

# WEP: A Reference Model and the Portal of Web Engineering Resources

Sotiris Christodoulou and Theodore Papatheodorou

High Performance Information Systems Laboratory  
Computer Engineering & Informatics Department,  
Building B, University of Patras, 26500 Rion, Greece  
{spc, tsp}@hpclab.ceid.upatras.gr

**Abstract.** This paper introduces the Web Engineering Resources Portal (shortly WEP), as a basic Reference Model and Guide for the Web Engineers. WEP provides a general classification of Web Engineering Resources under technologies, research results and tools. It consists of a Reference Model and a Resources Portal. The objective of WEP Reference Model is to provide a common basic terminology, a technical-oriented classification of WebApps, a specification of WebApps Logical and Physical Architectures, a classification of skills needed in Web Projects and a generic and adaptable Web lifecycle process model. WEP Reference Model provides the framework upon which Web Engineering resources will be classified and presented in WER-Portal. The aim of WER-Portal is to provide several and cross-referenced taxonomies of technologies, research results and tools and to facilitate Web Engineers to comprehend available resources, understand their role and appropriately use them during Development and Operation/Maintenance of Web Information Systems.

## 1 Introduction

Web Engineering is defined in [4], by experienced researchers in the field, as: “*The application of systematic, disciplined and quantifiable approaches to development, operation, and maintenance of Web-based Information Systems (WIS). It is both a pro-active approach and a growing collection of theoretical and empirical research in Web application development*”. But, what a WIS is? Holck [6] provides a good survey of WIS definitions around the literature, where we can observe confusion, because of diverse perspective and terms used. The first thing Web Engineers need is a common terminology on WIS and its components. To address this need, we provide in section 3.1 the WEP Basic Terminology & Definitions. We replicate the definitions of WIS and Web Applications here as well.

WIS is an information system, utilizing Web technologies to provide information (*data*) and functionality (*services*) to end-users through a hypermedia-based presentation/interaction user interface on web-enabled devices. *Web Applications (WebApps)* are the different functionality-oriented components of a WIS. Actually a WebApp is a WIS but in a small-scale, providing very specific information or functionality.

Moreover, we define the “planning, development, operation and maintenance of WIS” as a *Web Project*. Basically, it is a *lifecycle process model* to ensure successful

WIS development and evolving through a number of stages from investigation of initial requirements through analysis, design, implementation, testing and operation / maintenance. In each stage the process model specifies the activities that are carried out, the relationships between these activities, the skills needed (roles), the resources that are used, the results that are created, etc. The activities are carried out by *Teams of Developers* who are based on selected *Web Technologies*, take advantage of selected *Research Results* and use a number of *Tools*. This triplet constitutes the Web Engineering Resources (shortly WER), which actually includes anything available to developers to support the Web Project. Fig. 1 shows how WERs are produced and related to each other. However, WERs are not easily discoverable and understandable by developers and thus they are not used appropriately or at all during Web Projects.

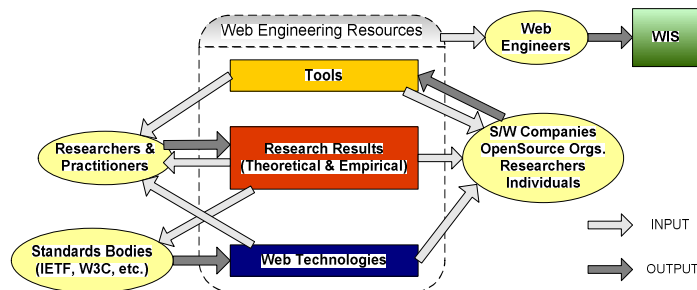


Fig. 1. Web Engineering Resources

Web Engineers' first choice for a Web Project is the lifecycle process among many and similar ones. Additionally, in several stages of the process, they have also to choose among several software tools, technologies and research resources. The problem is getting even bigger if we consider that tools and technologies (i.e. standards) are shifting extremely fast in the Web world and their volume is big. As Nambisan and Wang [11] state, "*Technology-related knowledge barriers are intensified by the fact that much of the Web technologies are not yet mature, making the task of choosing from among alternative technological solutions a challenging one*". Furthermore, Web Projects span to a variety of application domains and involve stakeholders of different backgrounds. Thus they have different requirements for methodologies, tools and technologies, even for different parts of the same WIS.

Finally, some research results, like [5], specify required skills for developers working on different parts of WIS development. However many real projects today are carried out with crucial roles or skills missing. Thus, not-skilled or inexperienced developers need help to quickly understand what Web Engineering can offer to them.

Based on our extended experience for several years on building large-scale Web-based systems and on our study / research [2], we have concluded to the followings:

- In several stages of web development processes, developers are asked to consider carefully and choose correctly the appropriate technologies to base on their development. However, these processes are not providing any way to help achieve it. They assume (or require from) developers to have the appropriate technology knowledge and experience. This is not true for most Web Developers.

- Few research results are transferred to real-life Projects. Web Engineers need time to study the research results in the fields of Web engineering and others affecting it, like multimedia, data management, software engineering, networks, etc.
- Emerging technologies are used hesitantly in the beginning and it takes a lot of time for them to be adopted by a large part of web developers. They need time to study and understand new technologies in such a broad field.
- Developers need time to use and understand new tools, like development platforms, emerging languages, etc.

It is clear that Web Engineers have to continually be in a process of studying, understanding and experiencing (use and test) emerging tools and technologies. They need to exhaustively study the recent research results, in order to gain the knowledge, experience and skills to decide correctly. This is a time consuming task and it is hard for most Web Engineers to follow in the strict schedule of a Web Project. Thus, WERs are not used appropriately or at all during current WIS Projects. Cutter Consortium [3] provides some statistical data on Web Projects that prove this.

We strongly believe that there are solutions out there but are not easily discoverable and understandable by Web Engineers. Web Engineers need help and guidance in accessing the knowledge and experience of web development. Current solutions include: *Design patterns*, *Good practices* and *tutorials* on technologies / tools. What is missing? An overall view and structure of WERs under several taxonomies that helps you find what you need, and provide you with references to study and explore it yourself. Web Engineering is a complex information space that needs to be engineered, in order to provide the WERs to developers in a structured and meaningful way.

To address these needs we introduce the *Web Engineering Resources Portal* (shortly *WEP*). The main objective of WEP is to structure and present the WERs in a unified framework that provides a *Reference Model* and a *resources Portal*. We call it, *Portal* because it provides several and cross-referenced taxonomies of WERs, just like an Information Portal does. The aim of WEP is to constitute (1) for academics an important tool for the education of Web Engineering, and (2) for Web Engineers a reference model for the deployment of WISs and a guide in the discovery and apprehension of WERs. *WEP Reference Model* includes:

- (a) *WEP-Terms*: WEP Basic Terminology and Definitions of the main terms used in Web Engineering in order to clearly determine the semantics of these terms.
- (b) *WEP-Arch*: Technical-oriented *Classes of WIS components* (i.e. WebApps). Specification of the WebApps' *Logical Architecture* and *Physical Architecture*.
- (c) *WEP-Teams*: Specification and classification of skills needed in Web Project under abstract Team classes of stakeholders.
- (d) *WEP-Process*: A *WIS lifecycle process model* with three phases: Planning, Deployment and Evolution. It is a generic process model through which *WEP-Teams* are using *WERs* to develop and maintain a WIS based on the *WEP-Arch*. We keep this high-level process generic, easy for the developers to follow, comprehend and adapt to specific WIS requirements.

*WER-Portal*: Several *WER Taxonomies* through which Web Engineers will be able to easily and meaningfully locate research resources, web technologies and tools and understand their role during (1) WIS Development and (2) WIS Operation/Maintenance. The objective of *WER-Portal* is to facilitate Web Engineers to

comprehend and appropriately use available and emerging Web technologies / tools and to provide means to transfer knowledge (research results) and experience (patterns/good practices) in an easy and effective way. WER-Portal should be regularly updated in order to include new WERs.

This paper introduces the WEP Reference Model that provides the framework upon which the WERs will be classified and presented under the WER-Portal. This portal is currently under development. The full-expanded description of WEP Reference Model and some parts of the Portal's resources will be described in detail as a chapter of a book [1] that will be published soon.

### 3 WEP Reference Model

#### 3.1 WEP-Terms: WEP Basic Terminology & Definitions

In order for the readers to perceive the concepts outlined in WEP and web engineering, we have to share the same understanding of basic Web or non-Web terms.

##### A. General Terms

- *Data, Information, Software, Program, Application*: For these terms see [1].

##### B. Web Primitive Elements

- *World Wide Web* (known as "WWW", "Web" or "W3") as defined by W3C, is "the universe of network-accessible information, available through web-enabled devices, like computer, phone, television, or networked refrigerator...". The *Web* is a network-spanning information space in which the information objects, referred to collectively as *Web Resources*, are identified by global identifiers called *URIs* and are interconnected by *Web Links* defined within that space.
- *URI, Resource, Web Resource, Web Resource Manifestation, Web Link, Web Anchor*: These Web-oriented terms are used here as defined in [9].

##### C. Web Architecture Elements

- *Web User*: The principal using a web client to interactively retrieve and render *web resources* or *web resource manifestations*.
- *Web Publisher*: The principal responsible for the publication of a given *web resource* and for the mapping between it and any of its manifestations.
- *Web Site Publisher, Message*: As defined in [9].
- *Web agent* [8] is software acting on web information space on behalf of a person, entity, or process. Agents include servers, proxies, browsers, spiders, etc.
  - *Web Client (Web Request, User Session), Web Server (Web Response, Server Session, Cookie), Web Proxy, Web Gateway*: As defined in [9].

##### D. Web Architecture

*Web architecture* encompasses both protocols that define the information space, by way of *identification* and *representation*, and protocols that define the *interaction* of agents within Web. These three dimensions of Web architecture are discussed in [8].

##### E. Web Resource Structures

- *Web Page, Web Collection, SubSite, Web Site, SuperSite* as defined in [9].

## F. WebApp & WIS

- WIS or *Web-based Information System* is defined in WEP as an information system, utilizing Web architecture to provide information (*data*) and functionality (*services*) to end-users through a hypermedia-based presentation/interaction *user interface* on web-enabled devices. WISs vary widely in their scope, from informational systems to e-business transaction systems. A high-level *functionality-oriented taxonomy* of WISs is provided in [7]: “There are four general kinds of WISs: *Intranets, Web-presence sites, electronic commerce systems and extranets*”. Generally, a WIS deals with vast amounts of data, in heterogeneous sources and formats, and distributed functionality coded in different programming languages and platforms. Like traditional Information Systems, beyond a delivering (run-time) infrastructure, WISs should provide a development and maintenance infrastructure to allow the managing of its data, s/w functionality and agents. WISs are designed, developed and maintained in order to fulfill specific goals of targeted end-users. These goals are the cornerstones of a WIS Project.
- WebApp or *Web Application* is defined in WEP as a WIS component that covers at least one of end-users’ goals. WebApps are the different functionality-oriented components of a WIS. At run-time, a WebApp is comprehended by the end-users as a set of *WebPages*, that provides very specific information (e.g. the WebPages of a tutorial on JavaScript) or functionality (e.g. the collection of WebPages through which an end-user can order goods and pay with his credit card). A WebPage may provide access to more than one

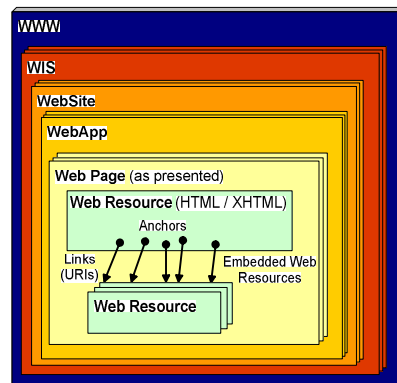


Fig. 2. From Web Resources to WWW

WebApp, e.g. an informative page that also includes at the top a form box for querying the search engine of the WIS, which is a separate WebApp. Actually a WIS is a large-scale WebApp that fulfills several end-users’ goals. That’s why most developers use these terms as synonymous, especially for small WISs.

Fig. 2 “visualizes” some of the above terms, from Web Resources up to WWW.

## G. Web Pages

According to the definitions of WIS and WebApps, the cornerstone of both is the *WebPage*. A WebPage [9] is a collection of information, consisting of one or more Web resources, intended to be rendered simultaneously, and identified by a single URI. More specifically, a Web page consists of a Web resource with zero, one, or more embedded Web resources intended to be rendered as a single unit, and referred to by the URI of the *main Web resource* which is not embedded.

- WebPage Main Resource* or *(X)HTML Page*: WebPages are built from a non-exclusive set of data formats, used separately or in combination (like (X)HTML,

CSS, PNG, Real Audio, SVG, JPG, MPEG). The WebPage Main Resource must be in a data format that can support linking mechanisms, identifying links to embedding resources or links to other WebPages. This data format is only limited by the WebClients implementations. Currently, the supported formats are: HTML and XHTML family markup document types. Thus, we also refer to the main resource as *(X)HTML Page*. These mark-up document types provide tag elements that can be grouped under tag sets of specific functionalities.

- *WebPages Tag Sets*: After studying the current (e.g. HTML, XHTML 1.0, XHTML Basic) and the near future ones (XTHML 2.0) we provide a general classification of WebPages tag sets in Table 1.

### ***G1. WebPages at WebServer side***

- *Static WebPage*: A Static WebPage and its embedded resources are stored under the WebServer as they delivered. Note that a Static WebPage can be “dynamic” and “active” on client-side, via ClientScripts, ClientPrograms or MediaObjects.
- *Dynamic WebPage*: Dynamic WebPages are produced dynamically (partly or as a whole), at run-time, upon user request by a *Server-Side Script (SSS)*, i.e. a *SESL-Page*, a *CGIProgram*, or a *JavaServlet*. All these technologies provide means to access various local or distributed data sources (e.g. DBs, directory servers, XML repositories, legacy systems), call local or distributed programs (e.g. web services) and manage workflow process (e.g. produce and manage Cookies) whilst being able to respond back to the client any information necessary as WebPages.
  - *SESLPage*: *SESL - Server-side Embedded Scripting Language*: Combine fixed or static data of (X)HTML with scripting code enclosed in special tags. This code can be in various languages (C, Perl, VBasic, etc.). When the *SESLPage* is requested by a WebClient, the *SESL* code is executed on the server and the dynamic content is mixed with (X)HTML static content to produce the WebPage to be sent back to the Client. Famous *SESLs* include: PHP, JSP, WebServer SSI, ASP, ASP.NET, ColdFucionML.
  - A *CGIProgram* is any program designed to accept and return data that conforms to the CGI specification. The program could be written in any language, including C, Perl, Java, or VBasic. Many (X)HTML pages that contain forms, use a CGI program to process the form's data once it is submitted.
  - *JavaServlet*: A small program that runs on a web server. The term usually refers to a Java applet that runs within a Web server environment. *JavaServlets* are becoming increasingly popular as an alternative to CGI programs. The biggest difference between the two is that a servlet is persistent. This means that once it is started, it stays in memory and can fulfill multiple requests.

End-users do not care whether a WebPage is static or dynamic. They request a series of WebPages in order to fulfill a goal. The decision whether a WebPage is better to be implemented as static or dynamic is a development issue.

### ***G2. WebPages at WebClient side***

Regarding the contents of WebPages delivered to WebClients, we classify them as Table 2 shows.

**Table 1.** HTML 4.01, XHTML 1.0 and Basic Tag Elements

General Tags		
Structure	<b>html</b> , <b>head</b> , <b>body</b> , <i>frameset(frame)</i> , <i>noframes</i> e.g. <code>html (head, body) or html (head, frameset(frame), noframes)</code>	
Head	<b>title</b> , <b>base</b> , <b>meta</b> , <i>style (StyleLang)</i> , <i>script (ScriptLang)</i> , <b>object(param)</b> (share objects across frames' WebPages) <b>link</b> : Links to alternate versions of a current document (written in another human language, designed for different media, for instance a version especially suited for printing), Links an external style sheet or script to a document, etc. <i>Link Types</i> supported by HTML: <i>Alternate, Stylesheet, Start, Next, Prev, Contents, Index, Glossary, Copyright, Chapter, Section, Subsection, Appendix, Help, Bookmark</i> (plus <i>Script</i> in XHTML).	
body and noframes tags can include the following tag sets:		
Structure & Presentation Tag Elements		
Core Text	Text	<b>p</b> , <b>pre</b> , <b>blockquote</b> , <b>div</b> , <b>h1-h6</b> , <b>address</b> <b>span</b> , <b>br</b> , <b>em</b> , <b>strong</b> , <b>dfn</b> , <b>code</b> , <b>samp</b> , <b>kbd</b> , <b>var</b> , <b>cite</b> , <b>abbr</b> , <b>acronym</b> , <b>q</b>
	List	<b>ul(li)</b> , <b>ol(li)</b> , <b>dl(dt,dd)</b> , <b>dir</b> , <b>menu</b>
Extra Text	Table	<b>table (caption, thead, tfoot, tbody, colgroup, col, tr, th, td)</b>
	Presentation	<b>tt</b> , <b>i</b> , <b>b</b> , <b>big</b> , <b>small</b> , <b>sub</b> , <b>sup</b> , <b>strike</b> , <b>s</b> , <b>u</b> , <b>font</b> , <b>fontbase</b> <b>hr</b> , <b>center</b>
	Misc	Edit: <b>ins</b> , <b>del</b> , Bi-directional text: <b>bdo</b>
Hypertext	<b>a</b>	
MediaObject	<b>img</b> , <b>object(param)</b> , <b>map(area)</b> , <b>iframe</b> Any multimedia format supported by Web Clients (natively or via Plug-ins) like, Raster Images (GIF, JPG, PNG), Vector Images (SVG, WebCGM), Animated Images (GIF), Animated Video (Flash, ShockWave), Video (MPEG, AVI, DIVX, MOV), Audio (MP3, WAV), Streaming Video (RealVideo), Streaming Audio (Real Audio), Docs (PDF, PS, MSOffice).	
ClientProgram	<b>object(param)</b> , <b>applet(param)</b> Small programs (except client scripts) that can be executed on Web Clients via Plug-ins, like: Java Applets, Flash, ShockWave, Active-X Controls.	
Interact Tag Elements & Events Attributes		
Form	<b>form</b> , <b>fieldset(legend)</b> , <b>isindex</b>	
	<b>input</b> , <b>select(optgroup,option)</b> , <b>textarea</b> , <b>label</b> , <b>button</b>	
Client-Script	<b>script (ScriptLang)</b> <i>Noscript</i>	
Events (attributes)	onload, onunload, onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onfocus, onblur, onkeypress, onkeydown, onkeyup, onsubmit, onreset, onselect, onchange These events' attributes may be used with most elements, esp. with Form ones. It is possible to associate an action with a certain number of events that occur when a user interacts with a WebPage. Each of the "intrinsic events" listed above takes a value that is a script, which will execute whenever the event occurs for that element. The syntax of script data depends on the scripting language.	
Embedded code of StyleLangs and ScriptLangs		
StyleLang	Any style sheet language (either embedded or in separate files linked through <b>link</b> tag or <b>src</b> attribute of <b>style</b> ) that attaches style (e.g. fonts, colors, content positioning) to different elements of structured documents (e.g., HTML, XHTML, XHTMLBasic and XML), such as HTML headers, links, tables or XML elements. Basic features: Input languages and Output devices. The most famous StyleLangs are: <b>CSS</b> (CSS2 that is build on CSS1) and <b>XSL/FO</b> . CSS can be applied to HTML and XML languages (like XHTML), while XSL/FO only to XML languages. Both support media-specific stylesheets for visual browsers, aural devices, printers, braille devices, handheld devices, etc.	
ScriptLang	Tiny program code (either embedded or in separate files linked through <b>link</b> tag or <b>src</b> attribute of <b>script</b> ) that can be executed on Web Clients natively, like: JavaScript, VBScript, JScript. Script data can be the content of the <b>script</b> element and the value of intrinsic event attributes.	

Elements in **Bold**: XHTML Basic 1.0 Tag Elements      B: Block level elements  
 Elements in *Italics*: Deprecated elements      I: Inline or "text-level" elements

**Table 2.** Client-side WebPages Classes

WebPage()	an (X)HTML file which may include the followings: (1) embedded StyleLang code (2) embedded ScriptLang code (3) link(s) to external StyleLang file(s) (4) link(s) to external ScriptLang file(s) (5) link(s) to embedded MediaObject file(s)
WebPage(F)	a WebPage() which also includes one or more Forms
WebPage(P)	a WebPage() which also includes one or more embedded ClientPrograms

Users can request a WebPage through a WebClient by typing a URI, by clicking on a link or by clicking on a button. They can request for either a Static WebPage, using an (X)HTML URI or a Dynamic WebPage, using an SSS URI. The response they get in both cases is a WebPage. In the case of a Dynamic WebPage request, WebClient may also get a *Cookie*, a text message produced by the SSS and transparently sent by the WebServer to WebClient. This message is stored locally and it is sent back to the WebServer each time the WebClient requests a new WebPage. Cookies are used for implementing personalization features of WebApps. Other kind of information may be also exchanged transparently between WebClients and WebServers, like user preferences and WebServer policy (e.g. P3P), at the first request.

Starting from a WebPage() users can follow the embedded links to request more WebPages. From a WebPage(F), they can either follow a hyperlink or fill the forms fields and press the submit button. Clientscripts may help them at this point. By pressing the submit button, they always request for a Dynamic WebPage, but their request includes also the form data. From a WebPage(P), users can interact with the Client-Program which is running locally within the WebClient. Some ClientProgram languages like Java, provide capabilities to create forms and hyperlinks within the programs GUI, and send requests back to the server. To summarize, a WebRequest can be: (X)HTML URI, SSS URI or SSS URI plus form data. All types of requests may include Cookies. The WebResponse is a WebPage and may include a Cookie.

## H. Web Meta-architectures

Web Architecture is extended by Web meta-architectures. Currently, the most important ones are *Semantic Web* and *Web Services*. For extended description see [1].

### HI. Semantic Web

The *Semantic Web* is not a separate Web but an extension of the current one, in which information is given a well-defined meaning, better enabling computers and people to work in cooperation. The principal technologies of the Semantic Web fit into a set of layered specifications called the RDF. The current components of this framework are the RDF Model & Syntax, the RDF Vocabulary Description Language (VDL) and the Web Ontology Language (OWL). All these languages are built on the foundation of URIs, XML, and XML namespaces. Fig. 3 presents Semantic Web Architecture and core Technologies.

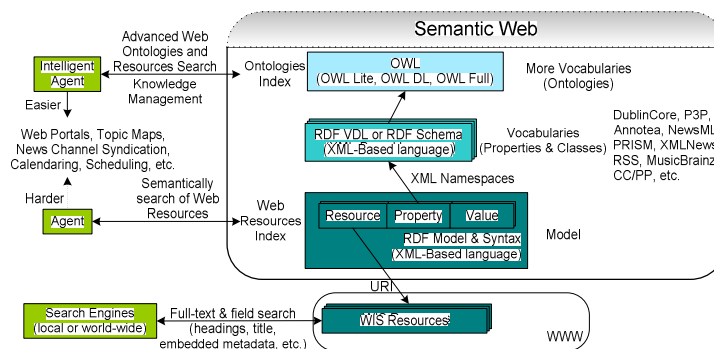


Fig. 3. Semantic Web Architecture and core Technologies

## H2. Web Services

*Web Services* are standard-based interfaces for software functionality, implemented as discoverable services on the Internet. While Semantic Web infrastructure is for discovering and consuming URI addressable data, Web Services infrastructure is for discovering and consuming URI addressable software logic. A *Web Service* is a software system, whose public interfaces and bindings are defined and described in a machine - processable format (WSDL). Its definition can be discovered (in a UDDI registry) by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages based on a messaging protocol (SOAP) conveyed by Internet protocols (like HTTP). Fig. 4 presents Web Services Architecture and core Technologies.

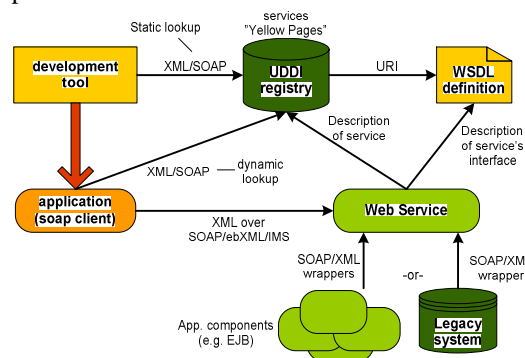


Fig. 4. Web Services Architecture and core Technologies

### 3.2 WEP-Arch: WebApps' Classes, Logical & Physical Architectures

WEP-Arch is a technology-independent reference model for the WebApps' technical-oriented Classes, Logical and Physical architectures. The aim of this component of WEP reference model is to facilitate WIS Project stakeholders, i.e. content / service providers, Web developers and end-users to comprehend:

- the diversity and complexity of WISs and WebApps.
- the logical architecture of WIS and WebApps
- the physical (run-time) architecture of WIS and WebApps

WEP-Arch provides a model and a context for understanding WIS, WebApps, their logical and physical components and the relationships among them. It integrates different conceptions of WebApps Classes under a common "reference architecture". While the concepts and relationships of WEP-Arch represent a full-expanded enumeration of the architecture components, the stakeholders are able to adapt the architecture in order to meet the goals and requirements of a specific WIS Project.

#### 3.2.1 WebApps' Technical-oriented Classes

Deshpande et. al. [4] provide a *functionality-oriented classification* of WebApps, that includes the following classes: Informational, Interactive, Transaction, Workflow, Collaborative work environments, Online communities / marketplaces, Web Portals

and Web Services. WebApps vary widely from small-scale, short-lived services to large-scale enterprise information systems distributed across the network, incorporating different technologies and data from heterogeneous sources and in various formats. It is obvious that the diversity and complexity of WebApps is high.

End-users classify a WebApp according to its functionality, which should facilitate them to achieve a specific goal. However, developers need to classify WebApps under “*technical-oriented*” classes, according to the skills, technologies and tools needed for designing, developing and maintaining them. After thorough study of current WebApps’ architecture and based on our experience in developing large-scale WISs, we provide such an abstract technical-oriented classification of WebApps as follows.

WIS is perceived as a large-scale Hypermedia Information Space that provides *information* and *services* to fulfill goals of targeted end-users. *Information* (as multimedia content in various formats, linked via hypertext metaphor and navigational structures) is provided by *Web Hypermedia Applications* through a collection of Static and/or Dynamic WebPages, while *services* are provided within these WebPages as *Web Front-End Applications* (WFA), through WebForms, ClientScripts & result WebPages and as *Web Interactive Applications* (WIA), through ClientPrograms.

- *Web Front-end Application (WFA)*: It is an integration application that uses the Web technologies to provide a web front-end (through WebPages(F)) to back-end information systems, business logic or legacy applications, e.g. registration forms, booking systems, web email, transaction applications etc. In this case, the application logic is mainly implemented using WebForms and ClientScripts on WebClient (e.g. Javascript code to validate users’ input) and SSS and programs on WebServer (e.g. PHP to query back-end data sources, call local or distributed web services and mix the results with (X)HTML code). A WFA may consist of one WebForm, many individual WebForms or a pipeline of WebForms (the results of a WebForm interaction is another WebForm page). Main Scope: Provide services (and content) of back-end systems over the web to end-users through WebForms interaction paradigm. Important part: The application logic (service-oriented applications). Skills needed: S/W Engineering, Network Engineering (distributed computing, web services), DB/Information Engineering, Web Publishing (Web Forms), Web Programming (Server-side scripting, client-side scripting).
- *Web Interactive Application (WIA)*: It is one or many ClientPrograms that use the Web infrastructure in order to reach end-users through a web-based GUI (that usually requires a plug-in installed on WebClient). Example of such applications include: interactive games, chat, interactive geographic maps, etc. Main Scope: Provide interactive experience over the web to the end-users. Important part: GUI and Performance (interaction-oriented applications). Skills needed: S/W Engineering, Human-computer interaction, Multimedia.
- *Web Hypermedia Application (WHA)*: it is the structuring of an information space in concepts of *nodes* (chunks of information), *links* (relations among nodes), *anchors*, *navigation contexts* (indexes, site maps, guided tours, etc.) and the delivery of this structure over the Web as WebPages. At server-side a WHA is a collection of Static and Dynamic WebPages() which incorporate the links and anchors of the WHA. A WHA can allow the navigation through the content of structured data sources like DB or XML repositories. Note though, that searching the same data

sources through a WebForm is not part of the WHA, but it is a WFA. Users interact with a WHA by following links. However, a WHA may include WebForms of WFA and ClientPrograms of WIA as embedded components in its WebPages. Interacting with these components is part of the other applications. **Main Scope:** Diffuse content over the web to the end-users. **Important part:** The content (content-oriented applications). **Skills needed:** Hypermedia Engineering, DB/Information Engineering, Web Publishing ((X)HTML, XML/XSL, CSS, RDF/metadata, web server admin), Web Programming (SSS), Multimedia.

WIS is a large-scale WHA, incorporating service-oriented (WFAs), interaction-oriented (WIAs) and content-oriented applications (WHA).

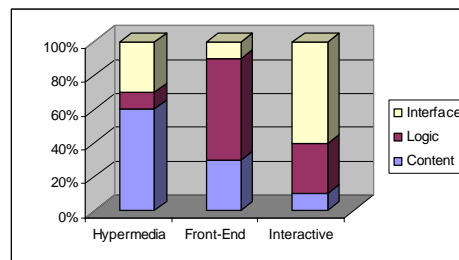
### 3.2.1 WebApps' Logical Architecture

We break-down WebApps into *Logical Components*, i.e. WebApps parts that can be designed and implemented to some extent independent. These components can be grouped into layers. Research results, practice and technology evolution shows that WebApps should be broken-down to three logical layers: *content*, *logic* and *interface*. Table 3 describes the Layers and Components of WebApps' Logical Architecture.

**Table 3.** WebApps' Logical Architecture (Layers & Components)

Logical Layers' Description & Components
<b>Content</b> layer includes the designs of data, metadata and WebApp navigation. For unstructured data it includes the type (e.g. image, video, animation, etc.). For structured and semistructured data, it includes the data schemata (like ER, XSD, OO, tree structure, etc.). Moreover, and for WHAs, it includes the navigational design as WebPages (nodes and navigational structure) and hyperlinks across them. Finally, it includes the metadata schemata for the WebPages and the individual Web Resources of the WebApp. Each data, metadata and navigation design constitutes a Content Component.
<b>Logic</b> layer includes the logical designs of all s/w components (program and script code) of the WebApp, like SSSs, ClientScripts, ClientPrograms, and server-side Programs. These Logic Components are designed to provide: data access logic, server-side logic (services of WFA), client-side logic (WIA ClientPrograms and WFA ClientScripts) and rendering logic (SSS of WHA and WFA).
<b>Interface</b> layer includes the designs of WebApp GUI through which end-users have access to the content and interact with the logic of the WebApp. Interface Components include the layout designs of the WebPages (WHA), the look-and-feel graphical designs of WebPages (WHA), the presentation designs of multimedia resources like video, animation, audio, etc. (WHA), the Web Forms designs (WFA) and the GUI design of ClientPrograms (WIA).

This three-layered architecture can also be expanded to a multi-layered one, where different layers of software implement the WebApp Logic and content is spitted to data and navigation layers. This isolated architecture facilitates Web developers to define, design and implement several reusable components in each layer (content, logic or interface components) and reuse them across several WebApps or even different WISs. Fig.5 shows the logical layers and their approximate importance for each class of WebApps.



**Fig. 5.** Logical Layers in each WebApps' Class

### 3.2.2 WebApps' Physical Architecture

The Physical Architecture describes the components of the WebApp at run-time. The *Physical Components* include implementations of Logical Components (like data sources, programs, stylesheets, etc.) and the run-time infrastructure of the WebApp (Web Client types, communication protocols, Web server and its modules, run-time environment of Logical Components, network infrastructure to access distributed resources, etc.). Contrary to logical architecture of WebApps which is simple and generic, the physical architecture is complex, multi-tiered and specific. The physical tiers do not associate directly to the logical layers. Instead, one Physical Component may implement Logical Components of various logical layers. The Physical Tiers, Components and Composite Components of WebApps are defined in Tables 4 and 5.

The Tiers of the physical architecture and their mapping to logical layers is shown in Fig. 6. Fig. 7 presents the physical components and composite components of each physical tier and shows how components are interoperate at high-level.

Finally, Table 6 presents the physical components for each class of WebApps. WebServer and WebClient physical components are common to all.

**Table 4.** Physical Tiers of WebApps

Physical Tiers' Description
<b>Data</b> physical tier includes the actual content of the WebApp, digitized and stored in a specific data format under certain systems/tools able to store and manage it (e.g. content management system, RDBMS, XML repository, directory server, (X)HTML files on filesystem, etc.). Legacy systems belong to this tier, as WebApps use them as black-boxes providing data. According to the data format and management system of data we distinguish three types: MediaObjects, Semi-StructuredData, StructuredData. Moreover, this tier includes data that does not correspond to Content Components, like XSL, CSS, ClientScripts, ClientPrograms. These are perceived as "data" for the WebApp at run-time, even though they implement Logic and Interface components, because they are delivered by the WebServer as files and used on WebClient side.
<b>DataAccess</b> physical tier includes the technologies (like SQL) and programs implementing the data access logic (e.g. ODBC, XML Parser, specific programs, etc.) which are used in order to access data at WebApp run-time. This physical tier implements the data access Logic Components.
<b>AppLogic</b> physical tier includes all programs that implement the core (local with regard to WebServer) Logic Components of WebApp (server-side Logic Components) and their run-time environment (e.g. an application server, J2EE, .NET, etc.).
<b>Distribution</b> physical tier includes the technologies (e.g. CORBA, SOAP) and programs (e.g. distributed web services) that implement the distributed Logic Components of WebApp and their run-time environment.
<b>Rendering</b> tier includes the code and run-time environment of CGIPrograms, SESLPages and JavaServlets.
<b>WebServer</b> physical tier includes the software and run-time environment of WebServer plus the administration and management tools (administration environment, log analysis tools, etc.). This tier may include the policy profile of the WIS stored in a format like P3P.
<b>Communication</b> tier includes the communication protocols and the supported data formats of the WebApp plus the network infrastructure to support the communication with web-enabled devices and WebClients.
<b>WebClient</b> tier includes the set of WebClients (browsers and devices) that are supported by the WebApp. Moreover, it includes the necessary plug-ins for MediaObjects and ClientPrograms and user preferences.
<b>GUI</b> physical tier includes the WebApp GUI through which end-users access content (view on screen, hear on speakers, etc.) and interact (with mouse, keyboard, etc.) with the WebApp client-side logic physical components. GUI Components include WebPages (layout and presentation style) and their embedded MediaObjects, Web Forms, ClientScripts and ClientPrograms.

**Table 5.** Physical Components & Composite Components of WebApps

Physical Components		Map to Logical Components
Data Sources	<b>Semi-StructData:</b> (X)HTML files, XML Native DB, XML Server/Repository, XML files (may have links to MediaObjects, ClientScripts, ClientPrograms, Stylesheets)	Content (data, links, metadata)
	<b>StructData:</b> DBs, Directory server, email server, Index server, legacy EIS, etc. (may include or link to MediaObjects, ClientScripts, ClientPrograms, Stylesheets)	Content (data, links, metadata)
	<b>MediaObjects :</b> Media files (images, video, animations, etc.), Docs (PDF, MSOffice, etc.)	Interface
	<b>MetaData:</b> Metadata of WebApp Web Resources (RDF/XML, OWL, RSS, etc.)	Content (metadata)
	<b>WebForm:</b> (X)HTML tags, XForms, etc.	Interface
	<b>StyleSheet:</b> CSS, XSL, etc.	Interface
	<b>ClientScript :</b> JavaScript, VBScript, Jscript, etc.	Logic
	<b>ClientProgram:</b> Java Applets, Flash, ShockWave, Active-X Controls, etc.	Logic, Interface
	<b>SSS :</b> SESL (PHP, ASP.NET, ASP, JSP, ColdFusion, etc.), CGIPrograms, Servlets (Java)	Logic (rendering)
	<b>AccessProgram:</b> Data Sources Access Programs or technologies, like XML parser, XQuery program, ADO.NET, LdapSearch, etc. These programs are based on a Data Source Querying technology, like SQL, XQuery, LDAP, JDO, JMS.	Logic (data access)
	<b>ServerProgram:</b> Any kind of S/W component that implements the core server-side logic (either local or distributed). One of them is invoked by a SSS and returns an output (possibly after invoking several others ServerPrograms). ServerPrograms include: 1) Individual Programs in any programming language (like JavaBeans, .NET Managed Classes etc.), 2) Infrastructure Servers (Application server like J2EE Server, .NET Framework, etc.) and 3) Distributed Programs (like WebServices, etc.)	Logic (local / distributed)
<b>WebServer</b>	Fundamental components for every WebApp responsible to provide responds and requests to each other. Moreover, they provide security (e.g. SSL) and policy (e.g. P3P) features of the WebApp.	Logic
<b>WebClient</b>		Logic
Physical Composite Components		Map to Logical Components
	<b>Static WebPage():</b> (X)HTML + embedded StyleLang code + embedded ScriptLang code + link(s) to external StyleSheets + link(s) to external ClientScripts + link(s) to embedded MediaObjects	Content (data, links, metadata), Logic (light client-side) Interface
	<b>StaticWebPage(F):</b> StaticWebPage() + WebForms (link to an SSS URI)	Content (data, links, metadata), Logic (form client-side), Interface
	<b>StaticWebPage(P):</b> StaticWebPage() + ClientPrograms	Content (data, links, metadata), Logic (client-side), Interface
	<b>DynamicWebPage():</b> a StaticWebPage() produced by an SSS (Input: SSS URI)	Content (data, links, metadata), Logic (data access, client-side), Interface
	<b>DynamicWebPage(P):</b> a StaticWebPage(P) produced by an SSS (Input: SSS URI)	Content (data, links, metadata), Logic (data access, client-side), Interface
	<b>DynamicWebPage(F):</b> a StaticWebPage(F) produced by an SSS (Input: SSS URI)	Content (data, links, metadata), Logic (data access, form client-side), Interface
	<b>DynamicWebPage(FR):</b> a StaticWebPage () or (F) produced by an SSS (Input: SSS URI + form data)	Content (data, links, metadata), Logic (data access, local/distr., client-side), Interface

**Table 6.** Physical Components and Composite Components per WebApp' Class

Physical Tiers	Hyperspace	Front-End	Interactive
<b>Data (CLI)</b>	DataSources, MediaObjects, StaticWebPages(), StyleSheets, ClientScripts	DataSources, MediaObjects, StaticWebPages(F), StyleSheets, ClientScripts	DataSources, MediaObjects, StaticWebPages(P), StyleSheets, ClientScripts, ClientPrograms
<b>Data Access (L)</b>		AccessPrograms	AccessPrograms
<b>AppLogic (L)</b>		ServerPrograms	
<b>Rendering (CLI)</b>	DynamicWebPages()	DynamicWebPages(F) or DynamicWebPages(FR)	DynamicWebPages(P)
<b>GUI (CLI)</b>	WebPages() with: StyleSheets, ClientScripts	WebPages(F) or () with: StyleSheets, ClientScripts (form)	WebPages(P) with: StyleSheets, ClientScripts, ClientPrograms

C= Content      L=Logic      I=Interface

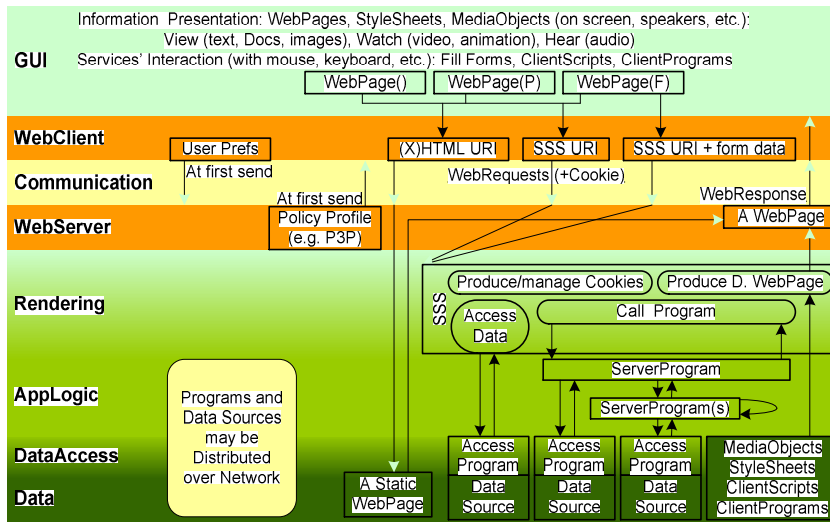
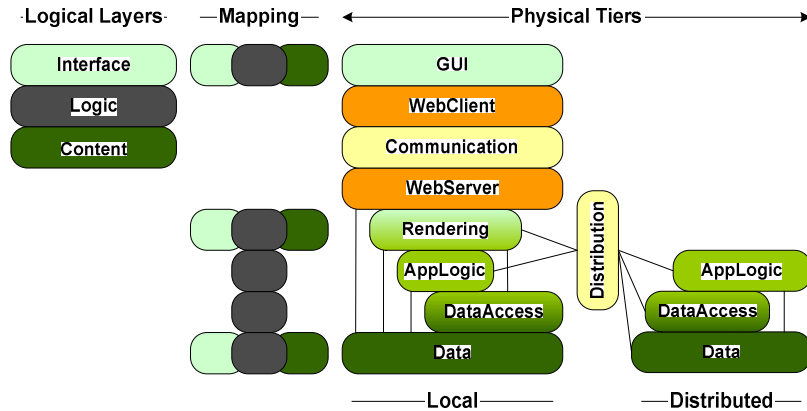


Fig. 7. Interoperation of Physical Components and Composite Components

### 3.3 WEP-Teams: WIS Project Teams

This section outlines the main stakeholders involved in the life-cycle of a WIS Project. They are classified under generic teams according to their skills on specific areas of expertise. This classification is included in WEP, because the composition of the Project team is crucial for the success of the Project and WEP goal is to give developers a roadmap how to do it. There are some research papers [5], that specify in more detail the required skills for developers working on different parts of a WIS Project.

The basic goal of the Web is to allow *content/services providers* easily diffuse content and services over the internet and make it available to the *end-users*. *Web Developers* are in-between. Two main bodies provide people for WIS Project Teams: (1)

*Customer*: the body that funds the Project and incorporates the content / service providers. It often represents the end-users during requirements gathering and testing stages. (2) *IT Organization*: the body responsible for deployment, operation and maintenance of the WIS. It often cooperates with External consultants or partners, providing additional skills not found in IT Organization personnel. It provides people with either Technical or Creative Design Background

**Table 7.** WEP-Teams: Basic and Hybrid WIS Project Teams

<b>BASIC TEAMS</b>	
<p><b>Project Manager:</b> Project Management Skills  <b>Domain Team</b>            Leader: Domain Overall Knowledge            Members' skills:            S1. Business Domain Expertise (legacy systems, guidance on achieving the business objectives)            S2. Content Domain Expertise (legacy content)            S3. Content Providers (textual content authoring, other content providing like photos, videotapes, etc.)            S4. Marketing Expertise in the Domain            S5. Legal, social, ethical issues of the domain  <b>Web Team</b>            Leader: Web Engineering            Members' skills:            S1. Requirements Engineering (analysis of Requirements, functional specifications / use-case modeling)            S2. Web-enabled device Publishing (in (X)HTML)            S3. Web Integration - Server-side Scripting (integration with data sources, programs or legacy systems)            S4. Client-side Scripting            S5. Web Testing Engineering (test platform design / implementation, metrics, acceptance criteria, etc.)            S6. Metadata design implementation and maintenance            S7. Web Site Administration</p> <ul style="list-style-type: none"> <li>• Install, configure and maintain Web Servers</li> <li>• Develop and Maintain "policies" (security/access rights) for the operation of the site</li> <li>• Collection and collation of feedback on site</li> <li>• Monitor/access log files to produce statistics</li> </ul>	<p><b>Content Team</b>            Leader: Hypermedia, DB and Information Engineering            Members' skills:            S1. Textual and Multimedia DB Engineering (Logical/Physical Designing, Data Access, Legacy Data Integration, DB Programming, DB Testing)            S2. XML Engineering            S3. Hypermedia Engineering (esp. Navigational Aspects)</p> <p><b>Logic Team</b>            Leader: S/W &amp; Network Engineering            Members' skills:            S1. S/W Engineering (Logic / Physical Designing, Programming, Legacy systems integration, Testing)            S2. Network Engineering (Distributed Computing – inter-application communication, web services)</p> <p><b>Interface Team</b>            Leader: Human – web-enabled devices interaction and Creative Design Skills (research, technologies, tools)            Members' skills:            S1. Electronic Publishing            S2. Web devices' Interface and Interaction Expertise            S3. Multimedia Engineering (Graphics Designing, Multimedia Content Digitizing, Animation Expertise)</p> <p style="text-align: right;"><i>Note: Engineering = experience and knowledge of research, technologies and tools</i></p>
<b>HYBRID TEAMS</b>	
<p><b>Project Planning Team:</b>            Project Manager, Web Leader, Domain Leader  <b>Project Management and Quality Assurance Team:</b>            Project Manager, Web Leader (Technical Manager), Domain Leader, Content, Logic &amp; Interface Leader</p>	<p><b>Analysis Team:</b> Web Leader, Web Team (S1), Domain Team, Content Leader, Logic Leader, Interface Leader  <b>Test Team:</b> Web Team (S5), Domain Team, End-users  <b>Evolution Team:</b> Web Team (Leader, S2, S6, S7), Domain Team (Leader, S3, S4, S5)</p>

First we construct the Basic Teams (expertise-oriented) with a Leader and several Members (see Table 8). The leader should have Team Management skills. A member may cover more than one skill. The number of the members and the priority of the required skills depend on the nature and the scale of the Project. We recommend forming small but high-skilled teams. A survey [10] across seven IT organizations shows that: *a development team of eight people incorporates: two from a technical background; two from a creative design background; two lead the team and the other two are a domain and a business expert.* Moreover, and based on the Basic Teams, we construct some very important Hybrid Teams. Notice, that all leaders of Basic Teams participate in analysis and quality assurance teams, because each Basic Team has a unique perspective of the WIS design and its relationship to Project objectives.

### 3.4 WEP-Process: A Lifecycle Process Model

In this section and based on the WebApps classes, logical layers and physical tiers, we specify *WEP-Process*, a Lifecycle Process Model, that supports developers during a WIS Project. It is a work discipline that describes *Who* is doing *What* and *How* in order to ensure successful WIS development and evolving. WEP-Process emphasizes on modularity, component-based development, extensibility, reusing, easy maintenance, interoperability and metadata support (semantic web). It is a data-centric and service-centric, rather than application-centric, development process. We kept WEP-Process generic, adaptable and easy for the developers to follow in order to achieve our main target: to transfer good practices and research results of web engineering to real web projects.

The cornerstones of WEP-Process are three:

1. WIS Structure design as a set of interconnected and interoperable WebApps, according to the WebApps' classes defined earlier, i.e. WHAs, WFAs and WIAs.
2. Identification and design of WebApps' Logical Components.
3. Design and Implementation of WebApps' Physical Components.

From developer perspective, it is very useful to break-down WebApps into a set of overlapping Components, each one having the following characteristics: (1) it can be developed, to some extent, independently of the others; (2) it is based on specific technologies and tools and (3) requires specific development skills. Components of the same class can be deployed by the same team, thus better exploiting the team skills, achieving best integration in each layer and tier, constructing maintainable WebApps and reusing design / implementation artifacts and experience.

WEP-Process consists of three *Phases*: A. WIS Planning Phase, B. WIS Deployment Phase and C. WIS Evolution Phase. We use the term "evolution" instead of "maintenance", which is used in Software Engineering literature, because in web engineering we have to think more about evolution and less about maintenance. If a WIS, no matter how good it is, it is just maintained and not evolved, after 2-3 years may become obsolete. Each phase consists of several *stages*. Stages include several *activities* that incorporate a *workflow* that diverse *team members* may go through to produce a particular set of *output artifacts* based on results artifacts of other activities (*input artifacts*). The workflows, at a more detailed level, describe a step-by-step procedure of how team members collaborate and how they use and produce artifacts, based on WERs. Throughout the workflow the developers have to use WERs, i.e. to take advantage of research results, to base on Web technologies and to use tools with specific guidelines. This will help developers understand the mission of each activity and make clearer to them how they can better exploit the available WERs.

WEP-Process is outlined in Table 8 (A. Phases, A1. Stages, a. Activities). *Project Management*, incorporating *quality assurance* is a special stage running in parallel with the Phases B and C. The Activities are briefly but clearly described. In the context of a real Project the Activities (i.e. team members, workflows, WERs used, input & output artifacts) should be described in detail.

**Table 8. WEP-Process**

<p><b>A. WIS Project Planning Phase (Planning Team)</b></p> <p>a. Definition of <i>Scope and general Objectives</i> of the WIS, including urgency of need, business value, etc.</p> <p>b. Definition of the <i>WIS Goals</i>. Initial requirements of the WIS are identified and documented.</p> <ul style="list-style-type: none"> <li>• Identify the <i>Users' Classes</i> and their profile (domain expertise, preferences, access devices connection speed, etc.)</li> <li>• Identify the <i>Users' Goals</i> for each user's class. We define two general types of goals: <ul style="list-style-type: none"> <li>◦ <i>Information-oriented Goals</i>: access and navigate through specific information of WHAs and</li> <li>◦ <i>Service-oriented Goals</i>: perform a complex task through interaction with WFAs and/or WIAs.</li> </ul> </li> </ul> <p>c. <i>Current Situation</i>. Identify existing data sources, legacy systems, S/W and H/W infrastructure, Human resources etc.</p> <p>d. Detail specification of the <i>WIS Deployment process model</i> (Planning and Timetables). In Phase B, we provide a general process model, which should be adapted to specific Project constraints and needs.</p> <p>e. Select a <i>Project Management Model</i> to control and track Project: Management structure, Decision-making structure, Quality Assurance mechanisms, progress monitoring, collaboration, ethical/legal/intellectual property issues, etc.</p> <p>f. Deploy the <i>Project Teams</i>. Specify what skills are required and identify roles (see WEP-Teams).</p> <p>g. <i>Budget estimation</i>.</p> <p>h. <i>Feasibility Analysis</i> – Economic justification, technical feasibility, legal considerations, Risk Analysis (to assess technical and management risks) and evaluation of alternative approaches [12], including time-related and marketing issues.</p> <p>Milestones: <b>WIS Goals, Current Situation, WIS process model, Management Model, Teams, Feasibility Analysis</b></p>
<p><b>B. WIS Deployment Phase (carried out by Project Teams based on WIS Deployment process model)</b></p> <p><b>Stage B1. Requirements Specification &amp; Analysis (Analysis Team)</b></p> <p>a. <i>Requirements Capturing</i>: (1) <i>Functional Requirements</i> based on WIS Goals, (2) <i>Non-Functional Requirements</i>, i.e. usability, reliability, security, performance, extensibility, maintainability, compatibility, etc. (3) Address the <i>Non-Technical Issues</i> such as business processes, legal, cultural and social aspects and (4) <i>Implementation Constraints</i>: required standards, implementation languages, resource limits, operation environments, required data formats, etc.</p> <p>b. <i>Use-case model</i> of Functional Requirements. It consists of users and use cases (in a modeling language like UML) and it is the integrated view of functional requirements analysis. Each use case (one per User Goal) shows step-by-step how users interact with the WIS. This activity specifies the functionality-oriented components of WIS.</p> <p>c. <i>Prototyping</i>: If the functional requirements specified by the use-case model are not very clear, you may consider including prototypes development. Prototypes could include: construction of representative WebPages, WebForms and their results pages, ClientPrograms short demo, etc.</p> <p>d. <i>WIS Structure</i>: A high-level preliminary structural design of the WIS as a set of <i>WebApps</i>. Usually, an Information-oriented Goal is served by one WHA, while Service-oriented Goals are served by WFAs and WIAs incorporated in WebPages of the WHAs. WIS WHA structure design includes the Home Page and second level WebPages (i.e. entry pages to WHAs) plus the individual WebPages that include WebForms of WFAs and ClientPrograms of WIAs.</p> <p>e. <i>Technical Specifications</i> of WIS: The construction of a technical report describing the following: <ul style="list-style-type: none"> <li>• Analysis of <i>Non-Functional Requirements and Non-Technical Issues</i>.</li> <li>• Analysis of <i>Current Situation</i> (existing data sources and legacy systems).</li> <li>• <i>Content Specification</i>. Specification of kind and amount of content need to be developed within Project.</li> <li>• <i>Logic Specification</i>. Specification of logic components (programs) need to be developed within Project (over Content and legacy systems) in order to satisfy the Service-oriented User Goals.</li> <li>• <i>Interface Specification</i>. Specification of targeted WebClients on specific web-enabled devices plus their presentation and interaction capabilities and constraints.</li> <li>• Analysis of <i>Implementation Constraints</i> and analysis of <i>Industry and Web standards</i> specific for the application domain. Core technologies selection, required development infrastructure and tools, etc.</li> </ul> </p> <p>f. <i>Acceptance Criteria Specification</i>. These will be used later during testing stage in order to validate the implemented WIS against Users' Goals and Non-Functional Requirements.</p> <p>Milestones: <b>Use Case Model, Prototypes, WIS Structure, WIS Technical Specifications, Acceptance Criteria</b></p>
<p><b>Stage B2. Logical Design</b></p> <p><i>Content/Logic/Interface Teams based on WIS Structure, WIS Technical Specifications and Use-case Model</i></p> <p>a. <i>Content Components Design</i> for each WebApp. <ul style="list-style-type: none"> <li>• <i>Data Components</i> are: UnstructData type specification (e.g. images, video, docs, etc.), SemiStructData designs (e.g. XML Schemata), StructData designs (e.g. ER, OO). Choose the better design methodology based on research results, good practices and data types. Some data components may be common across several WebApps (like copyright statement, etc.). These components belong to the WIS WHA.</li> <li>• <i>WHA Navigation Components</i> are: WebPages of WHAs (specify the data of each WHA WebPage based on the Data Components, specify classes of WebPages according to the semantics of their data, high-level links across WHA WebPages – low-level linking will be made during the authoring activity of implementation stage later), Navigational Structures of WHAs (design of WHA local access structures like menus, index pages, guided tours, etc.).</li> <li>• <i>WIS Navigation Components</i> are: Global WebPages (e.g. home page), global access structures (e.g. site map), cross-WHA high-level links, etc.</li> <li>• <i>Metadata Components</i> are: Metadata schemata per class of WebPages of WHAs, metadata schemata per individual WIS WebPages, metadata schemata per class of other WIS resources (like images).</li> </ul> </p> <p>b. <i>Logic Components Design</i> for each WebApp (see <i>Logic Layer</i> description).</p> <p>c. <i>Interface Components Design</i> for each WebApp (see <i>Interface Layer</i> description).</p> <p>Milestones: <b>Content Components, Logic Components, Interface Components</b></p>

<p><b>Stage B3. Physical Design</b> (based on <i>Logical Design</i> Components)  Team (Activities) : Web (a,d,f,g,h), Content (b,c,d,f), Logic (a,c,d,e) and Interface (a,b,f)</p> <p>This is the main development stage where the physical architecture of the WIS is designed and it will drive the implementation stage' activities. The following activities are not considered to be carried out sequentially but in parallel, as a decision in one tier may influence others. Thus, all basic development teams i.e. Web, Content, Logic and Interface Teams should work in close cooperation. The objective of these activities is to contribute to the overall <i>WIS Physical Architecture Diagram</i>, which is the Milestone of this stage. In many cases, developers may consider purchasing off-the-shelf development frameworks or re-using them from prior Web projects. Famous ones are IBM's WebSphere, Sun ONE (J2EE), BEA Logic and Microsoft's .NET framework. Many non-functional requirements of the WIS are usually covered by the features of these frameworks, and the developers don't have to implement them. However, non-functional requirements should be considered within each activity separately and at the WIS level.</p> <p>a. <i>WebClient Physical Components</i>: specify the set of WebClients (browsers and devices) that can access the WebApps and all necessary plug-ins for MediaObjects and ClientPrograms. Specify the delivery markup language (type of (X)HTML) of WebApps. Specify the StyleSheet, ClientScript and ClientPrograms language to use.</p> <p>b. <i>Data Physical Components</i>: for each Content Component and some Interface Components (e.g. MediaObjects) specify the data formats and their storage systems (data sources or repositories) able to develop, store it and manage it (e.g. content management system, RDBMS, XML repository, directory server, filesystem, etc.). Content Components of the same design methodology (e.g. ER or XML), is better to be physically designed under a common storage system, even though they belong to different WebApps. This will reduce the cost and complexity of the WIS development. During this activity we specify the structure and interface of the Static WebPages ((X)HTML files).</p> <p>c. <i>DataAccess Physical Components</i>: for each Data Physical Component specify the data access technologies (like SQL, XQuery) and the data AccessPrograms (e.g. ODBC, XML Parser, specific programs, etc.) which will be used by other physical components in order to access data sources at run-time.</p> <p>d. <i>Distributed Physical Components</i>: specify which server-side Logic Components will be physically implemented distributed with regard to WebServer. For these, specify the technologies (e.g. CORBA, Web Services, etc.), programming languages, and development / run-time environments. Moreover, specify which Data Physical Components will be distributed and what technologies will be used.</p> <p>e. <i>AppLogic Physical Components</i>: for each server-side Logic Component specify the programming language, and their development and run-time environments (e.g. an application server like Sun ONE (J2EE) or .NET Framework, etc.). It is recommended to base your design on few languages and a common environment. Moreover, extract common functionality of Logic Components and design them as common AppLogic Physical Components (like EJBs) that are re-used across different WebApps.</p> <p>f. <i>Rendering Physical Components</i>: specify the SSS technologies for the implementation of rendering Logic Components, plus its development and run-time environment. Specify the structure and interface of the Dynamic WebPages.</p> <p>g. <i>WebServer Physical Components</i>: specify the WebServer features, its modules, its run-time environment and its administration and management tools (administration environment, log analysis tools, etc.)</p> <p>h. <i>Communication Physical Components</i>: specify the communication protocols and the supported data formats of the WebApps plus the network infrastructure to support the communication with specified WebClients.</p> <p>Milestone: <b>WIS Physical Architecture Diagram</b></p>
<p><b>Stage B4. Implementation &amp; Unit Testing</b> (based on <i>Physical Design</i> Components)  Implementation and Unit Testing activities include the same activities' structure and Teams association as in <i>Physical Design</i>, except that content authoring is carried out by <i>Domain Team</i>. Integration activity is carried out by the <i>Web Team</i>.</p> <p>a. <i>Implementation of WIS Physical Architecture Diagram</i> Components as specified in <i>Physical Design</i> stage. This is not a sequential procedure, as some components may need to be developed before others. Such dependencies among components should be identified at the beginning of this activity. This activity includes run-time environments installing and configuring, programming, multimedia production and content authoring.</p> <p>b. <i>Unit Testing</i> of each component against the functional and non-functional (e.g. stress/capacity testing) requirements. This activity includes the design of test cases, the actual testing and the reporting of the results.</p> <p>c. <i>Integration</i> of all implemented components into the WIS and its WebApps.</p> <p>Milestone: <b>WIS up and running</b></p>
<p><b>Stage B5. Acceptance Testing</b> (<i>Test Team</i> based on <i>Acceptance Criteria</i>)</p> <p>a. <i>Test Cases Design</i> for WIS and per WebApps, based on methodologies and tools (e.g. stressing tools, etc.)</p> <p>b. <i>Test each WebApp</i> separately against Acceptance Criteria.</p> <p>c. <i>Test WIS</i> as a whole against Acceptance Criteria.</p> <p>d. <i>Report the results</i> of the testing activities. If the results are not what we expect we move back to either the logical design, physical design or implementation stage.</p> <p>Milestone: <b>Testing Reports</b></p>
<p><b>C. WIS Evolution Phase</b> (<i>Evolution Team</i>)</p> <p>a. Content update (data, metadata, links).</p> <p>b. WIS Infrastructure maintenance, e.g. correction of software bugs, new hardware and software installation, etc.</p> <p>c. <i>Enhancement</i>: lightly extends system functionality, minor adaptations to new technologies, etc.</p> <p>d. Logs analysis and statistics production.</p> <p>e. Collection and collation of end-users' feedback. Report emerging requirements for the WIS.</p> <p>f. Technologies, tools and research results monitoring. Report emerging technical requirements for the WIS.</p> <p>The results of the two last activities may indicate the need of a new Project, if the emerging requirements are by far different from the ones current WIS addresses.</p>

## 4 Conclusions & Future Work

Based on our extended experience on building large-scale WISs and on research on the field of current Web development we have outlined the main difficulties Web developers have on exploiting the Web Engineering Resources, i.e. Technologies, Research Results and Tools. WERs are not used appropriately or at all during current WIS Projects. Web Engineering is a complex information space that needs to be engineered, in order to provide the WERs to developers in a structured and meaningful way. To address these needs we introduce the *WEP*, in order to structure and present the WERs in a unified framework that provides a *Reference Model* and a resources *Portal*. The aim of WEP is to constitute for academics an important tool for the education of Web Engineering, and for Web Engineers a reference model for the deployment of WISs and a guide in the discovery and apprehension of WERs.

*WEP Reference Model* includes: (1) *WEP-Terms*: WEP Basic Terminology and Definitions, (2) *WEP-Arch*: technical-oriented classification of WebApps, WebApps' *Logical Layers* and the WebApps' *Physical Tiers*, (3) *WEP-Teams*: classification of skills and (4) *WEP-Process*: A *WIS lifecycle process model* with three phases: Planning, Deployment and Evolution.

*WER-Portal*, based on WEP Reference Model, will provide information and several taxonomies of *WERs*, and it will act as a *Guide* through which Web Engineers will be able to easily and meaningfully locate (1) Technologies, (2) Research Results in the diverse areas and topics, and (3) Tools ranging from complete solutions to small-scale programs. Moreover, Web Engineers will be able to understand each *WER's* role during WIS Development and WIS Operation/Maintenance. *WER-Portal* is currently under development. The full-expanded description of WEP Reference Model and some parts of the Portal's taxonomies and resources will be described in detail as a chapter of a book [1] that will be published soon.

The next step is to implement *WER-Portal* and evaluate it in real-life WIS Projects. Furthermore, we have to well-maintain it in order to always be up-to-date and incorporate emerging technologies, research results and tools. Finally, in order to achieve its objective, WEP must be enhanced by the contribution of other researchers in the field and work together to make it a stable "standard" for Web Engineering community.

## References

1. Christodoulou, S., and Papatheodorou, T. (2004). "Web Engineering Resources Portal (WEP): A Reference Model and Guide". *Book chapter in "Web Engineering: Principles and Techniques"*, to be published by Idea Group Inc. toward December 2004.
2. Christodoulou, S., Zafiris, P., & Papatheodorou, T. (2001). Web Engineering: The Developers' View and a Practitioner's. In *Web Engineering: Managing Diversity and Complexity in Web Application Development* (LNCS Vol.2016, Springer-Verlag).
3. Cutter Consortium. (2000). *Research Briefs*.
4. Deshpande, Y., Murugesan, S., Ginige, A., Hansen, S., Schwbe, D., Gaedke, M., & White, B. (2002). Web Engineering. *Rinton Press Journal of Web Engineering*, 1 (1), 3-17.

5. Hansen, S., Deshpande, Y., & Murugesan, S. (2001). A skill hierarchy for web-based systems development. In *Web Engineering: Managing Diversity and Complexity in Web Application Development*, LNCS Vol.2016, Springer-Verlag.
6. Holck, J. (2003). 4 Perspectives on Web Information Systems. *36th HICSS (Hawaii International Conference on System Sciences)*.
7. Isakowitz, T., Bieber, M., & Vitali, F. (1998). Web Information Systems. *Communications of the ACM*, 41 (7), 78-80.
8. Jacobs I., eds., "Architecture of the World Wide Web". *W3C Working Draft* 1 October 2003 (<http://www.w3.org/TR/2003/WD-webarch-20031001/>).
9. Lavoie B. and Nielsen H. F., eds., "Web Characterization Terminology & Definitions Sheet". *W3C Working Draft* 24-May-1999 (<http://www.w3.org/1999/05/WCA-terms/01>).
10. McDonald A., & Welland R. (2003). Agile Web Engineering (AWE) Process: Multidisciplinary Stakeholders and Team Communication. *Third International Conference on Web Engineering*, ICWE 2003, LNCS 2722, pp. 515-518. ISBN: 3-540-40522-4.
11. Nambisan, S., & Wang, Y.-M. (1999). Roadblocks to Web Technology Adoption? *Communications of the ACM*, 42 (1), 98-101.
12. Pressman, R.S. (1997). *Software Engineering: A Practitioner's Approach*. New York: McGraw-Hill.