Web Engineering: The Developers' View and a Practitioner's Approach

Sotiris P. Christodoulou, Paris A. Zafiris, and Theodore S. Papatheodorou

HPCLab, Computer Engineering & Informatics Department,
University of Patras,
26500 Rion, Patras, Greece
{spc, paz, tsp}@hpclab.ceid.upatras.gr

Abstract. The expanding role of the Web as a content and applications deployment platform and the appearance of new computing paradigms, such as thin-client computing, require now more than ever the introduction of concrete development frameworks. Although new approaches, technologies, tools, commercial applications appear daily, limited guidelines or frameworks exist that can assist Web developers in selecting the proper methodology and tools for the design, implementation and maintenance of flexible Web content and applications. Our work, triggered from our experience in implementing the Web presence of several large Greek Governmental organisations, attempts to address the major current and forthcoming problems that Web developers face. We propose a framework (RDF/XML based) that will act as a malleable development support environment, incorporating specific guidelines, which Web developers should always consider. The primary goals are achieving scalability (modular, component-based architecture), re-usability and technology independency in Web development. We focus on hypermedia content and applications.

1 Introduction

In this introductory section, we examine the concept of the Web application domain, as seen from the aspect of the developer, and the typical components of Web applications. The Web developer, once seen as a simple document author, nowadays is typically responsible for building complete Web sites (and sometimes distributed online environments), incorporating different Web applications, technologies and varied content. These applications can be distinguished into two basic categories, Web Hypermedia Applications and Web Software Applications.

A *Web Hypermedia Application* is the structuring of an information space in concepts of *nodes* (chunks of information), *links* (relations among nodes), *anchors, access structures* and the delivery of this structure over the Web. Therefore, the developer of such applications is often asked to *develop novel* hypermedia applications with new content, explicitly for the Web, *port or import* existing information from unstructured documents or other hypermedia systems into the Web or *map* (dynamically or statically) information sources stored within a logical, structured space (e.g. a database) into a collection of Web pages.

A *Web Software Application* is any "classic" software application that depends on the Web, or uses the Web infrastructure, for its correct execution. In most cases, the hypertext model is not appropriate for modeling the content of these applications, and other models (e.g. Object-Oriented, Entity-Relational) are employed. This category includes, among others, legacy information systems accessible through online gateways via a Web browser, such as databases (DBs), booking systems, knowledge-bases, numerical software, etc.

In accordance with the paradigm of on-line computing, with either thin- or full-clients, we consider that a generic Web application consists of three main parts: (a) *Information (content/data) Structure*, incl. analysis, design, structure, storage, management, reuse of data, (b) *Application Logic*, namely static and dynamic navigation, filtering, personalization (in Web Hypermedia apps) or more generic processing (in Web Software apps) and (c) *User Interface (UI)*.

This three-tier architecture can also be expanded to a multi-tier model, where the Application Logic is implemented by different layers of software. This approach of isolating content, application logic and interface is a tried and true one, and is recommended for most Web applications. Based on it, our work proposes an abstract architecture of modules that should be provided to the developer in order to support the life-cycle of each layer, emphasizing on easy maintenance, modularity, extensibility, re-use and interoperability of information or application components. Special attention is given to using a data-centric, rather than application-centric, development process.

For the data layer, we examine issues of effective content modeling (incl. development, management) of data resources, in order to construct "smart", human- and machine-readable (self-describing) Web resources, accessible from modern Web applications (content pool). We consider any kind of content, regardless of (i) complexity of the content structure (ii) media type (iii) potential application domain and (iv) type of Web application where the content may be used.

For the application logic and UI layers, we shall focus on Web Hypermedia applications. We examine application and UI modeling methodologies, that will allow the developer to implement apps that can transparently access and exploit the resources described above, as well as store their own application resources (content or logic) back into the data layer, enabling reuse by other Web applications.

On the issue of Web Software Applications (app logic, UI), we consider that this is rather a Software Engineering problem. However, in this case also common guidelines of Web Hypermedia application development can be used, such as:

- Focus on modular implementations, preferably using components
- Easily configurable query and results rendering mechanisms

2 Current Efficient Web Development

We consider that three main parts constitute a Web application: Information Structure, Application Logic and User Interface. This disjunction is clear in Figure 1, which illustrates the current Applications and Resources of the WWW. We distinguish four classes of Web applications, according to the implementation of these three parts. The applications of class *APP1* (a typical example is a collection of static HTML pages) are the worst regarding maintenance issues and re-use. For instance, if the developer wants to modify any of the three parts of the application, she/he has to modify every single resource. Thus, HTML authoring (or converting resources to HTML pages) is the most inefficient way of developing Web Hypermedia applications. However, most of the current Web developers still develop that way. At APP3, information structuring is embedded into the application logic, and so much of the knowledge about the information structure, which is utilized by the application logic, is lost once the application has been constructed. On the other hand, applications of type APP4 that separate information structure, application logic and interface are providing friendly maintenance and high reusability.

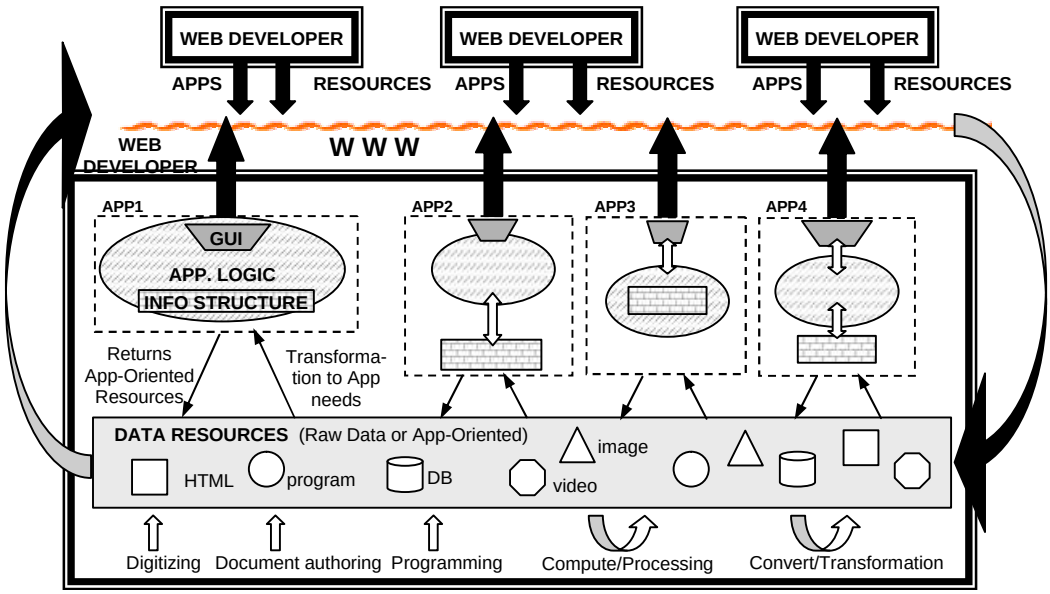

Fig. 1. Current Web Applications and Resources

For the development and management of Web Hypermedia Applications, the developers, in the best case, are using primary Data Resources, existing "Theoretical" Resources & Tools and they follow a Development & Maintenance Procedure.

2.1 Data Resources

As stated, the most common task for a Web developer is the development of a Web site, containing a set of Web applications. For the development of these applications, the developer is usually working on a set of heterogeneous and probably distributed *Data Resources* that have either been generated by other applications or are produced for the specific application needs. Such data resources are DBs, documents, HTML pages, multimedia files, 3D worlds files, programs, etc. The basic problem of the data resources from other applications is that they were designed, implemented or manipulated according to their application context. We call them "application-oriented" resources and they are difficult to be re-used. They usually need remodeling or transformation to the specific application needs. This requires much effort and time, and the result is again a collection of application-oriented resources.

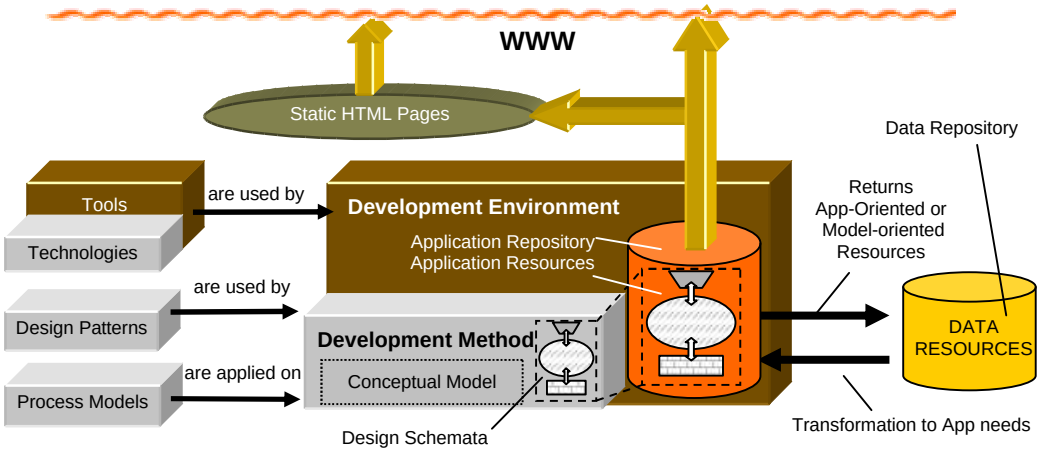

Fig. 2. "Theoretical" Resources and Tools in current efficient development

2.2 "Theoretical" Resources and Tools

Apart from the Data Resources, the developer can use existing "theoretical" resources (methods, models, etc.) and their corresponding tools (if available). Figure 2 illustrates how the developers can use the "theoretical" resources and their tools in order to implement a Web Hypermedia Application based on primary Data Resources.

The "theoretical' resources include:

- *Conceptual Models*: For the design of the three parts of the Web applications, the developer utilizes well-known and adopted conceptual models that mainly derive from Software Engineering and DB concepts, like Object-Oriented (OO), Entity-Relational (ER), Labeled Directed Graph (LDG) and Component-Based Design (CBD).

- *Development Methods*: Design methodologies for the construction of the conceptual, navigational and interface designs of the three parts and how they are related. Existing Development Methods are based on and incorporate a certain conceptual model.
- *Design Patterns* [1,2] for hypermedia applications development. Design patterns address the reuse of design experience at any of the several aspects of design and development: conceptual, navigation, and interface modeling; rhetorics; implementation, including specific development environments (e.g, Web); development process.
- *Development Process Models*: Various ways that the Development Methods can be applied in order to increase their performance. By process we mean activities that are carried out, the relationships between these activities, the resources that are used, the results that are created, etc. Such Process Models are the waterfall model [3] and the Spiral Model [4] in Software Development and IMPACT-A [5] in hypermedia development. Testing and Quality Assurance stages are included within the models.
- *Technologies*: Any technology that addresses data representation for storage/management/exchanging (e.g. XML, RDF, RDBMS, etc.) and application logic or interface implementation and management (e.g. CSS/XSL, HTML, DHTML, Java, Cookies, Plug-Ins, ASP, JSP, etc.).

Some of these "Theoretical" resources may have the support of a set of tools. Such tools could be:

- *Conceptual Model Tools*, e.g. RDBMS, OO design environment, etc.
- *A Development Environment.* A set of tools that supports the corresponding Development Method. It provides support for the storage (in the so-called *application repository*, see Figure 2) and the management of *design schemata* and *application resources*.
- *Technology Tools*: For each technology, a set of supportive tools is provided for improved exploitation (e.g. XML Toolkits, HTML editors, Java compilers, Visual Interdev for ASP, etc.).

The Development Method and the corresponding Development Environment constitute a Hypermedia Application Development and Management System (HADMS). In previous work [6], we proposed an evaluation framework for HADMS and we evaluated some representative systems (OOHDM[7,8], RMM[9], HSDL[10], STRUDEL[11], Vignette's StoryServer and Microsoft's FrontPage). The conclusions that were derived from this evaluation helped us identify the crucial issues of the development of Hypermedia Applications.

2.3 Development and Maintenance Procedure

Consider the steps that developers have to follow in order to implement a Web Application:

A. Theoretical Resources Decision Phase

1. Select one of the application types (we strongly recommend APP4)
2. Select the Development Method (incorporated Conceptual model)/Environment that better meets the application requirements. The developers may use the evaluation framework proposed in [6] in order to specify the requirements of the application under development and decide which method and environment to choose.
3. Select the Technologies that are more appropriate for supporting the requirements of the specific application.
4. Select a Process Model to apply on the Development Method.

B. Development Phase

5. Use the Development Method (Conceptual Model) / Process Model and the Design Patterns to design the conceptual, navigation and interface Design Schemata.
6. Use the Development Environment to implement and manage the Design Schemata.
7. Produce Application Resources either by transforming data resources or direct authoring.
8. Use the Development Environment to produce the final Web Application, which will be either a collection of HTML static pages or an interactive interface to the Application Resources or both.

C. Maintenance Phase

9. If the developers decide, e.g. for reasons of quality assurance or improved performance or adaptation to new requirements, to:
   - Replace the Application Type: Go to Step1.
   - Replace the Development Method (incorporated Conceptual model) / Environment: Go to Step2.
   - Replace some of the Technologies: Go to Step3.
   - Replace the Process Model: Go to Step4.
   - Modify the Design Schemata: Go to Step10.
   - Modify the Application Resources: Go to Step12.
   - Produce new Application Resources: Go to Step7.
10. Use the Development Method (Conceptual Model) / Process Model and the Design Patterns to modify the conceptual, navigation and interface Design Schemata.
11. Use the Development Environment to apply modifications to design schemata and propagate them to the Application Resources.
12. Use the Development Environment to manage the Application Resources.
13. Go To Step8.

3 Basic Requirements of Developers

It is obvious that different applications may have different requirements. For instance, a part of a Web site may contain only static pages, while another is a dynamic application over a multimedia DB. In such cases the developer may need to use different conceptual models, development methods, design patterns and technologies, that possibly do not inter-operate. For almost every application there is a development method

that can adequately cover its requirements. However, none of the methods can efficiently cover all application domains [6]. Thus, the developer should be able to use various conceptual models and development methods/environments for various applications.

The basic problems of the current HADMS are the followings:

- The development methods incorporate one conceptual model and they cannot support others.
- Most of the methods have no or poor development environment.
- There is no environment that can support more than one method.
- The development environments are not easily extensible, in order to efficiently incorporate upcoming tools.
- Most of the HADMS do not support efficient importing and exporting either for content or design.

In this work, we stress the lack of a development framework that can address the above problems, by covering the following basic requirements.

It has to be:

R1. *Modular*. The architecture of the framework should be based on a synthesis of abstract components (modules) that can interoperate through their interfaces, in order to support the features of the framework. This will facilitate the maintainability, extensibility and reliability of the framework, as the components can be maintained or replaced separately and by different persons.

R2. *Technology-independent.* The interfaces between the components of the framework should be developed with minimal dependency on technologies that may soon become obsolete.

It should support:

R3. *Several Conceptual Models.* Some applications are complex and require OO principles, while some others are simple and a labeled directed graph (LDG) is enough for modeling their concepts. Moreover, different models may be needed for designing e.g. the information structure and the interface of an application. Thus, the developer should be able to use various conceptual models for designing the three application parts.

R4. *Several Methods.* The developer should be able to use various methods for the development of different applications. For methods that provide an environment, this (or part of it) could be integrated in the framework (if it can be appropriately tailored).

R5. *Data Importing.* The framework should provide robust importing mechanisms from existing data resources.

R6. *Reusing of Information and Designs:* Developers should be able to reuse and exchange (through import/export facilities) information and designs across different applications developed with different models/methods. Moreover, the system should incorporate meta-data principles in order to build self-describing, machine-readable and human-readable content/application resources, for exchanging resources with other developers.

R7. *Maintenance/management of the applications.* Web application services and/or information usually change (and in some cases rapidly) throughout the life-cycle of the application. Thus the application must be built so that it can be modified, fixed, or maintained with little cost of time or money. The separation of the application's parts enable applications to be maintained more effectively. Basic maintenance facilities include efficient authoring environments for contents, development tools for supporting the implementation details of the targeted system (e.g. Web) of the application and finally collaboration facilities among the working group's members (like versioning, access control, etc.)

R8. *Quality Assurance (QA) of content and applications.* This involves the behaviour of the overall framework and of its individual layers, modules and tools. By dealing with the issues of QA in a per-layer level, in addition to using Development Process Models with embedded testing/QA stages, we establish efficient workflow paths for developers, avoid problems during the steps of the development process and minimize potential risks for the final output, either content or applications.

4 An RDF/XML Framework for Information and Hypermedia Application Management

In a previous work [6] we focused on defining and analyzing the requirements of hypermedia application development. Essentially, the needs of the Web developer can be described with abstract modules that are independent of the current development methods or models. In this work we propose a generic framework with a modular architecture (R1), based on the synthesis of interoperable modules, in order to satisfy the above requirements. Each module defines a *class of tools*, i.e. the set of tools that can contribute to the functionality of the module. Each module can incorporate several tools. Thus, for the implementation of the system, the appropriate tools for each module have to be selected, appropriately tailored and integrated into the framework. The main advantage is that when a newer or enhanced tool comes up, it can be integrated into the framework with minimal consumption of effort and time, while the developer benefits from the emerging technologies, methods and tools.

Apart from the architecture, we need to specify the data model that our framework will incorporate. The data model must provide interoperability, extensibility and reusability so that a range of applications can access the information data. In order to achieve this we employ core metadata principles. The main kinds of metadata are schematic, navigational and associative, which can be further categorized into descriptive (such as Dublin Core [12]), restrictive (such as PICS [13]) and supportive (such as dictionaries, thesauri and hyperglossaries). Making the repository self-describing and based on metadata structures ensures that knowledge about the repository structure is available to other applications.

The Resource Description Framework (RDF) [14] (see Figure 3) is both a foundation and a mechanism for encapsulating metadata sets and associating them with Web resources (along the lines of the Warwick Framework [15]); it provides interoperabil-

ity between applications that exchange machine-readable information on the Web. These characteristics led us to adapt the RDF Data Model in our framework as the primary mechanism for describing and interconnecting data, services and modules. The RDF data model defines a simple model for describing interrelationships among resources in terms of named properties and values. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources. As such, the RDF data model can therefore resemble an entity-relationship diagram.

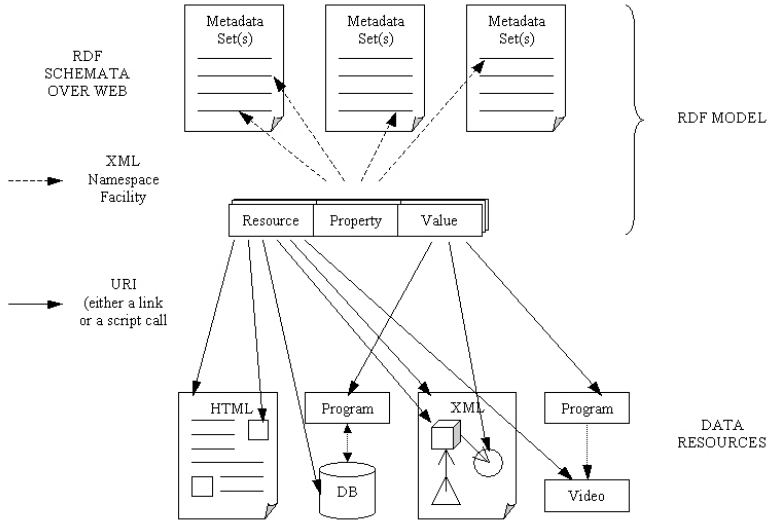

Fig. 3. Resource Description Framework (RDF) Model

A *Data Resource (DR)* in the RDF model is anything that can be referred and/or accessed through a Uniform Resource Identifier (URI) [16]. The declaration of the properties and their corresponding semantics are defined in the context of RDF as an *RDF schema*. A schema defines not only the properties of the DR (Title, Author, etc.) but may also define the types of DR being described.

RDF utilizes XML (eXtensible Markup Language) [17] as a common syntax for the exchange and processing of metadata. The XML syntax provides vendor independence, user extensibility, validation, human readability, and the ability to represent complex structures. By exploiting the features of XML, RDF imposes a structure that provides for the unambiguous expression of semantics and, as such, enables the consistent encoding, exchange, and machine-processing of standardized metadata. RDF also uses the XML namespace [18] facility to precisely associate each property with the schema that defines the property. A specific metadata set will be used to expose the interfaces of the modules/tools in the framework, thus providing the mechanism for their interoperability.

In summarizing, the design and implementation of the framework is based on the widely accepted standards of RDF, XML and URI. (R2). Metadata Sets (semantic schemata) are specified by RDF Schemata, the syntactic constraints (of metadata and structured DRs) by XML Schemata and the actual metadata and structured DRs are stored in XML. RDF and XML schemata are also valid XML files (according to their DTDs), enabling the application of XML content management tools to both RDF and XML schema management. XML technologies are constantly evolving and most prominent companies rush to support them in their products. This raises another strong requirement for developing and maintaining a modular framework, where the developer should be able test in practice the emerging technologies and tools, and decide which to adopt.

Our proposal/approach consists of four layers: Data Layer, Information Layer, Application Layer and Implementation Layer (see Figure 4). The separation of layers facilitates the applying of various development methods and conceptual models (R4).

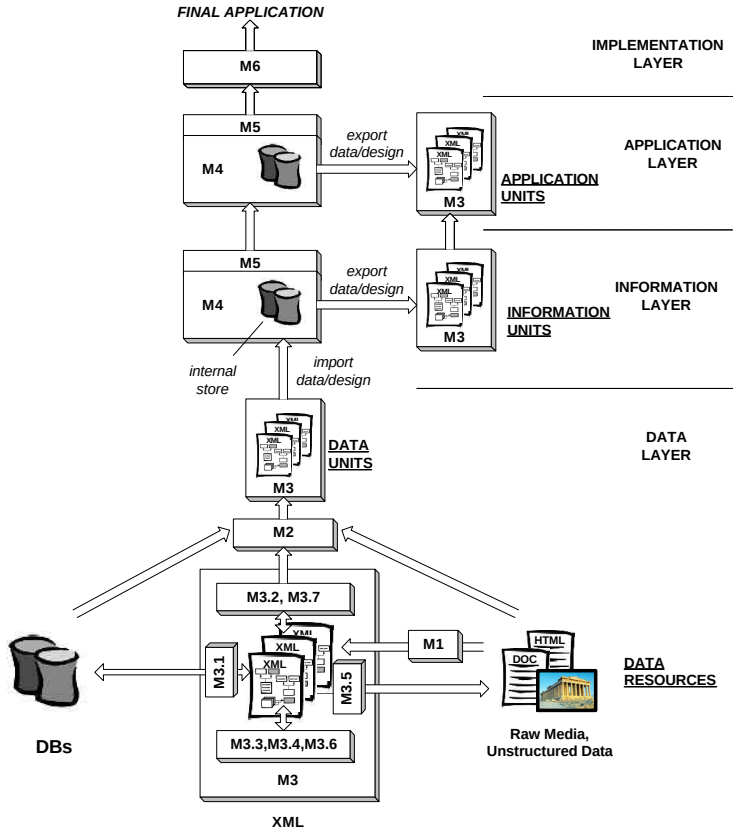

Fig. 4. The Modules of the proposed Framework

4.1 Data Layer

This layer provides management facilities of existing data resources (esp. hypermedia content). It provides a common way to access heterogeneous resources in a manner that is suitable for web applications. It converts the raw information into self-describing content, facilitating the importing or mapping of existing data resources into a web application. We define a *DATA-UNIT (DU)* as a self-describing DR. A DR and the sets of Metadata related to it form a DU. The main goals of this layer is to (a) provide services for querying and manipulating heterogeneous resources and (b) facilitate the exchanging and reuse of resources among developers.

The already existing resources, produced in the context of previous application development, are application-oriented, and in order to be reusable they have to be transformed into "neutral" resources ultimately (context-free content and structure). For instance, parts of text may have to be rewritten; the presentation has to be separated from the content (e.g. the transformation of an HTML file to XML/XSL files); the internal structure of the resource and the relationships with other resources has to be revised (e.g. when an HTML page is moved from a Web site to another, the link to the home page must change) etc. This is neither an easy nor a trivial task. The general guideline for facilitating the reusability of the resources is to keep textual information as context-free as possible and in structured forms (e.g XML, DB, etc.) and multimedia information (video, audio, images) in high resolutions and large sizes.

The Data Layer provides the following three modules. Next to the name of each module, we indicate the basic requirement(s) that the module meets.

M1. *Conversion Module.* (R5) A primary service is the transformation of unstructured textual information (HTML, PDF, MS DOC, etc.) to (semi) structured types (e.g. XML, SGML, DB, etc.). This will provide the ability for querying of data, referring to a part of the resource, reusing the whole or part of resource, etc. This service will be based on converters or more generic purpose tools, like structural and text pattern-matching software.

M2. *Metadata Module.* (R5, R6) A set of tools for specification, authoring and management of metadata for the DRs. It should support automatic extraction of selected metadata from the actual DR and strong management features for adaptability to future metadata standards. Moreover, the information on how to access or query structured DRs (even a DB) is exposed using metadata information. Finally, the relationships among DRs are specified through special metadata. Metadata information is stored in XML and it is based on RDF.

M3. *XML Module.* (R1, R2, R6, R7) This is the core of the framework. Essentially, this module is an XML Toolkit. Because of its importance and size we introduce the following sub-modules. Each sub-module consists of several tools. The users of the framework should select one of these tools according to their needs and the features of the tools.

M3.1. *XML-DB Module.* This module constitutes of a set of tools that can transform data and designs from DB to XML or vice-versa. Different approaches are needed according to the complexity of the designs and the type of data (e.g. binary data). XML and DTD can directly be mapped to object-oriented and hierarchical databases and can be mapped to relational databases using traditional

object-relational mapping techniques. Moreover, there are plenty of techniques and tools for transferring a database schema to a DTD or XML Schema. For instance, a promising standard is XMI (XML Metadata Interchange Format) that facilitates the exchange of UML (Unified Modeling Language) models and MOF (Meta Objects Facility) meta models among applications.

M3.2. *XML Parser Module.* Important features: validation on DTD or XML Schema, support of DOM, SAX, XPATH, XPOINTER, XLINK, XCatalog, etc.

M3.3. *XML Editor Module.* Important features: restrict editing according to DTD or XML Schema, validation on DTD or XML Schema, editing of DTD, XML Schema, XSL, RDF, etc.

M3.4. *XSLT Processor Module.* Important features: Support of XSLT1.0, input/output in DOM and SAX, multiple output documents, etc.

M3.5. *XSLT Formatting Objects Module.* The main feature is the supporting output formats.

M3.6. *XML Management Module.* Important transformations: XML to DTD, XML to XML Schema, DTD to XML Schema, XML Schema to DTD, XMLA to XMLB (by example), XML of DTDA to XML of DTDB. Other important management operations include: Pattern match/replace, find/manage tree differences among XMLs, etc. Moreover, utilities over textual information could be incorporated in this module, like spelling/grammar checkers, thesaurus, tools for semantic analysis and automatic extraction of relationships etc.

M3.7. *XML Query Module.* This module will provide querying and navigation mechanisms over the Data Units. The query mechanism could be applied on the metadata and/or the (semi) structured DRs like DBs and XML documents. Moreover, an XML reference mechanism should be supported to facilitate the mapping of DUs to application resources. Some features include the support of XPATH, XML Query (XQL), XML-QL, etc.

Obviously, rigorous testing and evaluation (R8) is required for the proper selection, tailoring and integration of tools within a module, as well as for the seamless interconnection of modules within the layer. This is the responsibility of the system/framework developer, who should construct a robust environment using individual parts, while paying special attention to topics such as interoperability, support for multiple languages, usability of the system by the content authors. *The same principle holds for all layers*, so that the results of each layer/module/submodule fulfill the requirements of the framework.

Additionally, Data Units should have well-prepared content and metadata, easily comprehensible to the end-user. This is the responsibility of the content authors, who can be assisted by tools such as statistics generators for missing (required or not) data fields, syntax checkers, content complexity analysers.

4.2 Information Layer

In this layer the developer analyses the information space of the application domain and designs the information structure using the more appropriate conceptual model (OO, ER or LDG) and the available design patterns. There is no concern for the types

of users and tasks, only for the information domain semantics (ontologies). Afterwards, the developer can implement the design through a design tool (RDBMS, OODBMS, XML Schema Editor, etc.). The data stored in the instance of the design is either authored by the information provider or extracted from the DUs of the Data Layer (through the XML Query Module over the metadata and/or data of the DUs). This facilitates the importing/reusing of existing information and the rapid information structure development. The structure and the information itself should be context-free (as far as possible), in order to increase the reusability of the information.

The design tools keep the design and data of the model in their native format. In order to facilitate the reuse of data and designs developed with different models, the solution is to have a common format that can express all the conceptual models in a way that supports interoperability, manageability, reusability and machine understanding (R3). This common format is XML Schema. With XML Schema the developer can import, for instance, an LDG design in an OO design and then manipulate it with OO design capabilities. On the other hand, the developer will be able to convert, e.g. an OO design to ER or a LDG, losing of course much of the powerful OO capabilities but making the designing much simpler. The conceptual design in XML Schema and the corresponding data in XML comprise an *INFORMATION-UNIT (IU)*, because it can be easily transferred among developers and applications as a unit.

Regarding the issues of QA (R8), the same principles as in the Data Layer hold.

Information Layer provides the following modules (that should closely co-operate with XML Module – M3):

M4. *Conceptual Model Design Module.* (R3, R4, R6, R7) A tool for supporting the *design and storing* of the conceptual model (e.g. OODBMS, RDBMS, XML Schema Editor). It should support (a) importing of designs in XMI, XML Schema or DTD, (b) the association (importing or mapping) of data elements of the design (objects attributes, rows of tables or XML elements) to DUs metadata or data through the XML Query Module or XML Parser Module (by query or reference), (c) authoring of new content and updating of existing through an authoring environment that may be automatically derived from the design of the conceptual model, (d) management operations on the design of the information space and propagation of these to the data of the model (e.g. merging of two attributes of an OO class, or elements of an XML schema, etc.) and support of management operations on data - both content and structure. Thus, important features include: Design importing/exporting from/to XMI, XML Schema and DTD, Data importing/exporting from/to XML, evolution of the design and propagation to data, efficient authoring environment etc.

M5. *Access Module.* (R4) This module provides querying, navigation and reference mechanisms over the data of the Information Layer. Important features include: a query service on the conceptual models (e.g. SQL for ER) and a reference mechanism for the data of the conceptual model. The XML Query Module can be used also for accessing the XML format of design and data of Information Layer.

4.3 Application Layer

In this layer the developer analyses the types of intended users of the application and their tasks and designs the navigational structure (application logic of a hypermedia application) over the information space. For the navigational structure, we adopt the OOHDM development method, where it is described in terms of a *navigational schema* and *a context diagram.*

The schema should support at least the basic concepts of hypermedia systems i.e. nodes, links, anchors and access structures (such as indexes, guided tours, etc.). Nodes and links represent logical views on conceptual data model defined during information layer. By defining the navigational semantics in terms of nodes and links, we can model movement in the navigation space independently of the conceptual model, while modifications into the conceptual model can have an impact in the navigational one, which the environment should propagate automatically.

Once the navigation classes have been decided, it is necessary to structure the navigation space that will be made available to the user. In OOHDM this structure is defined by grouping navigation objects (i.e. nodes, links, context classes and other nested navigational contexts) into sets called *contexts*. The navigation structure of the application is defined in a *context diagram*, which shows all the access structures and contexts defined for this application, and the possible ways of navigation between them.

The navigational schema and the context diagram can be designed by using the more appropriate conceptual model. Design patterns describing known navigation solutions may be applied. The Application Layer uses the Conceptual Model Design Module of the Information Layer. The data stored in the instances of the designs is extracted from the Information Layer, through its Access Module. On the other hand, the Access Module of the application layer provides querying and reference mechanisms over the data of this layer.

The data and designs of this layer, when represented in XML/XML Schema, comprise an *APPLICATION-UNIT (AU)*, because it can be easily transferred among developers as a unit and can be re-used in various implementation environments.

To enhance the quality of application units, unnecessary complexity in the context diagrams should be avoided. Metrics such as mean number of steps until the user finds the most "obscure" information can be calculated and utilized, e.g. in order to determine the granularity of the categories of a thematic catalog, on top of specific content

4.4 Implementation Layer

In this layer, we address issues of the applications' user interface (UI) that are obviously based on the capabilities of the implementation environment (e.g. WWW-HTML pages, WWW-Cocoon Servlet, WWW-ASP, a CD-ROM, etc.). In this layer the developer *specifies interface aspects and behavior* of navigational classes (nodes, links, access structures) according to the selected implementation environment. The Interface Layer provides the following module:

M6. *Implementation Module.* (R7) Tools for supporting the implementation of the interface aspects and behavior of the Navigational Classes, using technologies provided by the implementation environment (e.g. for the WWW-HTML through HTML templates that contain HTML code mixed with references and/or queries to the Navigational Schema and the Context Diagram). For the Web, such tools could be HTML editors, javascript debuggers, Java development environments, etc.

Especially important is the implementation of unambiguous user interfaces (R8). The tasks that the user can perform should be crystal clear and workflow should be straightforward. Help should be one click away and the use of wizard-driven user interaction offers significant advantages, especially for mainstream visitors. An approach that can be followed by implementers is the task-centered user-interface design [19], which emphasises analysis of both tasks and users. The same approach can be employed by the system/framework developers in the creation of the other layers' UI, which in effect are the framework front ends for content and application authors.

4.5 Collaboration Module

Additionally, we introduce a general module, which facilitates the collaborative work over the resources of the framework.

M7. *Collaboration Module.* (R7) A set of collaborative tools that enable people to work together during the design, development and maintenance phases. This module maintains the relationships between a working group’s members, tasks and information and provides services that may include: Multi-user access, Access control, Member activity tracking, Versioning, Notification control etc.

4.6 Overall Testing and Quality Assurance

Even though we attempt to provide QA within the layers of the framework, there is always the need to evaluate the final outcome, which can be a content category of a Web site or an entire application. To assess the quality of WWW site content, one can measure criteria such as easy navigation, improved availability and precision of information, enhanced guidance of the user throughout the navigation process. We summarise several practical approaches which, in addition to traditional testing by closed user groups, can highlight, among others, problems in content structure, navigation, flow and interaction of the application:

- Use of log analyzers, parsing the paths of users throughout content and applications and measuring data such as number of hits, depth of traversal graph, time (links) spent within the site, time (links) spent within specific categories, traversal patterns.
- On-line questionnaires for the users, requesting information on the usability of the site, attempting to evaluate and measure overall user satisfaction

- Features (mini-polls) embedded in the content/application pages, such as simple "voting" for correctness of the results of an operation (e.g. "See Also" links) and for rating the style and overall quality of the content presented (e.g. from 1 to 5)

Problems detected can be solved on a modular basis, depending on the layer which is involved. Additionally, this new knowledge can be embedded within the framework (e.g. in a design pattern), in order to improve the performance of the design and implementation process.

5 Implementation

To implement the proposed framework, the following tasks have been scheduled and are in progress:

- For each module, review of the available technologies and tools. (*continuous task*)
- Selection of the most appropriate ones and tailoring to support the interfaces of the modules (see Table 1 for tools that have already been selected and tested).
- Implementation of the glue logic and the Web interface of the framework. The framework is running under an Apache Web server (in Windows NT and Solaris 2.7), through the Cocoon Servlet (Apache XML Project), and will soon provide a robust set of XML-related libraries and supportive tools. Perl and Java are used as the scripting/programming languages, because of their ease of use, wide-spread appeal and the large (and ever increasing) library of available code.
- Testing of the framework with various methods, models and on different application domains.

Table 1. Tools already incorporated in the Framework

Modules	XML Apache xml.apache.org	IBM XML http://www.alphaworks.ibm.com/	OTHER
M1			Rhythmyx XSpLit (www.percussion.com)
M2			SiRPAC (www.w3c.org)
M3.1		XLE, Visual XML Creation	
M3.2	Xerces Java	XML4J3.01	XP (www.jclark.com)
M3.3		Xeena 1.1, Visual DTD	XML Spy (www.xmlspy.com)
M3.4	Xalan Java	Lotus XSL	XT (www.jclark.com)
M3.5	FOP Java (for PDF)		
M3.6		DdbE, XtransGen, Visual XML Transformation, PatML, XMLTreeDiff, X-IT	
M3.7		Visual XML Query	
M4		XMI Toolkit	
M6			Allaire HomeSite (http://www.allaire.com)
M7			RCS, MS SourceSafe

As application domains we have selected Culture (the primary Web site of the Hellenic Ministry of Culture, "ULYSSES" at http://www.culture.gr) and Sports (the primary Web site of General Secretariat of Sports in Greece), which both feature:

- extensive content, ranging from thousands of static HTML pages to integrated DBs (e.g movable artifacts of the Acropolis of Athens)
- various Web applications, including thematic catalogues, user-profile based navigation, search systems, metadata-based search systems, geographical map-driven navigation.

Using the framework and the guidelines presented in this work, we are proceeding with the restructuring of existing applications and content from both sites, into components that are gradually incorporated within the joint framework.

Significant progress has been achieved within the last months. Entire content categories of the WWW sites have been remodeled (e.g. transformation of more than 1500 existing HTMLs with associated metainfo into an E-R DB). Common metadata sets have been added to content from heterogeneous sources, including HTML-XML document collections and DB stores, thus constructing a pool of content (DUs/IUs) and enabling transparent access for Application Units development.

Prototype AUs have also been implemented, offering services such as guided tours, thematic catalogs, site-map generators on top of DUs/IUs. The interfaces to the AUs (implementation layer) consist of Java applets, Javascript & HTML files, Flash applications. The new version of ULYSSES is expected to be online by Fall.

6 Conclusions

In this work we presented the current status of Web development and outlined the major problems, current or upcoming, that a Web developer faces. In spite of numerous tools and technologies, several basic requirements of developers are still not being met. We summarized these requirements and proposed a more formal framework that could support them. We emphasized on the issues of content/application reuse, ease of maintenance and interoperability.

This framework builds on a set of guidelines that a developer should always consider, and intends to provide a generic development support environment. Its basic characteristics are: scalability (modular architecture formed by modules/tools) and technology independence (ability to replace tools in order to incorporate forth-coming techniques or technologies). The core technologies are XML and RDF.

The viability of this framework is already being explored through an implementation scenario, which is based on the content and application/services of two of the largest Web sites in Greece. This process also allows us to examine various technical solutions for the diverse modules and services of the framework. This hands-on experience and alternate technical implementations of the framework will also be disseminated to the Web developers' community.

References

1. Gamma, R., Helm, R., Johnson and J. Vlissides, “Design Patterns: Elements of reusable object-oriented software”, Addison Wesley, 1995.
2. Rossi, G., Schwabe, D., and Garrido, A., “Design Reuse in Hypermedia Application Development”. Proceedings of Hypertext'97, Southampton, UK, 1997.
3. Royce, W., “Managing the Development of Large Software Systems,” Proc. Int’l Conf. Software Eng., IEEE CS Press, 1987, pp. 328-338.
4. Boehm, B., “A Spiral Model for Software Development and Enchancement”, Computer, May 1988, pp. 61-72.
5. Lowe D., Bucknell A., and Webby R., "Improving hypermedia development: a reference model-based process assessment method", Proceedings of ACM Hypertext '99, Pages 139-146, February 21 - 25, 1999, Darmstadt Germany.
6. Christodoulou, S., Styliaras G. and Papatheodorou, T., "Evaluation of Hypermedia Application Development and Management Systems", ACM Hypertext '98, Pittsburgh, PA, USA, 1998
7. Schwabe, D. and Rossi G., “The Object Oriented Hypermedia Design Model”, Comm. of the ACM, Vol. 38, #8, pp45-46 Aug. 1995.
8. Schwabe Daniel, Rita de Almeida Pontes and Isabela Moura, "OOHDM-WEB: An environment for implementation of Hypermedia Applications in the Web”, SIGWeb Newsletter, June 1999, Vol. 8, No. 2
9. Isakowitz, T. Stohr, E. and Balasubramanian, P., “RMM: A Methodology for Structured Hypermedia Design”, Communications of the ACM 38 (8), 1995, pp. 34-44.
10. Kesseler, M., “A Schema-Based Approach to HTML Authoring”, W3 Journal.
11. Fernadez, M. Fiorescu, D. Kang, J. Levy, A. and Suciu, D., “STRUDEL: A Web-site Management System”, ACM SIGMOD 1997.
12. Dublin Core Metadata for Resource Discovery, Internet RFC 2413, http://www.ietf.org/rfc/rfc2413.txt
13. Platform for Internet Content Selection, http://www.w3.org/PICS/
14. Resource Description Framework (RDF) Model and Syntax, http://www.w3.org/TR/REC-rdf-syntax/
15. Lagoze, C. Lynch, A. and Daniel, R., “The Warwick Framework: A Container Architecture for Aggregating Sets of Metadata”, Warwick Metadata II Workshop, http://cs-tr.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR96-1593
16. Berners-Lee, Fielding, Masinter, “Uniform Resource Identifiers (URI): Generic Syntax”. Internet Draft Standard August 1998, RFC2396.
17. Extensible Markup Language (XML) 1.0; World Wide Web Consortium Recommendation, http://www.w3.org/TR/REC-xml.
18. Bray, Hollander, Layman, “Namespaces in XML”; World Wide Web Consortium Recommendation, http://www.w3.org/TR/1999/REC-xml-names-19990114.
19. Lewis, C., Rieman, J., “Task-centered User Interface Design: A practical introduction”, ftp://ftp.cs.colorado.edu/pub/cs/distribs/clewis/HCI-Design-Book