Some Trends in Web Application Development

Mehdi Jazayeri

Mehdi Jazayeri is the founding dean of the faculty of informatics and professor of computer science at the University of Lugano, Switzerland. He also holds the chair of distributed systems at the Technical University of Vienna. He spent many years in software research and development at several Silicon Valley companies, including ten years at Hewlett-Packard Laboratories in Palo Alto, California. His recent work has been concerned with component-based software engineering of distributed systems, particularly Web-based systems. He is a coauthor of Programming Language Concepts, (John Wiley, 3rd edition, 1998), Fundamentals of Software Engineering, (Prentice-Hall, 2nd edition, 2002), and Software Architecture for Product Families (Addison-Wesley, 2000). He is a Fellow of the IEEE.

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

Some Trends in Web Application Development

Mehdi Jazayeri
Faculty of Informatics
University of Lugano (USI)
Lugano, Switzerland
www.inf.unisi.ch

Faculty of Informatics
Technical University of Vienna
Vienna, Austria
www.infosys.tuwien.ac.at

Abstract

A Web application is an application that is invoked with a Web browser over the Internet. Ever since 1994 when the Internet became available to the public and especially in 1995 when the World Wide Web put a usable face on the Internet, the Internet has become a platform of choice for a large number of ever-more sophisticated and innovative Web applications. In just one decade, the Web has evolved from being a repository of pages used primarily for accessing static, mostly scientific, information to a powerful platform for application development and deployment. New Web technologies, languages, and methodologies make it possible to create dynamic applications that represent a new model of cooperation and collaboration among large numbers of users. Web application development has been quick to adopt software engineering techniques of component orientation and standard components. For example, search, syndication, and tagging have become standard components of a new generation of collaborative applications and processes.

Future developments in Web applications will be driven by advances in browser technology, Web internet infrastructure, protocol standards, software engineering methods, and application trends.

1. Introduction

The World Wide Web was introduced in the early 1990s with the goal of making it possible to access information from any source in a consistent and simple way. Developed at CERN, in Geneva, Switzerland, it was aimed at physicists and other scientists that generate huge amounts of data and documents and need to share them with other scientists. Hypertext was adopted as a simple way to both give access to documents and to link them together. The HTTP protocol was designed to allow one computer—the client computer—to request data and documents from another computer—the server computer—so that it could make that document available to the users on the client computer. In this way, the World Wide Web was viewed as a vast repository of information that provided access to a large number of users. This view of the Web was quite static and it has changed considerably over time. A first key observation was that the address that was considered to be a page of data on the server could in fact refer to a program that could be executed on the server and its results returned to the client. Today, the address could indeed refer to a sophisticated (Web) application being invoked. Currently, the Web is a powerful platform offering a vast array of tools and components to application developers. A new generation of applications offers users the opportunities to communicate, collaborate, and even update the capabilities of the application. Applications support individuals, small businesses or communities of users as well as large company businesses.

In a survey that captured the state of the art in Web application development in 1999, Fraternali described a Web application as a hybrid between a hypermedia and an information system. As a consequence, he stated the following requirements for Web applications[9]:

- the necessity of handling both structured (e.g. database records) and non-structured data (e.g. multimedia items);
- the support of exploratory access through navigational

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

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, i.e., for recommendation and filtering.

Since that survey was written, the Web has continued to evolve at a rapid pace and new challenges have surfaced. There are many ideas about what trends will dominate in the future. One all-encompassing concept, by no means universally accepted, is to consider the emerging trends as forming the basis of "Web 2.0". Compared to traditional Web applications—those of Web 1.0—Web 2.0 applications are dynamic, they invite user participation, and are as responsive to user requests as desktop applications. The idea of Web 2.0 is formulated comprehensively by Tim O'Reilley[19].

The metaphor of Web 2.0 is useful as a way of capturing the emerging trends in Web application development. Some example applications demonstrate the differences between Web 2.0 and earlier models of applications. For example, let us contrast Wikipedia with Encyclopedia Britannica. The former engages the help of users to create data and the latter draws a sharp distinction between producers and consumers of the data. As a second example, consider the old model of publishing Web sites to the current practice of blogging, in which the readers are invited to participate in a discussion. As a third example, we can regard the use of search engines for finding Web sites as opposed to the earlier approaches of remembering, guessing, or bookmarking Web site addresses. Search engines allow a more dynamic way to find Web sites.

What is Web 2.0, how did we get there, and what will come next (Web 3.0)? We provide some answers to these questions in this paper from a software engineer's point of view.

From the developments in software engineering, agile methods for processes and service orientation for architectures are particularly suited to Web application development and have been adopted rapidly. In this paper, we draw the parallels between traditional software engineering and Web application development. We review the progression of approaches and technologies for Web application development, review current trends, and offer a tentative forecast of future directions.

2. Foundations of the Web

Despite the enormous developments over the last decade, the fundamental principles upon which the World Wide Web was based have remained constant. Structurally, the World Wide Web is based on client-server computing, in which servers store documents and clients access documents. The same computer may act as a client and as a server at different times. The World Wide Web introduced three fundamental concepts on top of client-server computing: a method of naming and referring to documents (URL), a language for writing documents that can contain data and links to other documents (HTML), and a protocol for client and server machines to communicate with each other (HTTP).

URL A naming system is a fundamental component of computer systems, especially so for distributed systems. A naming system prescribes the way objects are named so that the objects can be identified and located. Depending on the characteristics of the naming system, objects may be searched for on the basis of their exact names only or on the basis of their attributes. For example, one might want to tell the system to "fetch the paper written by Alan Turing about intelligence test." The World Wide Web's naming scheme had the goal of uniquely identifying all objects stored on the computers on the Internet. The naming scheme is based on Uniform Resource Locators (URLs) which are composite names identifying the computer (IP address), the document in the file system of that computer, and a protocol with which to communicate with that object. URLs are now defined as a standard in IETF RFC 1630.

HTML The documents on the Web are written in the HyperText Markup Language. HTML documents contain content to be displayed, formating instructions that tell the browser how to display the contents of the document, and links to other documents. HTML has evolved along with browsers to achieve better visual presentations and standardization.

Initially, HTML was viewed as a language for instructing browsers what to display for humans. But as the number of documents written in HTML has grown, and as many applications started to generate HTML documents, computer processing of HTML documents became important. The extended markup language XML was created to standardize the definition of other specialized markup languages. XHTML is an XML compliant HTML which has become the dominant variant of HTML.

Currently, the Web Hypertext Application Technology Working Group (www.whatwg.org) is working on defining an evolutionary path for HTML and reconciling the discrepancies between XHTML and HTML. Other groups such as W3C are working on XHTML as a standard.

HTTP The communication protocol for the Web is the HTTP (See IETF RFC 2616) protocol. HTTP is a simple request-reply protocol. It defines eight basic operations: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, and CONNECT. The most used of these operations or methods are GET and POST. The method GET retrieves from a given URL the data associated with the requested URL. The method POST sends data to the program listening at the specified URL.

These simple concepts have proven to be surprisingly powerful. Application developers have found ingenious ways of using URLs to name a variety of things and not only documents. For example, one of the early ideas was to use the URL to name a program to execute on the server which would then produce output that would be returned to the client. Likewise, documents are used not only to contain information to be displayed by the browser but to contain code scripts to be executed (either in the browser or on the server). A whole array of languages have been created to write code to be executed by the browser (e.g. JavaScript) or on the server (e.g. PHP). Web applications take advantage of a variety of languages and design patterns to combine the capabilities of clients and servers.

3. Software Engineering and Web Applications

Software engineering was born in 1968 as a vision of taming the software development difficulties encountered in the 1960s. The early large software development projects were running into trouble, overspending their budgets and overextending their development schedules. Many projects were being abandoned because regardless of the amount of added resources, they were not making progress. It was becoming clear that large monolithic software, which was the only kind of software being built, had many limitations. Such software could not be extended beyond a certain size, could not be maintained, and therefore rarely met customer expectations, assuming it could be delivered in the first place. Software engineering promised a systematic approach to software construction based on modular designs, standard software components, and defined processes. This vision of software engineering is becoming more real as the years go by.

Web application development followed a history similar to software engineering but at a much more rapid pace. Early Web applications consisted of many scripts scattered over many files and lacked a systematic structure. Content, formatting instructions, processing instructions (referred to as *application* or *business logic*) were mixed together in the same file. Perhaps because Web applications are naturally distributed, however, they accepted and incorporated the concept of individual components that can be composed to build applications much more easily. Web applications commonly use standard components such as payment or registration modules. In this section, we look at several aspects of Web applications that are closely related to software engineering issues.

3.1. Static to dynamic

The early Web consisted of a set of documents which could freely link to each other. These were simple text files, which had static content and static links connecting to pages. Very early on, software engineers realized that the client-server architecture of the Web provided a powerful platform in which the browser could be a universal user-interface to applications that may run locally or remotely on a server. To maintain the browser-sever relationships, the server always returns a Web page to the browser, but this Web page could be generated programmatically as the result of processing on the server. For example, the server could retrieve data from a database, format them into an HTML page, and send the page to the client.

The idea of processing on the server and dynamically generated pages led to CGI (common gateway interface) applications, where the URL provided by the client referred to a "script" on the server. The script could be written in some specialized interpretive language such as PERL or even large compiled programs that would be run on the server to dynamically generate Web pages. This became cumbersome for many applications, because putting all of the content and formatting for a Web site into the code of an application, as software engineers know well, can become tedious and hard to manage. This realization led to a sort of hybrid model, where the language PHP was an early influence. With PHP, the typical structure is to create Web pages that have small bits of code directly in the text of the pages. At the time of a request, the code would be executed and its results inserted into the current document. This led to much more flexibility and easier reuse of page components, but over time people realized that it was often hard to make changes to code when it was spread out over all of the pages in a Web site. This led to a further refinement of the model, into what is typical in today's Web applications, where a central component in the application manages access to data, while the code which is dispersed in the HTML pages is limited to just what is displayed to the user (See next subsection).

The typical structure of a Web application consists of HTML data displayed to the user, client-side scripts that run on the client and may interact with the user, and server-side scripts that perform processing on the server and typically interact with databases. ECMAScript is the standards specification of an object-based scripting language intended

for client-side scripting (ECMA-262 and the equivalent ISO/IEC 16262). JavaScript (from Netscape) and JScript (from Microsoft) are implementations of ECMAScript. There are many languages for server-side scripts, including PERL, PHP, Python, and Ruby. Client-side Web scripts are embedded in HTML pages, run in the browser environment, and may provide interaction with the user. The objects in the script refer to user-interface entities such as windows and menus, or cookies. Scripts also react to user events such as mouse movements and clicks. Server-side scripts deal with other types of objects such as clients and files. The combination of client and server scripts provides a distributed computation that helps to build customised user interfaces for Web applications.

There are a number of frameworks for generating Web applications based on Web-based languages. We will discuss these in the next subsection.

3.2. Design and development of Web applications

From the software engineering point of view, the concerns of many Web applications are similar: they have a user interface, based in the browser, that interacts with the user, and they manage a possibly large amount of data, stored on the server, on behalf of the clients. It took a number of years before Web developers realized that the standard MVC pattern, well-known in software engineering, applies just as well to such Web applications. Once this realization was made, a number of frameworks have been developed to support the design and development of Web applications on the basis of the Model-View-Controller [20]. In the context of Web applications, the model is typically a set of object oriented classes to interact with the data store (usually a database), the controller contains the logic of the application or processes, and the view is a script which generates an editor (in our case, HTML pages). The controller is also responsible for preparing the data from the model for the view to be created. The controller creates the views and responds to queries made from the views.

A plethora of component-based Web engineering methodologies are now available [11, 18], supporting object-oriented composition [10], exploring aspect orientation and context-dependency [3], and searching for patterns using UML [1, 2]. Recently, more robust and agile frameworks have appeared in different programming languages, many of them based on the Model View Controller design pattern.

Ruby is a dynamic object-oriented programming language [23] that has become popular for writing Web applications. It supports the writing of complete programs and scripts that can be embedded in HTML files. It has an active community that has created many lightweight solutions. One of the best-known of these is Ruby On Rails[24], a framework for Web application development based on the model view controller pattern. To this, it adds a set of powerful functionalities such as scaffolding, active record, migrations, routing, environments, and many helper functions.

The *scaffolding* is a Ruby on Rails facility through which the developer uses scripts to generate the first version of an application. The scaffolding generates models, views and controllers as needed based on user-defined database schema. The framework supports an agile development methodology[4]. With the help of scaffolding, you start your application with a basic set of files which give you the essential operations on your model, namely: show, edit, create, update, and delete. These are colloquially referred to as CRUD (create, read, update, delete).

ActiveRecord is a library which allows the developer to interact with the backend database by managing only Ruby objects, making the development easier as the developer operates completely in the Ruby environment, without explicitly using the SQL(Structured Query Language) language. ActiveRecord is Ruby's way of providing object-relation mapping to the developer. Object-relational mapping is supported by many frameworks.

Migration is a way of managing the database schema by a set of Ruby classes. Changes in the database schema during software maintenance are automatically supported by migration.

The *routing* facilities of Rails maps a URL query to the desired controller and action to handle the query.

Rails provides three environments by default: development, testing, and production. These different working environments simplify the work on the same code at different stages of implementation in parallel. Further, deployment on different servers, with different databases and operating systems are specified once and handled automatically by the framework. Rails thus separates the application development and deployment processes explicitly.

Popular programming languages are now supported by their own Web application frameworks. J2EE is the Java Web application framework which evolved into a Model View Controller with the introduction of Struts. Struts is an open-source framework of the Apache Foundation. Its views system is implemented by Java Server Pages (JSP), controllers and models are typically Java Beans or Java Servlets.

Seaside is a powerful framework that extends the power, flexibility, and simplicity of Smalltalk to Web application development. Django is based on Python.

The common characteristic of these frameworks is that they support the model-view-controller, object-relational mapping to map databases to programming language objects, and generators for creating routine parts of the application.

The current generation of Web languages and frameworks reflects the frenetic pace of technological development in the area. New languages and frameworks are created and enhanced with new features to match newly available technologies or user requirements. In the traditional programming language world, there was a period of consolidation once every decade or so during which a new language would emerge that reflected the results of experience with different earlier languages. No such consolidation has occurred yet in the Web area.

3.3. Releasing a Web application

One of the reasons for the popularity of Web applications among developers is the ability to release a version without having to distribute and install the new version on client computers. The browser acts as a universal client and the application exists only in one copy on the server. Managing releases is a laborious and expensive part of traditional software engineering. The situation is dramatically different for Web applications.

In a desktop application, adding features and fixing bugs requires either a new version to be installed or a patch to be applied. This upgrade process is cumbersome and expensive. Users have to be notified of the availability of the update; they must apply the update; the update may interfere with other applications on the client, for example by installing an incompatible version of a shared library; the provider of the application never knows if all the clients have applied the update: at any given time, different clients may be running many different versions of the application. With Web applications, where the application exists in one copy only, all of these problems disappear. Features and bug fixes can be added to a running application by the developer as soon as they are ready, and instantly all users will have access to the upgrade. This allows developers to focus on improving the application rather than dealing with complex maintenance and upgrade issues, and it benefits application users because they get immediate access to the latest version of the software.

Web application development is therefore more open to agile methods because even small units of functionality may be made available to users instantly rather than having to be bundled with other functionality subject to an arbitrary release schedule. As opposed to desktop applications that may have release cycles of several months or even years, it is not unusual for Web applications to be updated several times a day. This move to centralized applications reverses the trend towards distributed computing that was motivated by the spread of personal computers in the 1980s. Clearly, some applications are better suited to being centrally provided as a service rather than being installed on every desktop computer on the planet. For example, as personal computer applications such as word processing have become more and more complex, for many users it is more productive to access a central server for these services. Today, you can manage your calendar, mail, to-do lists, photo albums, word processing, spreadsheets, and bookmarks with the help of Web applications. Once your data is stored on a central sever, sharing and collaboration becomes much easier to do. Another major problem of distributed processing, backing up of dispersed user data also becomes much simpler when the data is centrally located.

3.4. Deployment

Where do Web applications run? The server environment can be proprietary or open source. Web application development has been driven by a move towards open source and standardized components. This trend has spread also to the server environment where Web servers run. Considering that there are many small organizations, small companies and non-profit organizations, that run their own Web servers, there is both business reasons and technical reasons for the move to open source. The standard, bare-bones, Web server environment is commonly referred to as LAMP. Each of the four letters in LAMP stands for one component of the environment. The components are:

- Linux for the operating system;
- Apache as the Web server;
- MySQL as the database server;
- Perl or Python or PHP as the language.

More generally, LAMP is deployed in a three-tier architecture. The client is represented by a standard Web browser. The middle tier is an application server that itself contains several levels of software: operating system, which is usually Linux or FreeBSD; a Web server, which is usually Apache or Lighttpd; an interface software that implements a CGI, which is now usually FastCGI; and finally languages for implementing applications, which are now commonly PHP, PERL, Python, or Ruby. The database layer may host an open source database manager such as MySQL, PostgressSQL, or SQLLite.

This standard environment can be put together relatively easily and cheaply. It is sufficient for many types of Web applications. More sophisticated applications, however, rely on proprietary software platforms. In the Java world, much software is available including JDBC, EJB, Java Beans, and especially, J2EE, which is a sophisticated framework for developing transaction-oriented database applications.

4. Modern Web applications: examples

Early Web applications offered mostly textual user interfaces and limited interactivity. Today's Web applications offer rich interfaces, are interactive, and support collaboration among users. Here we examine several applications that represent the current generation of Web applications, sometimes collectively called Web 2.0.

4.1. Google docs

Google docs and spreadsheets is a recent service offered by Google that provides the traditional word processing and spreadsheet functionalities as a Web application. They are streamlined services that support the most often-used features and do not support many features that are offered by commercial word processors. The interface looks very much like a typical desktop application. The user does not have to press a submit button after every change (a hallmark of the first generation Web applications). The user's data is automatically saved in the background. You can even drag a piece of text in the window.

In addition to the usual word processing features, Google docs offers features that are associated with Web 2.0. A document may be shared with other users so that different people may collaborate on editing it. Naturally, the document may be searched using keywords. Additionally, documents may be tagged with terms the user chooses so that documents may also be searched for based on tags. Documents may be saved in a variety of formats (on the Google servers) and mailed to other users. Other users may be given read-only or read-write access to the document.

Tagging and collaboration are two features common to modern Web applications. (On-line word processing was initially offered by Writely, a company that Google acquired.)

4.2. Del.icio.us

Del.icio.us is a Web application that helps users manage and share their bookmarks. As the amount of information on the Web has grown, it has become more and more difficult to keep track of the information you find and want to remember for future reference. The bookmark feature in browsers was intended for this purpose but it has limited functionality. Del.icio.us lets users store bookmarks and tag those bookmarks with user-defined terms. The tags are therefore available to the user from anywhere on the Internet and they make it easier to search for bookmarks. Further, by sharing bookmarks and tags, users can help each other find related Web pages. The system can also suggest tags that other users have applied to the same document, thus giving the user ideas on how to classify a document.

Bookmarking sites such as Del.icio.us attempt to address the fundamental problem of the user's needs to master the enormous amount of unstructured data available on the Web. Bookmarks, hierarchically organized bookmarks, bookmarking sites, search engines, ant tagging are all different solutions to this problem.

4.3. Wikipedia

Wikipedia has become one of the most popular sites on the Internet. It is used by many as an authoritative source of information, from finding definitions of technical terms to explanations of current events. The key feature of Wikipedia is that its content is produced by users. Anyone can add or edit the information on Wikipedia. In contrast to a traditional printed or on-line encyclopedia that employs professional editors and writers to produce and structure and authenticate its content, Wikipedia relies on social structures to ensure the creation and correction of its content. The vast numbers of users of the Internet form a large pool of potential volunteers. The Wikipedia is updated constantly rather than following the multi-year release cycle of a traditional encyclopedia.

An innovative aspect of the Wikipedia application, considered a characteristic of Web 2.0 applications, is that it provides a platform for users to collaborate to create a valuable product. What makes Wikipedia valuable, its content, is indeed produced by the users themselves. This aspect creates what is called the network effect: The more users there are, the more useful the product becomes. Amazon already introduced early forms of user collaboration to enhance the product by encouraging users to provide book reviews.

Wikipedia is based on the conept of a wiki, described later.

4.4. Flickr

Flickr is a photo sharing site where users store their photos and tag them for future retrieval. Further, users may tag any of the photos on the site that are available publicly. Similar to Wikipedia, Flickr is a site that would have nothing without its users. As more and more users participate, the volume of content grows and tags allow photos to be found easily.

4.5. MySpace

MySpace is a site for social networking. A user registers and creates a profile detailing his or her or its characteristics (Profiles exist for animals and companies and products, presumably created by real humans). Each user's space is open to be visited by other users. Users seem to enjoy sharing

all kinds of information about themselves and to communicate and interact with other users. The basic thesis that makes MySpace work is that people like to interact with other people. MySpace provides a platform for social interaction, albeit in virtual space. Users have populated MySpace with a variety of multimedia documents including images and videos. The site constantly changes its appearance to maintain the interest of its users.

4.6. Blog Systems

According to Wikipedia, "A blog is a Web site where entries are made in journal style and displayed in a reverse chronological order." A fundamental feature of blogs is that it creates the ability of readers to interactively leave comments for others to see and comment on. Blogs (Web logs) first appeared on the Blogger.com system. A blog is a Web site managed by the user; content is added by "posting." These posts are often organized in categories and can be commented on by other users. The traffic on blogs is intense and bloggers often cite other blogs within their posts. Blogs therefore have very high link density. As of 2006, there exist over 60 million blogs on the Web.

Closely related to blogs is the concept of dynamic data feeds. Recently, the very primitive syndication scheme of RSS, which requires polling Web sites for updated XML content, has taken off among consumers. Many blogs use RSS to notify their readers of changes in the blog. Aggregator applications merge different RSS feeds to produce sites with richer content. One area of society that has been affected by blogs is news publishing. Blogs as a phenomenon has changed the traditional form of news delivery. Blogs offer a different paradigm than the traditional printed newspapers. They enable a new model for society to access up to date information about what is going on in the world, albeit without the newspaper's editorial process.

4.7. Wiki Systems

Wiki systems are a form of content management system that enable a repository of information that may be updated easily by its users. Wiki systems such as wikipedia.org are similar to blogs in principle as they are based on user participation to add content. The fundamental element of wikis is pages as in typical Web sites, as opposed to blogs in which basic elements are posts (which can be displayed together within the same pages). Wikis allow users not only to read but also to update the content of the pages. The underlying assumption is that over time the wiki will represent the consensus knowledge (or at least the opinions) of all the users. As blogs, wikis exhibit high link density. In addition, wikis have high linking within the same wiki as they provide a simple syntax for the user to link to pages, both to existing pages and to those yet to be created. Many wikis also provide authentication and versioning to restrict editing by users and to be able to recover the history.

4.8. Key components of modern Web applications

Studying these emerging applications, some features stand out as key common principles.

- Search
- Tagging
- User participation
- User interaction and collaboration

Indeed, these features have become standard components in modern Web applications. Searching and tagging are mechanisms to help users find their way through the mountain of information on the Web. Tagging helps structure the data so that searching can become more personalized and customized. User participation is used both to create and to structure the data. Finally, enabling user interaction and collaboration is the final goal of many tools. Another common component that supports collaboration and interaction is a buddy system which alerts users when their social contacts are on-line in real-time. This feature first appeared in instant-messaging systems and is now present in such applications as Google Mail and Skype.

We expect to see these features to become the abstractions provided by a new kind of Web-oriented middleware. The common thread in all these applications is the importance of content/data. A Web 2.0 motto is that the distinguishing characteristic of an application is no longer the computer processor or the operating system or the database but the content of the data store. As a result, there is a need for tools and policies to produce valuable data, provide access to the data, and mechanisms for data-interoperability.

5. Future Web developments

Semantic processing is the holy grail of computing. A main goal of computing is automation and to automate a process, we need to first understand it and then to specify it precisely. In programming language processing, the definition of syntax and its processing was attacked successfully relatively early on (1960s) but semantic definition of languages has continued to challenge researchers. Natural language processing and speech understanding have similarly been facing challenges in semantic processing. The Web in general, and Web applications in particular, also face the challenge of assigning meaning to the vast amount of data stored on the Web and its processing.

Currently, we consider the Web to contain a collection of documents in different formats. For example, the most successful application on the Web, Google search, does a "simple" textual search without any attention to the context or meaning of the text being searched. The task of semantic processing is left to the human user. But as the amount of information on the Web increases, we will need more and more to rely on computer processing of the information and this will require assigning semantics to the content on the Web. The semantic Web is one of the major efforts in this area. Just as XML has standardized the structural definition of data, the goal of the semantic Web is to standardize the semantic definition of the data. If the data on the Web is semantically defined, we may expect to see a whole semantically-oriented generation of Web applications such as semantic search and semantic Wikis.

5.1. Ontologies and folksonomies

Assigning semantics to the enormous amount of data on the Web is a daunting task. Even if we optimistically assume that once we have a standard semantic definition method all the newly defined data will adhere to it, there is still the problem of managing the semantics of the data already existing on the Web. This is the legacy problem of the Web. There are two types of approaches to the semantic definition of the Web data. The ontological approach relies on a top down (imposed) approach of defining ontologies that are used to classify data. For example, a tourism ontology may define classes such as lodging, price, season, tour, package, etc., each of which may further be defined in terms of subclasses. Thus, a program looking for a tour package in low-season in Paris may automatically search for and query the data. This approach requires the definition of standard ontologies which begs the question of who defines the ontologies and how is agreement reached on the "correctness" of these ontologies.

Another approach to semantic definition of Web data is based on so-called folksonomies. This approach is bottom-up and driven by users of the data who may tag the data according to what they view to be the semantics of the data. As many users continue to tag the data, a structure emerges that reflects the consensus of the users about the relationships of the data items and their relevance. This structure in general will be neither consistent nor hierarchical as an ontology and it may not necessarily reflect the wishes of the owners of the data. But it may be a more accurate description of the unpredictable and unstructured information on the Web. This approach requires suport for tagging of information. Collaborative tagging[12, 15] (as opposed to personal tagging) refers to the process of many users tagging the same data and using the combination of the tags for interpreting the meaning of the data. One consumer of the tagging data is other users. But we can imagine that eventually computer programs may also be able to use these tags. The work in [16] attempts to define the semantics of collaborative tagging as an example of how such processes may be specified to enable, among other things, automated processing of Web data.

The task of tagging all the available data is non-trivial. Some of the popular sites that have many visitors may be able to rely on the visitors to tag their data. The less popular sites, however, cannot count on the random user to tag the data. Even in the popular sites, the amount and kind of tagging will be uneven and unbalanced, depending on the whims of the visitors. If tags are to be relied upon seriously, a more systematic process of tagging is necessary. One way of doing this is to provide incentives to users to actively tag the data. An interesting approach used by von Ahn [25] is to involve the user in a game in which, to win, the user must tag as much data as possible. Engaging the millions of users of the Web to all work on this task helps to make the problem more manageable.

Even though in many discussions the two approaches of ontologies and folksonomies are contrasted as alternative approaches, we can clearly combine the two approaches in fruitful ways. One way would be to use tagging initially as a way to collect user input for constructing an ontology. Another is to consider tags as user requests for enhancing and evolving an ontology.

5.2. Technical trends in Web applications

The client-server paradigm has served the Web well. Initially, the idea was that the browser is a "thin" client and all the processing is done on the server. The server responds to the browser requests and each time that it computes what the client requested, it sends the results back to the client to be displayed. The task of the browser was only to interpret and display HTML content. But as Web applications started to perform more functions than simply retrieving data to be displayed, the total reliance of the browser on the server became a problem, leading to unresponsive applications. For example, if the user fills in a form, hits the submit button, and waits for the browser to send the form to the server to validate the information in the form, the user pays the price of network latency and the server has to spend processing power on something that in many cases could be done better by the client computer. Such observations led to adding more capabilities on the browser side. One of the first steps was JavaScript which supports sophisticated client-side processing by the browser. JavaScript has grown into a relatively large language that allows the client to perform many of the functions that were earlier done by the server and to provide a high degree of interactivity to the application.

Regardless of how much work is off-loaded to the

browser, the latency problem still exists when the browser has to wait for large amounts of data to be delivered by the server. Caching was the obvious first solution to this problem. Even early browsers had the possibility to cache some data. Caches were also used for local area networks to serve all the users on that network. But with the growth of multimedia data, the network latency and bandwidth problems called for other solutions. Content delivery systems were pioneered by Akamai Technologies to reduce the network costs of multimedia delivery. Akamai servers are located around the world and store multimedia content for their (company) clients. For example, CNN's images or videos would be stored on Akamai servers. When a (user) client accesses the CNN site, the multimedia URLs are converted to refer to the data at a location closer to the client.

One technique that has allowed browsers to provide more responsive operations to the user is to push some of the client-server communication to the background while the browser still provides interactivity to the user. AJAX (Asynchronous JavaScript and XML) refers to a set of techniques that do exactly this. As opposed to the synchronous communication of the original HTTP, AJAX uses an asynchronous protocol for client-server communication. The browser registers a set of call-backs for the server to use for updating the browser's data in the background. This technique has allowed applications to provide highly interactive services.

We expect to see continuing trends in browser sophistication and network delivery mechanisms. In the next two subsections, we review likely future developments.

5.2.1 Browser trends

Programming languages used for building Web applications encode the accumulated wisdom of the community about the best ways to build such applications. Over time, application trends and features influence new generations of programming languages that ease the development of more advanced features. Early Web languages, such as PERL, PHP, or JavaScript were aimed at supporting primarily the browser or the server. More recent languages such as Ruby are general purpose and cover the whole range of browser, application logic, and server functionalities.

Recently, untyped, dynamic languages (such as Ruby) have become more popular for Web development. This trend will probably continue as processor speeds increase and language implementations become more advanced. Such languages free the programmer from dealing with machine and operating system specific details so that the logic of the application becomes the focus of development.

The programming language or languages which enable next generation Web applications are important. Equally important is the environment in which they operate, the Web browser. The browser provides an execution environment on the client for Web applications to run. What was once a simple interpreter of simple text files with small bits of markup has now become a sophisticated rendering engine and application platform. Web browsers have become the primary operating environment for (Web) applications, possibly portending a paradigm shift in software deployment. For such a shift to occur, however, the following challenges must be overcome.

The browser has been the focus of much development in the Web community. The long term functionality included in the browser environment will most likely be driven by the most common applications run in the upcoming generation of dynamic Web applications. We can, however, already envision a number of basic developments.

For many reasons, including security, Web browsers have traditionally been given limited access to the local resources on the client computer. As we have said previously, the idea was that the main functionality of the application was on the server and the browser only had to provide a simple GUI interface to the user for accessing the functionality on the server. The inability of the browser to take advantage of local resources, however, is a hindrance to many Web applications. Cookies, which have been the only mechanism to store information within the standard browser environment, are a very limited form of key-value storage. Starting with the Flash plugin and then implemented natively in Internet Explorer, in the latest browser cookies have been augmented to allow for the storage model presented in WHATWG [26]. In this model, applications can store structured data with a key-value interface, but unlike cookies, the data is not sent to the server on every page request. In the new storage model, this session data or global data always resides locally with the browser, and scripts running within the client can explicitly access the data. This is expected to ease many of the subtle difficulties with session management in Web applications, and it will allow a number of new application models to surface.

Another aspect of browser technology that will continue to improve the capabilities of Web applications is the graphics sub-system now included in many browsers. The $canvas$ tag, first introduced by Apple in the Safari browser, is now a WHATWG standard that has been included in Firefox [7] and Opera. This block level HTML element provides client side scripts with a vector based 2-dimensional drawing context similar to the typical postscript or pdf model. This in-browser graphics environment gives Web applications flexibility in creating content that was once limited to server side generation. Spreadsheet charts and graphs, for example, can now be generated in the browser environment for greater interactivity as well as offline display and manipulation. This canvas is the beginning of moving what were once system level libraries into the browser environment.

We can foresee two obvious extensions of this trend. The first is a 3-dimensional graphics context and the second is canvas item-based event handling. Including support for a hardware accelerated 3-dimensional rendering context, which is hinted at in both the WHATWG specification and the Firefox roadmap, would turn the Web into a true 3D environment. Currently, the canvas widget produces an element that is equivalent to an image in the page, but more fine grain event handling would allow for the creation of more dynamic content such as custom interface components and interactive games.

5.2.2 Network infrastructure

As we have seen, there is a trend to move some desktop applications to the Web. If the desktop-to-Web trend is to continue, a major challenge will be the need for a permanent Internet connection or the ability of the application to deal with intermittent connectivity to the server. In order to address this issue, the WHATWG has also been standardizing on a set of events which can notify running applications of the network status. This would allow a spreadsheet, for example, to store the most recent data locally so that once connectivity is restored, it can ensure that no data will be lost. Furthermore, by notifying an application that it will be taken offline, both data and application logic could be stored locally so that at least some aspects of a Web application might be continued without access to a network. In this model, the Web becomes an application deployment mechanism and browsers an execution environment, as opposed to just a data surfing application.

The increasing use of RSS has gone hand in hand with the rapid growth of decentralized media production over the Internet. As we have seen earlier, blogs have created a new paradigm of news delivery. But the current method of first posting data to a Web site, and then having users poll for updates to the data, is really the only feasible way to distribute content to a large number of people. To address the scalability problems of this method of data distribution, an obvious approach is to exploit a peer-to-peer distribution scheme. BitTorrent is probably the most popular peer-to-peer file distribution protocol and it can be used for this purpose. BitTorrent allows any number of clients to simultaneously download a piece of content by creating a large, amorphous sharing tree. Both the Opera and Firefox [17] browsers are working on integrating this peer-to-peer download protocol so that torrents can be seamlessly downloaded from within the browser. This is the beginning for peer-to-peer (P2P) integration in the browser. In future browsers or browser plugins many other types of P2P services will be integrated. Data that was once published at a single, centralized Web server will move onto distributed P2P networks that allow for fast data access and better scalability.

Besides fast access to Web data, Web applications also need large amounts of storage. We expect to see public storage servers that are available to Web applications. One example of a recent service which acts as an example of 3rd party data source is the Amazon Simple Storage Service, or S3. In S3 Amazon provides its global storage network as a general purpose tool for any Web developer with scalable storage needs. With a simple Web services interface, which can be accessed by either a server application or a browser script, data can be stored and retrieved from the S3 service. Other such storage services are becoming available on the Web with simple Web service APIs. Some cost nominal amounts and others are even offered free of charge.

5.2.3 Fat clients versus device independence

We have discussed the trend towards supporting fat clients. The motivation is to enable the client to appear to provide as much functionality as possible to the user without having to communicate with the server. This trend is supported by the introduction of ever-more powerful personal computers. However, a countervailing trend is the ubiquity of the Web as an application and operating platform. This trend implies that more and more devices, from cell-phones to navigators may act as Web clients. Such clients are more suited to being thin clients than fully equipped personal computers. It is likely that the trend for the Web to reach out to a wider variety of devices will be a stronger trend, implying that Web applications will have to deal with a range of client capabilities.

Client capabilities will span ranges in the processing and storage capacities, (battery) power, in network transmission speeds, and even network connectivity characteristics. Some clients may be expected to be connected continuously, others only intermittently, and yet others may move from network to network. For a Web application to be able to handle all ranges of client devices is a significant design challenge. In addition, some devices of limited processing capability may bring their own unique features that may be exploited by the application. For example, personal devices such as cell-phones make it easier for the application to determine the user's location and context and thus to provide personal and customized services. Personalization, in general, is a goal and major challenge for Web applications that serve millions of users.

Early work on device independence (e.g. [14, 13]) for Web applications only considered the client's display capabilities. The situation has become considerably more complicated with the increasing capabilities of Web applications. The current idea of "Internet of Things" will stimulate developments in device independence.

5.3. Social Semantic Desktop

An interesting current approach is to combine the semantic Web with peer to peer communication. This is the case of the social semantic desktop [8] of which Gnowsis is an implementation [22, 21]. The social semantic desktop is an extension of typical operating systems which aims to bring to the desktop a "semantic" view of the file system and a collaborative infrastructure for different desktops to inter-communicate. Such an infrastructure is intended to support new applications in social networking, knowledge work, community management and discovery, file sharing, information discovery, knowledge articulation and visualization. Such a social semantic desktop is supported by ontologies that define the structure and relationships of information on the desktop, semantic-oriented wikis for maintaining and sharing information and a P2P infrastructure for communication among the desktops.

5.4. Sites to services

Another trend that is changing the way Web applications are built is the exploitation of Web services. A web service is a piece of functionality accessible over the Internet with a standard interface. The message format is encoded with XML and services are invoked with a remote procedure call style (RPC). The service is viewed as an object that provides interfaces to access its methods remotely. Many Web sites now offer not only their own application services to human users but also offer them as web services so that programs (e.g. other Web applications) can access the services automatically. Web services make it possible to build applications by composing services from many different Web sites. For example, Google search facilities or Amazon book databases may be accessed through Web services and thus be offered as components or basic features of Web applications.

The use of Web services to construct Web applications holds the promise of fulfilling many software engineering goals such as component-orientation and reuse. In the context of Web applications, component-orientation is indeed a powerful notion since components can provide services ranging from highly specialized to very generic. For example, the search service offered by Google is rather generic but the Google maps services offer access to data that was expensive to collect and required the use of satellites not available to every Web developer.

Google and many other data intensive sites, such as Amazon and the photo sharing site Flickr, are also providing Web developers with API's which allow for remote access to data and images. This allows 3rd parties to create new applications on top of this data, and it gives immense flexibility to users because their data is easily accessed from anywhere in the world with a browser and an Internet connection.

The combination of the semantic Web and Web services can indeed go a long way towards automated processing of Web data.

5.5. Protocols for accessing services

The HTTP protocol supports the accessing of an object on a server identified by a URL. As the level of sophistication of the object is increased to be a service, do we need a more sophisticated protocol for accessing it? This is currently an issue that faces Web application developers. There are two current alternatives being debated commonly referred to as REST and SOAP.

REST [5, 6] (Representational State Transfer) suggests a modest addition on top of HTTP. It uses HTTP as the protocol and URL as the naming mechanism. The only addition is to use XML for packaging messages. REST was first introduced at ICSE 2000. The idea of REST is to rely on the existing Web infrastructure.

SOAP (Simple Object Access Protocol), on the other hand, is an elaborate set of standards being developed to address all aspects of applications including security and transactions. SOAP, which was initially offered as a simple XML-based RPC mechanism, has grown to refer to the whole set of Web Services standards being developed by W3C. It attempts to be general and all-encompassing. For example, it does not prescribe a particular protocol such as HTTP. It leaves the choice of the protocol to be used to the application, based on the argument that the application knows best. It also provides a separate security service as opposed to REST which simply relies on the standard HTTPS. In general, REST relies on existing standards and is thus more supportive of inter-operability.

At the heart of the debate is whether relying on open standards is enough (REST) or application developers need the option to step outside the standards when necessary (SOAP). The former approach favors inter-operability and the latter favors flexibility. On the basis of this argument, it appears that SOAP might be more suited for Intranet applications in which one (in principle) has control over the environment. In the open environment of the Internet where inter-operability is essential, REST seems to be more suited. The debate will probably go on for a number years. The current status is that REST is simpler and available and thus used by most applications.

On the other end of the spectrum of inter-operability, OSGi (Open Services Gateway initiative) is defining open standards for a service layer on top of Java JVM. It provides for inter-node and intra-node communication among services.

6. Open issues

The World Wide Web has been not only a technological invention but also a cultural phenomenon. The open availability of the Internet means that Web applications have a potential user base that is large and varied in all aspects such as age, nationality, technical literacy, and so on. Thus, unlike desktop applications, Web applications can have sociological impact. Unlike in traditional software engineering, the users are engaged and feedback to the developer may be immediate. On the other hand, early requirements engineering for emerging applications is not really possible as many of these applications create an idea and grow along with their users' requests. From a software engineering view, the user can help in evolving the requirements in very significant ways. The agility of Web application development, combined with the fact that all user interaction with the application can be monitored, means that applications can be frequently released, both to improve the user experience and to keep the user engaged.

The heavy involvement of the user has other implications. First, the applications can have significant impact on the users' life. Second, the application and the users can work in a symbiotic relationship to produce value for society. We will discuss examples of both of these in this section.

6.1. Automation and humans

Advances in computing make it possible to increasingly automate processes that were once carried out by humans. For example, with the increasing use of software agents many tasks can be automated, leading to efficiencies of operations and productivity improvements. With the close interactions between users and Web applications, humans often become participants in these automated processes. The pervasiveness of the Web means that successful Web applications can have tremendous, sometimes unintended, social implications. For example, when humans are part of the process, we face natural limits of human performance. We can increase computer efficiency, requiring more and more interactions with humans. Indeed, as more tasks are automated, we can view the human-computer interaction as the computer generating tasks for humans. Here is a personal example.

On Jan. 1, 2007, I received an email requesting me to write a reference letter for a colleague who had applied for a faculty position at a university. The email was automatically generated by the university's recruitment Web application. I assume that once the candidate's application was complete, the application generated the email requests to the candidate's references. Computers are capable of generating lots of such tasks for us. The trend is towards overloading the human users with more and more tasks. While we have tools to detect thrashing of computer processes, we have no such tools to detect human loss of efficiency.

This situation is perhaps not unique to Web applications. Any kind of automation that involves previously human processes increases the pressure on humans to perform at machine speeds. Factory automation, as caricatured in Charlie Chaplin movies, certainly had this effect. What makes the situation more serious in this case is that the Web application contacts you by email. In earlier days you could escape the machine by leaving the office. Today, the machine follows you when you log on to the Web from home or when you are on vacation. With more advances, your email will seamlessly follow you through your cell phone.

We need to find and establish social policies and conventions as well as technical solutions to address this problem. But first, we must recognize that a problem exists and give priority to its solution.

6.2. Collective intelligence

The Web is the largest collection of data ever assembled. Some argue that the contents of the Web represent all of human knowledge.Whether this is true or is a goal, it is certainly true that the Web contains a lot of useful information, much more than any single human being could know. We need tools to collect, authenticate, and validate the information on the Web and then tools for processing that knowledge. Considering the sheer amount of data on the Web, it may be possible to have tools that will help us process that information and solve problems that are important to society, or at least use the data in highly informed decision making.

A pre-requisite step in this process is to define precisely the data on the Web and the processes that apply to that data. The semantic Web takes a step in this direction. Another step in this direction is presented in [16], which attempts to specify the semantics of collaborative tagging formally.

7. Conclusions

The paper has discussed the area of Web application development from a software engineering point of view. The Web is an attractive playground for software engineers where you can quickly release an application to millions of users and receive instant feedback. Web application development requires agility, the use of standard components, inter-operability, and close attention to user needs. Indeed, one of the important features of popular Web applications is to support user participation to add value to the application and collaborate with other users. Due to the wide reach of the Internet, Web applications reach users that are varied in

age, culture, language, education, interest, needs, etc. Providing for interaction and collaboration among such varied users poses many interesting challenges.

Recent Web applications have brought new emphasis to the role of (unstructured) data in applications. There are interesting questions regarding how to generate, interpret, structure, disambiguate, validate, search, and otherwise manipulate huge amounts of data. The value of many applications increases as the volume of their data grows. With the help of scalable data-interoperability, applications and their users can collaborate. There are many efforts currently under way to address these problems.

8. Acknowledgments

I would like to thank Cédric Mesnage and Jeffrey Rose at Università della Svizzera italiana who helped me appreciate the new world of Web application development and helped teach a course on the subject in 2006. Thanks also go to the students who put so much energy into that course. I also would like to thank the different generations of students and colleagues who worked on Web applications at the Technische Universität Wien, especially Robert Barta, Manfred Hauswirth, Markus Schranz, Engin Kirda, and Gerald Reif. I have learned a lot from each of them.

This work was performed in part in the context of Project Nepomuk, Number 027705, supported by the European Commission in Action Line IST-2004-2.4.7 Semantic-based Knowledge and Content Systems, in the 6th Framework.

References

[1] C. Atkinson, C. Bunse, H.-G. Gross;, and T. Kühne. Towards a general component model for web-based applications. *Ann. Softw. Eng.*, 13(1-4):35–69, 2002.

[2] D. Bonura, R. Culmone, and E. Merelli. Patterns for web applications. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 739–746, New York, NY, USA, 2002. ACM Press.

[3] S. Casteleyn, Z. Fiala, G.-J. Houben, and K. van der Sluijs. From adaptation engineering to aspect-oriented context-dependency. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 897–898, New York, NY, USA, 2006. ACM Press.

[4] A. Cockburn. *Agile software development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[5] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 407–416, New York, NY, USA, 2000. ACM Press.

[6] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Trans. Inter. Tech.*, 2(2):115–150, 2002.

[7] *Firefox Feature Brainstorming*, 2006.

[8] M. Frank and S. Decker. The networked semantic desktop. In *International Semantic Web Conference*, 2002.

[9] P. Fraternali. Tools and approaches for developing data-intensive web applications: a survey. *ACM Comput. Surv.*, 31(3):227–263, 1999.

[10] M. Gaedke and J. Rehse. Supporting compositional reuse in component-based web engineering. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, pages 927–933, New York, NY, USA, 2000. ACM Press.

[11] M. Gaedke, C. Segor, and H.-W. Gellersen. Wcml: paving the way for reuse in object-oriented web engineering. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, pages 748–755, New York, NY, USA, 2000. ACM Press.

[12] S. Golder and B. A. Huberman. The structure of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, April 2006.

[13] E. Kirda. *Engineering Device-Independent Web Services*. PhD thesis, Technical University of Vienna, 2002.

[14] E. Kirda and C. Kerer. Diwe: A framework for constructing device-independent web applications. In L. Baresi, S. Dustdar, H. Gall, and M. Matera, editors, *UMICS*, volume 3272 of *Lecture Notes in Computer Science*, pages 96–110. Springer, 2004.

[15] C. Marlow, M. Naaman, D. Boyd, and M. Davis. Position Paper, Tagging, Taxonomy, Flickr, Article, ToRead. In *Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland*, May 2006.

[16] C. Mesnage and M. Jazayeri. Specifying the collaborative tagging system. In *SAAW'06, Semantic Authoring and Annotation Workshop*, 2006.

[17] *MozTorrent Plugin*, 2006.

[18] T. N. Nguyen. Model-based version and configuration management for a web engineering lifecycle. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 437–446, New York, NY, USA, 2006. ACM Press.

[19] T. O'Reilley. *What is Web 2.0—Design Patterns and Business Models for the Next Generation of Software*, 2005. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

[20] T. Reenskaug. Models - views - controllers. Technical report, Technical Note, Xerox Parc, 1979.

[21] L. Sauermann, A. Bernardi, and A. Dengel. Overview and outlook on the semantic desktop. In *Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference*, 2005.

[22] L. Sauermann, G. A. Grimnes, M. Kiesel, C. Fluit, H. Maus, D. Heim, D. Nadeem, B. Horak, and A. Dengel. Semantic desktop 2.0: The gnowsis experience. In *The Semantic Web - ISWC 2006*, volume 4273/2006, pages 887–900. Springer Berlin / Heidelberg, 2006.

[23] D. Thomas, C. Fowler, and A. Hunt. *Ruby: The Pragmatic Programmer's Guide, Second Edition*. The Pragmatic Programmers, 2006.

[24] D. Thomas, D. H. Hansson, A. Schwarz, T. Fuchs, L. Breedt, and M. Clark. *Agile Web Development with Rails: A Pragmatic Guide, Second Edition*. The Pragmatic Programmers, 2006.

[25] L. von Ahn. *Human Computation*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2005.
[26] *Web Hypertext Application Technology Working Group*, 2006.

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE