# Preface

An enormous amount of material dealing with the Web and Web Services is available in books, online, and in various publications. Anybody interested in learning can find the material but suffers from the overload of information. Which information is important, up-to-date, and correct? How can I find information that constitutes the background for the problem facing me? Is the information relevant, up to date, and reliable? Can I simply apply what the published references recommended?

Inspired by similar questions, we decided to put together a systematic review of current Web technology and trends to meet two goals:

1. Provide information to people who need general, rather than in depth, technical knowledge such as information technology (IT) development mangers, software designers, architects, IT students, and project and program managers. Our ideal reader is technically inclined, with broad interests and management responsibilities.
2. Describe the logical development of business applications technology, from client-server to Web, Web Services, Portal, and computer Grids.

The book is practically oriented; it gives a large amount of industry focused advice and down-to-earth observations. The reader will probably need to seek more detailed information in specialized books, but he or she will be able to maneuver in the sea of available information.

It is not necessary to read the book sequentially, but there are advantages in doing so as the topics build on previously discussed topics.

Most importantly, acronyms and glossary terms are explained at the end of the text. Terms that we consider important are listed in the Index.

The reader should be familiar with the basics of current information technology, the fundamentals of Web architecture, and introductory Java. It is possible to understand the ideas without understanding the Java examples that appear in the text. We rely heavily on the XML concepts, and we envisage that the reader will consult the Web for additional information when reading this book.

We found many definitions and principles on the Web, and, where possible, we quote the source for the reader. Published Request for Information (RFC) material is used extensively throughout the book.

Some information will become obsolete with time, as many suggestions and design principles refer to the current industrial strength implementations. Many examples and practical observations refer to IBM WebSphere as the authors have experience with the product. We encourage readers to critically evaluate some of our more controversial advice.

## Promises and Expectations of Client-Server Computing

The promise of reduced complexity of software and of software development in the third and fourth generation of computers has simply not eventuated. The catch phrase of the client-server paradigm was the promise to reduce the complexity of the software by dividing electronic processing into a smaller portion performed on a client computer and a larger portion performed on a server. This way, the client (the computer, not the person) specializes in front end processing, leaving the logic to the backend server processing. This separation of duties was supposed to achieve what we call today the separation of concerns. The client interacted with the user and provided data validation, while the server processed the data and returned the result, often by connecting to a database. The idea of the Graphical User Interface (GUI) was born, and it was firmly allocated to the client. Nearly immediately, the user interface became much more sophisticated, filling the vacated space with GUI API's, event processing, and user interface standards. Thus, GUI processing grew into a specialized and complex area, demanding high skills and development costs.

Programming languages such as Visual Basic were able to produce functionally rich user interfaces with graphics, which the user more easily interpreted.

Previously unseen visual objects, such as tabs, pop up windows, buttons with graphical depiction or symbols of their function, and colorful navigational prompts filled the user's view.

On the server side, additional functionality also filled the void: generalized database connectors; lightweight processes called threads (which introduced additional complexity for synchronization); support for a large variety of devices and protocols; and ever-increasingly complex logic.

The development process for client-server applications did not become simpler either. The two subsystems were often developed in two or more different programming environments. For example, it was very common to develop client software in Visual Basic or Delphi, while the server software was developed in C or C++ (or both). The previously cohesive development team was broken into client and server specialists. The functional testing introduced a new, expensive, and difficult discipline: integration testing, where the client and server were seen in cooperation for the first time. Finding errors, bugs, and functional irregularities became complex and very expensive, as several specialists had to cooperate to resolve the problems.

The initial promise of reduced complexity was broken. The development managers were dealing with increased costs and were looking for a solution that would address the increased number of development personnel, ballooning testing and deployment costs. The developers were faced with a specialization dilemma — should I become a client or a server specialist? It should be remembered that the client specialist was also the user interface specialist, being responsible for the design of the user interface and its behavior. At the same time, the end users became accustomed to better usability and demanded ever-improving quality of the user interface, intuitive behavior, and sophisticated error management, where it should not matter whether the validation errors originated from the server or client processing.

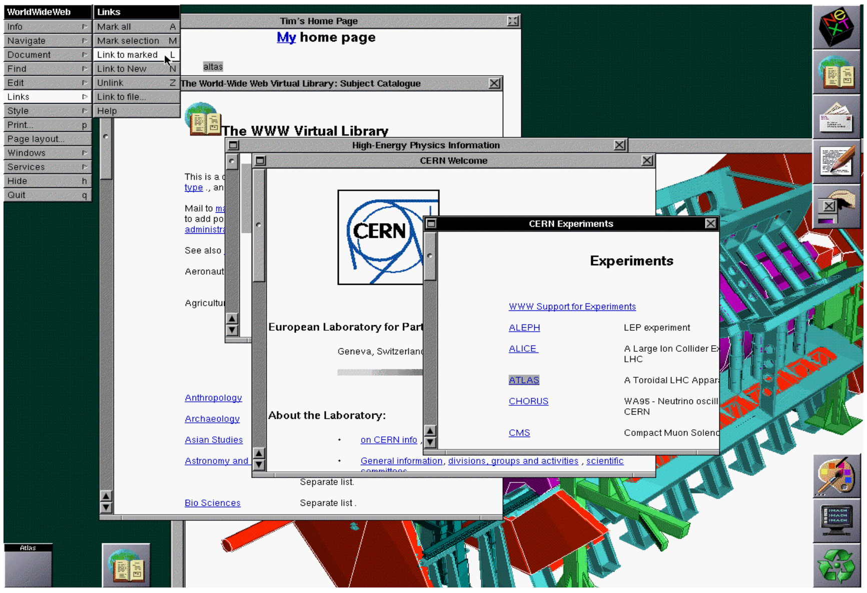
## The World of the Web

While client-server computing was evolving, the world of the Web was quietly assessing its strength. This occurred in 1989-1990. An interested reader can see one of the early Web sites on http://www.w3.org/History.html or http://www.w3.org/People/Berners-Lee/WorldWideWeb.html, or read about it in Weaving the Web by Tim Berners-Lee with Mark Fischetti (Berners-Lee & Fischetti, 1999).

Let us look at one of the first browsers, shown in Figure 1. It bears an unmistakable influence of what we call today a "thick" client — rich graphics, creative use of space, various visual clues, and many windows. The hyperlinks are the new elements, previously not used. Ted Nelson first coined the term hyperlink in 1965 in project Xanadu (Nelson, 1965)! Looking at the visual components, which became very common later, we notice that the Navigate menu (third from the top on the left side) has "back", "next", and "previous" options. The Style menu option (halfway down the first menu list) has a style sheet option where one could load different style sheets. The "X" box to close the window was copied from the NeXT toolbox and was later universally adopted by Windows. The "edit" option was also new. The browser was written in a dialect of C, called Objective-C. Note that the "Help" menu is also present as the last option on the second menu from the left. Many of these components are re-used in the current Web and Portal interfaces.

Very shortly afterward, the style of the interface became more restrained, less graphical, with limited expressiveness. This was caused by the amount of information that had to be transferred through the relatively slow communication lines. It is important to remember that by then, client-server computing had undergone a substantial change in that the connections between the Web client and the Web server were much slower than point-to-point, dedicated client-server connections.

*Figure 1. Berners-Lee's screen shot of the browser (1994) (http://www.w3.org/History/1994/WWW/Journals/CACM/screensnap2_24c.gif)*



This is a c
type ., an
Mail to ma
to add po
administra
See also
Aeronaut
Agricultu
Anthropology
Archaeology
Asian Studies
Astronomy and
Separate list.
Bio Sciences Separate list .

European Laboratory for Part
Geneva, Switzerland

About the Laboratory:

- on CERN info ,
- General information, divisions, groups and activities , scientific committees

Experiments

WWW Support for Experiments

| | |
|---|---|
| ALEPH | LEP experiment |
| ALICE | A Large Ion Collider Ex LHC |
| ATLAS | A Toroidal LHC Appar |
| CHORUS | WA95 - Neutrino oscill CERN |
| CMS | Compact Muon Solen |

The new Web technology brought some simplification, but only for a brief period. The browser was thin — essentially displaying a HTML document — and the server was rather primitive and very slow, the logic often implemented in a CGI[1] (Common Gateway Interface) script. Since the plain HTML/CGI combination provided services that were regarded as a step backwards (compared to more sophisticated thick clients), a method was found which brought the browser-based client closer to a thick client. Thus, client-side processing was introduced, constituting a full circle in the development. Java applets, Java scripts, and various other means were used to achieve functionally rich, dynamic user interfaces.

From the point of view of a development manager, the situation was not much better than before. A variety of skills were necessary in a development team, including a GUI designer, an HTML developer, a CGI script developer, and later a Java developer. However, a substantial gain was achieved — the whole system could be developed on a single machine, with a local Web server.

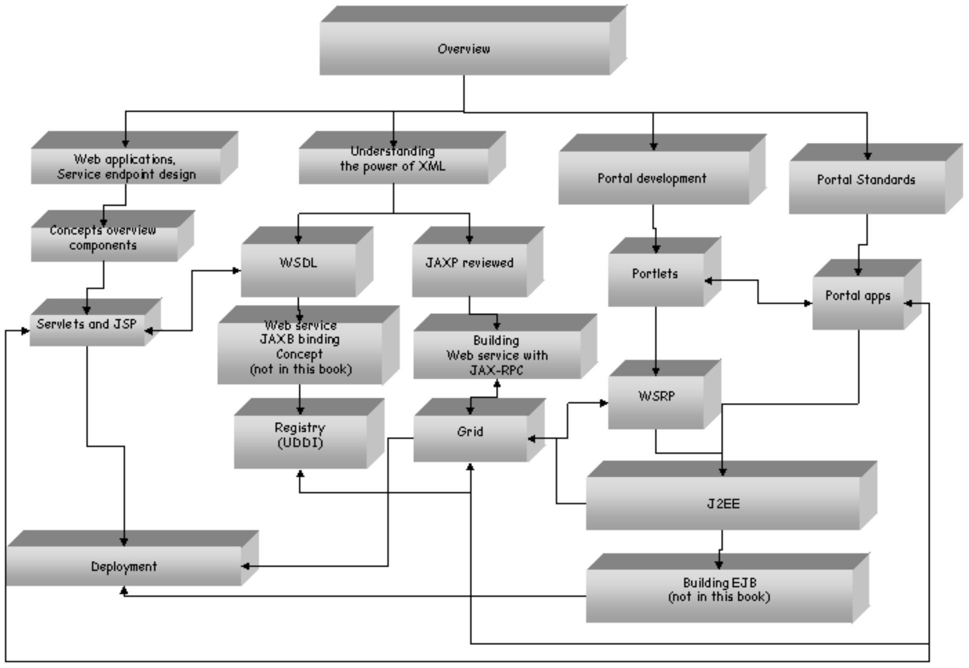
With the introduction of Java (and later .Net technology), the feeling of victory was also introduced: a single technology was used on both platforms, with a Java applet running in a browser and a Java application running on the server. It looked as if the technology was finally making life simpler, although the quality of the interface was still not as good as in the thick client-server processing. The net effect of the latest developments was that the enterprise's client-server systems were being re-implemented in the "Intranet" — applications designed for the Internet to be only accessible within the company network. Legacy systems continued to be used and maintained, and people wanted to use them in the "net" — here meaning Intranet — without the necessity to re-write them. A lightweight Web application would do precisely that — provide access to a backend system. This represented a critical point in thinking — suddenly people wanted to access a multiplicity of applications in one easy to use window. The idea of a Portal was born.

The necessity to provide integrated Web Services stems from the fact that modern businesses often evolve through mergers and acquisitions. These businesses have applications that are either redundant or reusable. However, integration of these applications is not trivial, mainly from the point of view of the number of connections. Consider n applications that are to be integrated; it will be necessary to build n*(n-1) interfaces. With every new application (n+1), one must test and maintain 2n (where n is the number of original applications) new interfaces. Obviously, the architect tries to find a solution where a new application requires only one new interface to the system. However, there is more to it; the integration must be done not only on the level of interconnectivity, but also on the user interface level. For example, two business applications that are related should be accessible from one screen, if possible. If there is some legal information or other textual information that is useful, such as business procedures, it should be also readily available. The industry answer to the previously-mentioned problem is the service-oriented architecture, which is described in the section called Service-Oriented Architecture. We will explain how the necessity of simplifying the development process contributed to the idea of Portal, Web Services, and computer Grids.

## Where to Start?

In Figure 2, we list the major subjects discussed in the book and their relationships. The reader can see the necessary prerequisites for the desired subject. The diagram shows that modern Web-oriented computing is reaching a stage where many components cooperate in the final product. It is important to realize this idea when designing, building, and managing the development of these systems. Understanding historical trends helps to keep current and future trends in perspective. We encourage the reader to look back before looking to today and onto the future.

*Figure 2. Roadmap of this book*


Overview
Web applications, Service endpoint design
Understanding the power of XML
Portal development
Portal Standards
Concepts overview components
WSDL
JAXP reviewed
Portlets
Portal apps
Servlets and JSP
Web service JAXB binding Concept (not in this book)
Building Web service with JAX-RPC
WSRP
Registry (UDDI)
Grid
J2EE
Deployment
Building EJB (not in this book)

This book arose from university courses taught by the authors at Monash University in Melbourne, Australia and from the practical managerial experience of the portal development manager. Most courses and textbooks in this field are targeted at either the low programming level of APIs for portals, J2EE components and Web Services, or high-level aspects of services and portal deployment in the enterprise information systems. We also realized that the majority of the textbooks focus on material around Web Services and ignore the fact that portals and Grid applications are in some ways the inevitable extension of Web Services.

This book uses Java, J2EE platform, Globus Toolkit, and IBM's WebSphere for Multiplatforms 5 to demonstrate the programming and deployment aspects of development of Web Services, portals, and Grid applications.

## Organization of the Book

At this point, the reader should be ready for the main body of the book, which is divided into four parts:

1. **In Section I:** The Toolset (Chapters I-III), we explain technicalities related to XML and Web Services Description Language (WSDL). The purpose of Chapter II is to give the reader basic knowledge necessary to read the rest of the chapters.

   Chapter III deals with Web oriented protocols, UDDI (Universal Description, Discovery and Integration), and SOAP (Simple Object Access Protocol). We explain relationship between UDDI version 2 and WSDL. In addition, we provide a simplified example of accessing UDDI Registry using JAXR (Java APIs for Registries). SOAP is described in terms of messaging — a widely used technology in Web Services.
2. **In Section II:** Web Services as Shared Resources (Chapters IV-VII), we describe Web Services and look at some implementation issues. Chapters V and VI discuss servlets and JSP (Java Server Pages). At this point, the reader will have the necessary knowledge for understanding the Web service design issues discussed in Chapter VII.
3. **In Section III:** Putting Portals on the Web (Chapters VIII-XVII), we utilize the knowledge of Web Services and develop the concepts of portals. We briefly touch on portal solutions for business processes, and then continue with portal development framework and the close relationship between portlets and servlets. Large portions of Chapters IX-

XIII are devoted to understanding portlets and portal concepts, their lifecycle, and messaging. Chapter XVI is dedicated to discussion about relevant portlet standards and issues related to accessing presentation-oriented Web Services. Chapter XVII is based on practicing manager's experience with installation and delivering portal solutions in large organizations. These issues are particularly relevant to the B2E and B2B solutions.

4. **In Section IV:** Grids as Virtual Organizations (Chapter XVIII), we introduce the concept of computer Grids, building on the previous two parts. As recently discussed in various Web communities, Web Services are seen as an enabling technology for Adaptive Enterprises where business and IT are synchronized to accommodate fast and efficient changes in real time. With the new incoming WS-Resource framework (authored by The Globus Alliance, HP and IBM), we feel that thorough understanding of Grid concepts would be beneficial to our readers. This seemingly long section will provide the reader with sufficient knowledge to allow him or her to appreciate new trends in creating Grid-based communities and applications running on the Grids.

## Endnote

1 The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. See also the following Web site for more information: http://hoohoo.ncsa.uiuc.edu/cgi/intro.html