resources, such as color background, graphical decorations, and fonts. Transformers give the opportunity to centralize the control of properties that in conventional WWW technology are defined implicitly on a per-entity basis.

Furthermore, transformers are used to extract particular views of a resource or element. During the translation process a transformer can inhibit the translation of portions of information on the basis of their attribute values (e.g., expiration date, creation date, relevance, and version). For instance, let us consider a hyper page containing a scientific paper that is put in three different contexts. Labeling the page elements with proper attributes, a different view of the contents can be proposed in each context: an abstract, a simplified version without mathematical formalisms, and a detailed presentation.

In conclusion, transformers are powerful mechanisms that allow the information provided by a site to be kept separate from the various occurrences in which such information can appear in different forms, within different contexts. By separating the information from its contextual occurrence, we facilitate reuse, implementation, and maintenance of complex data structures.

5 MEETING THE REQUIREMENTS

The WOOM modeling framework provides several advantages with respect to the standard Web technology. Its benefits can be analyzed with respect to the requirements presented in Section 3.

Rigor and formality By using an object-oriented model, the entities that compose a Web site and their relationships can be defined in a precise manner. The behavioral semantics of each element type is encapsulated within its definition.

Separation of concerns WOOM provides the resource and element concepts to model and structure site contents. Differently from the standard Web technology, in WOOM all the entities that compose a site from the server to the small grain elements are abstract data types. It is possible to extend the type hierarchy by using composition and inheritance. The modeling framework defines structuring constructs like containers, page sets, lists, trees, and supports the creation of user-defined constructs. The model provides predefined navigational patterns for resource structures. Thanks to the transformer mechanism, navigational patterns are defined separately from contents. The same mechanism allows the developer to keep the different views of the same information separate from the information itself.

WOOM provides an explicit model of the concepts of site and server, and their relationships with the site's contents. This clearly separates the issue of *network access* to the site contents from the internal organization and modeling of the data.

Modularity WOOM container types (lists, trees, etc.) provide a way to aggregate contents and partition the contents base. Container types have a well-defined interface; they can be used as building blocks to define more complex container structures. In addition, the server abstraction allows for a higher-level partitioning of the information domain.

Abstraction Modeling a Web site provides the developer with clearly defined abstractions. Abstractions can be distinguished into *entity-level abstractions* (resources, servers, site) and *structuring abstractions* (containers, page sets). WOOM clearly defines the relationships among the site constituents, hiding unimportant details and highlighting concepts that are left implicit in the low-level technology. For example, while the standard WWW

technology implements links like string attributes of anchor elements, WOOM models them as first-class objects. Management of implementation details is left to the translation process, as happens in the case of high-level programming languages [7].

Anticipation of change Design entities and their relationships can be modeled directly by using the WOOM framework. This way design changes can be tracked down to implementation and viceversa. In addition, modularization allows for easy management of groups of resources.

Generality Being an object-oriented model, WOOM exploits the inheritance mechanisms provided by object orientation to extend and manage abstractions and constructs. In addition, the transformer mechanism is a powerful, general-purpose mechanism that provides the developer with a tool for extending and customizing the model.

6 A TOOL FOR WEB DEVELOPMENT

We developed a prototype authoring tool, written in Java, that implements the WOOM model. The tool allows the developer to use WOOM constructs to create the resources that compose a Web site, to perform complex management operations, and to translate the WOOM instance into the standard WWW technology. The main components of the tool are presented in Figure 2.

A first component is the WOOM class framework. The framework provides the definition of basic WOOM entities and provides some predefined constructs. The class framework provides support for representing Web site design elements into the model. This is achieved by means of an integrated, yet separate, *design module*. The design module is a plug-in component that provides support for a specific design notation. Currently, the HDM notation is supported.

As a preliminary step in Web site implementation, the WOOM class framework is imported into the development application. Then, the developer uses the instances of the classes provided by the design module to represent the entities defined during the design phase. Once the design elements have been represented, the developer chooses the corresponding resource implementation. In order to implement each design element, the developer may use the predefined constructs offered by the WOOM class framework or create new application-specific constructs using inheritance, composition, and the transformer mechanism. After suitable constructs have been identified, *implementation links* that associate a design element with the corresponding implementation construct are created. Such links are used in tracking changes in the implementation to the site design and viceversa.

The next step consists of populating the site, by instantiating resource objects of the appropriate classes, and creating application links. Structural links are automatically managed by the semantics of structured objects that implements structured design elements. Once the site has been populated, the translation process is invoked on the site root container. The translation produces the site implementation in terms of files in the file system and servers that provide access to the site's contents.

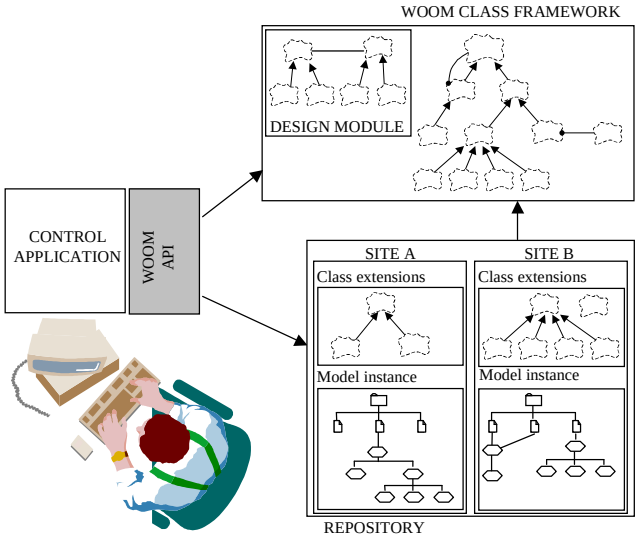

Figure 2: The WOOM-based authoring tool.

Web site maintenance and management operations are performed on the WOOM model instance. The WOOM framework provides support for a set of predefined tasks like syntax checking, link updating, resource restructuring, consistency checks, shared resource management, and design change management. The model instance, after it has been modified, must undergo a new translation process in order to reflect changes on its target implementation.

Web site instances, composed of site-dependent schema extensions (classes and transformers) and resource objects are persistently[2] stored in the *Repository* module. The control application accesses the WOOM schema and instances by means of the *WOOM API*. The control application is a Java application that uses the primitives and services offered by the API.

We are currently working on a graphical interface that allows the developer to access WOOM services in an intuitive and user-friendly way.

7 CONCLUSIONS AND CURRENT WORK

The development and evolution of Web sites is a significantly complex and time-consuming task. Unfortunately, the current approaches to Web site development not only are centered just on the implementation phase, but also lack high-level language abstractions as well as a comprehensive methodological development framework. In this paper, we presented a Web site development process that we used successfully in a number of projects. Our experience showed that the semantically poor abstractions and low-level mechanisms offered by the current WWW technology are inadequate to support the imple-

[2] Persistency is achieved by using the Java object serialization mechanism.

mentation and maintenance phases of the development process. Therefore, we developed a new object-oriented modeling framework that extends and enriches the Web basic abstractions and mechanisms by providing higher-level constructs. These constructs are used to implement higher-level design entities and provide the developer with powerful, customizable, and reusable patterns. In addition, the model supports complex management tasks by supporting an explicit representation of the relationships and links among the site constituents.

A prototype tool based on the model has been developed. The tool allows a developer to create a WOOM model instance and to translate it into a Web site implemented using standard WWW technology. The experience gathered so far on the use of the tool has shown that by following a clearly defined, tool-supported development process it is possible to produce high-quality Web sites with reduced effort. Web evolution is also facilitated.

Future work will include the development of a user-friendly graphical interface for the tool that will allow the site developer to use the features of the model more naturally. In addition, the set of predefined resource container types (i.e., lists, trees, sets, and indexes) will be extended to include more representation and navigation patterns that can be used with minimal modifications by the developer in a number of application domains.

Finally, we will address the issue of incremental Web translation. Presently, a complete target Web site is generated after each change in the high-level model. In an incremental schema, we would like to regenerate only the minimum amount of target information affected by the change.

REFERENCES

1. V. Balasubramanian, T. Isakowitz, and E. A. Stohr, *RMM: A Methodology for Structured Hypermedia Design*, Communications of the ACM, 38(8), August 1995.
2. T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen and A. Secret, *The World Wide Web*, Communications of the ACM, vol. 37, num. 8, August 1994.
3. D. Flanagan, *JavaScript – The Definitive Guide*, 2nd Edition, O'Reilly & Ass., January 1997.
4. F. Garzotto, L. Mainetti, and P. Paolini, *Hypermedia Design, Analysis, and Evaluation Issues*, Communications of the ACM, Vol. 38, No. 8, August 1995.
5. F. Garzotto, L. Mainetti, and P. Paolini, *Information Reuse in Hypermedia Applications*, Proceedings of ACM Hypertext '96, Washington DC, ACM Press, March 1996.
6. F. Garzotto, P. Paolini, and D. Schwabe, *HDM – A Model-Based Approach to Hypertext Application Design*, ACM Transactions on Information System, Vol. 11, No. 1, January 1993.
7. C. Ghezzi and M. Jazayeri, *Programming Language Concepts*, 3rd edition, Wiley, 1997.
8. C. Ghezzi, M Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, Prentice Hall, 1991.
9. J. Gosling and H. McGilton, *The Java Language Environment: a White Paper*, Technical Report, Sun Microsystems, October 1995.
10. Microsoft Corp., *FrontPage Home Page*, http://www.microsoft.com/FrontPage/
11. NetObjects Inc., *Fusion Home Page*, http://www.netobjects.com/
12. Netscape Inc., *Netscape Composer*, http://www.netscape.com/
13. D. Ragget, *Hypertext Markup Language 3.2 Reference Specification*, W3C recommendation, 1996.
14. D. Schwabe and G. Rossi, *From Domain Models to Hypermedia Applications: An Object-Oriented Approach*, Proceedings of the International Workshop on Methodologies for Designing and Developing Hypermedia Applications, Edimburgh, September 1994.
15. Sun Microsystems, *The Java Servlet API White Paper*, 1997
16. Taligent Inc.,*Building Object-Oriented Frameworks*, A Taligent White Paper, 1994.