Tools and Approaches for Developing Data-Intensive Web Applications: a Survey

Piero Fraternali

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci 32, I-20133 Milano, Italy
fraternali@elet.polimi.it

The exponential growth and the capillar diffusion of the Web are nurturing a novel generation of applications, characterised by a direct business-to-customers relationship. The development of such applications is a hybrid between traditional IS development and Hypermedia authoring and challenges the existing tools and approaches for software production. This paper investigates the current situation of Web development tools, both in the commercial and research field, by identifying and characterising different categories of solutions, evaluating their adequacy to the requirements of Web application development, enlightening open problems, and exposing possible future trends.

Categories and Subject Descriptors: H.5.4 [Information Interfaces and presentation]: Hypertext/Hypermedia; D.2.2 [Software Engineering]: Design Tools and techniques
General Terms: Application, Development
Additional Key Words and Phrases: WWW, HTML, Intranet

1. INTRODUCTION

Applications for the Internet in such domains as Electronic Commerce, Digital Libraries, and Distance Learning are characterized by an unprecedented mix of features that makes them radically different from previous applications of Information Technology [Myers et al. 1996]:

—Universal access by individuals with limited or no skills in the use of computer applications introduces the need of new man-machine interfaces capable of capturing customer’s attention and of facilitating access to information.

—Global availability of heterogeneous information sources requires the integrated management of structured and unstructured content, possibly stored in different systems (databases, file systems, multimedia storage devices) and distributed over multiple sites.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works, requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM Inc., 1515 Broadway, New York, NY 10036 USA, fax +1 (212) 869-0481, or permissions@acm.org.
In the recent years the World Wide Web has been elected as the ideal platform for developing Internet applications, thanks to its powerful communication paradigm based on multimediability and browsing, and to its open architectural standards which facilitate the integration of different types of content and systems.

Modern Web applications are conveniently described as a hybrid between a Hypermedia [Nielsen 1995] and an Information System. Like in Hypermedia, i.e., those applications commonly found in CD ROMs, kiosks, information points, and so on, information is accessed in an exploratory way rather than through “canned” interfaces, and the way in which it is navigated and presented is of prominent importance. Like in Information Systems, the size and volatility of data and the distribution of applications requires consolidated architectural solutions based on such technologies as database management systems and client-server computing.

Due to this hybrid nature, the development of a Web application is confronted to a number of applicative requirements, like:

- the need of handling both structured data (e.g., database records) and non-structured data (e.g., multimedia items);
- the support of exploratory access through navigational interfaces;
- a high level of graphical quality;
- the customization and possibly dynamic adaptation of content structure, navigation primitives, and presentation styles;
- the support of proactive behavior, e.g., for recommendation and filtering.

These requirements add up, and typically compete, with the technical and managerial issues of every software- and data-intensive application, which obviously remain valid also for large Web applications:

- security, scalability, and availability;
- interoperability with legacy systems and data;
- ease of evolution and maintainability.

As it happened in the past with other emerging technologies, like databases and object oriented programming languages, methodologies and software tools may greatly help mastering the complexity of innovative applications, by fostering a correct understanding and use of a new development paradigm, providing productivity advantages, and thus reducing the risk inherent in application development and migration.

The goal of this survey is to put forward the software engineering, architectural, and applicative issues of Web development (Section 2), classify and compare current development tools in light of such aspects (Section 4 to 10), formulate evaluations and perspectives by examining the relationship between state-of-the-practice solutions and relevant research areas (section 11, 12, 13 and 14), and summarize current choices for the developer based on the characteristics of the application (section 15).

2. PERSPECTIVES OF WEB DEVELOPMENT

The development of a Web application is a multi-faceted activity, involving not only technical questions, but also organizational, managerial, and even social and artistic issues.
2.1 Process

Although there is still no consensus on a general model of the lifecycle of a Web application\textsuperscript{1}, a scheme of the typical activities involved in the construction of a Web application can be obtained by interpolating the lifecycle models of traditional Information Systems and the proposals for structured Hypermedia design [Garzotto et al. 1995; Isakowitz et al. 1995; J.Nanard and M.Nanard 1995; Schwabe and Rossi 1995]. Figure 1 illustrates the lifecycle model used as a reference in this paper.

— **Requirements Analysis**: the mission of the application is established, by identifying the prospective users and defining the nature of the information base. In addition to the customary requirement collection and feasibility assessment tasks, Web applications designed for universal access require special care in the identification of human-computer interaction requirements, in order to establish the interaction mode most suitable for each expected category of users and for each type of output device that the users are expected to use to connect to the application (from hand-held personal communicators to high definition screens).

— **Conceptualization**: the application is represented through a set of abstract models which convey the main components of the envisioned solution. In the Web context, conceptualization has a different flavor with respect to the same activity in Information System design, because the focus is on capturing objects and relationships as they will appear to users, rather than as they will be represented within the software system. Although the notations may be the same (e.g., the Entity Relationship Model [Chen 1976]), usually the schemas resulting from the conceptualization of a Web application and a database application differ.\textsuperscript{2}

— **Prototyping and Validation**: simplified versions of the applications are deployed to the users for early feedback. The importance of prototyping is particularly

\textsuperscript{1}Among the current proposals there are: [Takahashi and Liang 1997; Atzeni et al. 1998; Fraternali and Paolini 1998]

\textsuperscript{2}A macroscopic difference is in the interpretation of relationships, which in database modeling represent semantic associations to be persistently recorded, whereas in Web modeling generally imply also a navigation possibility.
emphasized in the Web context, as it is in Hypermedia, because the intrinsic complexity of the interfaces requires a timely evaluation of the joint effectiveness of structure, navigation and presentation [Nielsen 1996; Bachiochi et al. 1997]. Typically, a prototype is built prior to design and on a simplified architecture, e.g., as a set of manually implemented pages containing samples of the application content, which emulate the desired appearance and behavior.

—**Design:** conceptual schemas are transformed into a lower-level representation, closer to the needs of implementation, but still independent of the actual content of the information base. Typically, the structural view of the application is mapped to the schema of a storage repository, the navigational view into a set of access primitives over the content repository, and the presentational view into a set of content-independent visual specifications (styles). The latter activity, called *visual design*, is of prominent importance in Web application development and is emerging as an autonomous discipline [Sano 1996; Horton et al. 1996].

—**Implementation:** the repository is filled out with new content prepared by domain experts and/or with existing data stored in legacy systems; the actual interfaces, *pages* in the Web terminology, are constructed, by embedding repository content and navigational commands into the appropriate presentation style. The mapping of design to implementation requires the choice of the network language in which the application is delivered (e.g., HTML, Java, ActiveX or a mix thereof), and the decision of the time of “binding” between content and application pages, which can be offline or online.

—**Evolution and Maintenance:** after delivery, changes in the requirements or bug fixes may require the revision of structure, navigation, presentation, or content. Changes are applied as high as possible in the development chain and propagated down to implementation.

The described process model caters for a variety of actual processes, whose applicability depends on the specific development context, including the available tool support, financial and time constraints, application complexity and change frequency. Generally speaking, with applications of limited size and fairly stable requirements, requirement analysis is directly followed by implementation, possibly after a number of iterations through prototyping. Conceptualization and design assume more relevance as the complexity and volatility of the application increase.

### 2.2 Models, Languages, and Notations

A Web application is characterized by three major design dimensions:

—**Structure:** describes the organization of the information managed by the application, in terms of the pieces of content that constitute its *information base* and of their semantic relationships.

—**Navigation:** concerns the facilities for accessing information and for moving across the application content.

—**Presentation:** affects the way in which application content and navigation commands are presented to the user.

To support the representation of application features during the development lifecycle, notations with different levels of formality and abstraction can be used.
At the conceptual level, applications are described by means of high level primitives which specify the structural, navigational and presentational views in a way that abstracts from any architectural issue.

Structural modelling primitives describe the types of objects that constitute the information base and their semantic relationships, without commitment to any specific mechanism for storing, retrieving and maintaining the actual instances of such object types. Examples of notations that can be used to express structural features include the best-known conceptual data models, like the Entity-Relationship model [Chen 1976] and various Object Models [Rumbaugh et al. 1991].

Navigation modelling primitives express the access paths to objects of the information base and the available inter- and intra-object navigation facilities, again without committing to any specific technique for implementing access and navigation. The vast range of notations and techniques proposed for the more general problem of human-computer interaction specification can be employed for navigation modelling: data models extended with built-in navigation semantics [Garzotto et al. 1993; Issakovitz et al. 1995; Fraternali and Paolini 1998] or explicitly annotated with behavioral specifications [Kesseler 1995; Schwabe and Rossi 1995], first-order logic [Garg 1988], Petri nets [Stotts and Furuta 1989], finite state machines [Zheng and Pong 1992], and formal grammars [Jacob 1982] are among the viable options.

Presentation modelling aims at representing the visual elements of the application interfaces in a way that abstracts from the particular language and device used in the delivery. Many different techniques can be used, with a varying degree of formality and rigour, ranging from simple storyboard specification [Madsen and Aiken 1993] to the use of software tools [Myers 1995] and formal methods. The independent specification of presentation, separate from structure and navigation, is particularly relevant in the Web context, because the final rendering of the interface depends on the browser and display device and thus there may be the necessity of mapping the same abstract presentation scheme to different designs and implementations.

At the design level, structured or semi-structured data models are used to represent the features of an application in a way amenable to manipulation, query, and verification. Typically, design representations are maintained as relational or object-oriented schemata, to exploit the capabilities of database management systems, or as semi-structured objects, to cope with information having partial or missing schemata [Florescu et al. 1998].

At the lowest degree of abstraction, applications are represented by means of implementation-level languages, notably network languages directly interpretable by the user's browsers. At this stage, content, navigation and presentation are directly embedded in the physical marked-up texts or programs that make up the application.

2.3 Reuse

As in any other software process, reuse is an important aspect, related to the facility of building a novel application from existing artifacts [Biggerstaff and Perlis 1989]. Reuse may happen at all levels of the development process: conceptual schemas, design schemas, content and physical application pages can be reused across applications.
The most common form of reuse on the Web, as in Hypermedia, is content reuse, possibly facilitated by the presence of a structured repository, which enables the construction of different applications on top of the same information base. Reuse also occurs at the implementation level, where component-based libraries of self-contained elements (e.g., JavaBeans or ActiveX components) can be plugged into application pages to provide predefined functionalities (e.g., an electronic payment facility). Generation-based reuse focuses instead on reusing transformation techniques from design frameworks or partially instantiated implementations to full implementations (e.g., by generating application pages from page skeletons and database content).

2.4 Architecture

Architecture is the subject of design and implementation and reflects the spatial arrangement of application data and the spatio-temporal distribution of computation.

The minimal spatial configuration of a Web application is the so-called two-tier architecture, shown in figure 2, which closely resembles the traditional client-server paradigm. Differently from client-server, in the two-tiers solution clients (i.e., browsers) are thin, i.e., they are lightweight applications responsible only of rendering the presentation. Application logic and data reside on the server side.

A more advanced configuration, called three- or multi-tier architecture (figure 3), separates the application logic from data, introducing an additional distinction of responsibilities at the back-end side. The presence of one or more distinct application tiers enables the implementation of advanced architectures which integrate the traditional HTTP protocol and client-server application distribution protocols, for better performance, scalability, reliability and security.
An orthogonal architectural issue concerns the time of binding between the content of the information base and the application pages delivered to the client, which can be *static* when pages are computed at application definition time and are immutable during application usage; or *dynamic*, when pages are created just-in-time from fresh content. Dynamicity has further nuances: it may involve only content (navigation and presentation are static), or scale to presentation and navigation.

### 2.5 Usability

From the customer's perspective, usability is the most important quality of a Web application. Although a thorough discussion of the emerging Web usability engineering field is outside the scope of this paper (we refer the reader to [Nielsen 1996]), it is possible to identify a set of general criteria to use in the assessment of Web usability:

- **Degree of visual quality**: indicates the overall coherence of the presentation and navigation metaphors and the individual quality of the graphic resources.

- **Degree of customization**: measures the facility of tailoring the application interface to individual users or user groups. At one end of the spectrum, applications may have a fixed interface; at the other extreme, content, presentation, and navigation may be individually personalized.

- **Degree of adaptivity**: is proportional to the run-time flexibility of the interface; at one end, the application may be immutable; at the other, it may track the user's activity and change accordingly [Fraternali and Paolini 1998].

- **Degree of proactivity**: indicates the capability of an application to interact with the user on its own initiative. Applications can be passive, i.e., provide information only in response to user's requests, or proactive, i.e., able to push information to users upon the occurrence of relevant events.

### 3. TOOLS FOR WEB DEVELOPMENT

In response to the above described complexity factors of Web application development, many software vendors are providing instruments for supporting the construction of new applications, or the migration of existing ones.

Notwithstanding their common goal, the available products greatly diverge in many aspects, which reflects different conceptions of the nature of a Web application and of its development process.

To help understanding the state of the practice, we have grouped existing tools into six categories, which exhibit homogeneous features. This grouping is the result of a broad review, which has considered over forty products, listed in Appendix 15.

The individuated categories are:

(1) Visual editors and site managers;
(2) Web-enabled Hypermedia authoring tools;
(3) Web-DBPL integrators;
(4) Web form editors, report writers and database publishing wizards;
(5) Multi-paradigm tools
(6) Model-driven application generators.
The order of presentation of the different categories reflects the increasing level of support that tools in each category offer to the structured development of Web applications.

Category 1 contains productivity tools that are a direct evolution of HTML editors, which do not really support the development of large scale Web-database applications but are nonetheless interesting because they pioneered many concepts (like presentation styles and top-down site design) later integrated into more complex environments. The same limitation in the degree of support to structured development of large applications is shared by Category 2, which originates from a different application domain, off-line Hypermedia publishing, but recently added facilities for Web and database integration; the interest in these tools is motivated by their non-conventional (with respect to the standard software engineering practice) approach to application design and specific focus on navigation and presentation. Category 3 is the first one that explicitly addresses the integration of Web and databases to achieve a higher level of scalability, and includes very powerful, yet basic, products; these products can be used to implement applications on top of large databases, although at the price of a substantial programming effort. Category 4 takes quite a different, but still database-centric, approach to Web development, by addressing the migration of client/server, form-based applications; these tools aim at augmenting the implementor’s productivity in such tasks as form editing, report writing, and event-based programming; they offer a higher level of support with respect to category 3, but still concentrate only on the implementation phase. Category 5 contains a number of tools whose common feature is the integration of different development approaches and technologies, drawn from the previous four tool families. Finally, category 6 includes those products (actually two products) that provide a complete coverage of all the development activities, from conceptualization to implementation, by leveraging state-of-the-art software engineering techniques.

To complete the overview of Web application development, in Section 10 we include a mention to other classes of products which are not directly concerned with the specific focus of this paper on the design and maintenance of Web sites, but either cover fundamental architectural issues like distribution, reliability, performance, and security, or provide specialized facilities to users and site administrators, like advanced search functions and collaborative work support.

4. VISUAL EDITORS AND SITE MANAGERS

This class includes authoring and site management environments originally conceived to alleviate the complexity of writing HTML code and of maintaining the pages of a Web site in the file system. In a typical configuration these products bundle a WYSIWYG editor, which lets the user design sophisticated HTML pages without programming, and a visual site manager, which displays in a graphical way the content of a Web site and supports functions like page upload, deletion, and renaming, and broken link detection and repair. Among the many products in this category there are Adobe SiteMill and PageMill, NetObject Inc.’s Fusion, SoftQuad’s HotMetal, Claris Home Page, Macromedia’s Backstage Designer, and Microsoft’s FrontPage.

From a lifecycle perspective, these products address the implementation and
maintenance of Web sites; implementation is supported by content production functionalities and maintenance by site-level management facilities. The most advanced products (e.g., NetObject's Fusion and Microsoft FrontPage) also offer a rather rudimental approach to design, whereby it is possible to separate the definition of the hierarchical structure of a site from the authoring of content.

Automation concentrates on content production, by generating code from visual page designs. Support to code generation is particularly relevant in those tools (like Macromedia's DreamWeaver, SofQuad's HotMetal Pro 5.0, Allaire's HomeSite, and many more) that support the latest extensions of HTML, like Cascading Style Sheets (CSS) [World Wide Web Consortium 1998a] for content-independent presentation specification, and the Document Object Model (DOM) [World Wide Web Consortium 1998b], for the object-oriented representation of page elements and their manipulation via a scripting language.

Some tools are also able to generate part of the navigation logic, by automatically inserting navigation buttons into pages based on their position in the tree-like layout of the site (figure 4). Development abstractions are basically implementation-level and oriented towards structure and navigation (pages and links); some tools also provide presentation abstractions, in the form of page templates or "themes", i.e., groups of graphic resources that can be applied to multiple pages to obtain visual consistency (figure 5).

Reuse mostly happens at the implementation level: content objects (e.g., multimedia, graphics, but also object-based components, like JavaBeans and ActiveX controls) can be reused across applications.

The architecture originally supported by these tools is file-based and two tier, and the binding between an application and its content is static. Recently, the most advanced tools have added functionalities for the dynamic connection to live database data, as we discuss in Section 8.

Since development is mostly conducted by hand, the level of customization and the resulting visual quality of the final application is proportional to the effort spent and can be arbitrarily high; coherence of presentation and navigation metaphors is facilitated by visual templates and themes, although in this case customization becomes more expensive.

The features of these tools are shown in Table 6: in summary, they are an excellent solution for small- to medium-sized applications, where publishing large information bases stored in a DBMS is not the major issue. The lack of a design-level schema of the application independent of content imposes that the application features be defined instance-by-instance and is a major obstacle to scale-up and integration of large masses of data. These limitations, however, are going to become less sensible as these products become more integrated with Web-database tools.

5. WEB-ENABLED HYPERMEDIA AUTHORING TOOLS

Web-enabled Hypermedia authoring tools share the same focus on authoring as visual HTML editors but have a different origin, because they were initially conceived for the development of off-line Hypermedia applications, and have been extended to support the generation of applications for the Web in recent times. The best known representatives of this class of products are: Asymetrix's ToolBook II Assistant/Instructor, Macromedia's Director and Authorware, Formula Graphics'
Fig. 4. Hierarchical Site Layout in NetObject’s Fusion

Fig. 5. Graphic Styles in NetObject’s Fusion
Multimedia 97, Aimtech Iconauthor, and Allen Communication Quest. Most of these products have both Web export facilities and database connectivity, but these extensions are not always compatible, and often the developer must choose between the two possibilities. These products are characterized by the following features:

— The authoring metaphor used in the definition of the application. Examples of such metaphors are: flowchart (Quest, Authorware, IconAuthor), book (Toolbook), movie making (Director).

— The way navigation is defined, which may require programming in a scripting language or may be supported by visual programming and wizard tools coherent with the product’s authoring metaphor (e.g., in Director objects’ synchronization is defined by editing the score for the cast members of a stage, see figure 7).

— The type of database connectivity, which may range from support of an internal database, of an external database via gateway software (typically ODBC or JDBC), of an external database through DBMS API.

— The type of Web connectivity, which may be achieved by means of a plug-in application extending a Web browser or by exporting the Hypermedia application into a network language. Web connectivity may affect database connectivity, because in the present version of most tools Web export can be done only for applications not connected to a database.

— The export language, which may be HTML, Java, or a mix of both.

Hypermedia tools basically are authoring assistants and their focus is restricted to the ad hoc implementation of long-lived applications to be published once and for all in CDROMS or information kiosks. As a consequence, they still have little provision for a structured development lifecycle and offer limited support to application modelling, design, testing and maintenance.

To aid the implementation of large applications, some tools separate authoring in-the-large (i.e., the definition of the overall application structure) from authoring in-the-small (i.e., the detailed definition of the multimedia content of individual information nodes) [GMP95]. This distinction fosters an elementary form of design which helps reconfiguring the application structure and navigation independently.
of content. For example, tools like Quest, Authorware, and IconAuthor distinguish between a “design mode”, where information nodes are placed on a flowchart, and an “edit mode”, where each individual node can be edited. A parallel can be established between authoring in-the-large in Hypermedia authoring tools and the site layout view offered by some visual HTML editors: in both cases, design is done at the instance level, without the help of a content-independent application schema, and navigation is (in part) automatically generated from the abstract structure of the application.

Reuse is pursued both at the design level, by means of reusable substructures and application independent layouts, and at the implementation level, by leveraging libraries of scripts or multimedia components. The recent versions of most tools also offer interoperability with third-party components (for example inclusion of Java applets and ActiveX components).

Regarding the development abstractions, Hypermedia tools have a strong plus in the built-in facilities for explicitly modelling navigation paths orthogonal to application structure; for example, guided tours can be defined as sequential navigations based on associative properties of objects that may cut across the hierarchical structure of the application. Other navigational aids include application-wide search functions, glossaries, and access indexes.

Like visual HTML editors, Hypermedia authoring tools:

—Do not provide a schema for distinguishing the type of objects constituting the application (beside the low-level distinction between objects of different media
types). The main development abstractions are generic container objects (pages, icons, stages), whose definition intermixes structural, presentation, and behavioral aspects (and in tools not supporting the separate specification of navigation, also navigational features).

—Offer presentation abstractions in the form of page styles, i.e., sets of graphic resources applied uniformly to the entire application or to part of it.

Architecture is where Hypermedia tools show a substantial immaturity with respect to competitor solutions: all tools were originally conceived for offline, file-based publication, and later upgraded to external database connectivity and Web export. Most products offer database connectivity via external libraries using ODBC, but lose this feature after Web export. Moreover, translation from native proprietary format into a network language is still rudimental, and the most advanced features (like those obtained by exploiting the tool’s scripting language) do not carry over to the Web version of an application.

Conversely, usability is the major strength of this category of products, which focus on delivering very sophisticated user interfaces exhibiting a degree of control over graphic accuracy and multimedia synchronization hardly available with other means. The inherent navigational design paradigm, coupled to very effective and well-established aids like guided tours, and user-defined access indexes, which most tools offer for free or with little programming effort, contribute to the deployment of applications which are very effective and much closer to the kind of communication found in high quality, hand-developed Web sites.

Table 8 summarizes the Web-related features of Hypermedia authoring tools.
6. WEB-DBPL INTEGRATORS

The common denominator of the products in this category is to produce Web pages dynamically from information stored in a database, by integrating databases and Web technology at the language level.

Different network languages (notably, HTML and Java) are merged with a database programming language (DBPL), to obtain an intermediate application programming language capable of making the best of both the Web and database technology. Web-DBPL integration is pursued in three distinct ways:

— **HTML extensions** add new tags to HTML to embed both generalized programming constructs and database-specific capabilities. The developer writes page templates, which intermix declarative HTML tags and executable code. Products in this category can be further distinguished based on the language paradigm used to extend HTML (imperative, object-based, object-oriented). Page templates must be translated into pure HTML and this task is normally executed at run-time by an interpreter. The performance of the different products vary based on the process in which the interpreter is executed and on the way the connection to the database is managed. Examples of HTML extensions include: the Cold Fusion Web Database Construction Kit by Allaire Inc., Microsoft’s Active Server Pages (ASP) and Internet Database Connector (IDC), StoryServer by Vignette Corporation, HAHT Software’s HahSite, and Informix AppPages. In general, HTML extensions have a broader objective than the mere addition of database connectivity, and include primitives for overcoming well-known limitations of HTML, like the absence of control flow instructions, modularization mechanisms, and stateful interaction\(^3\).

— **DBPL extensions** take the other way round and add specialized functions to an existing DBPL, or to a general purpose programming language with a database interface, to enable the output of HTML code as a result of program execution. Developers write database applications which construct HTML output from data retrieved from the database. This approach requires database programming skills, but may yield better performance, when DBPL programs are executed and optimized directly by the DBMS. Examples of DBPL extensions include: Oracle’s PL/SQL Web Toolkit, Sybase’s PowerBuilder Web.PB class library, and Borland’s Web interface for the Delphi Client Server Suite.

— **Java extensions** add database connectivity to Java. The best known effort in this direction is the JDBC open standard, which offers an ODBC-like architecture for Java applications that want to interact with a DBMS. Differently from HTML extensions, connectivity is added in the form of a library masking under a uniform callable interface the details of interacting with different proprietary DBMSs and SQL dialects.

Differently from visual HTML editors and Hypermedia authoring tools, which offer small-scale application development functions to the non-programmers, Web-DBPL integrators are tools for the programmer, who may use them to simplify

\(^{3}\text{Stateful interaction is the capability of retaining the application status between two consecutive client's requests, which is not supported by HTTP/1.0.}\)
the task of gluing together Web pages and database queries by hand. Due to their focus on implementation languages, these products lack high level abstractions for describing applications and thus do not assist the developer in identifying the structure, navigation, and presentation of an application, which is directly implemented by manually writing the necessary HTML page templates or DBPL programs.

Reuse may be achieved either through the native constructs of the programming language, for example packages and modules, or through ad-hoc modularization mechanisms, for example inclusion between HTML page templates.

The target architecture is database-centered and three-tiered; the most sophisticated products also permit multiple database tiers and distributed queries.

Not having any specific support for presentation and navigation modeling, these products have no direct impact on the usability and graphical quality of the user interface, which must be designed separately.

The major difference between Web-DBPL integrators and visual HTML editors and Hypermedia authoring tools is that the former introduce a clear separation between the final application pages and the content repository, the schema of which acts as an implicit specification of the application structure.

This distinction has several consequences on the application development process:

— it permits a more effective manipulation of the information base, because changes to content can be performed without affecting navigation and presentation;
— it exposes the different object types that constitute the application, although these types are not elicited from the structural, navigational and presentation requirements of the application, but stem from the design of the underlying database;
— thanks to the explicit object typing, the uniformity and coherence of the presentation are facilitated, because all objects of the same type may be easily visualised in the same way. As a counterpart, customization at the instance level must be explicitly programmed, by identifying exceptional cases and treating them in ad hoc ways.

Being database driven, applications built with Web-DBPL integrators may leverage the reactive capabilities of the underlying DBMS to achieve adaptivity and proactivity, e.g., database triggers. For example, an Oracle trigger could react to user-generated events and produce HTML pages adapted to the specific situation.

In spite of the above mentioned positive side-effects, Web-DBPL integrators cannot be considered development tools in themselves, but are comparable to traditional client/server 4GL, because they provide a high level programming interface masking lower-level architectural details; as such, they are often used to build more sophisticated products, like the ones reviewed next.

Table 9 summarizes the most important features of Web-DBPL integrators.

7. WEB FORM EDITORS, REPORT WRITERS, AND DATABASE PUBLISHING WIZARDS

This category collects a large number of products, all having the common characteristic of leveraging traditional database design concepts and development tools to rapidly deploy both new and legacy data-intensive applications on the Web. A
<table>
<thead>
<tr>
<th>Process: Lifecycle Coverage</th>
<th>Structural design (design of underlying database)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Implementation</td>
</tr>
<tr>
<td></td>
<td>Content maintenance</td>
</tr>
<tr>
<td>Process: Automation</td>
<td>Web-database communication</td>
</tr>
<tr>
<td></td>
<td>Query shipment and result processing</td>
</tr>
<tr>
<td></td>
<td>Page formatting from query results</td>
</tr>
<tr>
<td>Abstractions</td>
<td>Design-level: database structures (tables, classes)</td>
</tr>
<tr>
<td></td>
<td>Implementation-level: pages, interface objects</td>
</tr>
<tr>
<td>Reuse</td>
<td>Reusable modules: page templates, procedures, classes</td>
</tr>
<tr>
<td>Architecture</td>
<td>Three/multi-tier</td>
</tr>
<tr>
<td></td>
<td>Dynamic binding of content to pages</td>
</tr>
<tr>
<td>Usability</td>
<td>No specific support for control of graphical quality</td>
</tr>
<tr>
<td></td>
<td>Coherence facilitated by page templates (HTML extensions)</td>
</tr>
<tr>
<td></td>
<td>Low customization, adaptivity and proactivity manually implementable through database triggers</td>
</tr>
</tbody>
</table>

Fig. 9. Synopsis of Web-DBPL Integrators

The first classification can be obtained by considering the functions offered by the tools, ordered by complexity and sophistication:

---

- **Database export**: support is limited to the publication of basic database elements, like tables and views, and is achieved by automatically mapping database content into a network language, notably HTML. The export can be either static or dynamic, the latter case requiring on-the-fly HTML generation, which is normally implemented by means of a Web-DBPL integrator. A certain level of personalization can be obtained, by applying sets of preferences to drive the output of Web pages. A typical example is the exportation of tables and queries in Microsoft's Access97.

- **Report writing**: more complex read-only applications can be obtained by defining custom reports, which can be exported to the Web either statically or dynamically. Dynamic publication requires a report interpreter working side by side with the Web server. Examples include: Crystal Report Print Engine, Oracle's Reports for the Web, Microsoft's Access97 report export facility.

- **Form-based application development**: assistance extends to the construction of interactive, form-based applications for accessing and updating data, and more generally for implementing business functions.

Form-based application development is where the highest number of products concentrate, and developers are faced by an impressive spectrum of alternatives. Among the reviewed products, we cite: Microsoft's Visual InterDev, Visual Basic 5, and Access97, Oracle Developer 2000, Inprise IntraBuilder, Sybase's PowerBuilder, Apple's WebObjects, NetDynamics Visual Studio, Asymetrix SuperCede Database Edition, and Allaire's Cold Fusion Application Wizards. The most relevant dimensions characterizing the different products are:

---

- the application delivery format, i.e., the format in which the application is received by the browser, which can be pure HTML, a combination of HTML and
Java, HTML plus proprietary component technology and scripting languages (e.g., client-side ActiveX objects, VBScript or JavaScript tags), or a totally proprietary format (e.g., ActiveX documents, Sybase’s PowerBuilder PBD files). The delivery format affects the client’s performance and the application’s portability, with minimal overhead and maximum portability in the pure HTML case, and maximum overhead and minimal portability when the application is deployed in a totally proprietary format interpreted by a browser’s plug-in application.

—The application development language(s), which may be different from the delivery language(s) to preserve existing software investments and programming skills and/or increase performance and portability. Overall, the spectrum of bindings between development and delivery languages is extremely varied, ranging from products offering a single development language and multiple output formats (e.g., Oracle’s Developer 2000 maps PL/SQL to HTML, Java, and Visual Basic), to products translating different input languages into the same output format (e.g., Microsoft’s Visual InterDev maps design-time ActiveX controls scripted in VBScript or JavaScript into pure HTML).

—The implementation paradigm, which can be 3GL programming, Rapid Application Development (RAD), wizard-based, or a mix thereof. Almost all tools offer RAD functionalities letting programmers define an application by dragging and dropping components onto a form and by visually customizing their properties. Behavior customization, as necessary for the definition of complex forms, can be added by writing code in the tool’s development language. A few products (e.g., Visual InterDev, IntraBuilder, NetDynamics, Cold Fusion) also include a simplified development path for non-programmers, based on “wizards” enabling the automatic generation of code from sets of preferences selected through a visual interface (figure 10). Wizards are applied to obtain both components and complete applications; in the latter case they deliver “canned” database applications, in which the data structure and the interface logic are predefined (e.g., master-details data entry and visualization, drill-down reports, and so on).

—The execution architecture: the way in which an application can be partitioned among the different tiers of the Web architecture is even more varied than the choice of development and delivery languages, because object distribution (as supported by languages like Java and architectures like Corba and DCOM) permits the same functionality, e.g., a database driver, to be located either in the client or in the application server.

From a software engineering point of view, products in this class are to Web development what Integrated Development Environments (IDEs) and Rapid Application Development tools were to traditional application programming: instruments for boosting individuals’ and teams’ productivity in the implementation phase. The best tools also help process management, testing, and maintenance by leveraging standard software engineering techniques like source and version control, configuration management, and visual debugging.

With one exception reviewed later, form-based application generators do not provide upper-CASE capabilities: they do not encompass conceptual modeling nor support the model-driven design and implementation of applications. Typically, data structures, business logic, and interfaces are specified and designed separately
and then implemented with the tool of choice.

Component-based reuse is also intrinsic to most products, which leverage the various object-level interoperability standards emerging on the Web. Components can be either client-side, in which case they encapsulate typical interface controls (e.g., menus, edit fields, grids, and so on), or server-side, in which case they typically wrap database access and data formatting primitives.

The development abstractions are interface-oriented and drawn from the traditional client/server, database-centric world: query and data entry forms, input/output controls, record lists, and reports. The structure of the application is dictated by the forms that compose it and is generally isomorphic to (a subset of) the database schema, especially when forms are automatically derived from database tables and views through database wizards. More complex patterns can be obtained by exploiting canonical structures like table lookups and master-details, or by hand-programming non-standard combinations of objects. The navigation semantics is deeply embedded within the application and hyperlinks are emulated by means of form-to-form connections. In some cases, navigation is automatically inferred by the tool from database structure, like for example in automatically generated master-details and drill-down applications, where database foreign key constraints are translated into HTML hyperlinks. Presentation is addressed as an independent issue only in those products that provide a notion of "presentation style" (e.g., models in Access97, themes in InterDev, master pages in IntraBuilder)
### Table 11: Overview of Web Form Editors, Report Writers, and Database Publishing Wizards

<table>
<thead>
<tr>
<th>Process: Lifecycle Coverage</th>
<th>Implementation</th>
<th>Automation</th>
<th>Abstractions</th>
<th>Testing</th>
<th>Implementation-level: forms, reports, controls</th>
<th>Reuse</th>
<th>Architecture</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process:</td>
<td></td>
<td>Source code generation from RAD and wizard tools</td>
<td>Source code generation from RAD and wizard tools</td>
<td></td>
<td>Implementation-level: forms, reports, controls</td>
<td></td>
<td>Multi-tier, integration with application servers</td>
<td>Canned interfaces based on query/result display loop</td>
</tr>
<tr>
<td>Automation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dynamic binding of content to pages</td>
<td></td>
<td></td>
<td>Low customization, no adaptivity, no proactivity</td>
</tr>
<tr>
<td>Reuse</td>
<td></td>
<td>Plug-in components: client-side and server-side</td>
<td>Source code generation from RAD and wizard tools</td>
<td></td>
<td>Implementation-level: forms, reports, controls</td>
<td></td>
<td></td>
<td>Canned interfaces based on query/result display loop</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dynamic binding of content to pages</td>
<td></td>
<td>Multi-tier, integration with application servers</td>
<td>Low customization, no adaptivity, no proactivity</td>
</tr>
</tbody>
</table>

Fig. 11. Synopsis of Web Form Editors, Report Writers, and Database Publishing Wizards

which can be uniformly applied to all application pages to obtain a consistent look.

On the architectural side, all products target multi-tier architectures, and the most recent generation (e.g., NetDynamics Visual Studio) is tightly integrated with specialized middleware products, called application servers, which we review in section 10.1.

Differently from Web-DBPL integrators, which are neutral products that can be used to shape any kind of application, form-based generators and database publishing wizards deliver complete software solutions inclusive of user interface logic, and thus their underlying interaction model influences the user perception of the application quality. The evaluation must consider the application context in which a tool is used and consequently the kind of users addressed. In an intra-business or business-to-business context, these products are essentially vehicles for Web-enabling traditional client-server applications. In this setting, interface quality is proportional to the similarity between the look and feel of original client-server applications and their Web counterparts, because the peculiarities of Web interaction (free browsing, associative links, multimedia, user-level customization) are less important than adherence to interface standards well-established among users. Conversely, if these products are targeted to user-oriented applications for the Internet, they must be compared to Web and Hypermedia authoring tools, in which case their form-to-form navigation pattern and the limited graphic control imposed by wizards are a source of rigidity.

Table 11 overviews the main features of Web form editors, report writers and database publishing wizards.

### 8. MULTI-PARADIGM TOOLS

The ongoing trend of evolution of Web development tools sees an increasing convergence between visual editing and site management products, Web-HTML integrators, and client-server products (reviewed in Section 4, 6 and 7, respectively).

Multi-paradigm tools integrate solutions from the abovementioned categories into a unified development framework. The most typical configuration is the one in which visual HTML editing and site administration are extended with external components which provide database connectivity, or with enhanced HTML genera-
**Process: Lifecycle Coverage**

- Design (limited to hierarchical layout definitions)
- Implementation and maintenance

**Process: Automation**

- HTML generation from WYSIWYG page editing
- Generation of commands for hierarchy navigation
- Web-database communication
- Source code generation from RAD and wizard tools

**Abstractions**

- Implementation-level: pages, links, presentation styles, interface objects, tables, queries

**Reuse**

- Plug-in components, reusable presentation styles

**Architecture**

- Three and multi-tier
- Static and dynamic binding of content to pages

**Usability**

- High graphical control through manual authoring
- Coherence through use of templates and presentation styles
- Low customization, adaptivity and proactivity manually implementable through database triggers

---

**Fig. 12. Synopsis of Multi-Paradigm Tools**

Mult-paradigm tools do not introduce novel approaches into Web development, but combine already established features to broaden their spectrum of applicability with respect to single-paradigm products. Therefore, they offer advantages (summarized in Table 12) that are the sum of the strengths of their components (implementation productivity, top-down design, three-tier architecture, high graphic control), but do not overcome the major limitation of the products reviewed so far, i.e., the lack of a high level, content-independent representation of the site in terms of structure, navigation and presentation.

9. **MODEL-DRIVEN WEB GENERATORS**

Model-driven Web generators are at the top of the proposed categories, because they provide the highest level of automation and lifecycle coverage, by applying conceptual modeling and code generation techniques to the development of Web applications.

This category comprises a few commercial tools, which exhibit different conceptual models and code generation techniques. We review Hyperwave Server 4.0, and Oracle’s Web Development Suite.
Hyperwave [Hyperwave Information Management 1998] is an advanced document management environment which permits remote users to browse, annotate, and maintain documents distributed over the Web. Hyperwave has a very basic, yet powerful, high level model of a Web application, which is considered as a set of document collections organized hierarchically. Collections may contain sub-collections and documents, and have different behavior based on their type. Documents in a collection can be linked and annotated with a number of meta-data. Links are managed by the Hyperwave server, so that any hypertextual structure can be superimposed over a set of otherwise independent documents.

From the description of collections, links and meta-data, Hyperwave generates a Web interface that enables both user-oriented and administrative functions like collection and link browsing, searching and notification, remote document management, fine-grain version and access control, collaborative work, and personal annotations. The generated interface has a default presentation, which can be personalized using a proprietary template language.

Although document-oriented, the Hyperwave server relies on database technology and on a multi-tier architecture to store meta-data and manage links spanning multiple servers.

A more database-centric approach is taken by the Oracle Web Development Suite, which comprises Designer 2000 [Gwyer 1996; Barnes and Gwyer 1996], a CASE tool for generating Web applications from augmented Entity-Relationship diagrams.

Designer 2000 is an environment for business process and application modeling, integrated with software generators originally designed to target traditional client-server environments, namely Oracle Developer 2000 [Hoven 1997] and Visual Basic. The Web Generator enables previous applications developed with Designer 2000 and deployed on LANs to be ported to the Web, as well as the delivery of novel applications directly on the Internet or on intranets. The Web Generator takes its inputs from the Designer 2000 design repository and delivers PL/SQL code that runs within the Oracle Web Server to produce the desired HTML pages of the application. More specifically, three inputs drive the generation process:

—A Web-enhanced database design: database design diagrams specify the structure of the database in terms of tables, views, foreign key relationships, and integrity constraints. These constitute the schema of the future Web application. A few visual features can be specified in the schema: for example, column definitions can be supplemented with caption text and display format (e.g., a pop-up list). Moreover, some integrity constraints (e.g., valid ranges) can be attached to columns and tables, and the Web Generator can be instructed to produce code for checking them on the server via PL/SQL or on the client via JavaScript.

—The definition of applications and modules: modules correspond to basic application units; each module consists of a sequence of tables, linked by foreign key relationships (figure 13). The order of tables in a module determines the sequence of HTML pages that will be produced for that module. Navigation is established by drawing links between modules: the designer may define which modules can be reached by a given module and introduce fictitious modules acting as hierarchical indexes over other modules.

—The user preferences: user preferences are parameters that can be set to govern
the presentation of the generated application; they can be defined either globally, at the module, or at the component level. Example of preferences are colors, headers and footers, background images, and help text.

From these inputs, the WEB Generator produces fixed-format Web pages; one set of related pages is generated for each module and links between different modules are turned into hyperlinks between the HTML startup pages of modules.

Model-driven generators apply to Web development the fundamental principles of software engineering; differently from all other product categories, application are first modelled at the conceptual level and then implemented through code generation. This approach and its benefits are comparable to the ones delivered by CASE tools for object-oriented application development, and range from reduced development effort, reverse engineering, and multi-level reuse.

However, Hyperwave and Designer 2000 diverge in the underlying conceptual model, and both exhibit limits in the description of Web applications.

Hyperwave adopts a simplified hypermedia model, which is well suited to represent navigation, but lacks a proper structural model; as a consequence, the information base is reduced to a set of flat documents annotated with meta-data.

Conversely, Designer 2000 draws the development abstractions from the database world and adapts such concepts as entities and relationships to Web modeling, by adding to them some navigation and presentation flavor. This attempt to twist a database conceptual model and apply it to Web application is responsible of some limitations in the usability of the resulting applications, where the absence of proper modeling abstractions for navigation and presentation limits the exploitation of the
communication capabilities of the Web.

The limitations of commercial Web generators are addressed by a few research prototypes, namely Araneus [Atzeni et al. 1997], Autoweb [Fraternali and Padini 1998], Strudel [Fernandez et al. 1998], WebArchitect [Takahashi and Liang 1997], W3I3 [Ceri et al. 1998], HSDL [Kesseler 1995], RMC [Diaz et al. 1995], and OOHDM [Schwabe and Rossi 1995], which are discussed in Section 12.

10. MIDDLEWARE, SEARCH ENGINES, AND GROUPWARE

Beside support to conceptualization, design, and implementation, Web development requires tackling other issues which, although outside the focus of the paper, are critical for the delivery of effective applications: performance, availability, scalability, security, information retrieval, and support to collaborative administration and usage.

These needs are served by ad hoc tools or by specialized functions integrated into Web design products. In the sequel we briefly review the most important features of three categories of products: application servers, search engines, and groupware and collaborative design tools.
10.1 Middleware

Industrial-strength data-intensive applications require not only proper design tools but also a solid architecture, good performance, availability, scalability, and security. These goals have prompted for the extension of the original two-tier HTTP architecture to more sophisticated configurations, encompassing three, and even multiple, tiers. The key to multi-tiered applications is the capability of separating data, interfaces, and application logic, and of distributing each aspect to distinct network nodes. Such distribution leverages Internet-enabled application protocols (like Corba Internet InterOrb Protocol–Corba IIOP [Object Management Group 1996]–and Microsoft’s Distributed Common Object Model–DCOM [Microsoft Corporation 1996]) and the native remote procedure call capabilities of network languages (notably, Java’s Remote Method Invocation–RMI [Sun Microsystems 1995]).

The in-depth presentation of the alternative architectural and technological options for multi-tiered applications is outside the scope of this paper and is thoroughly addresses by many authors (see, for example, the special issue of IEEE Internet Computing on Internet Architectures [Benda 1998] for a recent review of the state of the art, and Byte’s special report on networked components [Montgomery et al. 1997] for a comparison of the Corba’s and Microsoft’s architectures).

However, also in the tool market several vendors are addressing multi-tiered architectures, by offering specific products, called application servers, which are middleware facilities either integrated into HTTP servers, or working side by side to them. Example of application servers are NetDynamics, Lotus Domino Server, Oracle 8i Java Server, Sybase’s Jaguar CTS and Inprise’s VisiBroker for Java.

Application servers do not directly impact the client-side design process, but offer several extensions to the standard functions of HTTP engines which can be used to support server-side development:

—Efficient execution of server-side programs that implement the business logic of the Web application. Products differ in the development languages supported, which commonly include C++, Java and scripting languages like Perl, Visual Basic and JavaScript. Efficiency is improved by replacing the Common Gateway Interface (CGI) between the Web server and the application programs by means of optimized protocols and architectures, like FastCGI (http://www.fastcgi.com) and Java Servlets [Chang 1998].

—High performance client-server communication. Such capability may be based on a comprehensive architecture for application distribution, on programming language features, or on proprietary remote procedure calls. Rather than using HTTP, enabled clients may bypass the Web server and directly connect to the application server using either open protocols, like Corba/IIOP and Java RMI, or proprietary protocols, like Microsoft’s DCOM, Domino’s Notes Remote Procedure Calls (NRPC), and Sybase’s Jaguar CTS remote procedure calls.

—Optimized and extensible connection to multiple external data sources. This functionality includes the pooling of connections to a database across multiple clients, the caching of result sets, and the possibility to extend the application server with plug-in gateways to heterogeneous data sources like Enterprise Resource Planning (ERP) and legacy systems (examples of built-in gateway components and programmable toolkits are NetDynamics 4.0 Platform Adapter Com-
—Flexible and dynamic load balancing of client requests by means of automatic replication of server functions. High-volume incoming traffic may be dynamically routed to multiple instances of server-side functions, implemented either as separate threads of the application server, or as external processes, possibly residing on remote hosts. Dynamic redirection also enables transparent failure handling: user's request for an unavailable service can be routed to a replica of the same function, either dynamically spawned or statically instantiated at system configuration time.

—Implementation of a secure infrastructure for both data and messages. Security may be granted by a variety of means: user logging and authentication over both HTTP and non-HTTP connections (e.g., by supporting the X.509 open standard certificate format, or authentication via third party directories), message and content encryption (e.g., by supporting the IETF Secure MIME standard and RSA/RC2 cryptography), and password quality testing.

—Transactionality, i.e., the capability of performing atomic and recoverable transactions over a single or multiple tiers. This feature, traditionally offered by distributed database servers and TP monitors, requires the application server to implement write ahead logging and two-phase commit protocols.

10.2 Search

The proper design of structure and navigation normally results in Web sites with a self-evident organization and consequently reduces the need of full-text searches over the information base [Halasz 1988]. However, Web applications offered to the general public must also consider the needs of casual readers and of readers with highly specific interests, for which content search is the most effective interaction paradigm.

Designing the search functions for a data-intensive Web site is an orthogonal issue with respect to the design of structure, navigation and presentation, and is supported either by ad hoc functions integrated into Web development tools, or by specialized products, called search engines.

Examples of Web development tools bundling integrated search functions are Hyperwave (which comes also with a separate commercial search engine) and Lotus Domino Designer. Among the numerous standalone search engines, we mention Verity Search97, Harvest, OpenText's LiveLink Search and Spider, AltaVista Search, QuarterDeck's WebCompass, and Excite for Web Servers.

Search engines basically consist of two main components: a user interface and query processor, whereby users can pose queries and obtain a ranked list of pages whose content satisfies the query; and an indexing component (also called spider or crawler) which creates and maintains indexes over the data sources.

The available commercial products differ on a variety of dimensions: the kind of queries they support (keyword-based, boolean, natural language, fuzzy); the customizability of the display of results; the flexibility of the index creation and maintenance process (e.g., the possibility to schedule the updates of indexes differently for different data sources or to analyze different file formats); and the adherence to the so called robot-exclusion standard, by which webmasters can deny access to
crawlers at their sites.

For a broad review of commercial search engines and a comparison of their features the reader may refer to [Lange 1997].

10.3 Groupware

Collaboration requirements affect Web development in several ways, going from the concurrent construction of a Web site by geographically distributed development teams, to the provision of limited content editing and interaction functions directly to end-users, to real-time interaction integrated into a Web site.

In the commercial market several tools offer groupware capabilities, including

—concurrent access control via content locking and checkin/checkout procedures;
—distributed file systems for concurrent content upload;
—offline collaboration tools like calendars, schedules, notification lists, and discussion groups;
—real-time collaboration facilities, like virtual rooms, white boards, chats, and video conference.
—full-fledged workflow support, including offline and online collaboration facilities and workflow modeling and implementation.

Example of design tools supporting collaborative development are Hyperwave, TeamSite, and Lotus Domino Designer. Products for the definition of virtual workteams are reviewed in [Wong 1998], and a comprehensive survey of Web-enabled workflow systems is contained in [Miller et al. 1997].

11. EVALUATION

Table 15 summarizes the features of the categories of Web development tools described in Sections 4 to 9, in the light of the perspectives of Web development discussed in Section 2. To complete this picture, developers must also consider middleware products, which address the enhancement of performance and security and offer a platform for implementing advanced business logic, and search engines and groupware tools, which add specialized functions orthogonal to the design process.

The state of the practice summarized in Table 15 can be better understood if compared with the current situations of mature software technologies, like object-oriented systems and databases (see figure 16).

In these contexts, development tools cover the entire spectrum of the application lifecycle and approach development from the right angle: they exploit well established abstractions suited to the specific context, like the conceptual and logical models for database design, and the object oriented notations for OO systems, and follow well-proven development guidelines, like those described in [Ceri et al. 1993] for database design and in [Rumbaugh et al. 1991] for object oriented design. Pictorially, mature tools and approaches are represented as “light cones” that enlighten the various phases of the development cycle, without crossing the borders of their target application field.

The situation of Web development is different and typical of a not yet mature technology (it could be easily compared to the OO tool market in the eighties):
<table>
<thead>
<tr>
<th>Life-cycle coverage</th>
<th>Visual Editors</th>
<th>Hypermedia Tools</th>
<th>Web-DBPL Integrators</th>
<th>Form Editors</th>
<th>Multi-paradigm Tools</th>
<th>Model-driven Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implement., design, site design, link maintenance</td>
<td>Implement., design (Authoring in-the-large)</td>
<td>Implement., maintenance</td>
<td>Implement., hierarchical site design, link maintenance, debugging</td>
<td>Implement., hierarchical site design, link maintenance, debugging</td>
<td>Conceptual, design, implement., maintenance, reverse eng.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Automation</th>
<th>Generation of HTML</th>
<th>Generation of HTML/Java</th>
<th>Database connection, query management, result formatting</th>
<th>Generation of HTML/Java</th>
<th>Generation of HTML, database connection</th>
<th>Generation of design schemas, navigation commands, interfaces</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Abstractions</th>
<th>Page, link, presentation style</th>
<th>Authoring metaphors</th>
<th>Table, page elements</th>
<th>Form, report, client-side and server-side control</th>
<th>Page, link, presentation style, form, table</th>
<th>Entity, relationship, module, table, column, collection, link</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Reuse</th>
<th>Components, presentation styles</th>
<th>Libraries, skeletons, components, styles</th>
<th>Page templates, DBPL units</th>
<th>Client-side and server-side components</th>
<th>Components, presentation styles, templates</th>
<th>Modules, preferences, collections, links</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Default Architecture</th>
<th>2-tiers, static</th>
<th>2-tiers, static</th>
<th>3-tiers, dynamic</th>
<th>3-tiers, dynamic</th>
<th>3-tiers, dynamic</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Support to usability</th>
<th>Good graphic control and coherence (manual)</th>
<th>Very good graphic, navigation, synchronization control (manual)</th>
<th>Interface neutral, proactivity through triggers</th>
<th>Canned interfaces</th>
<th>Good graphic control and coherence (manual and with templates)</th>
<th>Predefined interfaces, low graphic control</th>
</tr>
</thead>
</table>

Fig. 15. Synopsis of the Different Categories of Web Development Tools
most products limit their focus to implementation, with some provision for design (as represented by the shorter “light cone” in figure 16); a few tools are trying to cover the lifecycle in a broader way, but do so by approaching development from an unnatural angle, typically using models, abstractions, and processes drawn from other contexts. The “slanted” approach of these tools and approaches is represented by the light cone covering all the phases of development, but having an inclination due to its origin from the database area.  

12. RESEARCH PERSPECTIVES

Web application development has recently gained much attention not only in the commercial field but also in the research community, where several projects are addressing the extension of the capabilities of site design tools in various directions and proposing innovative design processes and methodologies.

In this Section we briefly review a number of projects which have proposed solutions for overcoming some of the limitations currently experienced by state-of-the-practice commercial tools: Araneus [Atzeni et al. 1997; Atzeni et al. 1998; Atzeni et al. 1998], Autoweb [Fraternali and Paolini 1998], Strudel [Fernandez et al. 1998], Web Architect [Takahashi and Liang 1997], and W3I3 [Ceri et al. 1998].

The common denominator of such efforts is the goal of supporting the activities of data-intensive Web design from conceptualization to maintenance, by properly distinguishing between the different dimensions of Web design, organizing the development activities into a structured process, and providing tools for partially automating repetitive development tasks.

However, beside such commonalities, each project has its special focus and best addresses only some of the issues left open in the currently available commercial tools.

---

Fig. 16. A Pictorial View of the State-of-the-Practice of Web Development Tools

---

4A similar phenomenon took place in the early days of object-orientation, when many proposals were put forth to adapt Structured Analysis / Structured Design concepts to object oriented implementation.
<table>
<thead>
<tr>
<th></th>
<th>Araneus</th>
<th>AutoWeb</th>
<th>Strudel</th>
<th>Web Architect</th>
<th>W3i3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lifecycle Coverage</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conceptualization</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Logical design</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Prototyping</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Implementation</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Restructuring</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Rev. engineering</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Process Automation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generation of supporting</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>database</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generation of mapping to</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>legacy databases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generation of navigation</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>structures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generation of HTML pages</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>from templates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generation of HTML pages</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>from abstract specs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Modelling Abstractions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structure model</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Derivation model</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Navigation model</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Page composition model</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Presentation model</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Reuse &amp; Components</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plug-in components</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Reusable presentation styles</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Application skeletons</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Default Architecture</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two tiers static</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Three tiers static</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Three tiers dynamic</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Automatic caching</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td><strong>Support to Usability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Navigation uniformity</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Presentation uniformity</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Usability guidelines</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

Fig. 17. Synopsis of the Reviewed Research Projects in Data Intensive Web Development

In Table 17 we summarize the features of the various projects using the same categories as for commercial tools (see Table 15), and in Table 18 we underline the special focus and strong points of each project.

For completeness, we conclude the overview of research efforts with a presentation of the background research on Web development, which has taken advantage of contributions from several fields, among which hypermedia and databases are prominent.
<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Araneus</strong></td>
<td>Querying and restructuring of HTML-based sites</td>
</tr>
<tr>
<td></td>
<td>Full design methodology</td>
</tr>
<tr>
<td></td>
<td>Orthogonal structure, navigation, composition and presentation modeling</td>
</tr>
<tr>
<td></td>
<td>Tool support</td>
</tr>
<tr>
<td><strong>Autoweb</strong></td>
<td>Full design methodology</td>
</tr>
<tr>
<td></td>
<td>Orthogonal structure, navigation, and presentation modeling</td>
</tr>
<tr>
<td></td>
<td>Relational representation of both data and meta-data</td>
</tr>
<tr>
<td></td>
<td>Full CASE support</td>
</tr>
<tr>
<td></td>
<td>Tool-supported usability guidelines</td>
</tr>
<tr>
<td><strong>Strudel</strong></td>
<td>Querying and restructuring of semistructured data</td>
</tr>
<tr>
<td></td>
<td>Declarative site definition</td>
</tr>
<tr>
<td><strong>WebArchitect</strong></td>
<td>Full design methodology</td>
</tr>
<tr>
<td></td>
<td>Scenario-based modeling</td>
</tr>
<tr>
<td></td>
<td>Role-based specification of content</td>
</tr>
<tr>
<td></td>
<td>Tool support</td>
</tr>
<tr>
<td><strong>W3I</strong></td>
<td>Full design methodology</td>
</tr>
<tr>
<td></td>
<td>Orthogonal structure, derivation, navigation, composition and presentation modeling</td>
</tr>
<tr>
<td></td>
<td>User modeling and profiling</td>
</tr>
<tr>
<td></td>
<td>Adaptive behavior through Web business rules</td>
</tr>
</tbody>
</table>

Fig. 18. Advanced Features of the Reviewed Projects in Data Intensive Web Development

13. RESEARCH PROJECTS IN DATA INTENSIVE WEB DEVELOPMENT

13.1 Araneus

Araneus⁵ [Atzeni et al. 1997; Atzeni et al. 1998; Atzeni et al. 1998] is a project of Università di Roma Tre which focuses on the definition and prototype implementation of an environment for managing unstructured and structured Web content in an integrated way, called Web Base Management System (WBMS). The WBMS should allow designers to effectively deploy large Web sites, integrate structured and semi-structured data, reorganize legacy Web sites, and Web-enable existing database applications.

On the modeling side, Araneus stresses the distinction between data structure, navigation, and presentation. In structure modeling a further distinction is made between database and hypertext structure: the former is specified using the Entity-Relationship model, the latter using a notation that integrates structure and navigation specification, called Navigation Conceptual Model (NCM).

Conceptual modelling is followed by logical design, using the relational model for the structural part, and the Araneus Data Model (ADM) for navigation and page composition. ADM offers the notion of page scheme, a language-independent page description notation based on such elements as attributes, lists, link anchors, and forms. The use of ADM introduces page composition as an independent modeling task: the specification of data and page structure is orthogonal and therefore different page schemes can be built for the same data.

Presentation is specified orthogonally to data definition and page composition,

---

⁵The project Web site is http://poincare.inf.uniroma3.it:8080/Araneus/araneus.html
using an HTML template approach.

The development process follows two tracks: database and hypertext. Database design and implementation are conducted in the usual way using the Entity-Relationship Model and mapping it into relational structures. After that, the Entity-Relationship schema is transformed into a NCM schema; this shift requires several design activities. The next step, hypertext logical design, maps the NCM schema into several page-schemes written in ADM. Finally, implementation requires writing page-schemes as templates in the Penelope language [Atzeni et al. 1997], which specifies how physical pages are constructed from logical page schemes and content stored in a database, in a way similar to commercial template-based HTML-SQL integrators.

Araneus includes several tools to support the automation of the abovementioned design tasks. A specific point to the Araneus project is the integration into the WBMS of languages and tools for querying and restructuring HTML data, so that the designer can deploy a new site, Web-enable a database application, and “reverse-engineer” a legacy HTML site, all within one system.

From the architecture viewpoint, Araneus offers both static and dynamic page generation.

Currently, the system does not support user profiling and personalization. Finally, no support is offered for event-based reactive processing, except for the recomputation of materialized pages following database and schema updates.

13.2 Autoweb

Autoweb6 [Fraternali and Paolini 1998] is a project developed at Politecnico di Milano with the goal of applying a model-driven development process to the construction and maintenance of data intensive Web sites.

Autoweb consists of three ingredients:

—A Web modeling notation called HDM-lite, which is an evolution of previous hypermedia and database conceptual models, specifically tailored to the Web.

—Two transformation techniques, which address the mapping of conceptual schemas into relational database structures, and the production of application pages (in HTML and Java) from data and metadata stored in the database.

—A set of design-time and run-time CASE tools that completely automate the design, implementation, and maintenance of a Web application.

The Autoweb conceptual model, called HDM-lite, includes primitives for the independent specification of structure, navigation and presentation. Differently from other projects, presentation is specified at an abstract level totally independent of the implementation language, and automatically mapped to HTML (a prototype implementation of the mapping to Java has also been developed). This makes presentation styles reusable across applications, as well as across different object types within the same site.

HDM-lite presently neither supports the orthogonal composition of the page, whose content is inferred from the structure schema, nor a language for data derivation. An original feature of Autoweb is the storage into a relational DBMS not only

---

6The project Web site is http://www.ing.unico.it/autoweb
of application data, but also of metadata about the navigation and presentation schemas, which greatly enhances the possibility of quickly adapting the output pages to the user’s needs, even at run-time.

The main focus of Autoweb is the automation of the development process. An Autoweb application is constructed starting from an HDM-lite schema, with a tool called Visual HDM Diagram Editor; presentation specification is assisted by the Visual Style Sheet Editor tool, which permits the designer to define presentation styles applicable to conceptual objects in a WYSIWYG manner. The conceptual model is automatically translated into the schema of a relational database for storing application data, and into a metaschema database containing a relational representation of the site’s structure, navigation and presentation. The site is populated either by using an automatically constructed data entry application, produced by the Autoweb Data Entry Generator, or by manually defining a mapping between the automatically generated relational schema and the schema of the pre-existing database. As the last step, application pages are dynamically constructed from database content and metadata by the Autoweb Page Generator, in such a way that all the prescriptions of the conceptual model are enforced.

During such development process, reuse happens at two levels: partially instantiated application skeletons and abstract presentation specifications.

Autoweb has a three-tiers architecture, featuring both static and dynamic page generation. Dynamic generation is optimized by a flexible caching systems, that caches and refreshes pages of selected types based on user’s requests.

A unique feature of Autoweb is the provision for interface design guidelines within the design tools: uniformity of navigation and presentation is enforced by the design and page generation tools, at the level of both individual object types and of the entire application.

13.3 Strudel

Strudel is a project of AT&T Labs [Fernandez et al. 1998], which aims at experimenting a novel way of developing Web sites based on the declarative specification of the site’s structure and content.

The core idea of Strudel is that of describing both the schema and the content of a site by means of a set of queries over a data model for semi-structured information.

Content is represented using the Uniform Graph Model, a graph-based data model capable of describing objects with partial or missing schema. As a starting point of the construction of a site, external data sources, e.g., HTML files or relational databases, are translated by means of wrappers into the Strudel internal format. In this way, it is possible either to restructure an existing HTML site or Web-enable a legacy data repository.

Then, the design of a Web site requires writing one or more queries over the internal representation of data, using the Strudel query language (StruQL). Such queries permit the designer to select the data to be included in the site, and the links and collections of objects to be provided for navigation. In this way, Strudel separates the description of content from the definition of the structure and navigation of the site. Presentation is added as a separate dimension by means of HTML

7The project Web site is http://www.research.att.com/sw/tools/strudel
templates; these mix HTML presentation tags and special-purpose tags, which are bound at HTML generation time to the objects resulting from the site definition queries. The templates determine the rendering of the site definition queries in HTML.

The specification of navigation is somewhat intertwined to structure and presentation, because navigable links and index collections are specified together with the queries that define the site, and because the structure of HTML pages and their links depend on the templates that describe the presentation.

Presently, Strudel has a two-tiers static architecture, in which queries are evaluated and transformed into HTML pages in advance. However, it could be possible to evaluate queries and render their result dynamically.

The declarative definition of structure and content by means of queries opens the way to personalisation: different sites or different versions of the same site can be built on top of the same content simply by changing the StruQL site definition queries.

13.4 Web Architect

WebArchitect [Takahashi and Liang 1997] is a project aimed at developing methods and tools for the construction of Web-Based Information Systems (WBIS). The authors propose a structured design process that goes through the analysis, design, construction and maintenance of a Web site.

Analysis includes both static and dynamic modeling; the former is conducted with the Entity Relationship Model, the latter requires the identification of scenarios, in the tradition of object-oriented modelling [Jacobson 1994]. During ER modeling, entities are classified according to the different role they play in the definition of the site (agent, product, or event). Design is conducted in parallel to scenario analysis and aims at pinning down the structure and navigation schema of the Web site. Design results are represented using a variant of the Relation Management Data Model by Iskowitz [Diaz et al. 1995], which incorporates the roles of the entities forming the Web site.

WBIS implementation and maintenance are supported by WebArchitect and PilotBoat. The former tool supports the definition of the structure and navigation of the site, as well as the maintenance of metadata about the site’s resources; the latter is a client application permitting the user to browse an application based on meta-links implementing the navigation semantics specified in WebArchitect. Meta-links are navigable connections stored outside the application objects, which are managed by extended HTTP engines supporting the special-purpose methods LINK and UNLINK.

13.5 W3i3

W3i3 (WWW Intelligent Information Infrastructure) [Ceri et al. 1998] is a project of the W3i3 Consortium, a partnership of four European industries and one Academic institution, funded by the European Community. The goal of W3i3 is to advance Web modeling and development tools for data intensive Web applications, with a special focus on user profiling, personalization, and reactive behavior.

---

8 The project Web site is http://w3i3.polimi.it
W3I3 modeling distinguishes the five perspectives of structure, derivation, navigation, page composition, and presentation, and includes models and languages for specifying a site under these perspectives.

The development process stresses the automatic generation of application pages from the conceptual model stored in a relational database, and content stored in external structured or semi-structured data repositories.

The architecture is multi-tier: application content can be distributed across different data repositories which are integrated into a global view of the site obtained by mapping the conceptual schema into a relational representation.

A special feature of W3I3 is the integration of user modeling and business rules: users are explicitly modeled through demographic and psychographic variables and business rules are used to map users or user groups to personal views of the site computed dynamically.

Presently, W3I3 is in the implementation phase; a prototype version of both the design tools and the runtime environment supporting site personalization and Web business rule has been constructed using the Autoweb core technology.

14. BACKGROUND RESEARCH

Web design tools and approaches owe much to the debate on hypermedia modelling and design, semistructured data modelling, and hypermedia development tools proposed even prior to the advent of the Web. In the following Sections, we review some of the most important contributions.

14.1 Modeling Notations

Historically, most of the modeling notations adopted by current Web development methodologies stem from the evolution and hybridation of previous conceptual models for database and Hypermedia design.

The common ancestors of many subsequent proposals are the Entity Relationship Model [Chen 1976], in the database field, and the Dexter Model [F.G. Halasz and M. Schwarz 1994], in the Hypermedia area.

The Dexter Model originates from the effort of providing a uniform terminology for representing the different hypertext structuring primitives offered by hypertext construction systems; the core of the model is the ordered representation of a hypertextual application at three levels: the storage, within-component, and runtime level. The storage level describes the network of nodes and links of the hypertext, without details on the inner structure and content of nodes, which is the focus of the within-component layer. The runtime level deals with the dynamics and presentation of the hypertext.

The modeling concepts available at the storage level are very basic: components describe the pieces of information that constitute the hypertext, and can be either atomic or composite, and links are a special kind of component, used to represent navigable paths. Although not conceived for hypertext design, the Dexter Model advocated many concepts, like the distinction between the structure, navigation, and presentation of a hypertext, whose influence has been long-lasting.

Several subsequent contributions started from a criticism of the Dexter Model, and added more complex forms of hypertext organization and more powerful navigation primitives [Garzotto et al. 1993], time and multimedia synchronization features.
Hardman et al. 1994, formal semantics of navigation, and a structured design process [Isakowitz et al. 1995]. Among these evolutions, HDM [Garzotto et al. 1993] and RMM [Isakowitz et al. 1995] have been particularly influential on the design of Hypermedia applications.

HDM and its variants [Schwabe et al. 1992; Garzotto et al. 1993; F. Garzotto et al. 1991; F. Garzotto et al. 1993] shifted the focus from hypertext data models as a means to capture the structuring primitives of hypertext systems, to hyper-text models as a means for capturing the semantics of a Hypermedia application domain. HDM integrates features of the Entity Relationship Model and of the Dexter Model, to obtain a notation for expressing the main abstractions of a Hypermedia application, their internal structure and navigation, and application-wide navigation requirements. Web structure is expressed by means of entities, sub-structured into a tree of components. Navigation can be internal to entities (along part-of links), cross-entity (along generalized links), or non-contextual (using access indexes, called collections [Garzotto et al. 1994]).

RMM (Relationship Management Methodology) [Isakowitz et al. 1995] evolves HDM by embedding its Hypermedia design concepts into a structured methodology, splitting the development process into seven distinct steps and giving guidelines for the involved tasks. RMM's data model (called RMDM) structures domain entities into slices, and organizes navigation within and across entities using associative relationships and structural links.

Most of the recent proposals for Web modeling are built on top of the Entity Relationship Model and of Hypermedia design models (notably HDM and RMM), adapted to the specificity of the Web context: Araneus and WebArchitect draw from RMM and the Entity-Relationship model, Autoweb and W3I3 have proposed a Web-specific evolution of concepts first proposed by HDM.

Finally, another source of inspiration to Web modeling comes from the research on the representation and querying of semi-structured data, i.e., data with partial or missing schema. The proposed data models, thoroughly reviewed in [Florescu et al. 1998], express Web content by means of relations, labeled graphs, hypertrees and logic. Araneus and Strudel are examples of Web content management systems based on semi-structured data models and query languages.

14.2. Processes

Web development processes have evolved in parallel to Web design notations and have the same hybrid origin from the Information System and Hypermedia fields.

In Hypermedia, the evolution from the creative definition of content to the content-independent organization of the structure of a Hypermedia application is attributed to the work on HDM [F. Garzotto et al. 1993], which stresses the difference between authoring in the large, i.e., designing general structure and navigation, and authoring in the small, i.e., deciding the layout and synchronization aspects of specific component types.

HDM, however, did not prescribe a formal development lifecycle, which was first advocated by RMM [Isakowitz et al. 1995], where the seven activities of Entity Relationship design, slice design, navigation design, conversion protocol design, interface design, behavior design, and implementation and testing are proposed as a standard process. The first three activities provide a conceptualization of the
Hypermedia application domain in terms of entities, substructured into slices, and navigable relationships. Conversion protocol design is a technical activity which defines the transformations to be used for mapping the conceptual schema into implementation structures. In addition to defining the development lifecycle, RMM also gives guidelines for slice and navigation design, the two tasks most particular to Hypermedia design.

**OOHDM** [Schwabe and Rossi 1995] takes inspiration from object-oriented modeling and simplifies the RMM lifecycle to only four steps: *domain analysis*, *navigation design*, *abstract interface design*, and *implementation*. In domain design, classical object-oriented techniques are used, instead of the Entity Relationship Model. Navigation design adds specific classes (e.g., node, link, index) to represent different forms of navigation. The same is done for presentation, which is described by means of classes (e.g., button, text field) added during interface design. Implementation then fleshes out the classes identified during design with code in the implementation language of choice.

In the Web context, most methodological proposals concentrate on visual design and usability criteria [Sano 1996], much as in the Hypermedia field authoring guidelines were the first concern of development. The requirement of application scale-up drives the most recent contributions, like Araneus, Autoweb, Web Architect, and W3I, which organize the development process into activities drawn both from the above mentioned Hypermedia methodologies and from traditional object-oriented and database design methods.

### 14.3 Other Design Tools

Beside the projects described in Section 13, other research efforts have conducted to the development of prototype of hypermedia and Web design tools.

**RMC** [Diaz et al. 1995] gives CASE support to the design and implementation phases of the RMM methodology. It consists of a diagram editor, which assists the input of RMDM schemas, and of a code generator, which outputs HTML pages from data stored in an internal repository. Pages are produced offline in a pre-compiled fashion. RMC is not based on a database architecture and thus does not provide facilities for scaling-up and updating the information base.

**HSDL** [Kesseler 1995] has an architecture similar to RMC, but starts from a HDM-like design model, augmented to support a fine-grain personalization of navigation and presentation semantics. HSDL objects are annotated by means of programs in a scripting language, called *expanders*, which drive the production of HTML pages. Expanders can be attached both to schema elements and to instances, to provide uniformity and exception handling at the same time. Expander programming, although powerful, is a low-level task which resembles the programming of page templates in Web-DBPL integrators. The difference is that HSDL, like RMC, does not rely on database technology to store the application content, but has an internal repository.

### 15. CONCLUSIONS

Although the present situation of the market of Web development tools is characterized by a substantial overlap between the different classes of products (despite some difference in the level of maturity), an attempt can be done to identify a
<table>
<thead>
<tr>
<th>Category</th>
<th>Visual editors</th>
<th>Hypermedia tools</th>
<th>Web-DBPL integrators</th>
<th>Form editors</th>
<th>Multi-paradigm tools</th>
<th>Model-driven generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Scale Business to Customer</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business to Business</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Large Scale Business to Customer</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 19. A Match Between Categories of Web Development Tools and Types of Applications

plausible choice based on the application requirements. Figure 19 summarizes the adequacy of each product category to a specific kind of application:

—Small scale business-to-user applications: this category includes such applications as companies' points of presence, training and education, infocenters, and so on. Last-generation visual editors and site managers (category 1) seem the appropriate solution, because they ensure the high level of visual accuracy and customization necessary for application targeted to the general public, coupled to productivity tools substantially reducing the development effort. This role could be undermined in the near future by Hypermedia authoring tools (category 2), which share the same focus on presentation quality and are even more effective in the design of navigational and multimedia interfaces. The current limit to the applicability of both these classes of solutions is the size and volatility of the information base; if this has to be kept in a database, then presently these tools do not provide the adequate means to integrate databases and the Web for design, implementation and maintenance. In this case, multi-paradigm tools (category 5) may be a better choice.

—Intra-business or business-to-business applications: this category comprises all legacy information systems and EDI applications and is characterized by a different kind of users, already trained to the transactional and form-based interaction paradigm. In the present situation, Web form editors and database publishing wizard (category 4) and model-driven generators (category 6) offer a powerful opportunity for migrating existing applications to Intranets, and this technical advantage largely balances the limited exploitation of the communication capabilities of the Web. However, novel intra- and inter-business applications are emerging (for example, Hypermedia for technical documentation and computer based training) which demand the integration of large masses of data, hypertextual multimedia interfaces, and deployment on the Web. The gradual introduction of these applications may promote the evolution of the interaction paradigm for conventional information systems as well.

—Large scale business-to-user applications: this is the most challenging area, comprising such applications as electronic commerce, virtual libraries, and all sorts of Internet services. Presently, it seems that no specific product or class of products is fully addressing the analysis, design, implementation and evolution of this
kind of applications, which require the same communication paradigm and interface quality as small scale user-oriented applications and the same performance and scalability as client-server database applications. In this scenario, neutral, implementation-oriented products like Web-DBPL integrators (category 3) and flexible, multi-paradigm tools (category 5) seem the most adequate choice, although development and maintenance with these tools still require a substantial coding effort. We are convinced that the best of these tools can be obtained by using them in conjunction with a model-driven development approach, based on design notations and processes like the ones described in the previous Section; after the supporting database has been designed, visual page designers and Web-DBPL integrators can be used to generate the application’s physical pages.

As several research projects are demonstrating, large scale Web applications have more dimensions than the mere structure of the underlying database, which prompt for a refocusing of the existing conceptual models and software development processes, to achieve the level of maturity of a consolidated software field.

REFERENCES


APPENDIX

List of URLs of Reviewed Products (alphabetic order)

42. Visual Basic 5.0, Microsoft, http://www.microsoft.com/vbasic