Internet Tools and Services - Lecture Notes

Attila Dr. Adamkó

Internet Tools and Services - Lecture Notes

Attila Dr. Adamkó

Publication date 2014

Table of Contents

Colophon

The curriculum supported by the project Nr. TÁMOP-4.1.2.A/1-11/1-2011-0103.

Part I. Internet Tools and Services

Table of Contents

Chapter 1. Introduction

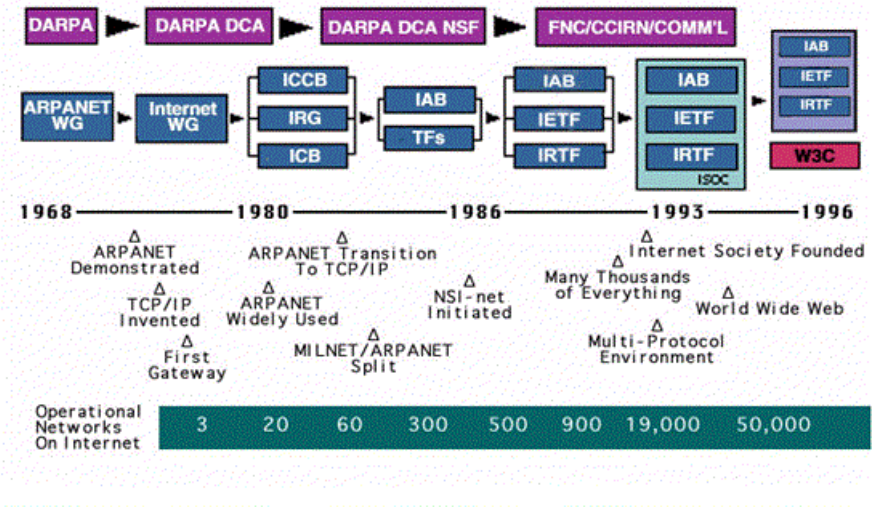
The Internet's history cover nearly 50 years from its born until our days. Its an interesting story and its evolution contains important milestones at least every decade. The last decades have seen considerable technological advances in the this sector. The current stage, the IoT (Intenet of Things), is far-far away from its initial version which was prepared by the invention of the telegraph, telephone, radio, and computer. The initial goal was clear, a connection was required between machines which forms a communications network that could exist even if parts of it was incapacitated. The story begun in 1966 at DARPA (originally ARPA which states for Advanced Research Projects Agency which is changed to DARPA as Defense Advanced Research Projects Agency in 1971). They created the ARPANET, the first *packet switching* network for host-to-host communication. ARPANET was funded by the United States military after the cold war with the aim of having a military command and control center that could withstand nuclear attack. The point was to distribute information between geographically dispersed computers. ARPANET created a communications standard (Network Control Protocol (NCP)), which defines the basics for the data transfer on the Internet today.

This network was the Internet’s forerunner before the public version was appeared in 1969. The original ARPANET grew into the Internet. The Internet embodies a key underlying technical idea, namely that of *open architecture networking*. In this approach, the choice of any individual network technology was not dictated by a particular network architecture but rather could be selected freely by a provider and made to interwork with the other networks through a meta-level "Internetworking Architecture". In an open-architecture network, the individual networks may be separately designed and developed and each may have its own unique interface which it may offer to users and/or other providers. Think about wired and wireless network solutions to get an imagine of it. The original communication standard, NCP did not have the ability to address other solution than the original ARPANET, so it need to be replaced. The new protocol would eventually be called the *Transmission Control Protocol/Internet Protocol* (TCP/IP) and appeared in 1972. However, the widespread presence of the Internet is dated into the mid of the '80s when the presence of PCs and workstations are started growing.

A major shift occurred as a result of the increase in scale of the Internet and its associated management issues. To make it easy for people to use the network, hosts were assigned names, so that it was not necessary to remember the numeric addresses. The *DNS (Domain Name System)* permitted a scalable distributed mechanism for resolving hierarchical host names into an Internet address. The increase in the size of the Internet also challenged the capabilities of the routers. New approaches for address aggregation, in particular classless *inter-domain routing (CIDR)*, have been introduced to control the size of router tables. Nowadays, after thirty years, there are still several researches for making these algorithms much more better, reliable and faster.

An other important piece in this picture is the role of *documentation* which established a series of notes for proposals and ideas. That was the *RFC (Request for Comments)* which is the way till nowadays to share feedback between researchers. The key its free and open access nature, all the specification and protocol documents are easily accessibly to everybody. The method is still using its original concept just the way of the publication is changed. At first the RFCs were printed on paper and distributed via snail mail. As the File Transfer Protocol (FTP) came into use, the RFCs were prepared as online files and accessed via FTP. Now, of course, the RFCs are easily accessed via the World Wide Web.

In the last three decade there are several organizations and work groups were appeared to help the standardization of the Internet. No longer was DARPA the only major player in the funding of the Internet. This evolution could be seen in the following figure (from the www.internetsociety.org website):



Standardization of the Internet

Internet Tools and Services

The Internet covers large, international Wide Area Networks (WAN's) as well as smaller Local Area Networks (LAN's) and individual computers connected to the Internet worldwide. The Internet supports communication and sharing of data, and offers vast amount of information through a variety of services and tools. **The major Internet tools and services** are:

- Electronic mail (email)
- Newsgroups
- Internet Relay Chat (IRC)
- Telnet and SSH
- File Transfer Protocol (FTP and FTPS, SFTP)
- World Wide Web (www)

Electronic mail , most commonly referred to as **email** or **e-mail** since ca. 1993, is a method of exchanging digital messages from an author to one or more recipients*E-mail clients* allow you to send and receive electronic mail messages. To use e-mail on the Internet, you must first have access to the Internet and an e-mail account set up (mostly free of charge) that provides you with an e-mail address. Valid e-mail address consists of a username and a domain name separated by the @ sign.

An email message consists of three components: the message *envelope*, the message *header*, and the message *body*. The message header contains control information, including, minimally, an originator's email address and one or more recipient addresses. Usually descriptive information is also added, such as a subject header field and a message submission date/time stamp. Network-based email was initially exchanged on the ARPANET in extensions to the File Transfer Protocol (FTP), but is now carried by the Simple Mail Transfer Protocol (SMTP), first published as Internet standard 10 (RFC 821) in 1982. In the process of transporting email messages between systems, SMTP communicates delivery parameters using a message *envelope* separate from the message (header and body) itself.

Newsgroups are often arranged into hierarchies, theoretically making it simpler to find related groups. The term top-level hierarchy refers to the hierarchy defined by the prefix before the first dot. The most commonly known hierarchies are the Usenet hierarchies. ***Usenet*** is a news exchange service similar to electronic bulletin boards. Usenet is older than the Internet, but the two are commonly associated with one another since most Usenet traffic travels over the Internet. A Usenet newsgroup is a repository usually within the Usenet system, for messages posted from many users in different locations. The term may be confusing to some, because it is in fact a discussion group. In recent years, this form of open discussion on the Internet has lost considerable ground to browser-accessible forums and social networks such as Facebook or Twitter.

Internet Relay Chat (IRC) allows you to pass messages back and forth to other IRC users in real time, as you would on a citizens' band (CB) radio. It is mainly designed for group communication in discussion forums, called *channels*, but also allows one-to-one communication via private message as well as chat and data transfer. IRC is an open protocol that uses TCP. An IRC server can connect to other IRC servers to expand the IRC network. Users access IRC networks by connecting a client to a server. The standard structure of a network of IRC servers is a tree. Messages are routed along only necessary branches of the tree.

Telnet allows you to log into another computer system and use that system's resources just as if they were your own. Telnet was developed in 1969 beginning with RFC 15, extended in RFC 854, and standardized as Internet Engineering Task Force (IETF) Internet Standard STD 8, one of the first Internet standards. However, because of serious security issues when using Telnet over an open network such as the Internet, its use for this purpose has waned significantly in favor of SSH (**Secure Shell**). SSH uses public-key cryptography to authenticate the remote computer and allow it to authenticate the user.

File Transfer Protocol (**FTP**) is a standard network protocol used to transfer files from one host to another host over a TCP-based network, such as the Internet. FTP is built on a client-server architecture and uses separate control and data connections between the client and the server. FTP users may authenticate themselves using a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that hides (encrypts) the username and password, and encrypts the content, FTP is often secured with SSL/TLS ("FTPS"). SSH File Transfer Protocol ("SFTP") is sometimes also used instead, but is technologically different and based on the SSH-2 protocol.

The **World Wide Web**, usually referred to simply as the Web, is a solution for displaying, formatting and accessing multimedia information over a network such as the Internet. It is a system of interlinked hypertext documents which allow related subjects to be presented together without regard to the locations of the subject matter. Hyperlinks function as pointers to information, whether the information is located within one website or at any site throughout the world. A website is a set of files residing on a computer (usually called a server or a host). Web sites do not have to be connected to the Internet. Many organizations create internal Web sites to enhance education, communications and collaboration within their own organizations. You access the site with software called a Web browser which displays the files as "pages" on your screen. The pages can contain files of text, graphics, sounds, animation, interactive forms-almost any form of multimedia-and they can be downloaded to your computer. Webpages are written in Hyper Text Markup Language (HTML).

Recently, the Web has become the predominant form of Internet communication (with the exception of e-mail), far outstripping the use of other systems such as gophers, newsgroups or ftp sites. It is already becoming a significant factor in many organizations' approaches to internal and external communications and marketing. The Web provides an immensely popular and accessible way to publish electronically, offer services or simply express your creativity.

The Web hides all of the underlying technology from the user. When you access a webpage, your browser locates and brings you the data. You do not have to worry about where the information is located, and the browser manages all storage, retrieval and navigation tasks automatically. The Web can handle many forms of Internet communication, such as FTP, Gopher and Newsgroups and Usenet, replacing the need for many other tools for using the Internet.

However, the story does not end in here. The Web continuously changing and new technologies are emerging. The next biggest invention is the Semantic Web which is currently just a little bit more than a vision. The technologies are existing but the implementation is partial. It it became reality the Web become one of the most important service of the Internet.

Summary - How does the Internet work?

If we need to conclude this section, we can say that it starts with protocols and finish in architectures. The most dominant parts are listed in the following section:

- Protocols – standardized rules that define how computers communicate and exchange data
- IP address – unique number used to identify computers on the Internet
- Domain name – structured naming system to locate computers on the Internet
- URL – uniform naming scheme that specifies unique addresses of Internet resources

- Client and server – computing architecture used by most Internet services

The Internet is a packet-switching network that uses TCP/IP as its core protocol. TCP/IP is a suite of protocols that govern network addresses and the organization and packaging of the information to be sent over the Internet:

- TCP – flow control and recovery of packets
- IP – addressing and forwarding of individual packets

An IP address is a unique address assigned to each computer connected to the Internet. It is used by TCP/IP to route packets of information from a sender to a location on the Internet. IP address consist of four sets of numbers ranging from 0 to 255. As we mentioned earlier, its hard to remember if we use several locations on the Internet. Domain Name System (DNS) allows the use of easier to remember domain names instead of IP addresses to locate computers on the Internet. Domain Name Resolvers scattered across the Internet translate domain names into IP addresses.

Domain names have two parts: the first part names the host computer while the second part identifies the top level domain. Top level domains (TLD) are identifying the type of host. It could be a generic Top Level Domain, like

- com – commercial/company site
- edu/ac - educational/academic
- gov – government site
- org– non-profit organization
- mil – military sites
- int – international organizations
- net – network providers

or a Country Code Top Level Domain, like .hu for Hungary.

All the other protocols are responsible for a given application and resides in a higher level of the IP stack. The most important protocols:

- HTTP (Hypertext Transfer Protocol) - for accessing and transmitting World Wide Web documents
- FTP (File Transfer Protocol) - for transferring files from one computer to another
- Gopher Protocol - for accessing documents via Gopher menus (no longer widely used)
- Telnet Protocol - allows users to login to a remote computer
- Secure Shell (SSH)
- SMTP (Simple Mail Transfer Protocol) for sending and managing electronic mails (e-mail)

This list shows that the Internet is serving all the major functionality what a user needs. We can imagine from this short introduction that the field covered by the title of this subject is far more greater than a book could be. The main focus is put on the HTTP part an the related technologies. We need to underline this is not limited only serving static HTML documents. Its far-far beyond the original goal of the Web. In the remaining part of this book we will discuss the story of the Web, the provided services and the supporting technologies and theoretical background.

Chapter 2. History of the Web

The Internet is defined as a network of networks which data trafficking accordingly to the TCP/IP stack of protocols. Inside it the most-known resource certainly is the World Wide Web. As defined in W3C's website,

> The World Wide Web (WWW, or simply Web) is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (URI).

In other words, it is a virtual space that provides a number of resources accessed by identifiers. These resources are instances of hypermedia - as said Tim Berners-Lee in 1989.

However, the story begun at a little bit earlier. In *1945*, *Vannevar Bush* authored the article *"As We May Think"* in which he first proposed his idea of the *Memex* machine. This machine was designed to help people sort through the enormous amount of published information. His article described a Memex as a "device in which an individual stores his books, records and communications and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory."

As we may see, this is about 30 years before the invention of the personal computer and about 50 years before the appearance of the World Wide Web. Bush's idea was originally a storage and retrieval device using microfilm and users are allowed to make links, or "associative trails," between documents. The machine was to extend the powers of human memory and association. Bush's article greatly influenced the creators of what we know as "hypertext" and how we use the Internet today. *Ted Nelson* created the term "*hypertext*" in *1967*.

After 22 years later, a proposal for information management was appeared and referenced Nelson's "Getting it out of our system" work and established the basic concept of a system which is currently known as the *World Wide Web*. The proposal estimated 6 to 12 month to realize the first phase of the project with only two people. The work was started in October 1990, and the program "WorldWideWeb" first made available within CERN in December, and on the Internet at large in the summer of 1991. *Tim Berners-Lee* introduced this project to the world on the **alt.hypertext newsgroup**. In the post he said the project "aims to allow links to be made to any information anywhere".

He originated the idea of sharing and organizing information from any computer system in any geographical location by using a system of hyperlinks (simple textual connections that "linked" one piece of content to the next) and established ***three key technology*** to manifest it:

- *Hypertext Transfer Protocol (HTTP), a way that computers could receive and retrieve Web pages*,
- *HyperText Markup Language (HTML), the markup language behind every Web page*,
- *URL (Uniform Resource Locator) system that gave every Web page its unique designation*.

1. Web 1.0 - the Read Only Web

Web 1.0 was an early stage of the conceptual evolution of the World Wide Web. Its like external editing, where the content is prepared by the webmaster and users are simply acting as consumers of content. Thus, information is not dynamic, technically, Web 1.0 concentrated on presenting, not creating so that user-generated content was not available.

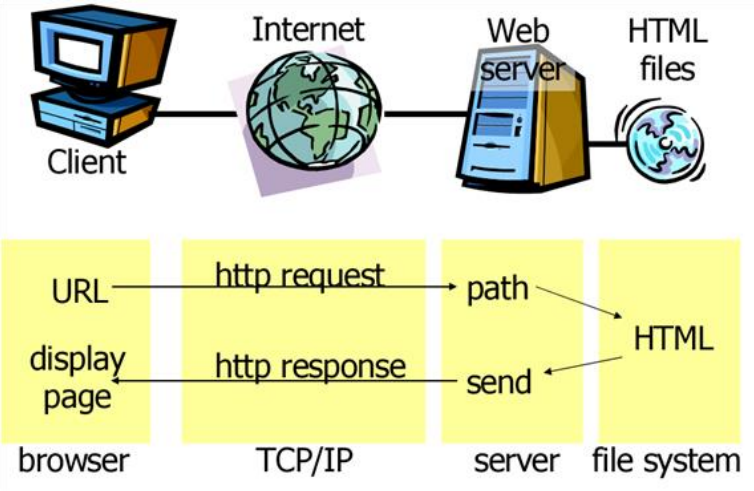
Web 1.0 pages are characterized by the following:

- Static pages instead of dynamic HTML.
- The use of framesets.
- The use of tables to position and align elements on a page. These were often used in combination with "spacer" GIFs (1x1 pixel transparent images in the GIF format).
- Proprietary HTML extensions, such as the <blink> and <marquee> tags (introduced during the first browser war).

- Online guestbooks.
- HTML forms sent via email.

The first webpage could be found in the following location: http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html

While the first phase of Web 1.0 could be summarized by the following figure:



Phase 1 of Web 1.0

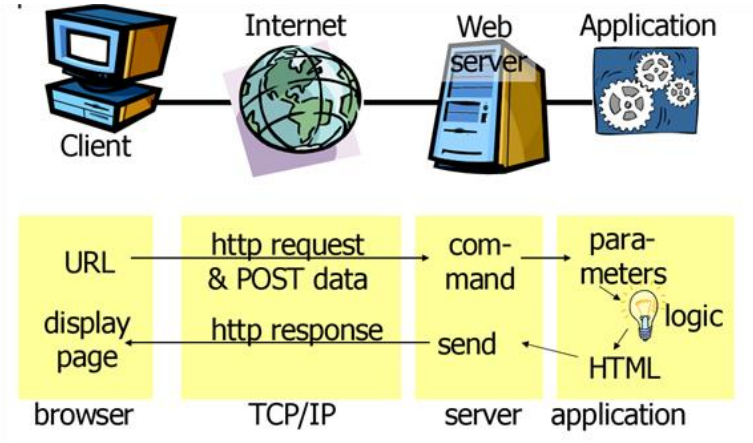
This results the birth of the first websites. A *website* is a collection of pages which contains images, videos and other asset that are hosted on the web server. The web servers consist of asset that are accessible via the Internet, cell phones or LAN. However, at this time websites are also referred to pages, because it may comprises different pages with different view and information. A webpage contains a written document in HTML, which is accessible and transported via Hypertext Transfer protocol (HTTP). The web page that is delivered to the user exactly as stored. The protocol transfers information from the web server to display it to the web browser. The webpage can be accessed through the uniform Resource Locator (URL), which is called the address. The URLs of webpages is usually organize in a hierarchy order, although the hyper link between them is a channel that helps the reader in the site structure and guide the reader's navigation on the site, which is generally contain a HOME PAGE where all the pages are been linked to each other.

Naturally, this early solution became obsolete just in few years. The disadvantages, like the lack of interactivity and personalization made the move. New technologies are appeared and static pages are transformed to dynamic ones. The goal was to introduce some "minimal" services to the sites like *searching, computing or communication*. The most relevant keyword for them were: HTTP-POST, CGI (Common Gateway Interface), SSI (Server-side Include) and Perl.

Common Gateway Interface (CGI) is a standard method for web server software to delegate the generation of web content to executable files. Such files are known as CGI scripts or simply CGIs; they are usually written in a scripting language. Perl was the most common language to write CGI application , but CGI application can be written in any language that has a standard input, output, and environment variables - like PHP , Bourne Shell (U*IX) or C. An example of a CGI program is one implementing a wiki, where the user agent requests the name of an entry. The program retrieves the source of that entry's page (if one exists), transforms it into HTML, and sends the result. If the "Edit this page" link is clicked, the CGI populates an HTML text area or other editing control with the page's contents, and saves it back to the server when the user submits the form.

SSI (Server Side Includes) are directives that are placed in HTML pages, and evaluated on the server while the pages are being served. They let you add dynamically generated content to an existing HTML page, without having to serve the entire page via a CGI program, or other dynamic technology. We can imagine it as a simple programming language, but SSI supports only one type: text. Its control flow is rather simple as well, choice is supported, but loops are not natively supported and can only be done by recursion using include or using HTTP redirect. Apache, nginx, lighttpd and IIS are the four major web servers that support this language.

The second phase of Web 1.0 can be seen in the following figure:



Phase 2 of Web 1.0

The last step in this evolution is arrived when traditional services are appeared on the Web as online services. In that phase the data mostly originates from databases and complex computations are done in the server side. The result is a comprehensive application environment where not just simple interactions could be made but combined workflows are possible. At this point again new tools are appeared, like session handling what is mostly implemented by cookie handling. A cookie, also known as an *HTTP cookie* is a small piece of data sent from a website and stored in a user's web browser while the user is browsing that website. Every time the user loads the website, the browser sends the cookie back to the server to notify the website of the user's previous activity. Cookies are used in several ways to achieve better user experience, like personalization and maintaining user data through a workflow or in multiple visits. However, there is an other usage scenario for cookies: tracking. Tracking cookies may be used to track users' web browsing activity. By analyzing the log data collected in, it is then possible to find out which pages the user has visited, in what sequence, and for how long. This opened a new line of privacy handling on the Web. Advertising companies use third-party cookies to track a user across multiple sites. In particular, an advertising company can track a user across all pages where it has placed advertising images or web bugs. Knowledge of the pages visited by a user allows the advertising company to target advertisements to the user's presumed preferences. Nowadays, a new EU directive is in charge to protect users and websites meed to show what kind of data is collected and process by them.

However, cookies have some drawbacks as well. There are several methods for cookie theft and session hijacking. Where network traffic is not encrypted, attackers can therefore read the communications of other users on the network, including HTTP cookies as well as the entire contents of the conversations, for the purpose of a man-in-the-middle attack. An attacker could use intercepted cookies to impersonate a user and perform a malicious task. An other problem appeared with the cross-site scripting. If an attacker was able to insert a piece of script to a page on a site, and a victim’s browser was able to execute the script, the script could simply carry out the attack. This attack would use the victim’s browser to send HTTP requests to servers directly; therefore, the victim’s browser would submit all relevant cookies, including HttpOnly cookies, as well as Secure cookies.

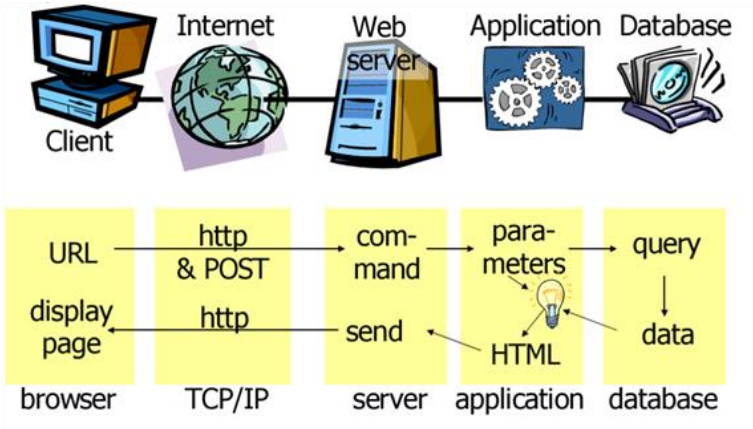
Naturally, cookies at the first time were a wonderful solution to overcome of the stateless manner of the HTTP protocol. It made available to handle workflows, introduce shopping carts on websites and made authentication easier. Besides privacy concerns, cookies also have some technical drawbacks. In particular, they do not always accurately identify users, they can be used for security attacks, and they are often at odds with the Representational State Transfer (REST) software architectural style. This is why alternative solutions are appeared.

A more precise technique is based on *embedding information into URL*s. The query string part of the URL is the one that is typically used for this purpose but this solution also have some drawbacks - like sending the same URL twice could cause problems if the query string encodes preferences and it was change between the two (same URL) request.

Another form of session tracking is to use web forms with *hidden fields*. This technique is very similar to using URL query strings to hold the information and has many of the same advantages and drawbacks. Most forms are handled with HTTP POST, which causes the form information, including the hidden fields, to be appended as extra input that is neither part of the URL, nor of a cookie.

The HTTP protocol includes the basic access authentication and the digest access authentication protocols, which allow access to a web page only when the user has provided the correct username and password. If the server requires such credentials for granting access to a web page, the browser requests them from the user and, once obtained, the browser stores and sends them in every subsequent page request. This information can be used to track the user.

All of these could be imagined with the following figure:



Phase 3 of Web 1.0

This third phase made the Web as a dominant platform which worth the investment. Centralized software could reach millions of users with one simple installation without the update process's nightmare. Just a simple client is needed, a web browser only. Its based on simple solutions:

Core Web Features: HTML, HTTP, URI

Newer Technologies: XML, XHTML, CSS

Server-Side Scripting: ASP, PHP, JSP, CGI, PERL

Client-Side Scripting: JavaScript, VBScript, Flash

Downloadable Components: ActiveX/Java

This online presence could be used to made available several services, made effective advertisements and finally to make money. All the economics behind Web 1.0 was very simple: everything is based on the traffic, advertisements and the most simple one: Insanity. An Internet company's survival depended on expanding its customer base as rapidly as possible, even if it produced large annual losses. The mantra was very short: “Get large or get lost”. At the time of August 2000, nearly 20 million websites were online.

The Dotcom Bubble Burst: January 14, 2000

The dotcom bubble had been growing since 1997. The excitement surrounding the web caused share prices to soar. Cisco became the world’s largest company, worth $400 billion (now $100 billion). $1 billion per week of Venture Capital money flowed into Silicon Valley. AOL took over Time Warner for $200 billion.

In January 2000 it reached its peak when the Dow Jones Industrial Average closed at a record level never reached before or since. On March 10 the NASDAQ Composite Index also reached an all-time high. Soon after, the markets began to crash and with it went many of the start up companies bankrolled during the dotcom boom. Between March and September 2000, the Bloomberg US Internet Index lost $1.755 trillion!

Where Web 1.0 went wrong was the misunderstood of the Web's dynamics. All of the development was relied on the old software business models, users were locked to APIs. Software was sold as an application and not as a service, so they were sold to the Head and not to the Tail as Web 2.0 solutions are do. The dynamics underlying the Web contains the Long Tail, the social data, the network effects and wisdom of the Crowds.

2. Web 2.0 - the Read/Write/Execute Web

Web 2.0 is a technology shift that provides a user level interaction that was not available before in the web environment. However, Web 2.0 was introduced in 2004 as a second generation of the World Wide Web that is focused on how information is shared among people. The word 2.0 comes from the software industry, which describe the transition from static HTML pages to dynamic webpages organized based on serving the web application users. Web become much more dynamic and interactive e.g. online communities. However, it is even easier to share information on the web. Popular websites that offer free services include Wikipedia, Google, and Facebook etc.

There are many definitions of Web 2.0. Wikipedia - as a prominent example for Web 2.0 - says:

> Web 2.0 is a term often applied to a perceived ongoing transition of the World Wide Web from a collection of websites to a full-fledged computing platform serving web applications to end users. Ultimately Web 2.0 services are expected to replace desktop computing applications for many purposes.

While this two sentences are properly outlines the primary concepts and we know that the Web is growing rapidly, and at such rate, websites continue to grow and more features are added, we need to see an other definition originating from Tim O'Reilly from 2005:

> Web 2.0 is the network as platform, spanning all connected devices; Web 2.0 applications are those that make the most of the intrinsic advantages of that platform:
>
> *delivering software as a continually-updated service* that gets better the more people use it
>
> *consuming and remixing data from multiple sources*, including individual users, while providing their own data and services in a form that allows remixing by others
>
> creating *network effects* through an "architecture of participation,"
>
> and going beyond the page metaphor of Web 1.0 to *deliver rich user experiences*.

From these two definitions we could derive the basic concepts arrived with the Web 2.0 expression:

Concept of Web 2.0

There are three main parts:

- **Rich Internet application**

  (RIA) — defines the experience brought from desktop to browser whether it is from a graphical point of view or usability point of view. Some buzzwords related to RIA are Ajax, Flash, Java FX (and the retired Silverlight as well). However, GWT, Vaadin and ExtJS are also related buzzes.

- **Web-oriented architecture**

  (WOA) — is a key piece in Web 2.0, which defines how Web 2.0 applications expose their functionality so that other applications can leverage and integrate the functionality providing a set of much richer applications. Examples are feeds, RSS, Web Services, mash-ups. (discussed detailed in the next chapter)

- **Social Web**

  defines how Web 2.0 tends to interact much more with the end user and make the end-user an integral part. In other words, let your users create your data, filter the data and create their own apps using your data.

As such, Web 2.0 draws together the capabilities of client- and server-side software, content syndication and the use of network protocols. Web browsers may use extensions to handle the content and the user interactions. Web 2.0 sites provide users with information storage, creation, and dissemination capabilities that were not possible in the environment now known as "Web 1.0".

2.1. Rich Internet application

A Rich Internet Application (RIA) is a Web application that is designed to deliver some key features and functions normally associated with desktop applications, which will help the user in accessing them. RIAs generally split the processing across the Internet/network divide by locating the user interface and related activity and capability on the client side. However, RIAs usually run inside a Web browser and normally do not require software installation on the client side to work. An RIA allows the client system to handle local activities, reformatting, calculations etc.

Characteristic of rich Internet application (RIA):

- *Performance impact*: In today's modern days depending on the applications and network characteristics, RIA is know often perform better that traditional apps. However most application that avoid round trip to the server by taking data and processing it locally on the client are likely to be faster to see. In order words offloading such data that have already been processed to the client machines improve server performance.

- *Better feedback*: Most Application using the RIA provides users with fast and accurate feedback. Due to their ability to change a part of the application without reloading, users can get to know more about the real time confirmation of an action, information and error messages. etc.

- *Partial page updating*: some web application pages are loaded once, when some one is updating something on the page; it will be automatically be sent to the server which makes the changes more easier and then resend the entree page. However there is no way HTTP and HTML can active this process. In traditional web based application a user is limited. However user has to wait for the entire page to reload even with the bandwidth connections, waiting times annoyed users. But RIA introduces some additional technologies, which can perform this task without any waiting time. Such technologies are the real-- time streaming etc.

- *Direct interaction*: According to competerworld.com "In a traditional page-- based Web application, interaction are limited to a small group of standard controls e.g. radio buttons, checkboxes and form fields. This severely hampers the creation of usable and engaging applications. An RIA can use a wider range of controls that allow greater efficiency and enhance the user experience. In RIAs, for example, users can interact directly with page elements through editing or drag-- and-- drop tools. They can also do things like pan across a map or other image."

Benefits of Rich Internet application (RIA).

Rich Internet application (RIA) offer organization a proven, cost effective way to deliver modern application with real business benefits,like:

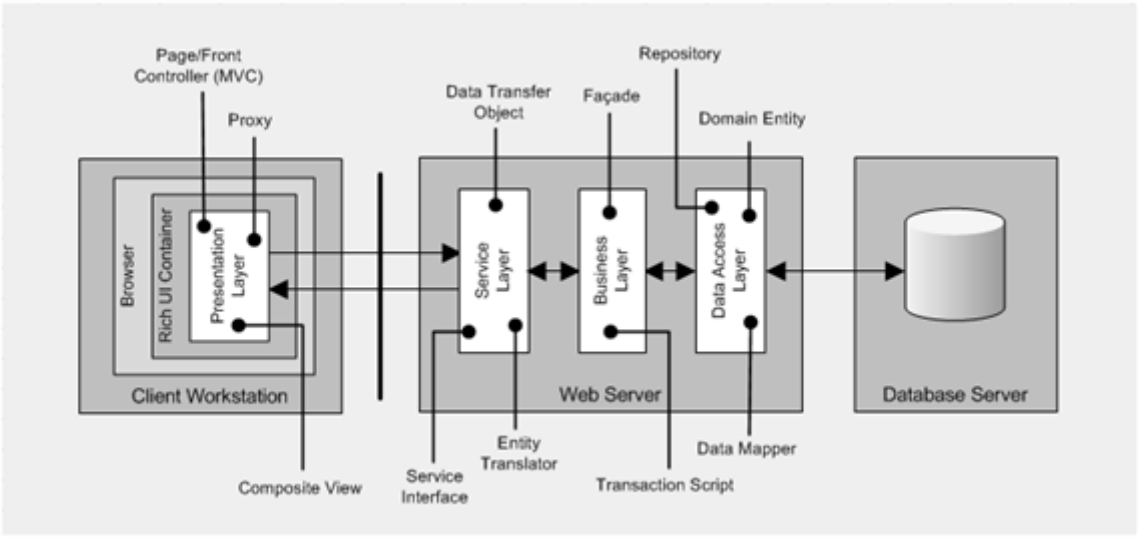
- Offer users a richer, more engaging experience.

- Keep pace with users' rising expectations.

- Increase customer loyalty and generate higher profits.

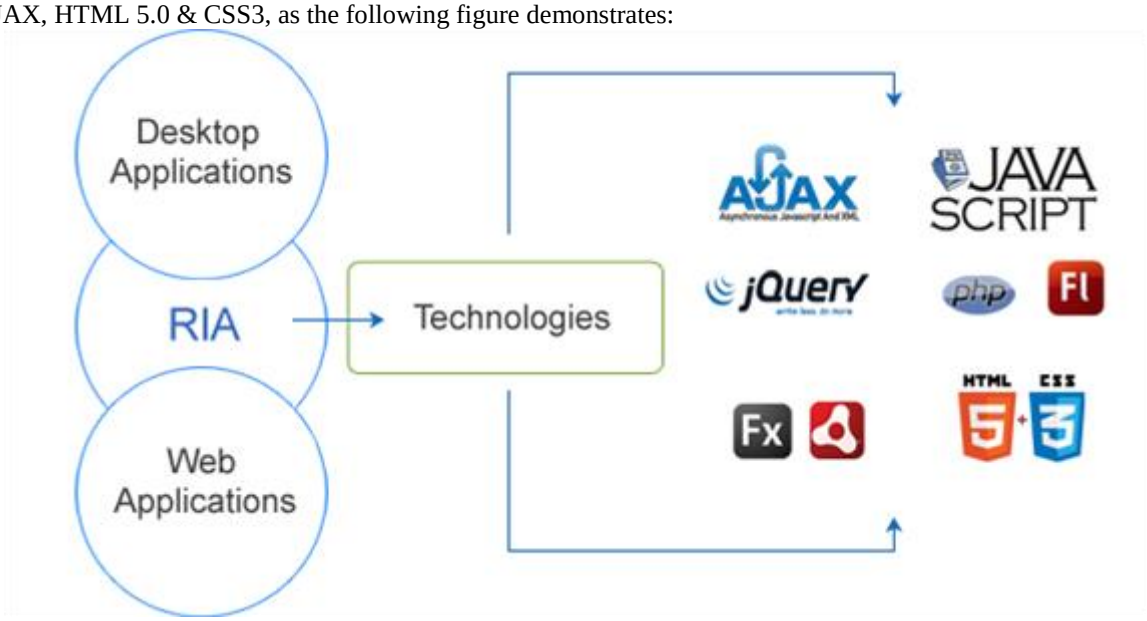
- Leverage existing personnel, processes and infrastructure.

Rich Internet applications are basically web applications designed to acclimatize and deliver function usually associated with desktop applications. The main feature of RIAs are that they do not need a software installation and run solely on a web browser. The code behind the RIA is devised in a way to identify and adjust accordingly. One striking feature of an RIA (in comparison to other Web-based applications) is the client engine that acts as an interface between the user and the application server. This can be seen in the following Figure:



RIA pattern

The most well-known tools used for RIAs are Adobe Flash / Flex, Adobe Air, Java Script, Struts, PHP, JQuery, AJAX, HTML 5.0 & CSS3, as the following figure demonstrates:



RIA technologies

2.2. Social Web

The Social web encompasses how websites and software are designed and developed in order to support social interaction. These online social interactions form the basis of much online activity including online shopping,education, gaming and social networking websites.

The social Web developed in three stages from the beginning of the '90s up to the present day, transforming from simple one-way communication web pages to a network of truly social applications. During the "one-way conversation" era of online applications in the mid '90s, the web was used socially at this time. In the mid '90s, some companies (like Amazon) made great progress in advancing online social interaction by storing information as well as displaying it. This led to the rise of read-write web applications, allowing for a "two-way conversation" between users and the individual or organization running the site.

The first social networking sites were introduced prior to social media sites. A *social networking site* is an online plate form that is usually created by an individual, describing his/her interest public. However this process enhances other people from different environment to get to know each other. Social networking site in general allows user to post their personal information, such as photographs, videos, and blog. Such sites, which are extremely popular, are usually an interactive site, which would allow users to be able to chart or share ideas

to other people across the web. A social site could be described as a great way to get in touch with a large group of people. However, if a user have any information that he wants to share; he can simply post it on the dash board which is known as the profile. Furthermore networking sites like this have different rules for creating connections, but some times they often allow users to view the connections of a confirmed connection and even suggest further connections based on a person's established network.

However some social networking websites like LinkedIn are used for professional connections, while sites like Facebook are in line between private and professional. There are many networks that are built for a specific user base, such as cultural, political groups within a given area, or even traders in financial markets. A social networking site can be seen as a public or semi-- public profile page, dating sites, fan sites etc.

It good to know, there are differences between social networking sites and social media sites. A social networking site is seen as a public or semi public site where as social media site are those site that could be used for broadcasting and let's anyone see your content — or at least, assumes someone you are not friends with might be interested in it. The focus is on voting up the most relevant content beyond the creator's neighborhood. Social network(ing) prefers to limit interaction and control to that first degree sphere; you might have some content visible to anyone, but mostly to identify people as relevant members of your close circle. Social Media are tools for sharing and discussing information. Social Networking is the use of communities of interest to connect to others. You can use Social Media to facilitate Social Networking.

Which sites/tools fall into which category. LinkedIn? Social networking. YouTube? It's social media. And what about Twitter and Facebook? Twitter and Facebook are Web 2.0 sites with the whole package. They straddle the Social Media and Social Networking divide perfectly.

Major types of websites

Blog : The term blog comes from the word weblog. Until 2009 blogs were usually the work of a single individual, occasionally of a small group, and often covered a single subject. More recently "multi-author blogs" (MABs) have developed, with posts written by large numbers of authors and professionally edited. This type of site is usually displayed in a reverse chronological order, such as the most recent post or upload appears first. The rise of Twitter and other "microblogging" systems helps integrate MABs and single-author blogs into societal news streams.

Wiki : These websites are created to serve as a detail way of passing descriptive information to the society, e.g. WIKIPEDIA. Text is usually written using a simplified markup language or a rich-text editor. While a wiki is a type of content management system, it differs from a blog or most other such systems in that the content is created without any defined owner or leader, and wikis have little implicit structure, allowing structure to emerge according to the needs of the users. Trustworthiness and Security - the two biggest attribute for wikis. Critics of publicly editable wiki systems argue that these systems could be easily tampered with, while proponents argue that the community of users can catch malicious content and correct it - Trustworthiness. While vandalism is affecting security. The amount of vandalism a wiki receives depends on how open the wiki is.

Social : Social network site is a site that enable user to create a public profile within that website and form relationship with other users of the web, however it is referred to a profile site. Social site on the Internet is describes the community based site where it brings people together to talk, share ideas, share interests, make new friends, etc. However this type of collaboration and sharing of data is often referred to as social media. Below are examples of social site; Facebook, twitter, YouTube, instagram etc.

If we would like to summarize to social side of the Web 2.0, we could found the following three concepts:

Users are creating data:

Amazon's reviews, Del.icio.us's bookmarks, Flickr's photos, Yahoo, Google's indexed web pages, Technorati's blogs, FriendsReunited's friends, Wikipedia's information

Users are creating data from your data:

Programmatic access to data (Web Services, RSS, FOAF, etc.). Apps showing how useful your data is compared with your competitor's. Adds value to your data.

Data filtering based on user behaviour

Recommendation engines, ranking algorithms, tagging.

2.3. Characteristics of Web 2.0

Using Web 2.0 sites the users - and sometimes visitors - have the ability to add some changes to webpages, allowing users to do more than just retrieve information. By increasing what was already possible in "Web 1.0", they provide the user with more user-interface (RIA), software and storage facilities, all through their browser. This has been called "network as platform" computing. Major features of Web 2.0 include social networking sites, user created websites, self-publishing platforms, tagging, news feed, social bookmarking and reviewing, like in popular sites such as amazon, zappos, eBay where shoppers are allowed to leave reviews about products. The following figure shows the most well-known representation of Web 2.0:

Web 2.0 Tagcloud

The key features of Web 2.0 include:

1. Folksonomy - free classification of information; allows users to collectively classify and find information (e.g. Tagging to provide somehow meta-information.)

2. Rich User Experience- dynamic content; responsive to user input (RIA)

3. User as a Contributor- information flows two ways between site owner and site user by means of evaluation, review, and commenting.

4. Long tail - Business aspect. Services offered on demand basis; profit is realized through monthly service subscriptions more than one-time purchases of goods over the network. (e.g. PayPerClick)

5. User Participation - site users add content for others to see (e.g. Crowdsourcing, Recommendation, Videocasting)

6. Basic Trust - contributions are available for the world to use, reuse, or re-purpose

7. Dispersion - content delivery uses multiple channels (e.g. file sharing, permalinks, RSS); digital resources and services are sought more than physical goods

With the emergence of Web 2.0, content can be easily shared and its offers all users the same freedom to contribute. However, this opens the possibility for serious debate and collaboration, it also increases the incidence of "spamming" and "trolling" and links back to the previously mentioned question about Trustworthiness and Security. Naturally, this possible downside should not let influence the good side of the Web.

Web 2.0 is important because it can easily grab attention, presenting the best output, which can triggers the attention of customers when visiting a site. It is easy for customers, it has been into studied since the first development of sites, that website that use simpler technique and strategies are been considered as popular sites in the web arena. Further researches shows that it takes few mines for a probable customer to decide whether he/she needs the information. Finally the third thing, market expansion. Expanding the market can be consider as a way of passing information about a product to a bigger environment. It is referred to as a process of offering a product service to a wider section of an existing market. Website are usually serve as a way of expanding business process, it boost a business, which has a low level to a higher level.

In Web 1.0, users were just spectators; users take information that the website provide by just reading it while with Web 2.0 users became more friendly to website by interacting with it. One of the most applicable reasons to have Web 2.0 is that it provides better functionality for interaction with websites. However web 2.0 websites, which revolutionized social networking, are Facebook, MySpace and Twitter. Thanks to the advanced enhancement of Web 2.0 these social networking websites which offer the users better interaction with each other whereby they can share ideas, comments, videos, links and much more. Furthermore in Social book marking and social networking are much more compatible with Web 2.0 because they have revolutionized the way the Internet is being used. However if you hire a web 2.0 service then you can let them do all the work for you. A user can use their services to get on any popular social networking website, you can write, and share information and even get to know more about any subject you want to. Web 2.0 is the greatest online development since the initial World Wide Web and it is making heavy changes in the way Internet technology is being used in today's world.

Advantage of Web 2.0 tools

LinkedIn

It is a social networking site that allows professional people to be connecting with professionals as well. However LinkedIn allows co-workers, customers, potential employers, previous colleagues, or potential clients to be connected with each other unlike personal social sites, where people focus on sharing photos and interests. LinkedIn users are more like keep on users where they detailed their employment and educational history. Furthermore, users of LinkedIn can recommend other LinkedIn user, which could be considered synonymous with a referee or employment reference. LinkedIn provides a way to opens avenues far beyond publishing your CV on the Internet; businesses can use LinkedIn as a way to finding potential employees with the exact talent and skills, which they require, and as a recruitment tool. However they can as well use it to generate sales leads, by finding out who are the key players within target organizations. Moreover users believe that competition is probably on LinkedIn, one could use public information about their employees to your own competitive advantages.

Features:

- Members or users can post photos and view the profiles and photos of other users.

- Members can view how many people have searched for and viewed them recently, although more detailed information requires a paid upgrade to a premium account.

- Employers can list jobs and openings and search for potential candidates.

Facebook

Nowadays, Facebook is one of the most popular social networking sites. It allows a user to create a profile, upload videos and photos as well. According to the statistics made by the Nielsen group, Internet users within the United States spends more time on Facebook than other sites. This makes it to a perfect business medium, where the profit originates from advertising. While doing business, Facebook bringing together all the Web 2.0 parts.

Wikis - By definition, a wiki is a collaborative space that can be edited by anyone with access to the site. This notion of participation and cooperation creates a more productive, usable information portal for all affiliated members. Facebook has rebranded this concept as *'Groups'*.

Blogs - When a user writes a *'Note'* on Facebook, they are expressing their thoughts or opinions in a given manner. A collection of these *notes*, in reverse chronological order, can be classified as a 'weblog' or *blog*. The offline concept of a diary has been around for centuries.

User-Generated Content (UGC) - Once again, the term may seem rather self-explanatory, but it does need some clarification. UGC is content created by the user - it is not production quality. Examples include *photos, videos, and audio clips*.

API - It is a way to integrate services for the data. This is what Facebook has done with their platform.

Micro-blogging - This new phenomenon is essentially a mini-form of blogging. Recently made popular by companies such as Twitter and Tumblr, micro-blogging is a way to provide a short message (usually less than 200 characters) about your life, mood, or current state via the web, e-mail, text, or IM. To meet demand in this area, Facebook launched *'Status Updates'*, which is simply another way of labelling micro-blogging.

Widgets - A widget is an embedded device that provides some level of value to the publisher. This is somewhat akin to what Facebook has done with their *'F8 Platform'*, and more notably '*Applications'*. Once a user adds a given *'Application'*, it appears on their profile page, where other users can see it and interact with it (or even add it themselves).

RSS - The concept of the '*News Feed'* acting as an *RSS reader*. Having said that, Facebook has started to integrate actual RSS protocol within the site as well. Anyone now has the ability to subscribe (via RSS) to another user's 'Notes', in many cases.

On top of all these obvious examples, Facebook also makes extensive use of *AJAX* (**A**synchronous **Ja**vaScript and **X**ML) throughout the site. This creates a more intuitive, enjoyable user experience. However, there are other features as well, like Nearby. Nearby technology tells you when your friends are nearby so you can get in touch with them easily. However Facebook will let you *share your location* with friends and other people that are not even related to you as well. This feature is hard to categorize because it goes beyond the simple Web 2.0 concept.

2.4. Further directions - Crowdsourcing

Crowdsourcing has become one of the ways in which the social Web can be used collaborative efforts, particularly in the last few years, with the dawn of the semantic web and Web 2.0. Crowdsourcing is the practice of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, and especially from an online community, rather than from traditional employees. This process is often used to subdivide tedious work to use crowd-based outsourcing or to fund-raise startup companies (*crowdfunding* e.g. Kickstarter) and charities, but it can also apply to specific requests, such as , a broad-based competition, and a general search for answers, solutions.

Facebook has also been a mode in which crowdsourcing can occur, as users typically ask a question in their status message hoping those that see it on his or her news feed will answer the question, or users may opt to use the poll option now available to obtain information from those within their friends network.

Continuing the travel back in time, we have found that the wisdom of the crowd could be found in several points. try to think about tagging (Del.icio.us, connotea.com) or voting systems (Digg.com, Reddit.com) or search engines (Google's PageRank). It can be summarized in one sentence: Decisions by the many better than decisions by one. However, the meaning of Crowdsourcing and derivatives are based on the first paragraph's point of view. The idea is to take work and outsource it to a crowd of workers. Famous Example: Wikipedia.

Instead of Wikipedia creating an encyclopedia on their own, hiring writers and editors, they gave a crowd the ability to create the information on their own.

Pros & Cons

Crowdsourcing's biggest benefit is the ability to receive better quality results, since several people offer their best ideas, skills, & support. Crowdsourcing allows you to select the best result from a sea of 'best entries,' as opposed to receiving the best entry from a single provider. Results can be delivered much quicker than traditional methods, since crowdsourcing is a form of freelancing. You can get a finished video within a month, a finished design or idea within a week, and microtasks appear within minutes.

Clear instructions are essential in crowdsourcing. You could potentially be searching through thousands of possible ideas, which can be painstaking, or even complicated, if the instructions are not clearly understood. Some forms of crowdsourcing do involve spec work, which some people are against. Quality can be difficult to judge if proper expectations are not clearly stated.

3. Web 3.0 - beyond the Semantic Web, a way to global SOA?

Seeing Web 2.0's advantages, we could ask from ourselves what could be the next shift? Try to imagine the following situation: You have not seen any new movies in a while and feeling all energetic, you make up your mind to go see a movie and have a late night dinner afterward. You are in the mood for some action adventure and an Italian delicacy. At first instant, you pull out your tablet, turn it on, open a web browser and immediately search for cinema, movies, and restaurant information. Without knowing what movies are showing in cinemas near you, you spend time reading short descriptions about movies which fall under action adventure before deciding. You sometimes even watch trailers about each movie showing to help make your choice easier. Although this might sway your decision in what movies you decide to watch-if there are less movies in the category you have decided-, you proceed anyway. Also, you may want to check for location, customer reviews and ratings for possible nearby restaurants. In all, you end up visiting several websites with a near or final conclusion in mind before heading out the door.

Some web experts are quite certain that Web 3.0-the next generation of the web after Web 2.0- will make such task like searching for movies or restaurants quicker, faster and easier. They believe that multiple searches will be a thing of the past and with complex search terms, the web can do the rest. Using the previous example, one could type "I would like to see an action adventure movie and then have dinner at an Italian restaurant. What possibilities do I have?" In this scenario, your response would be analyzed by the Web 3.0 browser after searching for all possible answers that match your criteria providing an organized search result for you.

Anyway, there is more to it. Most of these internet expert are certain that Web 3.0 browser will act like a personal assistant which is attentive in learning what one's interest is. They believe that the more you use your browser, the more your browser becomes knowledgeable about your questions. In the end, you might even be able to ask open questions to your browser such as "where is the best place for dinner nearby?" or "where is the best Italian restaurant in town" Looking up your records and taking into account your likes and dislikes, and also using your current location and geo-tagging, your browser would then suggest a list of possible nearby restaurants or eateries.

3.1. Graph Search - an other way of search

If you type in google what graph search is, you would most probably get results related to Facebook graph search. Though searching has been around for some time, it is not as natural as we would want. It is still dependent on keywords which results in articles on the web related or unrelated to our intended search. With this in mind, future web pioneers had to think outside the box. Facebook as a leading online presence moved a step further and developed Facebook graph search. To understand how graph search is different from normal searches, I'd like to shed more light on it.

Graph Search, popularly known as Facebook graph search, is a search engine combined with Facebook's social graphs. Using the search engine, natural language queries are processed from raw data and return information based on a user's network of friends, connections, or related information depending on the search. Current usage of graph search include but not limited to online marketing, job searches, common interest, dating, to name a few.

Below are a few examples;

- Most liked restaurants by friends living in Debrecen.
- Games fans of Harry Potter like.
- Debrecen alumni who like Titanic.
- Single ladies in Kassai utca.
- People in Debrecen who like Arsenal.

With graph search, several concepts of a search can be shared and correlated. These ties, which consist of search variables which are dependent on each other include education, hobbies, location, jobs, employer, marital status, gender, religion, interest and age. With graph search, organisations and individuals act as nodes which can be linked to one another.

With the idea of an emerging Web 3.0, the future looks promising for Facebook's graph search.

3.2. The Road to Web 3.0 through Web 2.0

Several jargon and internet buzzwords have made it to public consciousness and sub consciousness but of all these that evolved, Web 2.0 is by far the best known. Though most people may have heard of it in several ways than one, only a few have an idea what it really means. While several of those who have no idea of what it is all about suggest it is nothing more than a strategy but online marketers created to persuade venture capitalist (according to investopedia, "An investor who makes available capital either to startup ventures or supports small companies that wish to expand but do not have access to public funding") into investing millions of dollars into websites or startups. Without disputing the fact that Dale Dougherty of O'Reilly Media coined the phrase "Web 2.0" in 2004, there was never a decision if a "Web 1.0" existed.

Characteristics of Web 2.0 include but not limited to:

- Users and sometimes visitors have the ability to add some changes to webpages. Popular sites such as amazon, zappos, ebay allows shoppers to leave reviews about products. This helps future visitors get information that can be easily read.
- With the emergence of web 2.0, content can be easily share. Another example is youtube which allows users to create and upload a video to its site for visitors to watch.
- With a good internet connection, users who subscribe to websites can receive notifications via RSS (Really Simple Syndication) feeds.
- Access to the internet using handheld devices like smartphones and tablets. This way, internet has moved beyond mere desktop computers.
- Using interactive web pages to link people together thus bridging the gap of face to face meeting. Facebook-a popular social networking site- makes it easier for users to keep in touch with one other. It also helps users make new friends and find new friends too.
- Content which were inaccessible digitally is now easily accessible and available.
- With the emergence of "mashup" capability, users who are not professionals can create different applications using a mix of several software. Google maps is a popular example as it can be incorporated in different web applications and websites.

Moreover, think of Web 1.0 as the earlier stage of the World Wide Web which consisted of webpages that were connected by hyperlinks. Think of it as a source of information which information can be gotten but no change or contribution to such information is allowed. The exact definition of Web 2.0 has evolved over time, but with social networking and online interactions, Web 2.0 is focused on the ability of users to share and contribute information through social media, blogs, etc.

3.2.1. Folksonomy and Collabulary

As we listed in one of the Web 2.0's key principles, Folksonomies are the first step for the semantic version of the Web. Its an Internet-based Information Retrieval methodology or in other words, a collaboratively open-ended labels for categorizing content (webpages, photographs, links, etc.). The labels have a new name: Tags, and labeling have Tagging. It could be threated as people's classification management, where a folksonomy is accessible as a shared vocabulary which is familiar to its primary users.

It has several advantages, like dramatically lower categorization costs and quick respond to changes. Folksonomies are unsystematic, unsophisticated and open-ended (tags are created and applied on the fly). In spite of the various tagging abilities, the global process usually produces results comparable to the best professionally designed systems. Moreover, in enterprise level, the "emergent enterprise taxonomy" created by the employees could be seen easily.

However, there are disadvantages as well. The criticism shows several problems with

* polysemy (words with multiple meaning),

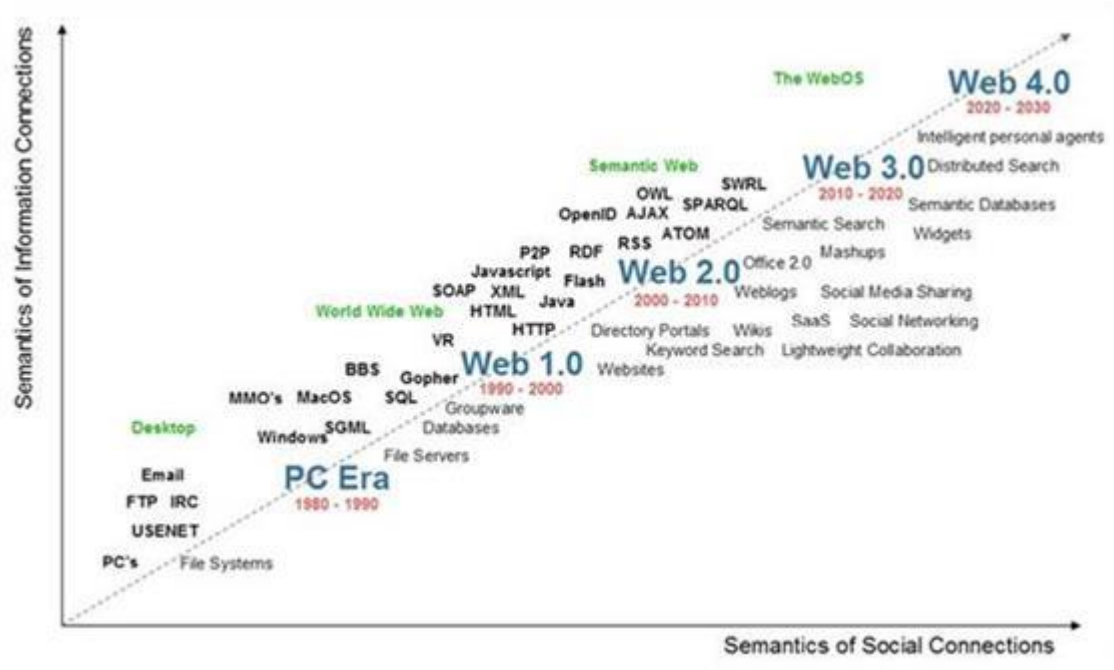
* synonyms (words with the same or similar meaning).

Over these, there are other factors, like plural words between the tags, and the "meta noise" which decreases the system's information retrieval with the false tagging.

The solution is a compromise between folksonomies and taxonomies (controlled vocabularies). This is the Collabulary. A collabulary arises similarly to what a folksonomy does but is developed collaborating with domain experts. It could avoid errors that inevitably arise in native, unsupervised folksonomies.

3.3. Basics of Web 3.0

Even though most users have not yet gotten a grasp on what Web 2.0 is about, others are already thinking ahead trying to figure out what comes next. There have been several questions to which we do not know the answer but yet they are still being asked. What exactly will Web 3.0 have that would separate it from Web 2.0? Would it be different from how we use the web today? Would it happen like a boom that we would not even notice it has already begun?



Timeline of the Web

Source: solutions.wolterskluwer.com/blog

People with extensive understanding about the internet believe that Web 3.0 would be like having an assistant who knows almost practically everything about a person with answers to unrestricted information stored on the internet. These experts also believe that even though Web 2.0 connects people using the internet, Web 3.0 will use the information from the internet to make the connection. However, questions have often been asked as to whether Web 3.0 will replace the current web as we have it or if it would exist separately.

As complicated as this concept may sound, an example may shed more light. Everyone loves to pamper themselves once in a while. Maybe going to the sauna or taking a nice vacation so as to watch relax on the beach and watch the sun rise and set. As usual, you set aside a budget of €1,500 for your vacation. Your desire is to find a flight deal, a nice restaurant and a comfortable place to stay without entirely spending all of your budget.

Currently, all you have to do is a little or sometimes more research to find the best holiday options available. You would research destinations, holiday deals, cheap flights, and budget meals until you find the right option that best suits you. Often, you may browse hotels, hostels, and car rentals as presented by several search engines. The entire search process of getting a holiday that satisfies your needs may take a few hours, days or weeks depending on your effort.

However, internet experts believe that Web 3.0 will let you sit back while the internet does all the work. The painstaking process of Web 2.0 will be a thing of the past. Using a search engine, Web 3.0 will be able to narrow down your search by gathering your data, analyzing those data and presents the data back to you in such a way that compares it quickly. With this, Web 3.0 will be intelligent enough to understand information from the web.

Currently, a search engine on the web, for example google, is not quite intelligent enough to be able to understand search. What it actually does is browsing through millions of webpages that contain relevant keywords related to your search terms. Search engines are unable to differentiate relevant or irrelevant webpages related to your search. What is does is display a webpage with a keyword in your search term. For example, if you typed in a search engine the word “Carina”, your results would be from webpages about the car “Toyota Carina” or names, products, and things bearing Carina.

It is believed that with a Web 3.0 search engine, not only would you find related keywords but the context of your request would be interpreted as well. Using our holiday destination as an example, if you do decide to search for “Budget holiday for a week under €1,500”, aside just displaying the results, information about restaurants, or upcoming activities related to your search might also be displayed in a Web 3.0 browser. The entire internet would not only be treated as an information centre but also as a massive database.

3.4. Approaches to Web 3.0 - APIs, SOA and semantics

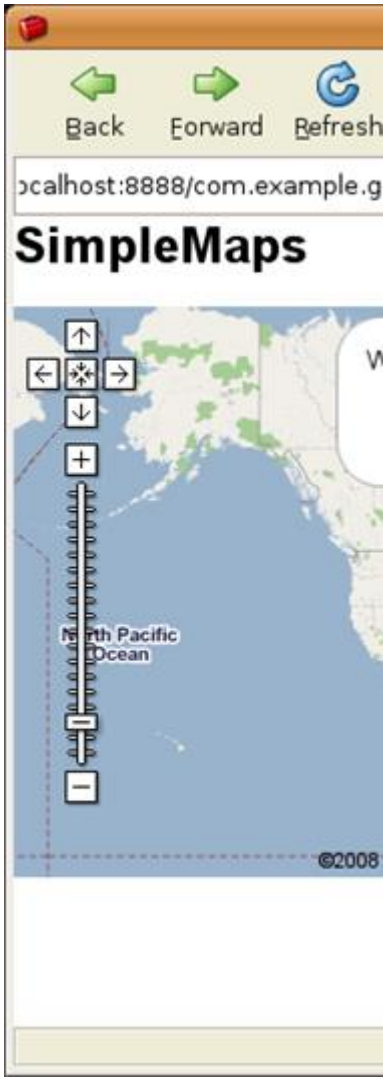
The future of the web is unknown to all. Most web experts believe that the experience will be relevant as well as enabling users have a distinctive profile. In this case, such distinctive profile will be based on individual’s browser history. With this concept, it would be easier to modify searches to suit every individual. They believe that if two users performed that same search using the same search term, both would get different result based on their individual profiles. It is slightly similar to graph search but they are not the same.

Presently, technologies for applications such as these are not yet mature. They are still in production and testing phase. Most still use a trial and error method which is not as efficient as what the future of Web 3.0 is intended to be. Experts believe that Web 3.0 will be founded on **Application Programming Interface (APIs)** to achieve a global SOA (Service-Oriented Architecture).

What is an API?

An **API** can be defined as an interface designed in a way that allows developers to create applications and take advantage of a certain set of resources. Many Web 2.0 sites include APIs which present developers with access to certain capabilities and unique site’s data. Popular social networking site Facebook allow developers to use their API for reviews, games, etc.

Among all current trends of the web as we have come to know it, mashup seems most likely the trend that would aid in the fast development of Web 3.0. A mashup is the process of combining two or more applications making them become one. An example is combining google maps with a hotel review site. Now this new site would not only show reviews about hotels but also display their locations for visitors to see.



An example of mashup using API

What is SOA?

The SOA term becoming widely used, but there's not a lot of precision in the way that it's used. There are at least *two different approaches* what we could identify. At first, the *technological side* which is a very technical perspective in which architecture is considered a technical implementation. The World Wide Web Consortium (W3C) for example refers to SOA as

> A set of components which can be invoked, and whose interface descriptions can be published and discovered

Second, a much more general interpretation, based on the Component Based Development and Integration (CBDI) forum's definition:

> The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface. (CBDI)

It highlights that SOA is much more than just an architectural pattern, it is a style. It highlights that any form of service can be exposed with a Web services interface. However higher order qualities such as reusability and independence from implementation, will only be achieved by employing some science in a design and building process that is explicitly directed at incremental objectives beyond the basic interoperability enabled by use of Web services. The CBDI forum advices that we need to think on it as a framework to understood what constitutes a good service. Two obvious sets could be identified here:

- Interface related principles—Technology neutrality, standardization and consumability.

- Design principles—These are more about achieving quality services, meeting real business needs, and making services easy to use, inherently adaptable, and easy to manage.

Business management and IT management may have a better understand the cost and benefits after they realized what are the difference when a system is not designed for this purpose. It is important to know if a service is to be used by multiple consumers, (as is typically the case when a SOA is required), the *specification needs to be generalized*, the service needs to be *abstracted from the implementation* (as in the earlier dotcom case study), and *developers of consumer applications shouldn't need to know about the underlying model and rules*.

If a service is SOA enabled we can say it is:

- reusble
- abstracted
- formal
- relevant
- published

With SOA it is critical to implement processes that ensure that there are at least two different and separate processes—for provider and consumer. For the consumer, the process must be organized such that only the service interface matters, and there must be no dependence upon knowledge of the service implementation. If this can be achieved, considerable benefits of flexibility accrue because the service designers cannot make any assumptions about consumer behaviours. For the provider, it needs to develop and deliver a service that can be used by the Service Consumer in a completely separate process. The focus of attention for the provider is therefore again the interface—the description and the contract.

CBDI concludes that there are three major process areas which we need to manage:

- **The process of delivering the service implementation.**
  - 'Traditional' Development
  - Programming
  - Web Services automated by tools
- **The provisioning of the service—the life cycle of the service as a reusable artefact**.
  - Commercial Orientation
  - Internal and External View
  - Service Level Management
- **The consumption process.**
  - Business Process Driven
  - Service Consumer could be internal or external
  - Solution assembly from Services, not code
  - Increasingly graphical, declarative development approach
  - Could be undertaken by business analyst or knowledge worker

This process view that we have seen is a prerequisite to thinking about the type of architecture required and the horizons of interest, responsibility and integrity. From the architectural point of view, Service-oriented

architecture is a design pattern that consists of discrete pieces of software with some functionalities and other applications can utilize these functionalities. This pattern does not require us to use some specific product or a platform. A service provides some functionality and this can be used by other large software applications to complete its use. A service is the base of this architecture. Services are the structures which have ability to interact with each other. In other words they are the listener of the other side of the phone which is an endpoint. In the classical layered architecture layers interact with each other. This interaction and hierarchy of this system should be constructed very properly for this system to work proper. Therefore we can say that SOA simply means that these layers are created as services. For example if we make a service which provides us the data, then we have accomplished the functionality of data layer by this. Afterwards we can also use this service from other applications as well. Achieving this we have a flexible architecture to use.

However, not all experts agree. Some however do believe that Web 3.0 will start from scratch. They believe that it would rely on some new programming language and not HTML. They suggest that starting from the scratch would be a lot easier than following the current trend of Web 2.0. The man responsible for the web as we have come to know it has a different theory of what he feels the future of the World Wide Web will be like. He refers to it as a semantic web and his work is mainly referenced when experts talk about Web 3.0. But then again, what exactly do we mean by a semantic web?

3.5. One aspect for the Web's future: Semantic Web

According to semanticweb.org,

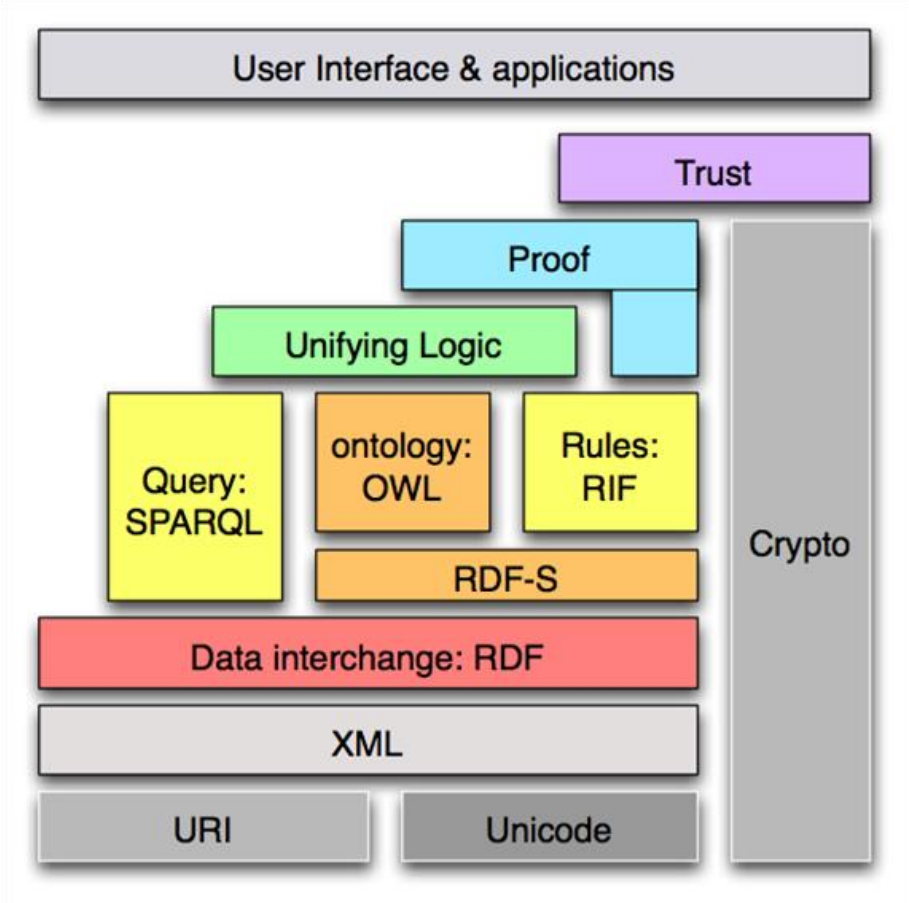
> The **Semantic Web** is the extension of the World Wide Web that enables people to share content beyond the boundaries of applications and websites.

However, w3.org also defines it as

> The **Semantic Web** provides a common framework that allows **data** to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF).

The World Wide Web was invented by Tim Berners-Lee in 1989. This happened 20 years after the first connection was established over what we know today as the internet. At that time, Tim worked as a scientist at the European Organization for Nuclear Research (CERN) and realized that it was difficult for scientists to share information with one another. With this discovery, the need for potential computers to be connected together was born. He disputes that Web 1.0 or Web 2.0 is nothing more than a meaningless jargon by maintaining claims that the intention of the World Wide Web is to do everything that both Web 1.0 and Web 2.0 is supposed to do.

The **semantic web** is a vision of what Tim Berners-Lee future of the web is supposed to be like. Presently, the web is structured for humans and not computers. Computers are unable to interpret information on the web but with semantic web, computers can interpret information on the web using software agents crawling around the web searching for relevant information. In Berners-Lee concept, semantic web would consist of metadata which would make the task of search by computers intelligent and more comprehensive. The most well-known figure describing the SemanticWeb's layered architecture is the following (from the talk of Tim B.-L. in 2006) :



Semantic Web's layered architecture

Before going any further, I would like to walk us through.

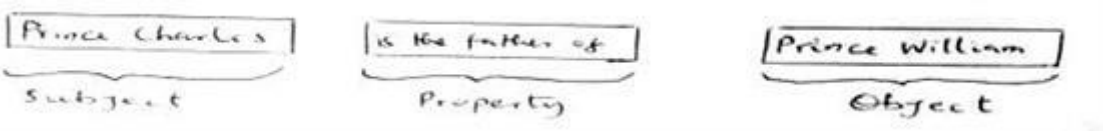
3.5.1. Why use Semantic Web?

With our previous example of searching for a movie, we have come concluded that most of the work done would be by the user. However, with Semantic web search, your choice would be directly entered and the web searches for the best choice to suit your need. In this case, the web would be intelligent enough to know your preferences and if you did have a bad experience from a particular theatre or restaurant, it would not use it when compiling your search result.

In this case, search in Web 3.0 would not be done by reading descriptions or reviews like a person would do but by a thorough search through its **metadata** that defines what the Web needs to present your data. According to whatis.com "**Metadata** is defined as data which describes other data. It is also a prefix used in information technology which means "an underlying definition or description"". Metadata sums up basic information about data thus making it easier to find and work with specific occurrence of data. For example, a metadata for a document would include title, author, file size, date created, etc. In webpages, metadata contain descriptions of the page's content as well as keywords linking the content to the page.

In Semantic Web, experts believe that the metadata of a page are not visible while the page is being read but visible to computers. Metadata, according to Tim Berners-Lee would let the web become a giant database.

3.5.2. XML, RDF and URI



A simple relationship analogy

From the photo above, it is easier to figure out what the sentence "Prince Charles is the father of Prince William". We know that they are both from the royal family, and there exist a relationship between them. We can also tell that father is a parent, and that William is the son of Charles. It is easier for humans but not so easy for the computer to understand such text. To help the computer understand what such sentence means, machine-readable information describing who Prince Charles and Prince William are needs to be added to make their relationship clearer. The tools needed are the **eXtensible Markup Language (XML)** and **Resource Description Framework (RDF).**

XML is a markup language used to create common information formats, share the format and data on the web. It is a standard way of describing data that enables a user to request a program, gather data and make comparison. Its a markup language and may seen similar for HTML at first sight, they need to be separated. While HTML used to describe the content of a web page and how it is to be displayed and interacted with, XML describes the content - maybe for a web page - in terms of what data is being described. XML and HTML are used together in many web applications and sometimes, XML may appear within an HTML page.

RDF on the other hand, does exactly what its name suggest. It provides a framework which describes resources using XML tags. An RDF description, sometimes referred to as "data about data" can include the authors of the resource, date of creation, updating, sitemap, information that describes content in terms of audience or content rating, keywords for search engines, subject categories, etc. By identifying the resource, using the above diagram as an example, the computer would not confuse Prince William with Prince Harry or Princess Diana, or with other members of the royal family.

To achieve this, RDF makes use of **triples** written as XML tags to express such information. These triples are the **subject, property and object**. Very often, people call these the subject, predicate and object. Presently, RDF exists in the web and it is part of the RSS feed creation.

Benefits of RDF

Some likely benefits of RDF include but not limited to;

- By providing a consistent framework, RDF will encourage the providing of metadata about Internet resources.
- Because RDF will include a standard syntax for describing and querying data, software that exploits metadata will be easier and faster to produce.
- Applications will be able to exchange information more easily due to its standard syntax and query capability.
- Searchers will get more precise results from searching, based on metadata rather than on indexes derived from full text gathering.
- Intelligent software agents will have more precise data to work with.

In the above example, the computer understands that there are there exist a relationship between the two objects. However, it does not know what the objects are nor how they relate to one another. Let's see how these objects relate to one another.

3.5.3. Identifying resources: URI

With the framework that XML and RDF provide, a computer still needs a very direct, specific way of understanding what these resources are and to do this, RDF uses **Uniform Resource Identifiers (URIs)** to direct the computer to a document (object) that represents the resource. The most common form of URI is the Uniform Resource Locator (URL), which begins with *http://*.

According to Margret Rouse,

> *A URI is the way you identify any of those points of content, whether it be a page of text, a video or sound clip, a still or animated image, or a program. The most common form of URI is the Web page address, which is a particular form or subset of URI called a Uniform Resource Locator (URL).*

An example of URI is http://www.w3.org/Images/WWW/w3c_main.gif

The above web address identifies a file that can be accessed using the Hypertext Transfer Protocol ("http://") which resides on a computer named "www.w3.org" that can be mapped to a unique Internet address. The computer's directory structure indicates that the file is located at the pathname of "/Images/WWW/w3c_main.gif."

Character strings that identify File Transfer Protocol FTP addresses and e-mail addresses are also URIs (and, like the HTTP address, are also the specific subset of URI called a URL). Other sources define a **Uniform Resource Locator (URL)** as "*the unique address for a file that is accessible on the Internet*".

Another kind of URI is the **Uniform Resource Name (URN)**. A URN is a form of URI that has "institutional persistence," which means that its exact location may change from time to time, but some agency will be able to find it. Very often, people get confused when speaking about URL and URN. A URN functions like an identity or person's name, while a URL resembles that person's street address. In other words, the URN defines an item's identity, while the URL provides a method for finding it.

More exactly, as defined in 1997 in RFC 2141, URNs were intended to serve as persistent, location-independent identifiers for web resources, allowing the simple mapping of namespaces into a single URN namespace. The existence of such a URI does not imply availability of the identified resource, but such URIs are required to remain globally unique and persistent, even when the resource ceases to exist or becomes unavailable.

Example,

`urn:issn:0167-6423`	The scientific journal *Science of Computer Programming*, identified by its serial number.
`urn:isbn:0451450523`	The 1968 book *The Last Unicorn*, identified by its book number.
`urn:lex:eu:council:directive:2010-03-09;2010-19-UE`	A directive of the European Union, using the Lex URN namespace.

Since RFC 3986 in 2005, the use of the term has been deprecated in favor of the less-restrictive "URI", a view proposed by a joint working group between the World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF). Both URNs and uniform resource locators (URLs) are URIs, and a particular URI may be a name and a locator at the same time.

URNs were originally intended to be part of a three-part information architecture for the Internet, along with URLs and uniform resource characteristics (URCs) as a metadata framework. However, URCs never progressed past the conceptual stage, and other technologies such as the Resource Description Framework (RDF) later took their place and became the "official language" of the Semantic Web but they cannot make the web accessible to the computer on their own. There is a growing need for more languages.

3.5.4. Semantic Web languages: RDFS, OWL and SKOS

One of the many obstacles of Semantic Web is that computers and humans do not have the same kind of vocabulary. We have used natural languages our whole life so it is almost completely easy to understand the connections between different words and concept and make a meaning out of it. Unfortunately for humans, we cannot just give the computer a dictionary, an almanac and a set of encyclopedia and let it learn all of it on its own. For the computer to understand words and the relationship between them, it needs to have **documents** describing all the words and **logic** in order to make required connections.

In the world of Schematic Web, this relationship come from **schemata** and **ontologies** which are the related tools which helps the computer in understanding human vocabulary. A schema is a method for organizing information while an ontology is the language that describes objects and the relationship between them. With RDF tags, a documents creator must declare which ontologies are referenced at the beginning of the document by providing access to schemata and ontologies included in these documents as metadata.

Below are the schema and ontology tools used on Semantic Web:

- **RDF Vocabulary Description Language Schema (RDFS)**

"*RDFS is extending RDF vocabulary to allow describing taxonomies of classes and properties. It also extends definitions for some of the elements of RDF*." (Maret Obitko, 2007). For example, the resource Prince William is a subclass of the class *Royals*. A property of Prince William could be *handsome*.

- **Simple Knowledge Organization System (SKOS)**

  "is a W3C standard, based on other Semantic Web standards (RDF and OWL), that provides a way to represent controlled vocabularies, taxonomies and thesauri."(Juan Sequeda, October 31, 2012). A **controlled vocabulary** is a list of term agreed upon by an organization or community such as Monday, Tuesday, Wednesday are days of the week. A **taxonomy** is a controlled vocabulary organized in hierarchy such as Phones as concepts and smartphones or mobile phones as subclasses because they are both phones while a **thesauri** is a taxonomy with more information about each concept including preferred and alternative terms ("Phone" in English, "Telefon" in Hungarian).

  - For example-in reference to SKOS-, with the Royal Family, a narrower term for Prince William could be William, while a broader term could be The Duke of Cambridge.

- **Web Ontology Language (OWL)**

  extends RDF and RDFS and its primary aim is to bring the expressive and reasoning power of description logic to the semantic web. With OWL, new classes can be constructed based on existing information. There are three levels of complexity in OWL namely Lite, Description Language (DL) and Full.

However, ontologies are very difficult to create, implement or maintain and depending on their scope, they can be enormous thus defining a wide range of concepts and relationships making it difficult for developers to focus on them and not logic or rules. Another pitfall for the semantic web would be the disagreement over what roles these rules play. Even so, some critics believe that such project is extremely impractical.

Going back to our original example about deciding on watching a movie, here is how the Semantic web could make the steps easier;

- Each site would have pictures and text so people can easily read them, and metadata so computers can read them describing available movies from different cinemas.
- Every metadata would make use of RDF triples and XML tags so that all attributes of the movie (like showing time, cast, and languages) would be machine-readable.
- Businesses would give the computer the vocabulary needed using ontologies to describe these objects and their attributes. Several e-commerce sites would use the same ontologies thus making all metadata a common language.
- Each cinema showing a movie would use appropriate security and encryption measures to protect customer info.
- All metadata found on different would be read by computerized applications and enable comparison of information and verify that the sources of such information is accurate and trustworthy.

In conclusion, much of Web 3.0 is far from reality but more of theory yet this has not stopped experts from guessing what might come next. Since the web is enormous, adding all this metadata to existing pages would be quite tasking

3.6. References

Bibliography

Web resources

[SOA] *Understanding Service-Oriented Architecture - David Sprott and Lawrence Wilkes CBDI Forum*. Understanding Service-Oriented Architecture .

[Web 3.0] *World Wide Web 3.0*. What is Web 3.0? What Will Web 3.0 Be Like? . Web 3.0: The Third Generation Web is Coming . A Brief Introduction to Web 3.0 .

[Web 2.0] *World Wide Web 2.0*. What is Web 2.0? - Definition from Techopedia . What is Graph Search? - Definition from WhatIs.com . What is metadata? - Definition from WhatIs.com . Utilizing Web 2.0 in business .

[Semantic Web] *Semantic Web*. W3C Semantic Web Activity Homepage . W3C Semantic Web Activity Homepage . semanticweb.org . Semantic Web - W3C . What is Resource Description Framework (RDF)? - Definition from WhatIs.com . RDF Schema RDFS - Introduction to ontologies and semantic web - tutorial . Introduction to: SKOS - Semanticweb.com . Web Ontology Language OWL - Introduction to ontologies and semantic web - tutorial .

[Web 1.0] *The World Wide Web Consortium*. History of the Web – World Wide Web Foundation . Uniform resource identifier - Wikipedia, the free encyclopedia . What is URL (Uniform Resource Locator)? - Definition from WhatIs.com .

[RIA] *Rich Internet applications*. QuickStudy: Rich Internet applications . About RIAs . What is Rich Internet Application (RIA) .

Chapter 3. Web Applications and Mashups

1. Web Applications

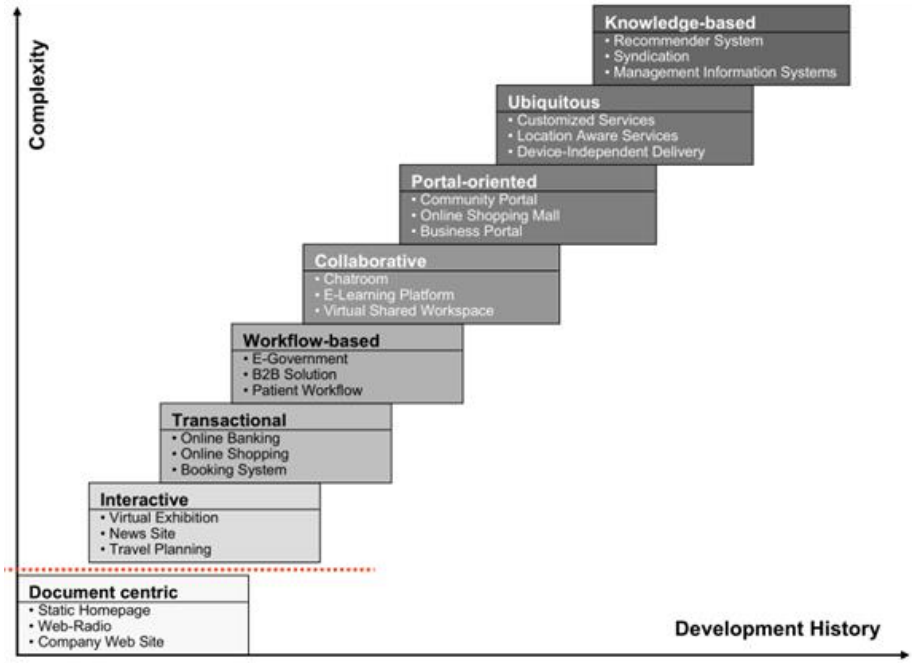
The definition of Web Application is a very similar way as the Web 2.0. We can find several articles and papers about it and its a challenging task to approve only one for it. In that way, we can use the outline from Wikipedia which collects it from several sources and describes it as the following: "A web application or web app is any application software that runs in a web browser or is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a common web browser to render the application".

While this description contains most of the necessary keywords its useful to see the most frequently referenced one too: "A Web application is a *software system* based on *technologies* and *standards* of the World Wide Web Consortium (W3C) that provides *Web specific resources* such as *content* and *services* through *a user interface*, the Web browser." - Source: Kappel, G., Pröll, B., Reich, S., Retschitzegger, W. (eds.), Web Engineering – Systematische Entwicklung von Web-Anwendungen, dpunkt.verlag, 2003 (in German).

It implies two things, one from the software aspect, saying that static HTML pages are not Web applications, and one from the interface aspect, outlining that Web services are not Web applications.

The evolution of the Web implies form its nature to include software solutions for serving content. As we seen in the earlier chapters, the third phase of Web 1.0 reached that level where we can say Web applications are present because they are not only simple "HTML file serving" solutions but complex multidisciplinary applications. Web 2.0 implies greater collaboration between websites, and greater interaction with website users. In fact, website might be the wrong term; as more and more functionality is provided via the browser, Web application is becoming a better description.

Web application categories



Web Application categories

On Figure 9 we can see the most widely used categories for Web applications. The development of a Web application can be started in any of these categories but complex Web applications can be assigned to several categories at once, e.g. Online Shopping Malls where transactional, workflow based and collaborative aspects could be founded at the same time. Naturally, new fields of applications are continuously created, e.g., location-

dependent services are appeared just in the recent past. Newer categories are generally more complex but this does not mean they can fully replace the older generation and each has its own specific fields of application, e.g. Mashups which will be discussed int the next chapter as a new category near by the portals.

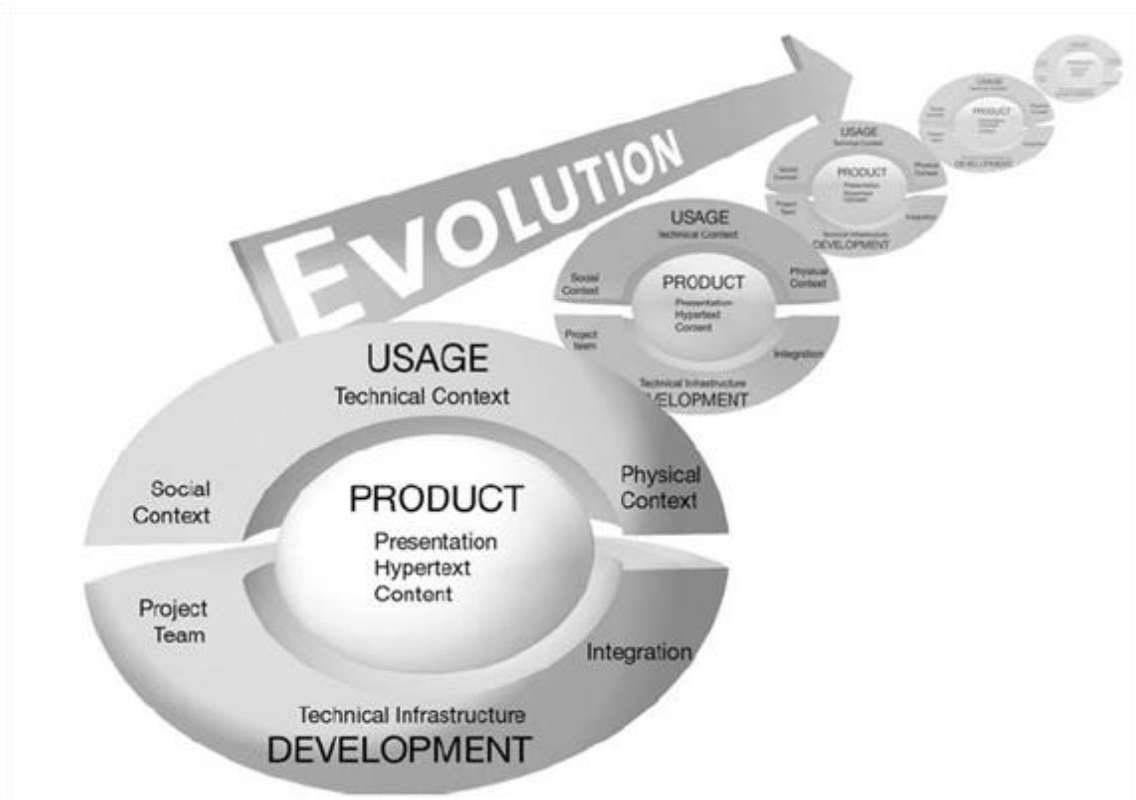
While the first category of them means the read-only web, the second category made the shift to the real web application development. This state could be treated as the starting point of the web applications. However, these category only means the usage of Common Gateway Interface (CGI) and HTML forms which offer a first, simple form of interactivity where pages and links to other pages could be generated dynamically according to user input. The real expansion arrived when more interactivity was achieved by the support of the database system forming the transaction-oriented part of the apps which made available the online transactions, e.g. online banking, shopping or booking.

All the remaining categories are really complex solutions are requires complex business solutions. Workflow-based Web applications allow the handling of workflows within or between different companies, public authorities, and private users where the appropriate Web services are guarantee the interoperability. This is the place where the e-... prefix is mostly applied: e-commerce, e-government, e-health, etc. Collaborative Web applications support cooperation in case of unstructured flow of activities and high degree of communication, like wikis, chat rooms or e-learning solutions. Portal-oriented Web applications provide a single point of access to separate, potentially heterogeneous sources of information and services.

The highest two categories are representing a more complicated ones. Ubiquitous Web applications provide customized services anytime, anywhere, for any device – this is the “ubiquitous access”. It has a very important precondition: knowledge of the context in which the Web application is currently being used in order to make dynamic adjustments to the Web application. Sadly, existing Web applications usually offer a limited form of ubiquity. Knowledge-based Web applications are forming the top and currently visionary layer where the goal is to present information on the Web not merely for humans, but in a machine processable form. Facilitate knowledge management on the Web based on Semantic Web technologies.

Characteristics of Web Applications

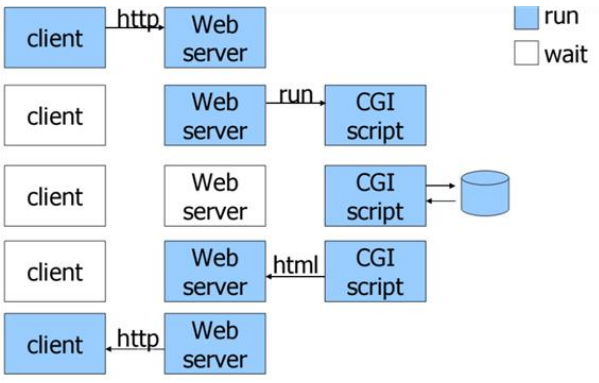
Web applications differ from traditional, not Web-based applications, in a variety of features. There are ones that traditional applications lack completely (e.g. non-linear navigation) or that are particularly pronounced in Web applications (e.g. frequency of updates). However, the presence and strength of a certain characteristic depend partly on the type of Web application, e.g., e-commerce systems vs. digital libraries. It means that proven methods from other related disciplines (e.g., software engineering or Hypermedia) are totally inadequate and thus new solutions have to be developed and have to be adapted to the needs of Web Engineering. All of these characteristics are well-formed on the following figure which is from the ISO/IEC 9126-1 standard for the evaluation of software quality.



ISO/IEC 9126-1

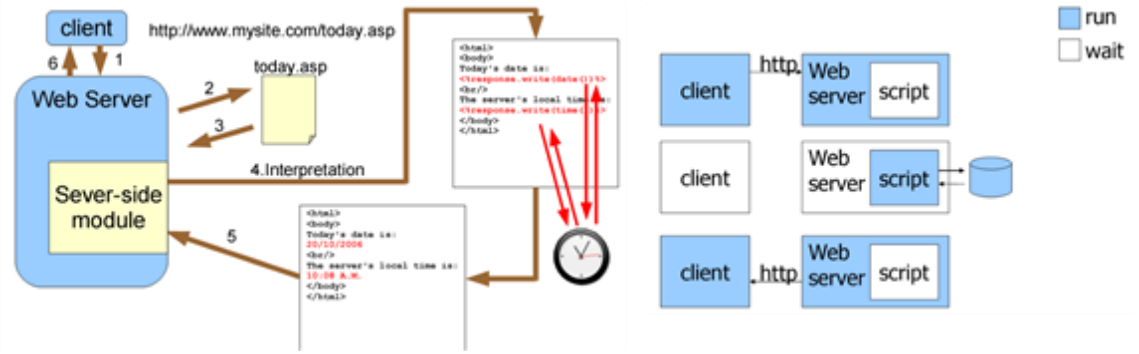
Traditionally, Web applications can be considered as a specific variant of client-server software where the client software is downloaded to the client machine when visiting the relevant web page, using standard procedures. In the early days of the Web each individual web page was delivered to the client as a static document, but from 1995, when Netscape is released its client-side scripting language called JavaScript, the World has been changed. Instead of sending data to the server in order to generate an entire web page, the embedded scripts of the downloaded page can perform various tasks. A little bit later an other technology is appeared to raise the user experience in a higher level, that was the Flash player plug-in from Macromedia. The real "Web application" term was introduced by Servlet specification of the Java language in 1999.

The evolution of the technologies could be seen in the following figures. The first one highlights the traditional Web 1.0 solution, the shift to the first dynamic webpages using CGI scripts:



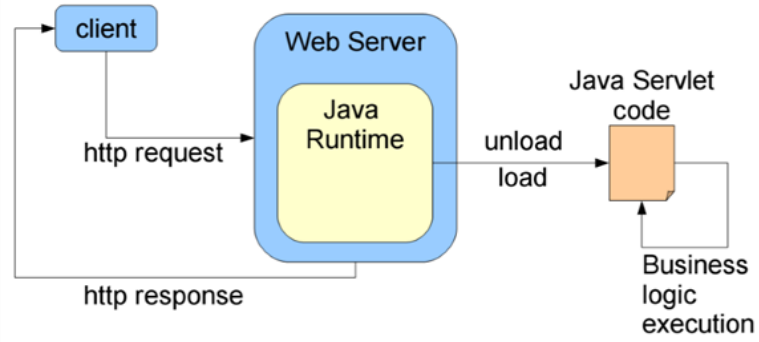
Web 1.0 phase 1

The second phase was the introduction of the server side programming languages:



Web 1.0 phase 2

The third phase was the introduction of the application servers:



Web 1.0 phase 3

At the first times that was a good strategy for application software companies to provide web access to software previously distributed as local applications. Depending on the type of application, it may required the development of an entirely different browser-based interface, or merely adapting an existing application to use different presentation technology. However, legacy application is not always extendible for Web presence, and if so, than the solution could be very fragile. In practice, we could found that a scalable and responsive Web application need to use Web technologies from the beginning rather than extending existing solutions.

Writing of web applications is often simplified by web application frameworks. These frameworks facilitate rapid application development by allowing a development team to focus on the parts of their application which are unique to their goals without having to resolve common development issues such as session handling, workflows or simple cases as user management. While many of these frameworks are open source, this is by no means a requirement.

The use of web application frameworks can often reduce the number of errors in a program, both by making the code simpler, and by allowing one team to concentrate on the framework while another focuses on a specified use case. Frameworks can also promote the use of best practices. The general description of Web applications and its supporting architecture could be found in the following chapter.

Advantages

- Web applications do not require any complex "roll out" procedure to deploy. A compatible web browser is all that is needed;
- They require no upgrade procedure since all new features are implemented on the server and automatically delivered to the users;
- Web applications could work together with other server-side solutions.
- They also provide cross-platform compatibility in most cases (i.e., Windows, Mac, Linux, etc.).

- With the advent of HTML5, programmers can create richly interactive environments natively within browsers. Included in the list of new features are native audio, video and animations, as well as improved error handling.
- Modern web applications support greater interactivity and greatly improved usability through technologies such as AJAX that efficiently exchange data between the browser and the server.
- Web applications allow for easier introduction of new user devices (e.g. smartphones, tablets) because they have built-in browsers and responsive themes could adopt the new layouts.

An enterprise level Web application involve groups including software engineers, database experts and Web designers as well. Modern solutions are strictly separates the responsibility between the teams but in the first times it was not so clear and different parts could be mixed in one file - like the early versions of ASP or JSP.

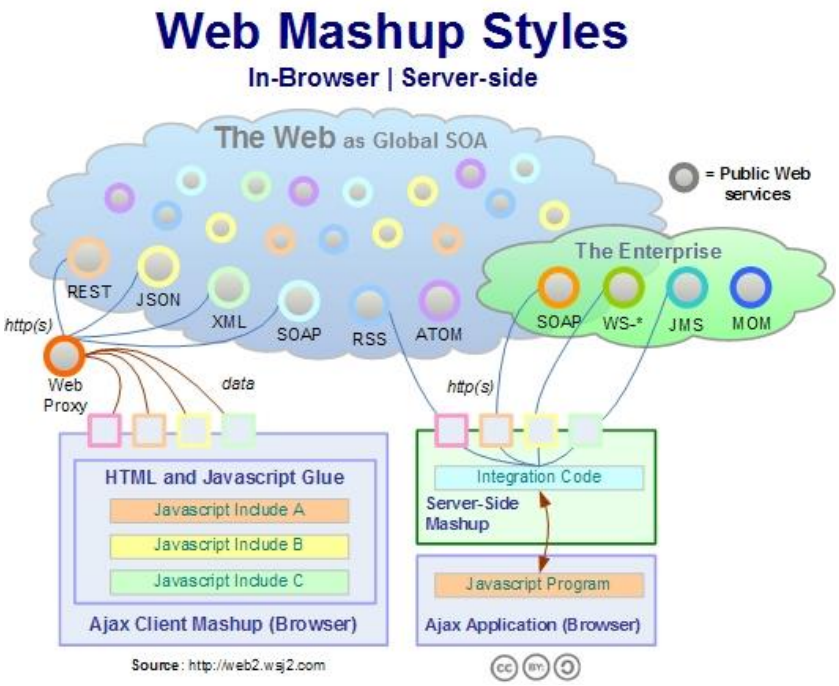
2. Mashups

The term mashup - as the original term from the music world - describes a Web application that combines multiple external sources (services) into a single application. The external sources are typically other web sites and their data may be obtained by the mashup developer in various ways including, but not limited to: APIs, XML feeds, and screen-scraping. Its like a bit of a buzzword in the Web 2.0 era. It's frequently mentioned in the same context as cloud computing and Web 2.0 while we can access data and tools in the cloud. The primary purpose of most Web mashups is to consolidate information with an easy-to-use interface. Because the combinations of Web applications are limitless, so are the possibilities of mashups. For example, Yahoo offers a mashup called Yahoo! Pipes that aggregates RSS feeds into a single page that can be navigated using a graphical interface. MyWeather.com offers weather information, while Amazon could serve information about new books and Youtube could be used to serve videos and Last.fm to include music recommendation. There are plenty of Mashup tools and editors that really allow users to start playing around with this possibilities. Internet mashup websites are *quickly growing for 2007-2008*. As of January 2008, there are approximately 12 new internet mashups launched each day. At this time, only a fraction of new mashups achieve significant popularity, but mashups are definitely here to stay. However, not all of the them remained online, Microsoft was closed its visual mashup creator (Popfly) in 2013 - 5 years later from its start. Google also closed its Mashup Editor in 2009, but they provided a solution for the users, they introduced the Google App Engine - as they say: Platform as a Service (PaaS).

An other well-known example from Google is the Maps API, where e.g. a Web forum could display what parts of the world the users are posting from but it can be used to rate areas in a city, delineate points of interest, or show roads that are undergoing construction. Google Maps has spawned thousands of mashup applications. In the previous section we also used it to show an example for the way to Web 3.0 which goes through APIs. This highlights that APIs are key stakeholders currently in the evolution of the Web.

A mashup is a technique by which a website or Web application uses data, presentation or functionality from two or more sources to create a new service. Mashups are made possible via Web services or public APIs that (generally) allow free access. Most mashups are visual and interactive in nature. To a user, a mashup should provide a richer, more interactive experience. To a developer, a mashup is beneficial because it requires less code, allowing for a quicker development cycle.

The overall concept could be seen in the following figure:



Mashups

Characteristics of mashups

We could infer the following outstanding characteristics of mashups, based on the concepts mentioned above:

- A mashup is *focused on adding value*, putting a tremendous amount of ready information at the fingertips of the final user.
- *Information + User experience should be efficiently combined*, using technologies such as *RIAs*.
- *It prioritizes the information in the order in which is most likely to be useful*. You have to know who you are talking to, what products they have registered, what the top service items are for those products, and then start trying to answer questions that they may have such as where the local service centers are for the customer's location.
- *Developing* a mashup must be measured in *hours or days*.
- Generally, it is *read-only*.
- Mashups are *agile views* into the data that they present. They are not an all-powerful editing surface capable of editing any and all data thrown at them. If someone wants to edit that data, they should go back to the application that created the data to do the editing. A *timestamp* may be important for improving the clarity and usefulness of the mashed data.

The 3 categories or the 5 styles of mashups

As we can see these could be combined at multiple levels in an application stack means that there are (at least) five places that mashups can take place. These five styles are:

- *Consumer or Presentation Mashup:* Information and layout is retrieved and either remix or just placed next to each other. Combine similar types of media and information from multiple sources into a single representation.
- Data Mashups:
  - *Client-Side Data Mashup*: Takes information from remote Web services, feeds, or even just plain HTML and combines it with data from another source. New information that didn't exist before can result such as

when addresses are geocoded and display on a map to create a visualization that could exist without the underlying combination of data.

- *Server-Side Data Mashup*: Databases have been linking and connecting data for decades, and as such, they have relatively powerful mechanisms to join or mashup data under the covers, on the server-side. While it's still harder to mashup up data across databases from different vendors, products like Microsoft SQL Server increasingly make it much easier to do. This points out that many applications we have today are early forms of mashups, despite the term. Of course, the more interesting and newer aspects of mashups happen above this level.

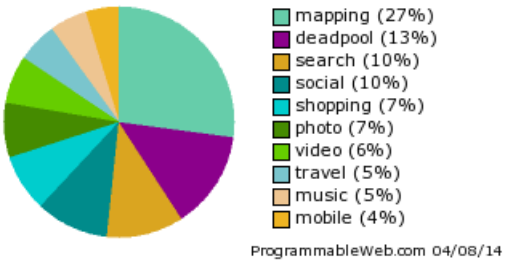
- Business Mashups:

  - *Client-Side Software Mashup:* This is where code is integrated in the browser to result in a distinct new capability. While a component model for the browser is only now being hashed out, there is considerable potential in being able to easily wire together pieces of browser-based software into brand new functionality.

  - *Server-Side Software Mashup*: A better place for integrating software since Web services can more easily use other Web services and there are less security restrictions and cross domain issues. As a result, server-side mashups like those that in turn use things like Amazon's Mechanical Turk or any of the hundreds of open Web APIs currently available, are quite common.

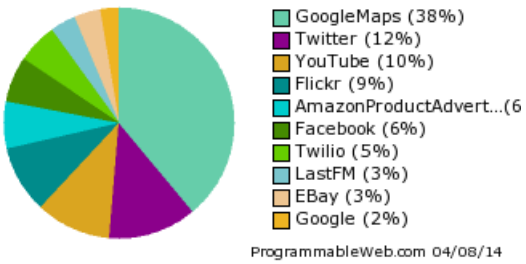
Mashup uses are expanding in the business environment. Business mashups are useful for integrating business and data services, as business mashups technologies provide the ability to develop new integrated services quickly, to combine internal services with external or personalized information, and to make these services tangible to the business user through user-friendly Web browser interfaces. Business mashups differ from consumer mashups in the level of integration with business computing environments, security and access control features, governance, and the sophistication of the programming tools (mashup editors) used. Another difference between business mashups and consumer mashups is a growing trend of using business mashups in commercial software as a service (SaaS) offering. Many of the providers of business mashups technologies have added SOA features.

The presence of these mashups can be easily followed by the http://www.programmableweb.com website which collects all the known mashups and APIs into one central place. It is useful to get some statistics as well. The following figure is listing the mostly used tags for mashups:



Top Mashup categories

followed by the most used APIs:



Top Mashup APIs

Everybody may have a feeling what are the most common usage of the mashups and APIs. All of them try to add some new information to the user but finding the relevant ones are not an easy task. The website shows

around 7500 distinct mashups and the average Mashups/Day is 1.5 at the time of this writing. Hard to follow these sites but we can found interesting ones, e.g. one of the highlighted mashup was the Audience Finder. The Audience Finder tool lets you find Facebook users that can be part of a custom audience for your social media marketing campaigns. This way, you can create an ad campaign that is very targeted towards people who are, right now, likely to be interested in your product or service. This is only one usage scenario where mashups and APIs could help us.

Architectural aspects of mashups

Some places refer to mashups as **web applications that combines data from more than one source, hiding this behind a simple unified graphical interface**. Despite for me this is true, I prefer another definition which widens the scope, referring to a mashup as a technique, as an **architectural style for building applications that combine data from multiple sources to create an integrated and improved experience to the user**.

The architecture of a mashup is divided into three layers:

- Presentation / user interaction: this is the user interface of mashups. The technologies used are HTML/XHTML, CSS, Javascript, Asynchronous Javascript and Xml (Ajax).
- Web Services: the product's functionality can be accessed using API services. The technologies used are XMLHTTPRequest, XML-RPC, JSON-RPC, SOAP, REST.
- Data: handling the data like sending, storing and receiving. The technologies used are XML, JSON, KML.

Architecturally, there are two styles of mashups: Web-based and server-based. Whereas **Web-based mashups** typically use the user's Web browser to combine and reformat the data, **server-based mashups** analyze and reformat the data on a remote server and transmit the data to the user's browser in its final form.

*Mashups appear to be a variation of a **façade pattern***. That is: a software engineering design pattern that provides a simplified interface to a larger body of code (in this case the code to aggregate the different feeds with different APIs). Mashups can be used with software provided as a service (SaaS). After several years of standards development, mainstream businesses are starting to adopt service-oriented architectures (SOA) to integrate disparate data by making them available as discrete Web services.

Web services provide open, standardized protocols to provide a unified means of accessing information from a diverse set of platforms (operating systems, programming languages, applications). These Web services can be reused to provide completely new services and applications within and across organizations, providing business flexibility.

The need for the mix: (REST + Traditional SOA) * Web 2.0

In summary, we need the mix before Mashups really take off. I believe both REST and 'Traditional SOA' will work together alongside all the other various ways of getting content on the web to achieve this. Traditional SOA Web Services + RESTful oriented architecture Web Services + RSS Feeds + Social Context + Screen Scraping = Future Mashups.

3. Mashups versus Portals

It is important to differentiate a mashup from simple (external site) data embedding. For instance, if I have a site that allows to see a video in YouTube I do not have a mashup, that is simple embedding. **Mashups are intended to add value to the final user, processing information consumed by any API/Web Service and displaying it in an integrated experience**.

When the whole concept of Portals first come out, the whole idea was that we could draw content from various sources using little Portlets, and personalize it to what we want. Portals are providing a platform to leverage off these services. If you think of Facebook as a platform, and all the widgets/gadgets as what makes the overall content, you have a mashup. Through a portal-like interface you have brought in content from many sources: Web Services (SOAP, XML-RPC), REST, RSS Feeds and even Screen Scraping. Google also had its own portal-like solution with iGoogle (aka Google Start Page). iGoogle was a Portal platform that left you add and configure your own widgets/gadgets (aka Portlets) to do what you want. However, Google shutdown iGoogle

after 8 year of usage in 2013 because the browsers are evolved to act as an application platform and its own Chrome browser could fulfill all the tasks what iGoogle did.

Both Mashups and portals are content aggregation technologies. Portals are an older technology designed as an extension to traditional dynamic Web applications, in which the process of converting data content into marked-up webpages is split into two phases: generation of markup "fragments" and aggregation of the fragments into pages. Each markup fragment is generated by a "portlet", and the portal combines them into a single webpage. Why it is different form the mashups that portal technology defines a complete event model covering reads and updates. Portal technology is about server-side, presentation-tier aggregation. A request for an aggregate page on a portal is translated into individual read operations on all the portlets that form the page.

In 2009, Gartner introduced the concept of the portal-less portal or the “lean portal”. Gartner defines a portal as

> a Web software infrastructure that provides interaction with relevant information assets (for example, information/content, applications and business processes), knowledge assets and human assets by select targeted audiences, delivered in a highly personalized manner."

It’s built using modern Web 2.0 technologies, such as AJAX, widgets, representation state transfer (REST) and WOA/SOA approaches. Lean Portal offerings from vendors like IBM WebSphere, Oracle WebCenter and Liferay replace the traditional container-oriented portal model while maintaining the main purpose of a portal — providing a personalized point of access that allows customers to find relevant information, read about business processes and reach people. According to Gartner, organizations who opted for a Lean Portal found that it delivered more than 80% of the required functionality within months of launching, without compromising security or advanced integration requirements.

While both portals and mashups are supporting integration of content provided by other websites and presentation, portals are offering more things like navigation between portlets, access control and personalization. The list of the advanced features are:

- *Single Sign-On* — enterprise portals can provide single sign-on capabilities between their users and various other systems. This requires a user to authenticate only once.
- *Access Control* — the ability for portal to limit specific types of content and services users have access to. Access control lists manage the mapping between portal content and services over the portal user base.
- *Integration* — the connection of functions and data from multiple systems into new components/portlets with an integrated *navigation between these components*.
- *Business process management* (BPM) and a mechanism for providing or integrating with workflow and BPM tools or platforms.
- *Federation* — the integration of content provided by other portals, typically through the use of WSRP or similar technologies.
- *Customization* — Users can customize the look and feel of their environment. Customers who are using EIPs can edit and design their own web sites which are full of their own personality and own style; they can also choose the specific content and services they prefer.
- *Personalization* — Personalization is more about matching content with the user. Based on a user profile, personalization uses rules to match the "services", or content, to the specific user. To some degree, you can think of the two like this: customization is in hands of the end user, personalization is not.
- Support for *multichannel and multi-device delivery*.

Moreover, not just these aspects are made the separation between them, the following table form the crowdsourced Wikipedia also highlights the primary differences:

	Portal	Mashup
Classification	Older technology, extension of traditional Web server model using well-defined approach	Uses newer, loosely defined "Web 2.0" techniques

Philosophy/approach	Approaches aggregation by splitting role of Web server into two phases: markup generation and aggregation of markup fragments	Uses APIs provided by different content sites to aggregate and reuse the content in another way
Content dependencies	Aggregates presentation-oriented markup fragments (HTML, WML, VoiceXML, etc.)	Can operate on pure XML content and also on presentation-oriented content (e.g., HTML)
Location dependencies	Traditionally, content aggregation takes place on the server	Content aggregation can take place either on the server or on the client
Aggregation style	"Salad bar" style: Aggregated content is presented 'side-by-side' without overlaps	"Melting Pot" style - Individual content may be combined in any manner, resulting in arbitrarily structured hybrid content
Event model	Read and update event models are defined through a specific portlet API	CRUD operations are based on REST architectural principles, but no formal API exists
Relevant standards	Portlet behavior is governed by standards JSR 168, JSR 286 and WSRP, although portal page layout and portal functionality are undefined and vendor-specific	Base standards are XML interchanged as REST or Web Services. RSS and Atom are commonly used. More specific mashup standards such as EMML are emerging.

4. Cloud Computing

Nowadays, a new model is appearing, this is cloud computing. Most businesses today do not need to buy data center hardware such as servers because they are affordably rented on a short term basis from a hosting company that provide turnkey implementations of web applications. In cloud computing model web applications are software as a service (SaaS). There are business applications provided as SaaS for enterprises for fixed or usage dependent fee. Other web applications are offered free of charge, often generating income from advertisements shown in web application interface. We could think about some popular sites which are using this philosophy, like DropBox, Google Applications, ... etc.

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. The data center hardware and software is what we will call a cloud. When a cloud is made available in a pay-as-you-go manner to the general public, we call it a public cloud; the service being sold is utility computing. We use the term private cloud to refer to internal data centers of a business or other organization, not made available to the general public. However, this topic could serve as a standalone theme for a book, so its not discussed further here.

5. References

http://www.informit.com/articles/article.aspx?p=1326602

http://www.programmableweb.com/mashups

http://en.wikipedia.org/wiki/Mashup

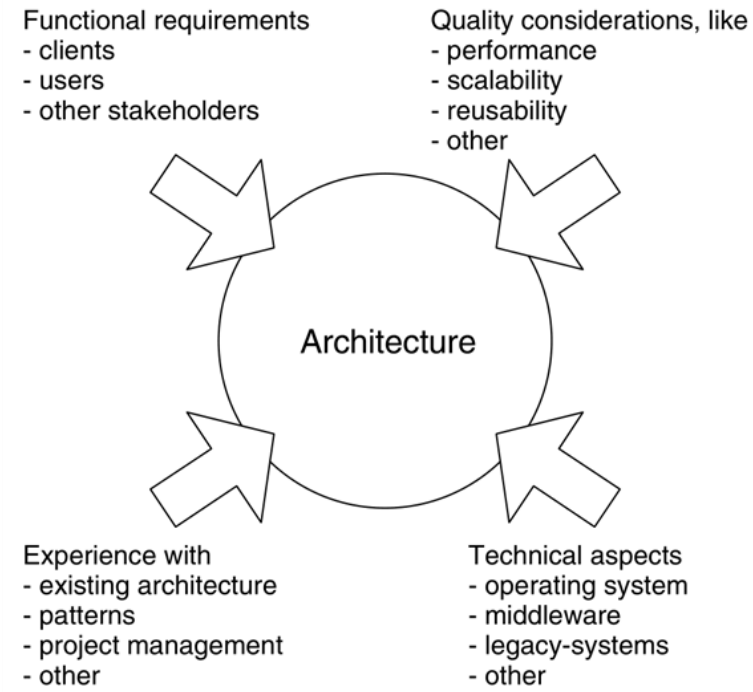
http://www.zdnet.com/blog/hinchcliffe/is-ibm-making-enterprise-mashups-respectable/49

Part II. Architectures for the Web

Architecture is defined as the structure of components, their relationships, and the principles and guidelines governing their design evolution over time. According to (Bass et al. 1998), the architecture of a software system consists of its structures, the decomposition into components, and their interfaces and relationships.

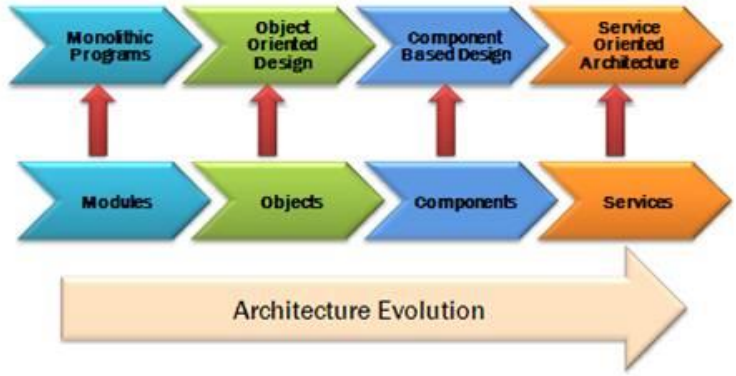
When we create architecture we try to break the functional requirements and quality requirements down into software components and their relationships through interfaces using an iterative approach. Depending on the point of view, we can emphasize and itemize different architectural aspects. Structuring software systems and breaking them down into different perspectives allows us to better manage the complexity of software systems, and the systems become easier to understand. In addition, the abstraction of system aspects facilitates the outline of possible and important architectural issues.

Considering the above properties of architecture we can easily see that architectural decisions are of enormous importance for the development of Web applications. The requirements of software and thus its architecture are subject to change. Technical and organizational constraints change during and after the development of an application. This is the reason why software systems are often referred to as "moving targets". Due to the changes in capability and performance objectives, interests of different stakeholders, technology improvements, and unanticipated situations, existing architectures may need to evolve to meet new capability requirements and constraints. The following figure shows the different factors and constraints influencing the development of an architecture according to (Jacobson et al. 1999).



Factors influencing the development of an architecture

The evolution of distributed architecture can be considered a sophisticated extension of client/server architecture because the Internet and World Wide Web caused a shift in the nature and ways of business transactions in place today. With this evolution, especially when it concerns to the Web, came a large quantity of services and tools. As these services and tools grew up in uses and complexity they came to be the subject of software engineering, a topic that was well developed by the time this applications became used. The evolution of architectures could be imagined with the following figure:



Evolution of Architectures

A number of architectures for specific requirements in several application domains have been developed in the past few years. The most widely used aspect is the layered view. Layering means that software systems are structured in several tiers to implement the principle of "separation of concerns" within a software system. Many frameworks in the field of distributed systems and Web applications are primarily structured by the layering aspect, which will be discussed in the next chapter.

The increasing distribution of software systems has led to the development of architectures and infrastructures addressing the distribution of data and messages. We could found several proposals to achieve an efficient way in this distributed nature. The most well-known architectures are:

- *Distributed Object Middleware (DOM):* This type of infrastructure allows to access remote objects transparently. It is based on the Remote Procedure Call (RPC) mechanism. Some DOM systems also enable objects on different platforms to interact (e.g., CORBA). Other examples of this type of system include Microsoft's DCOM (Distributed Component Object Model), or EJB (Enterprise Java Beans) by Sun Microsystems.

- *Message Oriented Middleware (MOM):* MOM systems offer functionalities for asynchronous transmission of messages. Asynchronous communication are sent to the receiver regardless of its status, e.g., the receiver may not be available when the message is sent. MOM ensures that messages are delivered nevertheless. Examples of MOM systems include Sun's JMS (Java Messaging Service) and Microsoft's MSMQ (Microsoft Message Queue).

- *Peer to Peer (P2P):* P2P stands for direct communication between two devices – the peers – in a system without using a server, i.e.,they communicate over a point-to-point connection. The peers are basically equal. P2P systems describe how the devices in such a network communicate and how they can "discover" each other.

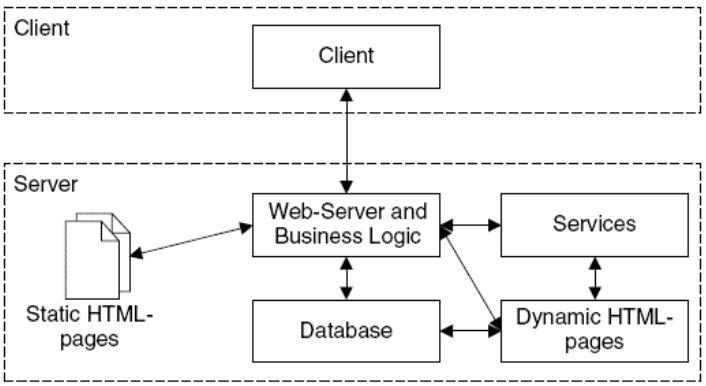
- *Service Oriented Middleware (SOM):* SOM enhances DOM systems by the concept of services. A service in this context is a number of objects and their behavior. These objects use a defined interface to make a service available for other systems/services. SOM defines communication protocols between services, and provides for location- and migration-transparent access to services, thus supporting a simple integration of services beyond platform boundaries. One example of a SOM is Sun's Jini system (http://www.sun.com/software/jini/). Architectures emerging within the field of Web services also belong to this category.

These architectures are applicable to distributed systems in general, which means that they are not limited to Web applications. Similarly, integration architectures address integration aspects on the content level and the application logic level and are commonly summarized under the term Enterprise Application Integration (EAI) architectures. This category also includes architectures which integrate existing applications as a whole. Alternatives to EAI are Web services which support the integration of services, i.e., application logics and contents. On the presentation level, a set of different systems is typically integrated by using portal architectures.

Chapter 4. Layered Architecture for Web Applications

The first question that should be answered: Why the web is suitable for developing applications? It is not a difficult question, of course. In the World Wide Web Consortium (W3C) Architecture of the Web Recommendation paper various examples are given, notably one about a user who wants to see the weather in a place where she wants to travel to. The nature of the Internet - trafficking data over protocols from network to network - is a powerful resource that make communication between different places very fast and easy. This way computer programs can be made to improve the relation between systems and people, and what is seen today is that it happened.

Web applications are distributed applications, and hence are at least two-tiered. They act as a special kind of client-server applications, where a large portion of the functionality is “pushed” back to the server side despite the fact that the Web does not define what is behind the server.

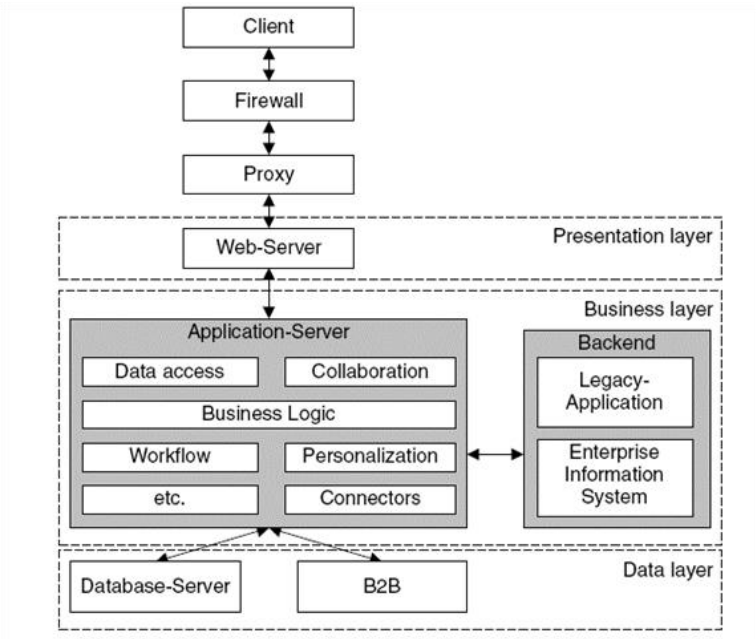


The basic two parts for Web Applications

The Web relies strongly in the client-server model, and it uses markup languages such as HTML and XML to transfer and represent data. Under it there are many programming and scripting languages that can dynamically process, modify and generate data, or give an user interface. This way, the development of Web applications can be put under the cover of software engineering but need to be extended. Web applications are multidisciplinary (software engineering, database modeling techniques, network computing, and effective interface design). They are built in a continuously changing environment where requirements are unstable and the user community is wider than before. Web applications handle information from various sources (text, graphics, video, audio) dealing with structuring, processing, storing and presenting this information.

1. The Three Layers Model

The nature of the Web is layered: it has formats over protocols and uses a client-server model. Therefore, it is natural that a layered architecture would be suitable for developing to the Web. We learnt that this model overcame the two layered client-server because of its scalability. Many different approaches to the aim of developing applications with different layers had been used along the years, but a clear pattern seems to appear frequently in various of them: the Three Layers Model (according to Kappel et al. 2006).



The standard three layered architecture for Web Applications

This model of web application development is very similar to the Service Layer/Domain Model/Data Source Layer set of design patterns from Martin Fowler's collection, but receiving different names . In fact, the idea (usually named 3-tier architecture, or expanded into n-tier architecture) is very general and widespread, so in this paper only the most common assumptions and uses are examined.[1]

Its conception is three layers, one over the other, being the application the set of them working together. The most external of them is the View Layer, that is the visible part of the application, the one that interacts with the user. The layer in the middle is the Business Logic Layer, which serves as an intermediate between the View (or presentation) and the innermost layer, that is the Data Layer. This last one is where all the data used by the application is stored.

The benefits of using layered models in software development are in the way that it is easier to know exactly what each part of the application does. It makes easier to construct the application, to debug it, and to maintain and reuse the code. For example, if it is needed to exchange some details in the presentation of the content but the business rules and data models do not change, only one layer is affected. Even for more complicated changes involving all of the application architecture there are benefices, so a plan can be created in the overall but specifying exactly where the changes need to be done.

1.1. The View Layer

The outermost layer in this kind of model deals with the presentation of the content and interaction with the user. It can be called view, presentation, UI. In this layer the application shows to the user what is needed to be seen and gives the tools for interaction. The exact kind of interaction depends on the application; one can create a web app that only shows information to the user without any kind of interaction, not even hyperlinks to be clicked, but such a case does not need an advanced architecture. In most cases the user will generate some input, send it for processing and then receive a feedback, that can be the final result or a step for further operations.

Following the example by W3C of the user that wants to see the weather in her trip destination, the presentation is where she sees the actual content. The content display is shown, and the user can interact with the provided controls to, for example, see weather in different periods of time or another places, and see pictures of it.

The technologies usually involved in this layer on the web development context are mainly the markup that is processed by the browser (HTML/ XHTML/ ...), the style of the page (CSS) and client-side scripts (Javascript/ Flash/ ...). All of these tools together can produce a rich environment for user interaction and content display. Of

[1]Sometimes tier is different from layer. For it, layers mean the logical distinction of the applications integral parts, and tiers mean the physical structures in where the application runs (networks, computers, servers). But in this work the two words are used as synonyms, and with the meaning of layer, not being the objective to discuss physical tiers.

course it can be said that server-side scripts can be used to generate content, but at the final level these scripts produce the HTML that will be shown be the browser, so this role of the development can be subdivided: the content generation is created by the business logic layer (the next topic to be discussed) and then it is passed to the view layer, maintaining the logical division of the application. The browser shows the content initially written or produced by the server-side scripts, and the client-side scripts are able to modify that content. A Javascript code, for example, can be used to validate form data or even to create drag and drop interfaces. It interacts with HTML through a DOM tree, that is a representation of the document in memory. HTML5, the present (2014) trend for web development is praised for its flexibility, specially where it touches the concept of responsiveness, that is the ability to change the content disposition according to the screen size. This matters because, in current days, the availability of a page in different screen sizes and devices is extremely important. Having many possibilities like desktops, tablets, smartphones, wearable devices and even augmented reality or voice user interface, the range of technologies and targets for the view layer is very wide, and it shows both the importance of it to the user and reinforce the need of a logical division of the application for supporting such variety.

This layer communicates with the business logic layer under it, passing the information from the user and controlling it, then giving back any response it produces, not leaving any decisions of the application's logic to be resolved by the UI. This passing of information is usually done through forms, like a user log-in in a system by giving username and password, but there are other ways. AJAX is an asynchronous way to pass information to the server and get responses. The cited a-synchronicity comes from the fact that in a form the content needs to be passed and then the response will come after a page refresh, but with AJAX the requested information, that is the result of the user's action will come in the actual page. It saves time and gives to the user the impression that the application is really interacting with him.

1.2. The Business Logic Layer

The central layer of the model deals with the logic of the program. It receives data from the upper level and transforms it, using in the inner application logics. It also retrieves data from the deepest data level and uses it to the logics. And, by integrating these two processes, it can do modifications in both levels as well.

The Business Logic Layer contains the determinant part of the application logic. It includes:

- performing all required calculations and validations
- managing workflow
  - state management: to keep track of application execution
  - session management: to distinguish among application instances
  - user identification
  - service access: to provide application services in a consistent way
- managing all data access for the presentation layer

The Business Logic Layer is generally implemented inside an Application server (like Microsoft Transaction Server, Oracle Application Server, or IBM WebSphere). The Application server generally automates a number of services like transactions, security, persistence, connection pooling, messaging and name services.

Using the same example of the last session, of the user that wants to see the weather in a specific place, when the information is given by the user the application retrieves it and process. For example, the user wants to see the weather forecast for two days. The application receives its request from the UI and the data is sent to the server. A PHP script catches it and then make the calls for the lower level to get the needed data. When a response comes, being it the desired information or a failed request, it is dealt and then prepared to be sent again to the upper level.

The tools used in this level are usually server-side scripts like PHP,ASP.NET, Ruby or CGI. There is even a solution that uses server-side Javascript, called node.js. These technologies, following the information feeding that comes from the upper level, can do any computational pro- cessing that is needed. The CGI (Common Gateway Interface) scripts are an older technology that defines communication way between a server and the content-generated program, in this context called CGI script. One of the most remembered languages when

talking about CGI scripts is Perl. The other languages here cited have a similar approach, by running inside a server. PHP is related to Perl, being as well a scripting language and having similar philosophies. It is one of the most popular languages, being the implementation language of important content management systems as Drupal or Wordpress. Ruby have a large popularity too, especially with the framework Ruby on Rails. Applications as Github or Redmine are built using it. There are many others, of course, and different uses of them, one example being C used as CGI or the Java Server Pages (JSP).

1.3. The Data Layer

The deepest level in the layered architecture, the data layer deals with data retrieval from its sources. It is an abstraction to get the plain data, that can be in a wide variety of forms. Once again, it plays a huge role on the reusability and exchange of technologies: if one data source is changed to another, but the proper data is still the same, a good layered design can help by providing the same data to the upper level with the same interfaces, changing only its inner logic.

In the example given in this paper of the weather forecast, the requirement by the user for the next days forecast will come to this level as a request for the forecasts that it may have. Then a search will be made in the data using the given parameters, and then the data (or some information about not getting it) will be sent again to the upper level.

The technologies used in this layer are database management systems like MySQL or PostgreSQL for relational databases, NoSQL management systems like MongoDB or Apache Cassandra, or even plain XML files or text files. For the management systems usually an API will be used for making queries and retrieving data, and for the plain text ones a script will do the needed operations. Inside it there can be any level of sophistication desired by the application designer, so there can be integrity checks, stored procedures, and virtually anything needed to maintain the data in the desired state.

Deepest in the Data Layer: NoSQL and NewSQL

Inside the Data Layer, as it was outlined, many different technologies can be used. Most of the web applications currently active use relational databases, but now the market is seeing a change of paradigm in the form of the NoSQL. NoSQL is a general way to identify non-relational databases. Fowler summarises some common characteristics that NoSQL databases share:

- Not using the relational model
- Running well on clusters
- Open-source
- Built for the 21st century web estates
- Schemaless

The key points NoSQL supporters use to justify the need for it is that relational databases are not the best solution for any kind of problem, being a problem of its own the uses. They say it is heavy, slow, and non-scalable to use the relational databases, so the use of NoSQL can be a good way to solve these kinds of problems. The use of NoSQL nowadays seems related to startups that use innovating new technology and social web services such as Facebook and Amazon, that have a great amount of data[2] to deal with and have the need to find new ways to use it.

In fact, that is this demand of large data processing. Under the label of big data lies the concept of large quantities of data generated in the last few years and from different sources and in a variety of different formats. The processing of this kind of data leads to a wide range of uses, from healthcare to criminalistics inferences. Of course, new challenges arises with this perspective. The drawbacks come from the nature of the data - massive,

[2]"The increasing amount of data in the web is a problem which has to be considered by successful web pages like the ones of Facebook, Amazon and Google. Besides dealing with tera- and petabytes of data, massive read and write requests have to be responded without any noticeable latency"

HECHT, Robin and JABLONSKI, Stefan. NoSQL Evaluation, A Use Case Oriented Survey.

disperse, heterogeneous. This is why NoSQL can be seen as a solution - it thinks about this kind of problem, trying to solve it.

As of 2014, there are four important categories of NoSQL databases:

- *key-value stores*, that are basically hash tables
- *column family stores*, which aim is to deal with vast collections of data spread amongst many different machines
- *document databases*, versioned documents that are collections of other key-value collections
- and *graph databases*, where the data is presented as a graph, and then it is possible to divide easily into different machines and the queries are more data-specific than the relational ones

A topic that attracts attention when it comes to the issue of scalability and performance of databases is the so-called NewSQL. It is more a way to recognise " the various emerging database products at this particular point in time". The authors writing about it use the term as an identification of vendors that provide SQL databases that are high-performance and scalable, in the market that is also aimed by NoSQL providers. The aims of NewSQL are also related to the Big Data paradigm.[3]

2. The MVC pattern - useful but not a silver bullet

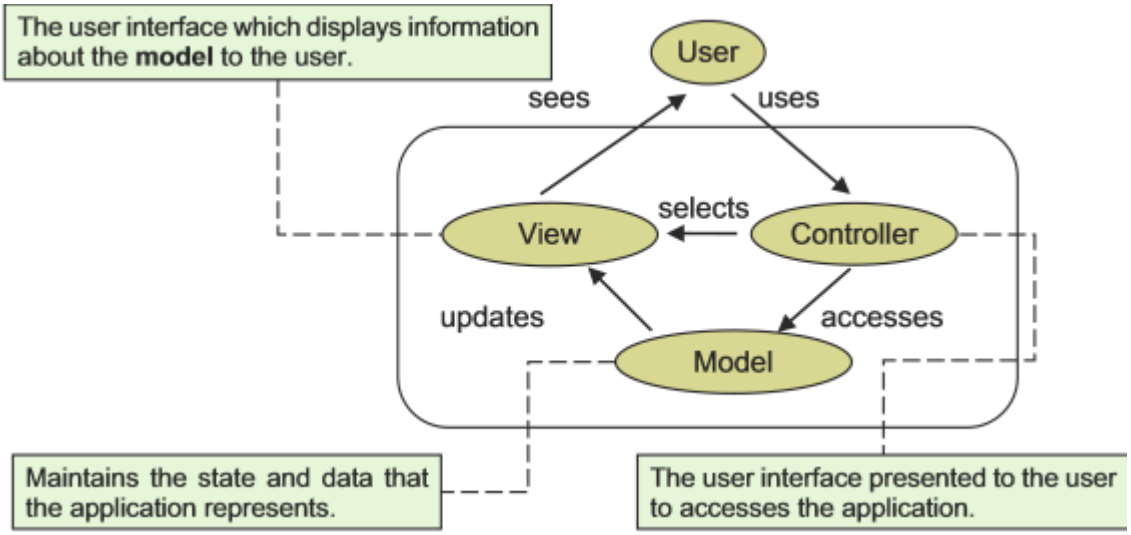
Design patterns try to suggest the way of the application design, while methodologies try to give suitable models for the application and its whole lifecycle. The main idea behind design patterns is to extract the high level interactions between objects and reuse their behaviour from application to application. Moreover, design patterns help to clarify the way that we can think about a Web application.

The MVC architecture has its roots in Smalltalk, where it was originally applied to map the traditional input, processing, and output tasks to the graphical user interaction model. However, it is straightforward to map these concepts into the domain of multi-tier Web applications. It can improve the application's usability, creating reusable code and helping to understand and clarify the functionality of the program. The MVC pattern is very simple, yet incredible useful. It could support:

- Efficient modularity: allows swapping in and out any of the components as the user or programmer desire. Changing one aspect of the program are not coupled to other aspects
- Reusability: it can support the reuse of previously created code if we act sensibly and design carefully.(Reduces risks of bugs coming from refactoring)
- Ease of growth: controllers and views can grow as the model grow
- Centralized controller: a main module is used to make control more manageable

[3]"The NoSQL projects were developed in response to the failure of existing suppliers to address the performance, scalability and flexibility requirements of large-scale data processing, particularly for Web and cloud computing applications. NewSQL and data-grid products have emerged to meet similar requirements among enterprises, a sector that is now also being targeted by NoSQL vendors."

ASLETT, Matthew. NoSQL, NewSQL and Beyond: The answer to SPRAINed relational databases.

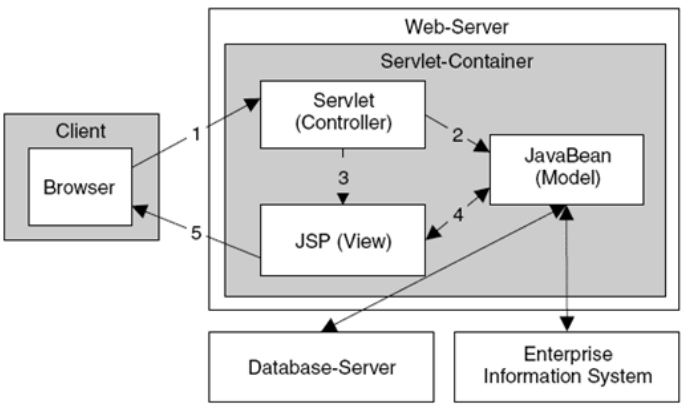


The MVC architecture

- **Model**: The model represents enterprise data and the business rules that govern access to and updates of this data.
- **View**: The view renders the contents of a model. It accesses data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes.
- **Controller**: The controller translates interactions with the view into actions to be performed by the model. In a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

Although MVC is undoubtedly a valuable and useful way to design Web applications, but not the only one. The MVC design pattern's importance lies in how could it help to achieve a clear separation of concerns and functional layers. Create a module for the database portion, create another module for the application code, and create a third module for the presentation code. That way, we can swap and change different elements at will, hopefully without affecting the other parts.

Several vendors have applied this design pattern in their solution. As we have seen that the Web application term appeared in the servlet definition from Sun, here we will show the classic version of the MVC pattern's usage in that context (according to Kappel et al. 2006).



The JSP-Model-2 architecture

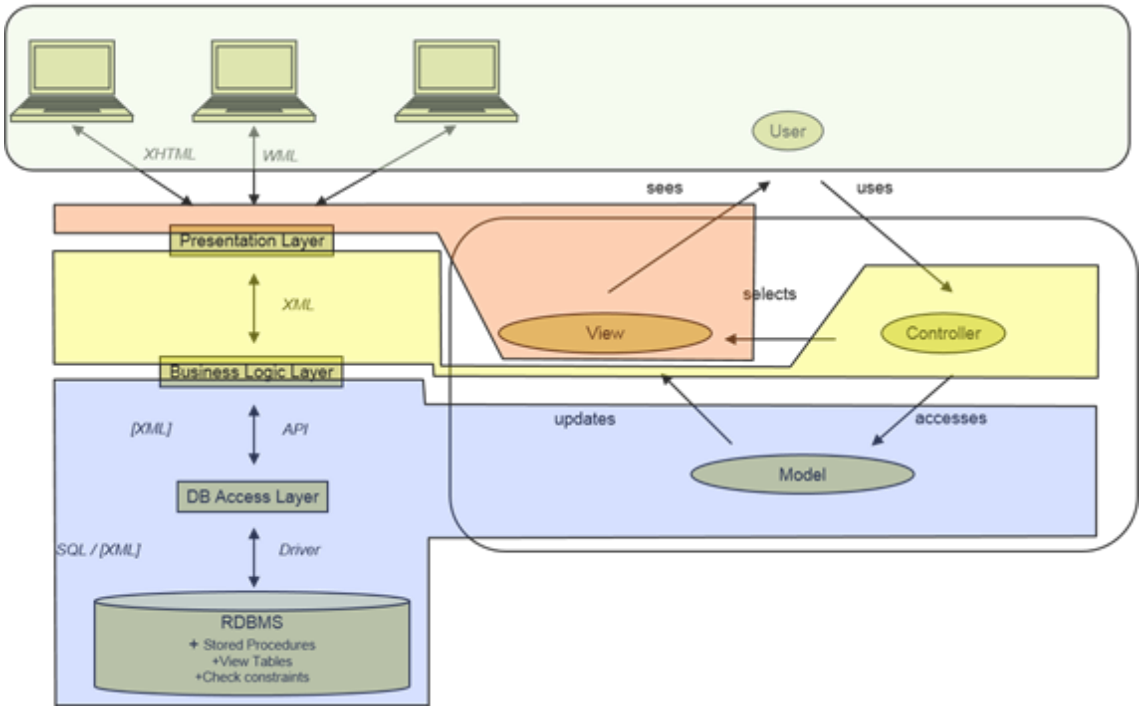
2.1. The Layered Architecture and the MVC Design Pattern

In the context of Web applications, by concentrating too closely on applying the MVC design pattern and nothing else, many other important aspects may be overlooked. This could lead to a fragmented and fragile solution, and as a result, is hard to maintain and execute further development. The MVC is a well known and widely used design strategy, but the problem is, how can we adopt this pattern into the layered model of Web applications?

At first sight the answer could say that we have three layers and three modules, the View module is equivalent with the Presentation layer, the Model module is equivalent with the Data layer, and the Controller module is equivalent with the Business Logic.

But if we take a closer look in the functionality of the Model and the Controller – discussed earlier in the paper – we could find that the Model represents business rules, and the Controller translates the interactions (HTTP GET, POST requests).

These suggest that the Controller is only a part of the Business Logic Layer and the Model forms the other part of the Business Logic Layer. We must take under consideration this heterogeneous composition of the Business Logic Layer through the development of a Web-based application.



MVC and the Layers

It should be obvious by now that MVC or any other design pattern is no more a silver bullet by itself than object-oriented programming is. It's just one part of a much bigger system. We know that MVC has several advantages:

- Clean separation of different functional layers
- Reduces maintenance costs
- Reduces risks of bugs coming from refactoring, graphics redesign
- Presence of a central controller raises overall software security level
- Controller can centrally perform tasks like access logging – without central controller such a task would affect the source code if all business logic actions.

but we also know the drawbacks:

- More software design cost (short term)

- More implementation cost
- Programmers cannot use some comfortable system features
- Programmers may feel they are forced to program a complicated way instead of quickly implementing.

Chapter 5. Architectures for Enterprise Level

Extensive advancements in enterprise computing has been taking place in the last decade and with the advent of the Web it is now possible for organizations to automate and integrate businesses and computer operations. Almost all enterprise organizations face the problem of integrating different applications and database systems at some point. Research has shown that during software development, a third of the time is dedicated to the problem of creating interfaces and points of integration for existing applications and data stores. In situations like this enterprise application architecture becomes very important. **Enterprise application architecture** allows an enterprise to integrate its existing applications and systems and to add new technologies and applications to the mix.

With the emergence of the Web EAI has gone beyond just merging applications within enterprises. Provision of services through the web has rapidly become a trend. External or internal systems, e.g., existing applications, existing databases and interfaces to external business partners, can be integrated into Web applications. There are two patterns that EAI systems implement:

- *Mediation (intra-communication):*

  Here, the EAI system acts as the go-between or broker between multiple applications. Whenever an interesting event occurs in an application (for instance, new information is created or a new transaction completed) an integration module in the EAI system is notified. The module then propagates the changes to other relevant applications.

- *Federation (inter-communication):*

  In this case, the EAI system acts as the overarching facade across multiple applications. All event calls from the 'outside world' to any of the applications are front-ended by the EAI system. The EAI system is configured to expose only the relevant information and interfaces of the underlying applications to the outside world, and performs all interactions with the underlying applications on behalf of the requester.

Both patterns are often used concurrently. The same EAI system could be keeping multiple applications in sync (mediation), while servicing requests from external users against these applications (federation). Enterprise Application Integration is related to middleware technologies such as message-oriented middleware (MOM), and data representation technologies such as XML. Other EAI technologies involve using web services as part of service-oriented architecture as a means of integration. Enterprise Application Integration tends to be data centric. In the near future, it will come to include content integration and business processes. For our point of view, the service-oriented architecture is the one which plays an important role, while this is the most frequently used in Web context.

1. Service-oriented architecture

In the IT industry a frequently used buzzword is ‘Service Oriented Architecture (SOA)’. The Web, IT literature and other resources provide different definitions and interpretations on SOA. Some refer to it as a “Sophisticated Product”; some as an “Architectural Approach” while others treat it as a mere marketing gimmick. Service-oriented architecture is somehow like a design pattern that consists of discrete pieces of software with some functionalities and other applications can utilize these functionalities. This pattern does not require us to use some specific product or a platform. A service provides some functionality and this can be used by other large software applications to complete its use.

In layman terms, Service Oriented Architecture provides a framework where different (Heterogeneous) platforms (Windows, Linux, and UNIX), technologies (.NET, Java) and applications (ERP, CRM, and SCM) operate in synchronization. More specifically, this is achieved through “Services” and “Web based Open Standards”.

Service is the base of this architecture. Services are the structures which have ability to interact with each other. In other words they are the listener of the other side of the phone which is an endpoint.

In the classical layered architecture layers interact with each other. This interaction and hierarchy of this system should be constructed very properly for this system to work proper. Therefore we can say that SOA simply means that these layers are created as services. For example if we make a service which provides us the data, then we have accomplished the functionality of data layer by this. Afterwards we can also use this service from other applications as well. Achieving this we have a flexible architecture to use.

Why SOA?

Reusability

This property of SOA is the most important benefit of this architecture. Base of this architecture is constructed with services so that it provides us functionality to multiple applications or clients. The purpose is actually to reuse them.

Connecting Heterogeneous Systems

Nowadays in our software world its almost impossible to create a common infrastructure. What I mean by this is technologies can be different and even if they are same the versions of them might be different as well. Thus SOA provides us a good communication quality.

Loose Coupling

Services are not connected with classes or libraries they are connected with contracts and endpoints. So that they are loosely coupled. A service which obeys the same schema structure can replace another service very easily. To give an example for this we can say that a Java service can be replaced with a .NET service or another service which obeys the same schema.

Facade

This architecture can be use to simplify interface of complex structures or complex services. This type of approach is used when the service is created to simplify technical usage for other developers to use. It can be also used to separate the service from work flow. It is enough to expose only the relevant data to the consumer.

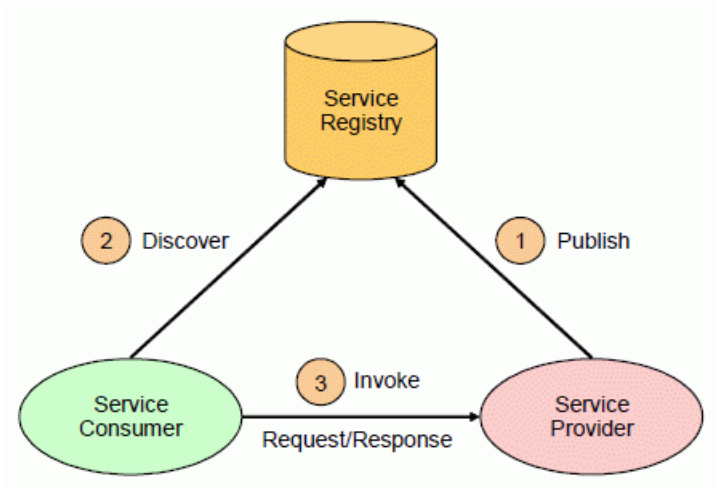
Abstraction

Every service is responsible for their own functionality. For example : If you want your service to produce passwords, than it only produces passwords while another service checks the username and password. In this way application is partitioned into simpler parts. This increases the readability and reduces the time of maintenance.

Where to use SOA then ?

This architecture should be preferred when there are multiple heterogeneous systems, applications or consumers. Otherwise achieving flexibility is an additional cost. Consider the scenario where Enterprise systems have different owners and multiple teams working on them. In order to improve visibility and accessibility, centralized control is required. The Service Oriented Approach is appropriate in such a scenario as it provides complete information about the system components through its Service Registry.

The common service-oriented architecture is illustrated in the following diagram:



Service-oriented architecture

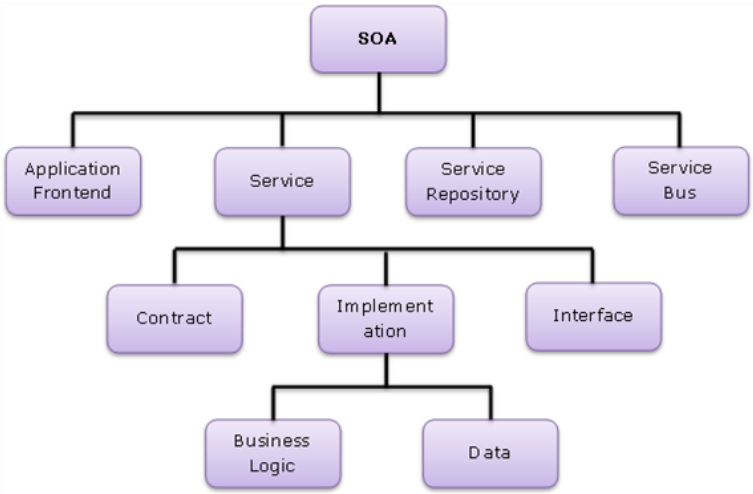
Service provider is responsible for providing the services and the details of the services. The service provider can decide whether the service needs to be secured or can be used by anyone. The cost implication and the traffic of using the service needs to be thought about. There will be an interface provided by the service provider so that any service that needs to be accessed can be achieved with the help of the interface.

The service provider can decide whether the services needs to be listed or not and what should be the agreement that should be set between the consumer for accessing the services. Most of the cases the service provider publishes the interface and the access details with the Service Registry

Service Registry or broker implementers should carefully think and decide on the implementation and access strategy of the services by the service consumers. The ownership of making the service interface public resides with the service registry. The implementers should consider about the scope that is involved. There are pubic service registries that can be accessed over the internet and there are private service registries which is accessible to only restricted or role based consumers

Service consumer ensures that it's possible to locate the service that is registered in the service registry and binds to the service provider is obtained. The service consumers invoke the service that is defined.

Elements of SOA:



SOA elements

These architectures have a number of benefits, including quicker and more cost-effective responses to changing market conditions. SOAs can also simplify connections to legacy IT systems. SOAs also have a number of challenges in implementation, such as managing the large number of services and how they interact with each other. There are also concerns about lack of testing and security levels of services. Overall, service-oriented architecture is a widely-used technology that is changing the way large businesses function today.

Principles of SOA:

1. Explicit Boundaries

2. Shared Contract and Schema, not Class

3. Policy-driven

4. Autonomous

5. Wire formats, not Programming Language APIs

6. Document-oriented

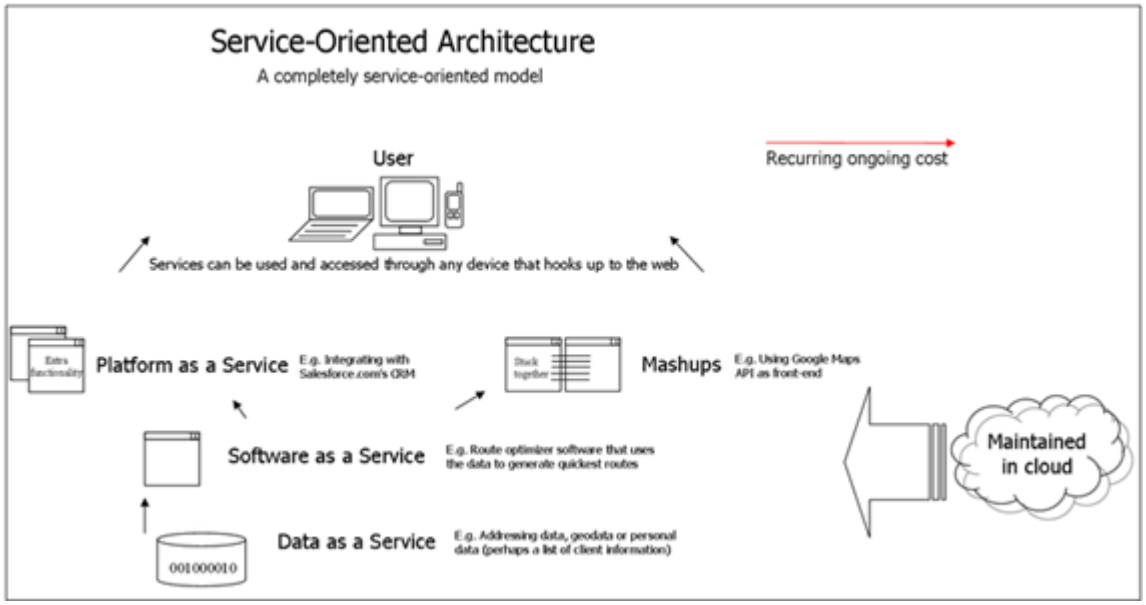
7. Loosely coupled

8. Standards-compliant

9. Vendor independent

10. Metadata-driven

The following figure is a good description about how could we imagine a completely service-oriented model:



SOA overview

1.1. Side note about Web-oriented architecture

Note

According to Gartner's definition of Web-Oriented Architecture (WOA) from 2008:

> WOA is an architectural substyle of SOA that integrates systems and users via a web of globally linked hypermedia based on the architecture of the Web. This architecture

emphasizes generality of interfaces (UIs and APIs) to achieve global network effects through five fundamental generic interface constraints:

1. Identification of resources
2. Manipulation of resources through representations
3. Self-descriptive messages
4. Hypermedia as the engine of application state
5. Application neutrality

Resource identification could be done by uniform resource identifier (URI), resource manipulation by HTTP, application state by links and self-describing messages could be identified by Multipurpose Internet Messaging Extensions (MIME) types.

If we familiar with REST than we can feel this is essentially a rehash of REST except for the fifth point on "application neutrality". After some back-and-forth discussions on the REST-discuss list [WOAvsREST], it turned out that "application neutrality" meant using generic media types such as the Atom Syndication Format for representations. Media types should be able to convey the type of representation, and by using generic types, applications lose describability.

It is unfortunate that Gartner chose a vague term like "application neutrality" to describe this. But on the positive side, they are at least talking about REST. Furthermore, Gartner archived the document in 2011 because some of its content may not reflect current conditions and said that its more easier to speak about Web APIs.

[GartnerWOA] *Tutorial: Web-Oriented Architecture: Putting the Web Back in Web Services*. Web page .

[GartnerBlogWOA] *WOA: Putting the Web Back in Web Services*. Web page .

[GartnerSOAgovernance] *Magic Quadrant for SOA Governance Technologies*. Web page .

[WOAvsREST] *Yahoo! groups: Gartner note on WOA*. Web page .

[RESTthesis] *Architectural Styles and the Design of Network-based Software Architectures*. Web page .

2. Representational State Transfer (REST)

Representational State Transfer is an architectural style that build on certain principles using the current web fundamentals. It generally runs on HTTP. It makes a stateless transfer. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. REST has been applied to describe desired web architecture, to identify existing problems, to compare alternative solutions, and to ensure that protocol extensions would not violate the core constraints that make the Web successful. Fielding used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI).

The REST architectural style is also applied to the development of web services as an alternative to other distributed communication types such like SOAP. REST is often used in mobile applications, social networking Web sites, mashup tools and automated business processes.

There are 5 basic fundamentals of REST services which are created for the Web.

1. Everything is a Resource.
2. Every Resource is Identified by a Unique Identifier.
3. Use Simple and Uniform Interfaces.
4. Communication is Done by Representation.

5. Every Request is Stateless.

Everything is a Resource

The first important point in REST is to think in terms of resources rather than physical files. You access the resources over some URI. for example:

- http://www.mysite.com/pictures/logo.png - Image Resource;
- http://www.mysite.com/index.html - Static Resource;
- http://www.mysite.com/Customer/1001 - Dynamic Resource returning XML or JSON content;
- http://www.mysite.com/Customer/1001/Picture - Dynamic Resource returning an image.

Unique Identifier

Informations are reached by using unique identifiers. In REST, we add one more constraint to the current URI: in fact, every URI should uniquely represent every item of the data collection. For example you can see below unique URI format for each customer is defined.

http://www.mysite.com/Customer/dupont

http://www.mysite.com/Customer/smith

and orders of customer ” dupont ” is defined like following:

http://www.mysite.com/Customer/dupont/Orders

Simple and Uniform Interfaces

To request and send data to those resources some HTTP methods are used. These are the HTTP methods :

GET - List the members of the collection (one or several)

PUT - Update a member of the collection

POST - Create a new entry in the collection

DELETE - Delete a member of the collection

Then, at URI level, you can define the type of collection, like this:

http://www.mysite.com/Customer to identify the customers or

http://www.mysite.com/Customer/1234/Orders to access a given order.

This combination of HTTP method and URI replace a list of English-based methods, like GetCustomer / InsertCustomer / UpdateOrder / RemoveOrder.

Representations

What you are sending via the network is actually a representation of the actual resource data.

The main representation schemes are XML and JSON but it could be CVS as well.

For example, here is how a customer data is retrieved with a GET method:

An example with XML format :

```
<Customer>
<ID>1234</ID>
<Name>Dupond</Name>
<Address>Tree street</Address>
</Customer>
```

Below is a simple JSON snippet for creating a new customer record with name and address:

```
{Customer: {"Name":"Dupont", "Address":"Tree street"}}
```

As a result to this data transmitted with a POST command, the RESTful server will return the just-created ID.

Clarity of this data format is one of the reasons why preferably to use JSON format instead of XML or any format.

Stateless

In REST concept every request is independent and each request doesn't have any information about the previous request. Every request should be an independent request so that we can enhance the requests being sent. As REST doesn't use much memory like SOAP so that we can make more independent requests. This Scalability in terms is important because a server request is unaware of any state we store and makes resource management easier.

Of course, there are some disadvantages of being stateless. Client has to add all the necessary information in request' which increases network traffic. It also highlights consistency in the behavior of the application server which could be handled difficultly because there may be many different Clients, the requests may come from the different content.

Independent request means that each request doesn't carry out any state information about another. A classic example of statelessness is the use of the HTTP protocol. HTTP protocol does not carry out data between requests so that this is usually achieved by using some kind of server variables to carry out the data using some programming languages. For example in Asp.Net it can be achieved by using variables such as viewstate, session, caching, query string vice versa.

There is one more important fact that need to be outlined: **Cacheable**. HTTP response by the client "cache" can be entertainment, so it sent Server RESPONSE to indicate whether the case is cacheable, it is important in terms of performance.

What are the advantages of rest;

- It is lightweight, can be easily extended.
- Inbound, outbound data size is very small.
- It is easy to design and easy implementation, it does not need any extra tools.
- Works over HTTP, platform independent.

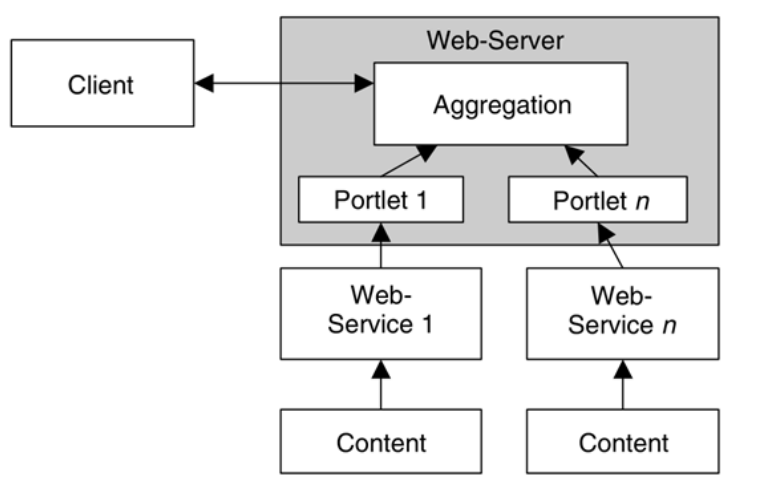
2.1. REST and RESTful

Representational state transfer (REST) is a style of software architecture. As described in a dissertation by Roy Fielding, REST is an "architectural style" that basically exploits the existing technology and protocols of the Web.

RESTful is typically used to refer to web services implementing such an architecture. Some says that "REST" is an architectural paradigm. "RESTful" describes using that paradigm. More precisely the term Representational State Transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Conforming to the REST constraints is referred to as being 'RESTful'.

3. Portal architecture - one of the SOA variants

Portals represent the most recent development of multi-layered Web applications. Portals try to make contents, which are distributed over several nodes of different providers, available at one single node providing a consistent look and feel. Portal servers are based on so-called portlets, which arrange contents and application logics in a navigation structure and layout appropriate for the portal. An independent aggregation component is used to integrate the set of different portlets into a uniform Web site. This aggregation can be specified by the portal provider, or manipulated by the user, e.g., through customization. The schematic architecture could be seen in the following figure (according to Kappel et al. 2006):



Portal architecture

Chapter 6. Web Services

Web services are the standardized way of communicating different web-based applications via network. By using Web services, your application can publish its function or message to the rest of the world. Web services can communicate with each other using XML file format without knowing how each other is implemented. Web based applications are communicating using the concepts as XML, SOAP, REST, WSDL and UDDI.

In the context of Service Oriented Architecture, Web Services are used to facilitate communication between service providers and service consumers.

W3C defines web service as a “software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

Different applications might be built with different programming languages, therefore there is a need for a method of data exchange that doesn't depend upon a particular programming language. In other words a web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.

Here are the common properties of web services:

- Web services are application components
- Web services communicate using open protocols
- Web services are self-contained and self-describing
- Web services can be discovered using UDDI
- Web services can be used by other applications
- HTTP and XML is the basis for Web services

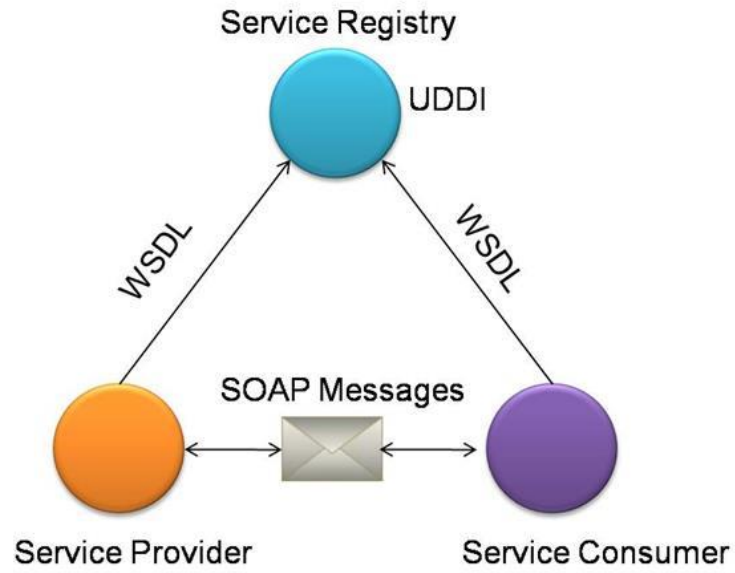
Usage of web services can be mentioned in two major parts. First one is ”reusable application components” and second one is to ”connect existing software”.

Reusable application components:There are sometimes some informations which many applications need to use. So why do we need to do these again and again. We can reuse the application components. Web services can offer some components like weather reports, currency conversion or language translation.

Connect existing software:You can connect your existing software to a Web service to utilize the data from which a web service provide.

Web Service Roles

Web service architecture can be expressed by mentioning some major roles of it. There are three major roles within the web service architecture:



Web Services overview

Service provider

This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

Service requestor

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

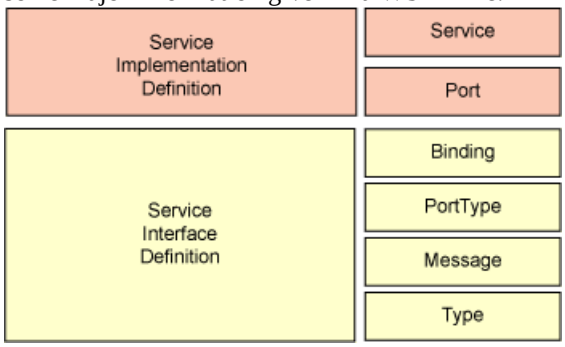
Service registry

This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized trade center for companies and their services.

1. Web Services Description Language (WSDL)

WSDL describes the functionality of a web service and it is an XML based description language. A WSDL file contains information about how a web service can be called, what parameters it expects and what data structure it returns. The WSDL describes services as collections of network endpoints, or ports. The abstract definitions of ports and messages are separated from their concrete use.

This is good for allowing the reuse of these definitions. As you can see in the *Figure* ... it describes the structure of a WSDL file graphically. Concrete and Abstract sections are separated for reuse. I am going to mention about some major information given in a WSDL file.



structure of a WSDL file

Service:Contains a set of system functions that have been exposed to the Web-based protocols.

Endpoint:Defines the address or connection point to a Web service. It is typically represented by a simple HTTP URL string.

Binding:Specifies the interface and defines the SOAP binding style and transport (SOAP Protocol). The binding section also defines the operations. There may be any number of bindings for a given portType.

PortType: Defines a Web service, the operations that can be performed, and the messages that are used to perform the operation. It is the most important element in WSDL. I can be compared to a module or a class in traditional programming.

Message: Typically, a message corresponds to an operation. The message contains the information needed to perform the operation. Each message is made up of one or more logical parts. Each part is associated with a message-typing attribute. The message name attribute provides a unique name among all messages.

Types:The types element defines the data types that are used by the web service. WSDL 1.1 allows the use of any typing system, but it uses W3C XML schema for maximum platform neutrality.

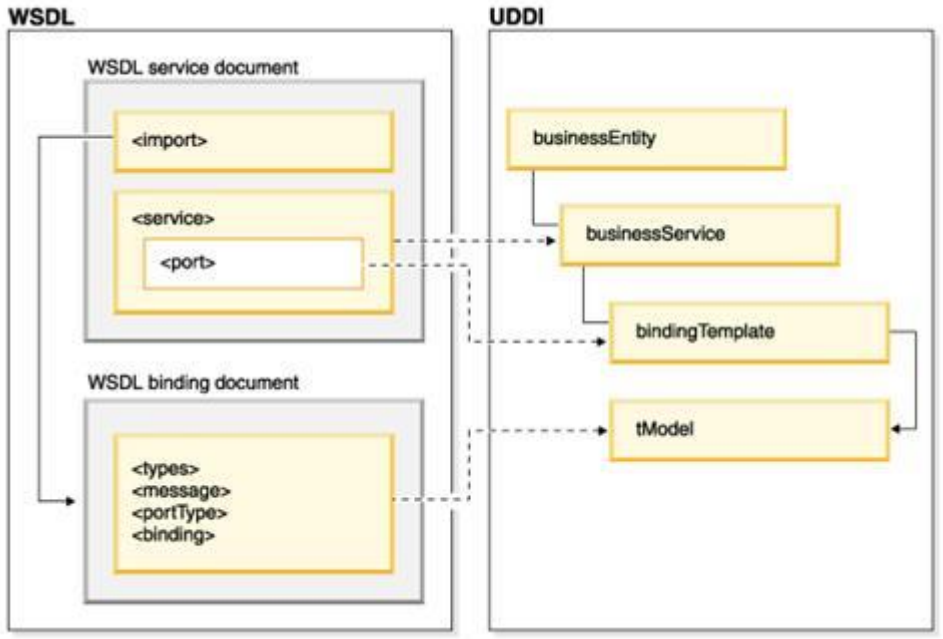
2. Universal Description, Discovery and Integration (UDDI)

UDDI is a web based distributed directory. It enables businesses to list themselves on internet and they also discover each other. It is similar to a traditional phone book's yellow and white pages. Companies that use the UDDI protocol can extend their market reach and find new customers while also finding other businesses that offer useful services to them. Because UDDI uses a standard format for describing business services, it is easy to search and find useful services offered from other businesses.

A UDDI registration consists of three components:

- White Pages — address, contact, and known identifiers.
- Yellow Pages — industrial categorizations using the standard classifications.
- Green Pages — technical information about services published by the business.

The Relation between WSDL and UDDI is shown in the *Figure*



UDDI

3. SOAP Web Services

Web services can be implemented with different technologies such as SOAP or REST. Simply, SOAP web services provide a business logic exposed via a service to a client. And it is done by a loosely coupled interface using XML. The interface to which a message is sent defines the format of the message request and response, and mechanisms to publish and to discover web service interfaces.

SOAP usually relies on several technologies and protocols such as HTTP or SMTP for transferring and transforming data. Some of this technologies are:

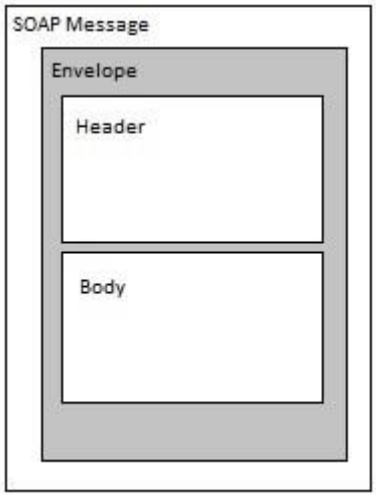
- Extensive markup language (XML) is the foundation of SOAP which is used to build and define the web services.
- Web Services Description Language (WSDL) describes the protocols, message types, interface and interactions of SOAP web services.
- Simple Object Access Protocol (SOAP) is a protocol that is used for exchanging data using XML.
- HTTP is the commonly used transport protocol for exchanging messages. Other protocols such as SMTP or JMS are also used.
- Universal Description Discovery, and Integration (UDDI) is optionally used to store and categorize SOAP web service interfaces (WSDL).

SOAP

Simple Object Access Protocol is the standard web services application protocol. It provides the communication mechanism to connect web services, exchanging formatted XML data across a network protocol. SOAP relies heavily on XML for defining an extensive messaging framework. This framework is independent of any particular programming model.

SOAP Messages

SOAP message is a one-way message such as a request from a client, or a response from a server. Every SOAP message contains an Envelope element, an optional Header element and a Body element.



SOAP message

Envelope

The SOAP XML Envelope defines specific rules for encapsulating data being transferred between computers. This includes application-specific data, such as the method name to invoke, method parameters, or return values. It can also include information about who should process the envelope contents

Envelope element has:

- a local name for the envelope

- a namespace name used for differentiating versions ("http://www.w3.org/2003/05/soap-envelope" for SOAP 1.2),
- Header and Body elements in its children property

An example:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and applies to the element's contents and all child elements.

Header

The optional Header element offers a flexible framework for specifying additional application-level requirements (like authentication, payment, etc.) about the SOAP message.

An example:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
<m:Trans xmlns:m="http://www.example.org/transaction/"
soap:mustUnderstand=”true” >
5
</m:Trans>
</soap:Header>
...
</soap:Envelope>
```

Body

Body element contains the actual message. For example:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

A GetStockPrice request is sent to a server. The request has a StockName parameter, and a Price parameter that will be returned in the response

4. SOAP vs REST

SOAP is a protocol specification for exchanging structured information in the implementation of Web Services. It uses XML for the message format. It is independent of the transport protocol (could be HTTP, FTP, TCP, UDP, or named pipes). SOAP based services strictly define the format of messages passed back and forth. A SOAP message contains the data, the action to perform on it, the headers, and the error details in case of failure. Security is provided by WS-Security standards and is end-to-end. It supports identity through intermediaries, not just point to point (SSL).

Representational State Transfer (REST) is an architecture style for designing networked applications. REST recognises everything a resource (e.g. User, Course, etc.) and each resource implements a standard uniform interface (typically HTTP interface), resources have name and addresses (URIs), each resource has one or more representation (like JSON or XML) and resource representations move across the network usually over HTTP.

RESTful web APIs (or RESTful web service) is a web API implemented using HTTP and principles of REST. RESTful API separates user interface concerns from data storage concerns. It improves portability of interface across multiple platforms and simplifies server components by making them stateless. Each request from client contains all the state information and server does not hold client context in the session.

One of the major benefits of SOAP is that you have a WSDL service description, like a contract between the two parties. You can pretty much discover the service automatically and generate a useable client proxy from that service description (generate the service calls, the necessary data types for the methods and so forth). With REST you'll have to implement such contract yourself in your code.

Note that with version 2.0, WSDL supports all HTTP verbs and can be used to document RESTful services as well, but there is a less verbose alternative in WADL (Web Application Description Language) for that purpose.

With RESTful services, message security is provided by the transport protocol (HTTPS), and is point-to-point only. It doesn't have a standard messaging system and expects clients to deal with communication failures by retrying. SOAP has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries.

One of the major benefits of RESTful API is that it is flexible for data representation, for example you could serialize your data in either XML or JSON format. RESTful APIs are cleaner or easier to understand because they add an element of using standardised URIs and gives importance to HTTP verb used (i.e. GET, POST, PUT and DELETE).

RESTful services are also lightweight, that is they don't have a lot of extra XML markup. To invoke RESTful API all you need is a browser or HTTP stack and pretty much every device or machine connected to a network has that. With web services that are generally SOAP-based, the request and response are hidden. SOAP requests must be interpreted as they are received at the server to determine the operation to perform and the arguments required to perform that operation. They are generally passed through as a parameter, which is essentially a function/method call. SOAP does not currently provide a mechanism for caching results, where as REST can (through the standard HTTP caching).

Essentially, the Traditional SOA way of doing things and the RESTful way of doing things would both yield similar results. However you might find that the Traditional SOA would allow you to do more powerfull things (and as a result is more complex) and the RESTful oriented architecture would allow you to easily do simple things (and this being much easier to implement) REST becomes an architecture style applicable to systems that would want operate with a Resource view. SOA, on the flip-side, has a different view of the web. It uses the web as a application infrastructure between service providers and consumers otherwise known as the Service view.

In conclusion, we can say that web services standards provide an open standards based communication framework that operates within the purview of W3C guidelines. Web Services provide a platform/technology independent communication methodology while SOA is an overall IT strategy framework that aims to provide business agility. Finally, which ever architecture you choose make sure its easy for developers to access it, and well documented.

Chapter 7. References

[1] W3C. Architecture of the World Wide Web, Volume One. http://www.w3.org/TR/webarch/

[2] W3C. Character Model for the World Wide Web 1.0: Fundamentals. http://www.w3.org/TR/charmod/

[3] W3C. Device Independence Principles. http://www.w3.org/TR/di-princ/#section-Introduction

[4] Microsoft. Microsoft Application Architecture Guide, 2nd Edition. ISBN 9780735627109, http://msdn.microsoft.com/en-us/library/ ee658109.aspx

[5] FOWLER, Martin. Patterns of Enterprise Application Architecture. ISBN 0-321-12742-0 11

[6] STAFFORD, Randy. Service Layer. http://martinfowler.com/eaaCatalog/serviceLayer.html

[7] RAMIREZ, Ariel Ortiz. Three-Tier Architecture. http://www.linuxjournal.com/article/3508

[8] Exforsys. Three Tier Software Architectures. http://www.exforsys.com/tutorials/application-development/ three-tier-software-architectures.html

[9] CHARLAND, Andre and LEROUX, Brian. Mobile Application Development: Web vs. Native. http://queue.acm.org/detail.cfm?\searchterm=web+ application&id=1968203

[10] BERNERS-LEE, Tim. Information Management: A Proposal. http://www.w3.org/History/1989/proposal.html

[11] FIELDING, Roy T., and TAYLOR, Richard N. Principled Design of the Modern Web Architecture. http://www.cs.helsinki.fi/group/java/s12-wepa/resurssit/ principled-design-of-the-modern-web-architecture.pdf

[12] GOMEZ, Jaime, CACHERO, Cristina and PASTOR, Oscar. On Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach. http://dlsi.ua.es/ ~ ccachero/papers/ieee.pdf

[13] MITCHELL, Scott. Creating a Business Logic Layer. http://msdn.microsoft.com/en-us/library/aa581779.aspx

[14] STRAUCH, Cristoph. NoSQL Databases. http://home.aubg.bg/students/ENL100/Cloud%20Computing/ Research%20Paper/nosqldbs.pdf,

[15] FOWLER, Martin. Key Points from NoSQL Distilled. http://martinfowler.com/articles/nosqlKeyPoints.html

[16] PAPAGELIS, Manos. WEB APP ARCHITECTURES: MULTI-TIER (2-TIER, 3-TIER) & MVC. http://queens.db.toronto.edu/ ~ papaggel/courses/csc309/docs/ lectures/web-architectures.pdf

[17] HENSCHEN, Doug. When NoSQL Makes Sense. http://www.informationweek.com/big-data/software-platforms/ when-nosql-makes-sense/d/d-id/1111811

[18] Rebelic. The four categories of NoSQL databases. http://rebelic.nl/2011/05/28/the-four-categories-of-nosql-databases/

[19] MELNIK, Sergey and DECKER, Stefan. A Layered Approach to Information Modeling and Interoperability on the Web. http://infolab.stanford.edu/ ~ melnik/pub/sw00/

[20] HECHT, Robin and JABLONSKI, Stefan. NoSQL Evaluation, A Use Case Oriented Survey. http://rogerking.me/wp-content/uploads/2012/03/ DatabaseSystemsPaper.pdf

[21] POH, Michael. 8 Next-Generation User Interface That Are (Almost) Here. http://www.hongkiat.com/blog/next-gen-user-interface/

[22] Apache.org. Apache Tutorial: Dynamic Content with CGI. http://httpd.apache.org/docs/current/howto/cgi.html

[23] ASLETT, Matthew. What we talk about when we talk about NewSQL. http://blogs.the451group.com/information_management/2011/ 04/06/what-we-talk-about-when-we-talk-about-newsql/

[24] ASLETT, Matthew. NoSQL, NewSQL and Beyond: The answer to SPRAINed relational databases. http://blogs.the451group.com/information_management/2011/ 04/15/nosql-newsql-and-beyond/

[25] IBM.com. What is big data?. http://www-01.ibm.com/software/data/bigdata/ what-is-big-data.html

[26] SAS.com. Big Data - What it is and why it matters. http://www.sas.com/en_us/insights/big-data/ what-is-big-data.html

[27] The Economist. Data, data everywhere. http://www.economist.com/node/15557443

Part III. Web Engineering

The Web is possibly the most used feature of the Internet, and of its many uses and features one that cannot be neglected is the scope of Web Applications. They are useful for a diversity of end-users and companies, achieving power that were commonly related to desktop software and then changing the traditional way of computer interaction, diversifying the platforms where software can be accessed. For example, moving from desktop computers to tablets and smartphones, or even diminishing the operational system role to become a simple interface to web apps. This way, the development of such web apps cannot be neglected to a "lower form of development", needing as software engineering as any other piece of software.

However, Web developers often use ad hoc, geek or technee-type approaches, which lack rigor, systematic techniques, sound methodologies, and quality assurance. This is why we need to mention first the most important factors for the quality.

Quality of Web Applications

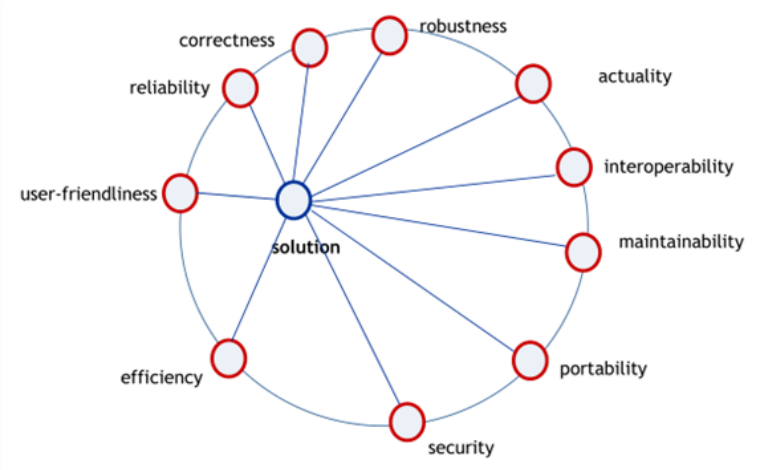
We can speak about external and internal qualities. External qualities are visible to the user and includes the following ones:

- correctness: a web application is functionally correct if it behaves according to the specification of the application
- reliability: the probability that the software will operate as expected, occurring software errors are not serious
- robustness: software behaves reasonably even in circumstances that were not anticipated in the requirements specification
- actuality: actuality of content must be guaranteed
- user-friendliness: easy to use by human (novice / experts)
- efficiency: economical handling of resources (time, storage space)
- security: system is protected from unauthorized access.

Internal qualities are concerning and visible to the developer:

- portability: a web application is portable if it can run in different environments
- interoperability: refers to the ability of the web application to coexist and cooperate with other systems
- maintainability: ability to modify a web application after it has been deployed, to correct errors or extend the web application

These factors are shown that each solution is a compromise and we need to made a tradeoff when choosing our own solution. This can be seen in the following figure.

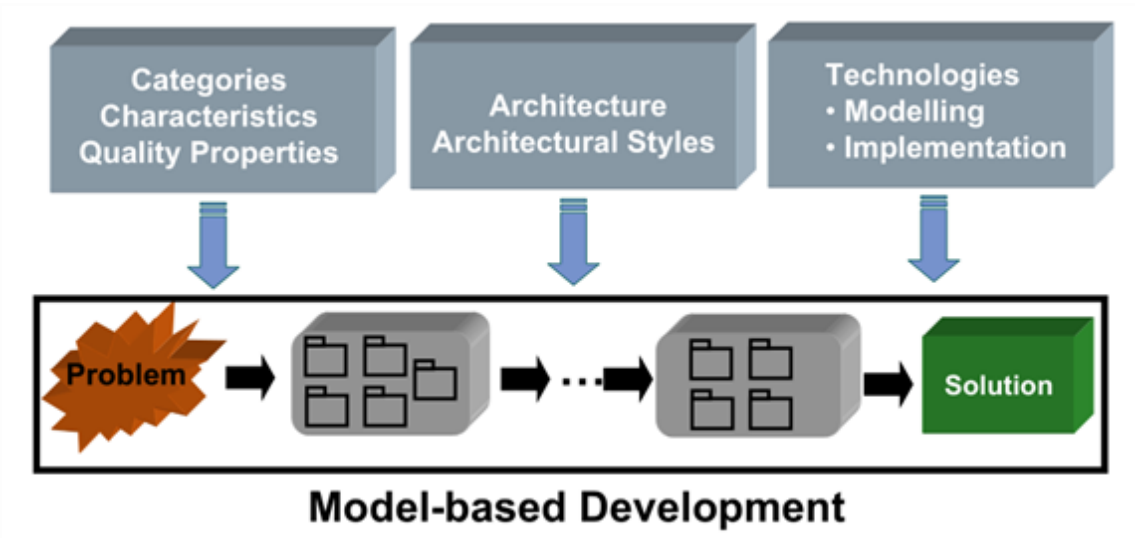


The tradeoff-circle for Web applications

The cost of bad design, shabby development, poor performance, and/or lack of content management for Web-based applications have many serious consequences. A survey in the end of 2000 showed that most of the Web-based projects are faced with serious problems which resulted delays, malfunctioning functionalities or budget overrun. As a result, developers, users, and other stakeholders have become increasingly concerned about the manner in which complex Web-based systems are created as well as the level of system performance, quality, and integrity.

Modern Web applications are full-fledged, complex software systems. Due to the evolution of Web technologies the Web has become a primary platform for developing applications. However, as these technologies evolve very fast, they might become obsolete soon. Developers of Web applications need sophisticated solutions that support the whole product lifetime of an application that is able to cope with the quick changes of the underlying technologies.

Therefore, the development of Web applications requires a methodologically sound engineering approach. Model-driven Web Engineering (MDWE) is a still emerging field aiming at providing sound model-based solutions for building Web applications that try to separate the abstract design (PIM) from the concrete technological platforms (PSMs). However, current MDWE approaches cannot provide solutions for all kinds of the requirements against a software system but the directions are clear.



Support for Web Application Development

Chapter 8. Web Engineering

MODEL-DRIVEN WEB ENGINEERING - MAGIC OR REALITY?

Modern Web applications are full-fledged, complex software systems. Due to the evolution of Web technologies the Web has become a primary platform for developing applications. However, as these technologies evolve very fast, they might become obsolete soon. Developers of Web applications need sophisticated solutions that support the whole product lifetime of an application that is able to cope with the quick changes of the underlying technologies.

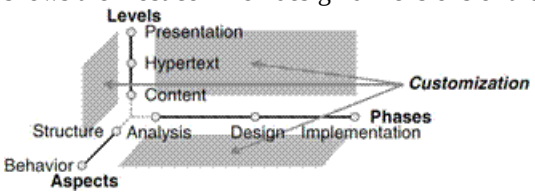
Therefore, the development of Web applications requires a methodologically sound engineering approach. Model-driven Web Engineering (MDWE) is a still emerging field aiming at providing sound model-based solutions for building Web applications that try to separate the abstract design (PIM) from the concrete technological platforms (PSMs).

Introduction

The evolution of Web technologies increased the presence of the Web in our everyday life starting from personal home pages through corporate portals and Web shops up to Web applications implementing complex business processes. This diversity of applications makes the selection of the appropriate technology and platform harder, in particular, when these applications should collaborate with each other. Some of them might be suitable in a given case, which are sometimes alternatives to each other, while they should be combined in other cases. As these technologies evolve very fast, they might become obsolete soon. It is very hard to predict, which technologies will be out tomorrow so business logic should be as independent as possible from technologies to allow change in the technology without affecting business processes and rules.

The modeling of such systems is a very complex process since (in an ideal case) it should take both existing and future technologies into consideration in order to create a flexible system which allows further development. Despite changing of the technology, models of the application domain remain almost the same since they capture business model and processes. The best practice tells us to keep the modeling of the application domain and the technological details separated.

Existing model-based Web Engineering approaches provide different methods and tools for both the design and the development of various kinds of Web applications. In order to reduce complexity, most of the methodologies propose the separation of different views (i.e., models) of the application into 3 levels: structural (or content), navigational (or hypertext) and presentational models. For more information see [7]. Figure 1 shows the most common design dimensions of the currently existing methodologies.



Design dimensions of Web applications.

In addition, some methodologies add some new models (or refine existing ones) to obtain a more fine-grained solution when modeling the application. Despite the separation, the levels should be interconnected in order to be able to capture the semantics behind the elements of the different models, e.g., the navigational objects are based on certain elements of the content model.

Beyond the creation of the models for the corresponding levels, Web application designers need to be aware of the various aspects of the systems to be modeled. Some applications are providing access to more or less static information hence they require much less behaviour modeling compared to systems that need to perform several complex business processes like e-commerce applications. Both structure and behaviour need to be modeled using a uniform notation that has to cope with the specific characteristic of each of the levels.

Current design methods offer some possibilities for modeling the levels and aspects mentioned above but they all has a unique approach (e.g., offering some model kinds that the others not) so this field is not standardized.

There is an other approach worth mentioning when talking about Web application design. Let this be either a fortune or an unfortune, there is no consensus in the literature about the general phases of the development which means that the order of steps involved in modeling the levels is up to the modeler.

1. MDA & MDE

The Model-Driven Architecture (MDA) is a software design approach that was officially launched in 2001. MDA is intended to support model-driven engineering of software systems. The MDA is a specification that provides a set of guidelines for structuring specifications expressed as models. Using the MDA methodology, system functionality may first be defined as a platform-independent model (PIM) through an appropriate Domain Specific Language. Given a Platform Definition Model (PDM) corresponding to CORBA, .Net, the Web, etc., the PIM may then be translated to one or more platform-specific models (PSMs) for the actual implementation, using different Domain Specific Languages, or a General Purpose Language like Java, C#, Python, etc. The principles of MDA can also be applied to other areas like business process modeling where the architecture and technology neutral PIM is mapped onto either system or manual processes.

One of the main aims of the MDA is to separate design from architecture and realization technologies facilitating that design and architecture can alter independently. The design addresses the functional (use case) requirements while architecture provides the infrastructure through which non-functional requirements like scalability, reliability and performance are realized. MDA envisages that the platform independent model (PIM), which represents a conceptual design realizing the functional requirements, will survive changes in realization technologies and software architectures. In other words, ”Design once, build it on any platform”.

MDA at a glance.

Following a long history of the use of models to represent key ideas in both problem and solution domains, MDA provides a conceptual framework for using models and applying transformations between them as part of a controlled, efficient software development process. The basic assumptions that governing MDA usage:

- Models help people understand and communicate complex ideas.
- Many different kinds of elements can be modeled, depending on the context. These offer different views of the world that must ultimately be reconciled.
- We see commonality at all levels of these models in both the problems being analyzed and the proposed solutions.
- Applying the ideas of different kinds of models and transforming them between representations provides a well-defined style of development, enabling the identification and reuse of common approaches.
- Provides a conceptual framework and a set of standards to express models, model relationships, and model-to-model transformations.
- Tools and technologies can help to realize this approach, and make it practical and efficient to apply.

Model Driven Engineering.

In order to raise the level of abstraction in program specification and increase automation in program development we need to use engineering approaches as well. The idea promoted by Model Driven Engineering (MDE) is to use models at different levels of abstraction for developing systems, thereby raising the level of abstraction in program specification. An increase of automation in program development is reached by using executable model transformations. Higher-level models are transformed into lower level models until the model can be made executable using either code generation or model interpretation.

MDE is often confused with Model Driven Architecture. MDA can be seen as OMG’s (Object Management Group) vision on MDE but MDE is wider in scope than MDA. MDE combines process and analysis with architecture. The MDE approach is meant to increase productivity by maximizing compatibility between systems (via reuse of standardized models), simplifying the process of design (via models of recurring design patterns in the application domain), and promoting communication between individuals and teams working on the system (via a standardization of the terminology and the best practices used in the application domain). A modeling paradigm for MDE is considered effective, if its models make sense from the point of view of a user

that is familiar with the domain, and if they can serve as a basis for implementing systems. The models are developed through extensive communication among product managers, designers, developers and users of the application domain. As the models approach completion, they enable the development of software and systems.

2. Domain Specific Models and Languages

In software development a domain-specific language (DSL) is a programming language or specification language dedicated to a particular problem domain and is not intended to be able to solve problems outside it. A DSL can be either a visual language, like the languages used by the Eclipse Modeling Framework, or textual languages. A sound language description contains an abstract syntax, one or more concrete syntax descriptions, mappings between abstract and concrete syntax, and a description of the semantics. The abstract syntax of a language is often defined using a metamodel. The semantics can also be defined using a metamodel, but in most cases in practice the semantics aren't explicitly defined, they have to be derived from the runtime behavior.

In model-driven engineering many examples of domain-specific languages may be found like OCL, a language for decorating models with assertions or QVT, a domain specific transformation language. However languages like UML are typically general purpose modeling languages. What is radically new is the idea of creating your own DSL for your own project. Domain specific languages have important design goals that contrast with those of general-purpose languages:

- domain specific languages are less comprehensive.
- domain specific languages are much more expressive in their domain.
- domain specific languages should exhibit minimum redundancy.

A model specified using a DSL is called a Domain-Specific Model (DSM). A complex system is usually described using multiple DSMs specified in different DSLs. These models refer to each other and have to be combined when executing them. Because complex systems ask for a lot of DSMs to model them, it is important to structure the modeling space, the different parts of the system as we can see on Figure 1.

Domain-specific modeling.

The main goal of domain-specific modeling is to raise the level of abstraction by specifying the solution directly using domain concepts. The final product (and maybe several intermediate artifacts, as well) are generated based upon these high-level specifications. It also allows the stakeholders and domain experts to concentrate to the domain only. Domain-specific languages (DSLs) are built in order to capture domain semantics. A very common (but not the only) way of defining DSLs is metamodeling. The previously mentioned Web application design methods contain notations that can be used for describing a model of a Web application so they can be considered as DSLs for Web applications hence.

3. Characteristics of Web Applications and Web Engineering

Web applications are applications accessed over a network based on technologies and standards of the World Wide Web Consortium (W3C). These software systems are used through the Web browser as the user interface. Web Applications are usually broken into logical chunks called "tiers", where every tier is assigned a role. Traditional applications consist only of 1 tier, which resides on the client machine, but web applications lend themselves to a n-tiered approach by nature. Though many variations are possible, the most common structure is the three-tiered application. In its most common form, the three tiers are called presentation, application and storage, in this order. A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP(.NET), CGI, JSP/Java, PHP, Perl or Spring) is the middle tier (application logic), and a database is the third tier (storage). The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.

Moreover, the architecture is only one aspect which differs from traditional applications. There are varieties of features missing from traditional applications. Navigation is one of the most important ones. Non-linear navigation is lack completely from those applications. Update frequency and deployment are other factors which are different from traditional routines.

The current situation of Web application development similar to the early days of software development practices, before it was realized that the development of applications required more than programming expertise. The top problem areas of large-scale Web application projects are the failure to meet business needs, project schedule delays, budget overrun, lack of required functionality, and poor quality of deliverables.

4. Web Engineering

The emerging Web engineering discipline deals with the process of developing Web-based systems and applications. This includes theoretical principles and systematic, disciplined and quantifiable approaches towards the cost-effective development and evolution of highquality, ubiquitously usable Web-based systems and applications. It fundamentally concerns the technology which enables the construction of Web applications.

Web Engineering includes the following areas:

- Web Process & Project Management Disciplines
- Web Requirements Modeling Disciplines
- Web System Design Disciplines, Tools & Methods
- Web System Implementation Disciplines • Web System Testing Disciplines
- Web Applications Categories Disciplines

Currently, Web engineering does not provide a unique and systematic approach to the development process containing process models, architectures, suitable techniques and methods with quality assurance. As a result, Web engineering is still struggling to establish itself as a reliable engineering discipline. The cost of poor reliability and effectiveness has serious consequences for the acceptability of the systems. One of the main reasons for the low acceptance of Web-based applications could be the gap between design models and the implementation model of the Web.

4.1. Web Engineering methodologies

In the past decade many design methods have been created: OOHDM, OO-H, UWE, W2000, WSDM and WebML are among the most popular ones. From a modeler's perspective, each of them offer some possibilities for modeling the levels and aspects mentioned above, and they all come with a guideline for the development process. On the other hand, today's situation is somehow similar to the well-known "object-oriented method war" of the 1990ies. That "method war" has ended with the unification of the different modelling notations which resulted in the UML so the real question is that can this strategy also work for the existing web engineering approaches or not.

Some of the existing Web application design methods (e.g., UWE, WebML) offer a metamodel, as well. This allows model-based development since one need to build models conforming to the appropriate metamodel in order to capture the structural, navigational or presentational structure of the application to be developed. However, in the most of the cases, these models mix the different levels of Web applications that results in a solution that might be appropriate for the given application domain but makes the reuse of models or model parts almost impossible.

4.2. Model-Driven Web Engineering

As the field of Model-Driven Web Engineering (MDWE) approaches follow the well-known Model-Driven Engineering principles, some methods create Computational Independent Models (CIMs, e.g., in the form of requirement models), almost all of them allows the creation of Platform Independent Models (PIMs) for structure, navigation, presentation or business processes and most of them provides a means of obtaining Platform Specific Models (PSMs) for various platforms (e.g., J2EE, .NET, Spring, Struts, etc.) that can further be transformed into code.

Model transformations are the most important operations in model engineering, describing how elements in the source model are converted into elements in the target model. This is achieved by relating the corresponding metamodel elements in the source and the target metamodels. Transformations can be classified into two

categories: vertical transformations (a.k.a. refinements) are defined between models of different abstraction levels (e.g., PIM—PSM mappings), while horizontal transformations are mappings between models of the same level of abstraction (e.g., for improving or correcting a model).

A modeling paradigm for MDE is considered effective if its models make sense from the point of view of the user and can serve as a basis for implementing systems. The models are developed through extensive communication among product managers, designers, and members of the development team. As the models approach completion, they enable the development of software and systems. In order to achieve our goals we need to find an effective way in the Model-Driven Architecture approach that defines system functionality using a platform-independent model (PIM) using an appropriate domain-specific language (DSL).

5.1. **MDWE and the DSL lifecycle.** A DSL life cycle can contain five development phases: decision, analysis, design, implementation and deployment. In practice DSL Development isn't a sequential process, the phases should be applied iteratively.

5.1.1. *Decision.* The development of a DSL starts with the decision to develop a DSL, to reuse an existing one, or to use a General Purpose Language (GPL). If a domain is very fresh and little knowledge is available, it doesn't make sense to start developing a DSL. In order to determine the basic concepts of the field, first the regular software engineering process should be applied and a code base supported with libraries should be developed.

5.1.2. *Analysis.* In the analysis phase the problem domain is identified and domain knowledge is gathered. The output of formal domain analysis is a domain model consisting of:

- a domain definition, defining the scope of the domain,
- domain terminology (vocabulary, ontology),
- descriptions of domain concepts, and
- feature models describing the commonalities and variabilities of domain concepts and their interdependencies.

The information gathered in this phase can be used to develop the actual DSL. Variabilities indicate what elements should be specified in the DSL, while commonalities are used to define the execution engine or domain framework.

5.1.3. *Design.* A DSL can be designed from scratch or it can be easier to base it on an existing language. If it is based on a language it mostly restricts and extends that language and the existing language-based rules or semantics are influencing the design procedure. If you design your DSL from scratch the basic building blocks are created in a natural language and/or examples. Fortunately there are tools which can help you to create an editor which would accept only elements in your language.

5.1.4. *Implementation.* For executable DSLs the most suitable implementation approach should be chosen. It could be an interpreter, a compiler/generator or a commercial off-the-shelf product. While the different approaches can make a big difference in the total effort to be invested in DSL development, the choice for a particular approach is very important.

One possible solution can be found in the Eclipse Modelig Project (EMP). Xtext is a component that supports the development of a DSL grammar using an Extended Backus-Naur Form (EBNF)-like language, which can use this to generate an Ecore-based metamodel, Eclipse-based text editor, and corresponding ANTLR-based parser. This tool makes our approach very effective for the production because the created DSL can be validated against the grammar.

5.1.5. *Deployment.* In the deployment phase the DSLs and the applications constructed with them are used. Developers and/or domain experts use the DSLs to specify models. These models are implemented with one of the implementation patterns presented in the previous section (e.g. the models are interpreted by an engine). Such an implementation results in working software which is used by end-users.

An optional or more exactly a final step may exist in this life cycle. The maintenance. While domain experts themselves can understand, validate, and modify the software by adapting the models expressed in DSLs, sometimes changes in the software may involve altering the DSL implementation. Because it is not a new idea

or decision this could not be a first stage in a life cycle, rather than a closing stage which could lead to a new cycle or only a small modification in the language.

5. Conclusions and summary

In this chapter we introduced the basics and current state of the various existing model-driven Web engineering solutions. We have seen that MDE approaches could be effective in Web application development. The models expressed by diagrams are a useful way to communicate ideas and design. Different developers may use different diagrams, like use cases by developers and DSLs by domain experts but the main goal is to lower the inconsistency by using only one model. Moreover, the involvement of domain experts inside the development process can support rapid prototyping and iterative software development processes like agile methods also. However, there is a lot of work to do by the Web Engineering community in order to provide proper solutions for Web application development.

6. References

Bibliography

[Koch2006] W. Schwinger and N. Koch. *Modeling Web Applications*. 2006.

[Ceri2002] S. Ceri, P. Fraternali, and et al.. *Designing Data-Intensive Web Applications*. 2002.

[Gomez2003] J. Gomez and C. Cachero. *OO-H method: extending UML to model web interfaces*. 2003.

[Koch2003] N. Koch and K. Kraus. *Towards a common metamodel for the development of web applications.*. 2003.

[Koch_2_2003] N. Koch and K. Kraus. *A Metamodel for UWE*. 2003.

[Schauerhuber_2006] A. Schauerhuber, M. Wimmer, and E. Kapsammer. *Bridging existing Web modeling languages to model-driven engineering: a metamodel for WebML.*. 2006.

[Schwabe_1998] D. Schwabe and G. Rossi. *An object oriented approach to web-based applications design.* . 1998.

[W3] *The World Wide Web Consortium (W3C)*. Web page .

Part IV. Internet of Things

Table of Contents

Chapter 9. IoT - The advanced level of the Internet

The Internet of Things (IoT) is the network of physical objects accessed through the internet as described by technology analysts and visionaries. These objects contain embedded technology to interact with internal states or the external environment. That is, when objects can sense and communicate, it changes how and where decisions are made, and who makes them. The Internet of Things can be seen as scenario in which objects, animals or people are provided with unique identifiers and the ability to automatically transfer data over a network without requiring human-to-human or human-to-computer interaction. This is a term that we are hearing nowadays, and will be hearing in the future as well.

Internet of Things can be identified as a network which has uniquely identified and communicating objects with a certain protocol. From an other aspect, we can say a system with a smart network which is consisted by communicating devices.

Internet of Things has evolved from the convergence of wireless technologies, micro- electromechanical systems and the Internet. In 1991 at Cambridge University, (nearly) 15 academics found a system which captures the coffee machine. Considering that days, this might be a freaky system because this system was sending 3 photos in a minute to all these academics' computers. Because its an online and real time work, this was the first usage of Internet of things in the history.

Connecting things to the Internet can be accomplished with the help of various protocols and standards, either adopted from the traditional Internet and telecommunications fields (WiFi and Bluetooth, Ethernet, 3G and LTE, HTTP), or specifically tailored to meet the constraints of the connected things (ZigBee and Z-Wave, as well as IETF's 6LoWPAN, RPL, and CoAP).

What is a Thing?

According to the IoT Clusterbook SRA (Sundmaeker et al. 2010), a "thing" in the Internet-of- Things can be defined as a physical or virtual entity that exists in space and time and is capable of being identified. A thing in the Internet of Things, can be a person with a heart monitor implant, a farm animal with a biochip transponder, an automobile that has built in sensors to alert the driver when tire pressure is low or any other natural or man-made object that can be assigned a unique identifier and provided with the ability to transfer data over a network.

An IoT object shall have both processing and communication capabilities, either embedded in itself or offered by an attached component . Depending on their functionality, these things generate, relay and/or absorb data. Based on their primary functionality, the IoT objects can be divided into the following three categories (ITU 2005):

- Identifying things assign a (unique) identity to an object.
- Sensing things transduce the physical state of the object and/or its environment into the (digital or analog) signal for storage and further processing.
- Embedded-systems things have an immediate access to the data sensed and processed by the systems.

Naturally, the categorization is non-exclusive; for example, sensors may have identities and embedded systems may include both identifiers and sensors. Different taxonomies are available for classifying the things in IoT; The following table summarizes the dimensions found in literature to characterize or classify the IoT things (Smith et al. 2009; Chaouchi 2010; OECD 2012).

Dimension	**Characteristics**
Mobility	Moveable vs. fixed
Size	From tiny microchips to large vessels
Complexity	Dumb vs. smart
Dispersion	Concentrated vs. dispersed
Power supply	Externally powered vs. autonomous
Placement	Attached vs. embedded
Connectivity patterns	Sporadic vs. continuous communication, narrow vs. broadband
Animateness	Non-animate vs. animate

Currently, according to actual research, today there is 10-11 billion devices connected to the Internet and it is expected to be 50 billion in 2020. According to the same research, in 2003 there was 0,08 communicating devices for each person, in 2020 its assumed to be 6,48. İn addition, in 2020, its assumed that 20 classical home type devices will create data transfer traffic, which is greater then 2008’s all internet data transfers traffic at all.

Definitions

A number of definitions have been provided for the term Internet-of-Things in recent literature.

Note

Original Definition

Bill Joy had envisioned D2D (Device to Device) communication, as part of his "Six Webs" framework (as far back as 1999 at the World Economic Forum at Davos); it wasn't until Kevin Ashton that industry got a second look at the Internet of Things.

In a seminal 2009 article for the *RFID Journal*, "That 'Internet of Things' Thing", Ashton made the following assessment:

> Today computers—and, therefore, the Internet—are almost wholly dependent on human beings for information. Nearly all of the roughly 50 petabytes (a petabyte is 1,024 terabytes) of data available on the Internet were first captured and created by human beings—by typing, pressing a record button, taking a digital picture, or scanning a bar code. Conventional diagrams of the Internet ... leave out the most numerous and important routers of all - people. The problem is, people have limited time, attention and accuracy—all of which means they are not very good at capturing data about things in the real world. And that's a big deal. We're physical, and so is our environment ... You can't eat bits, burn them to stay warm or put them in your gas tank. Ideas and information are important, but things matter much more. Yet

> today's information technology is so dependent on data originated by people that our computers know more about ideas than things. If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so.

—Kevin Ashton, *'That 'Internet of Things' Thing', RFID Journal, July 22, 2009*

As of 2014, research into the Internet of Things is still in its infancy. In consequence, we lack standard definitions for Internet of Things. With the potential for great mischief through hacking, security issues are pivotal to the success of systems-integration designs. A survey lists several IoT definitions as formulated by different researchers.

Kevin Ashton (born 1968) is a British technology pioneer who co-founded the Auto-ID Center at the Massachusetts Institute of Technology (MIT), which created a global standard system for RFID and other sensors.

In general, the following three complementary – and partly overlapping – visions can be distinguished (Atzori et al. 2010, Bandyopadhyay and Sen 2011):

1. The *Things oriented vision* focuses on the things' identity and functionality, which is in line with the original idea which was initially tied to the RFID and Electronic Product Code (EPC), other identification alternatives have emerged, and the concept of an identifiable object has been expanded to include also virtual entities. From this perspective, IoT is defined as:

   > Things having identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within social, environmental, and user contexts.

   or

   > A world-wide network of interconnected objects uniquely addressable, based on standard communication protocols. (EPoSS 2008)

2. The *Internet oriented vision* emphasizes the role of the network infrastructure and is concerned with the applicability of the available (and future) Internet infrastructure, including IP protocol stack and Web standards for the purpose of interconnecting smart objects. This perspective is promoted by, for example, the IPSO (IP for Smart Objects) Alliance 1 , Internet architecture (Gershenfeld et al. 2004), and Web of Things community, suggesting that IoT shall be built upon the Internet architecture, by adopting and, when necessary, simplifying the existing protocols and standards. From this perspective, IoT can be defined (following the definition by CASAGRAS project) as:

   > A global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities. This infrastructure includes existing and evolving Internet and network developments. It will offer specific object-identification, sensor and connection capability as the basis for the development of independent cooperative services and applications. These will be characterised by a high degree of autonomous data capture, event transfer, network connectivity and interoperability.

3. The *Semantics oriented vision* focuses on systematic approaches towards representing, organizing and storing, searching and exchanging the things-generated information, by means of semantic technologies (Toma et al. 2009; Barnaghi et al. 2012). According to this vision, the application of semantic technologies to IoT

   > promotes interoperability among IoT resources, information models, data providers and consumers, and facilitates effective data access and integration, resource discovery, semantic reasoning, and knowledge extraction" [through] "efficient methods and solutions that can structure, annotate, share and make sense of the IoT data and facilitate transforming it to actionable knowledge and intelligence in different application domains. (Barnaghi et al. 2012)

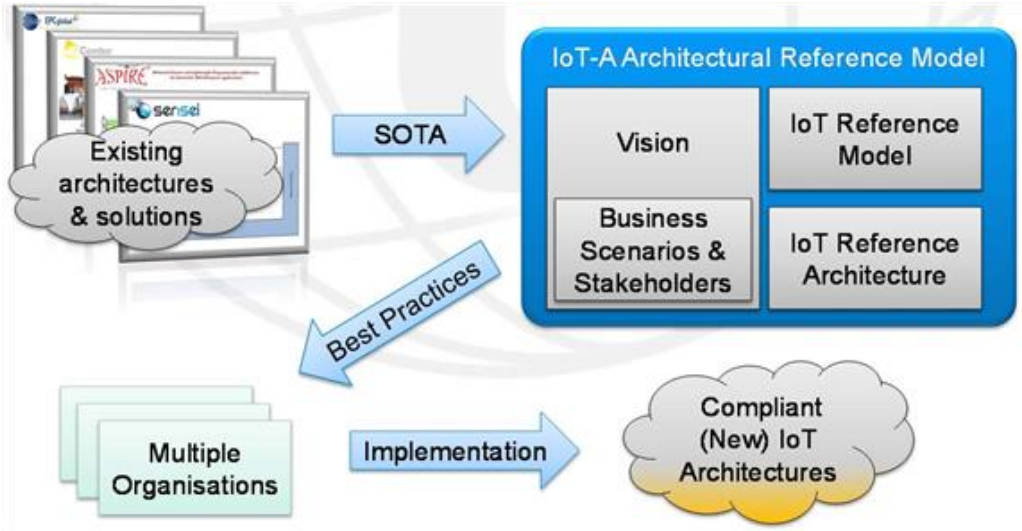
IoT research has its roots in several domains addressing different IoT aspects and challenges. These research domains include, for example, the radio-frequency identification (RFID), machine-to-machine (M2M) communication and machine-type communication (MTC), wireless sensor and actuator networks (WSAN), ubiquitous computing, and web-of-things (WoT). Furthermore, these technologies have been applied in many vertical application domains, ranging from automotive and machinery to home automation and consumer electronics. Thus, what is today known as IoT represents a convergence of multiple domains, and IoT can be seen as an umbrella term uniting the related visions and underlying technologies (Atzori et al. 2010).

1. The Architectural Reference Model

New applications can only reach its pinnacle if full collaboration between the previously mentioned vertical domains can be achieved. If we consider also the mentioned fact that IoT related technologies come with a high level of heterogeneity (specific protocols, specific applications) in mind, it is no surprise that the IoT landscape nowadays appears as highly fragmented as the following figure shows:

The IOT-A Tree (source iot-a.eu)

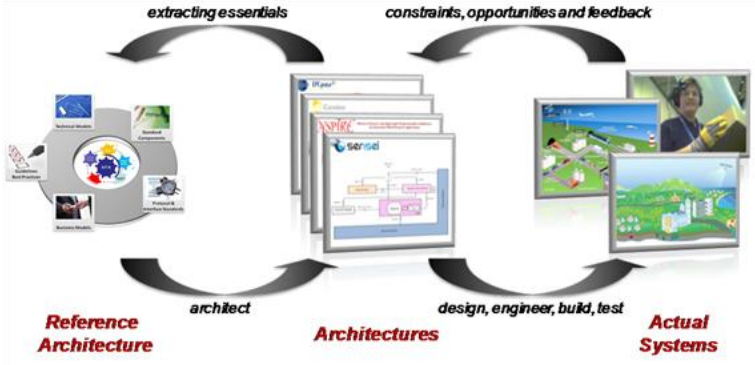
The vision of the Internet of Things IoT-A (http://www.iot-a.eu/arm) wants to promote, a high level of interoperability needs to be reached at the communication level as well as at the service and the information level, going across different platforms, but established on a common grounding. Based on this document, the following figure gives an overview about the building blocks:



IoT-A architectural reference model building blocks (source iot-a.eu)

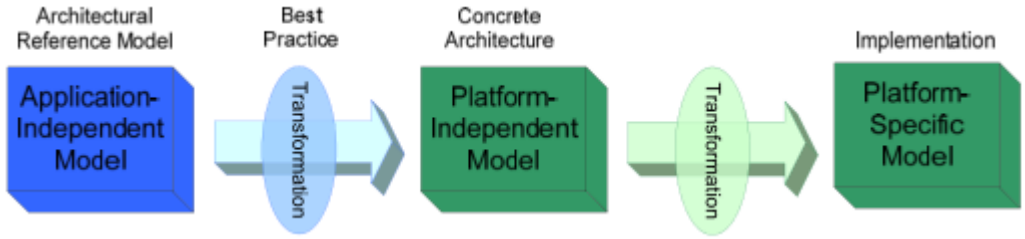
The IoT Reference Model provides the highest abstraction level for the definition of the IoT-A Architectural Reference Model. It promotes a common understanding of the IoT domain. The description of the IoT Reference Model includes a general discourse on the IoT domain, an IoT Domain Model as a top-level description, an IoT Information Model explaining how IoT knowledge is going to be modelled, and an IoT Communication Model in order to understand specifics about communication between many heterogeneous IoT devices and the Internet as a whole. The foundation of the Reference Model is the Domain Model that introduces the main concepts of the Internet of Things and the relations between these concepts. From this model, an other model is derived which holds the structure of all the data that is handled by an IoT system on a conceptual level. The third layer in this vision is the Functional model which groups of functionalities around the key concepts in the Domain model.

The IoT Reference Architecture is the reference for building compliant IoT architectures. As such, it provides views and perspectives on different architectural aspects that are of concern to stakeholders of the IoT.



Relationship between a reference architecture, architectures, and actual systems (adapted from Mueller).

Guidance in form of best practices can be associated to a reference architecture in order to derive use-case-specific architectures from the reference architecture. The role of the ARM is to guide the architect through design choices at hand, and to provide best practices and design patterns for those different choices. It can be visualized as an OMG process for MDA:



Relation of the Best-Practice-driven derivation of concrete architectures from an architectural reference model and the derivation of implementations from said concrete architecture. (source iot-a.eu)

2. IoT Application

Internet of Things is not the technology that we get form only a certain interest and investment. It is a way of turning the raw data to wisdom of humanity. While we saw the evolution of the Web in the previous chapters, IoT is a step forward in the Internet's side. The Web is in the application layer that sits on the hardware layer, sits on top of the Internet. At first, it was simply a read-only content. Then it moved on a phase that we can call as Transactional Web. It was the time that companies,holdings and factories were putting their wares online available and promote their branding. And all these improvements over time helped us create the Social Web. Facebook, Twitter and so on. The main idea is getting people communicate with each other using the Web.

With Internet of Thing, the next level of this improvements, Internet will not be recalled as people-centric, the machine surrounded by sensors will communicate and provide data too. Why it is a step forward? First of all it will be more mobile then the current web. Its not fixed anymore, it available 35.000 feet up in the air and also in the ground. Its open for new way like cloud and virtualization. IPv6 will be the dominant protocol of the next generation. It has 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses. We will see many sensors will be added to the Internet - it will be sensor-laden. We will have so many absolute information of temperature, pressure, location, position and so on. Maybe we will go through a phase where Internet is more about machines then people. Moreover, the Internet is doubling the size of itself in every five years. So five years from now there will be twice as many devices connected and addressed to the Internet as there are today.

Nowadays, 93% of global population have wireless connectivity. 294 million consumer electronic devices with Wi-Fi shipped in 2007. 1 billion in 2012. And not just the amount of smart devices are increased, the data traffic also exponentially increased:

In 2003: 1.8 petabytes

In 2007: 161 exabytes

In 2009: 487 exabytes

In 2010: ½ zettabytes

In 2011: 1 zettabyte (540.000X increase from 2003)

In 2012, just 20 regular house hold generated more traffic than the entire Internet did in 2007.

Why is this important?

We are separated from other species with our ability to communicate. So with communicating we can turn the raw data into information and turn it into knowledge and turn it into wisdom. With the invention of the greatest tool that human created ever, we can surround the planet and universe with sensors and smart machines, get the raw data and turn it into wisdom of humanity. Learn what’s going on in universe! So it will make our generation much more smart and intelligent.

The IoT technologies can be applied in a variety of domains of which the following will be discussed in this section:

- Automotive/Transportation applications, e.g., in-vehicle infotainment, eCall, parking meters, information sharing about road conditions and traffic density, road pricing, toll collection, taxation, pay as you drive (PAYD) car insurance
- Digital/Connected home including (home) consumer electronics, home automation, utilities/automated meter reading (AMR), and residential security
- Healthcare solutions including the monitoring solutions to support wellness, prevention, diagnostics, or treatment services.

Now, we could ask ourselves: What do trees, cows and shoes have in common?

They can be all parts of the Internet of Things. They are all new inhabitants in it. How? Its simple.

There is a company they stucked sensors to farmed cows and they collect 200mb data everyday about their dietary habits and likely behaviours.

There is a tree in Brussels which is surrounded by cameras and sensors. Its able to collect information about its environment. It has 3000 twitter followers.

There are shoes such like NIKE plus which has sensors at some specific places which can detect your movement, and the pressure. It is able to measure how much you walked or ran and send it.

Besides all these benefits that internet of things has brought to us, there will be some side effects as you may think. For example, we are willingly recording our private life with this connected devices. And decoding & processing this data will cause complications, securing and keeping this information private will be such a big issue as well.

In near future we will see that interactions and Internet will be mostly in every object and we will see that different objects will be used in some common purposes. We will see how human will respond to that. Fields of applications include: waste management, urban planning, environmental sensing, social interaction gadgets, sustainable urban environment, continuous care, emergency response, intelligent shopping, smart product management, smart meters, home automation and smart events.

For example, Songdo, South Korea, the first of its kind fully equipped and wired ubiquitous, or smart city is near completion. Nearly everything in this digital metropolis of smart homes is planned to be wired, connected and turned into a constant stream of data that would be monitored and analyzed by an array of computers with little, or no human intervention. Thus, Internet of Things, or embedded intelligence in things, with "smart systems that are able to take over complex human perceptive and cognitive functions and frequently act unnoticeably in the background" is a close reality.

Center of Songdo

3. Common patterns

3.1. Smart phones

Smart phones are a very common element in many IoT-related scenarios. They are on the one hand devices containing a multitude of sensors, but they also host apps (Active Digital Artefacts), services, and resources. E.g.: the GPS sensor is embedded in the phone itself. It is thus embedded sensor hardware. Its data is made accessible through the related On-Device Resource and the location service that exposes it. An app can be used to display the location information.

3.2. M2M interaction

Internet of things has been mostly associated to with machine-to-machine (M2M) communication in manufacturing and power, oil and gas utilities. *Machine to Machine* (M2M) is a term or label that can be used to describe any technology that enables networked devices to exchange information and perform actions without the manual assistance of humans. M2M communication is often used for remote monitoring. In product restocking, for example, a vending machine can message the distributor when a particular item is running low. M2M communication is an important aspect of warehouse management, remote control, robotics, traffic control, logistic services, supply chain management, fleet management and telemedicine. It forms the basis for a concept known as the Internet of Things (IoT).

Key components of an M2M system include sensors, RFID, a Wi-Fi or cellular communications link and autonomic computing software programmed to help a networked device interpret data and make decisions.

The most well-known type of M2M communication is telemetry, which has been used since the early part of the last century to transmit operational data. Pioneers in telemetrics first used telephone lines -- and later, on radio waves -- to transmit performance measurements gathered from monitoring instruments in remote locations. The Internet and improved standards for wireless technology have expanded the role of telemetry from pure science, engineering and manufacturing to everyday use in products like home heating units, electric meters and Internet connected appliances. Products built with M2M communication capabilities are often marketed to end users as being “smart.”

3.3. RFID gates and cards

Historically, the IoT term was introduced in a presentation where RFID was used to demonstrate the possibilities lying in this filed. The most well-know use case was the goods tracking scenario during the supply chain. However, we have increased its usage scenarios not just relying on goods tracking and logistic but passing gates.

RFID card

The technology of RIFD (Radio Frequency IDentification) is simply using radio frequencies to identify objects. Its mainly similar for a barcode and its reader. A generic term for technologies that use transponders to identify an object. The technology has its origins in the first part of the twentieth century and was initially used to identify military aircrafts as friend or foe. Today RFID technology has found various applications in security, logistics and maintenance, just to name a few of them.

How does it work?

We can separate the work and the ware, as two part. Tags and readers. The tag is looking like this:

RFID tag

The Reader may have many different forms but we can generalize it in such like this:

RFID reader

RFID readers are devices for reading out RFID tags via radio signals. Different versions of readers exist that are tailored to the targeted application. Examples are mobile reads or RFID gates for monitoring items that pass through a door. RFID tags consist of a chip for storage and computation and antenna for communication. We categorize RFID tags according to their energy supply, data storage capabilities and communication frequency. Tags without batteries are referred as passive tags. These tags harvest energy from the communication signal sent by the RFID reader to run their operations.

4. IPv6 and short-range protocols

IPv6‘s huge increase in address space is an important factor in the development of Internet of Things. **Internet Protocol version 6** (**IPv6**) is the latest version of the Internet Protocol (IP), the communications protocol that provides an identification and location system for computers on networks and routes traffic across the Internet. IPv6 was developed by the Internet Engineering Task Force (IETF) to deal with the long-anticipated problem of IPv4 address exhaustion.

IPv6 is intended to replace IPv4, which still carries the vast majority of Internet traffic as of 2013. As of February 2014, the percentage of users reaching Google services over IPv6 surpassed 3% for the first time.

Every device on the Internet is assigned an IP address for identification and location definition. With the ever-increasing number of new devices being connected to the Internet, the need arose for more addresses than the IPv4 address space has available. IPv6 uses a 128-bit address, allowing 2128, or approximately 3.4×1038 addresses, or more than 7.9×1028 times as many as IPv4, which uses 32-bit addresses. IPv4 provides approximately 4.3 billion addresses. The two protocols are not designed to be interoperable, complicating the transition to IPv6.

IPv4 vs IPv6

The main advantage of IPv6 over IPv4 is its larger address space. The length of an IPv6 address is 128 bits, compared with 32 bits in IPv4. The address space therefore has 2128 or approximately 3.4×1038 addresses. By comparison, this amounts to approximately 4.8×1028 addresses for each of the seven billion people alive in 2011.In addition, the IPv4 address space is poorly allocated, with approximately 14% of all available addresses utilized. While these numbers are large, it wasn't the intent of the designers of the IPv6 address space to assure geographical saturation with usable addresses. Rather, the longer addresses simplify allocation of addresses, enable efficient route aggregation, and allow implementation of special addressing features. In IPv4, complex Classless Inter-Domain Routing (CIDR) methods were developed to make the best use of the small address space. The standard size of a subnet in IPv6 is 264 addresses, the square of the size of the entire IPv4 address space. Thus, actual address space utilization rates will be small in IPv6, but network management and routing efficiency is improved by the large subnet space and hierarchical route aggregation.

Renumbering an existing network for a new connectivity provider with different routing prefixes is a major effort with IPv4. With IPv6, however, changing the prefix announced by a few routers can in principle renumber an entire network, since the host identifiers (the least-significant 64 bits of an address) can be independently self-configured by a host.

According to Steve Leibson, who identifies himself as “occasional docent at the Computer History Museum,” the address space expansion means that we could “assign an IPV6 address to every atom on the surface of the earth, and still have enough addresses left to do another 100+ earths.” In other words, humans could easily assign an IP address to every "thing" on the planet.

Short-range protocols

Sensor networking and M2M communication have become subject to special interest during the past few years. We present here both proprietary and standardized solutions, which have commercial deployments and concentrating on protocols that are being used for user/device monitoring, home and building automation and automotive applications.

	User/device monitoring	Home automation	Large building automation	Automotive
ZigBee	x	x	x	
Z-Wave		x		
Insteon		x		
EnOcean		x	x	
ONE-NET		x		
KNX		x	x	
LonWorks		x	x	
BACnet			x	
Modbus			x	
IEEE 802.11x (e.g. WiFi)	x	x	x	x
DASH7				x
IEEE 1902.1 (RuBee)	x			
Bluetooth (LE)	x			x
ANT/ANT+	x			
Infrared	x	x		

Short-range protocols for IoT (source: Oleksiy M, et al. 2013)

5. Security Issues Associated to IoT

- **Data tracking:**

  As consumers move towards adopting sensor-based gadgets or wearable devices, the risk of being tracked by companies also increases. They can monitor each and every move and send targeted advertisements. Similarly, if a hacker takes control of any connected devices, they can gain access to a ton of personal information. The more information attackers have in their hands, the more powerful they become.

- **The collection and sharing of personal data with third parties:**

  Another risk with IoT is access of consumer data by third party vendors. Even if you agree to share your data explicitly with a company, there is a possibility that your data might get merged with the IoT data that the same company secure from a third party. As a result, your data will be accessed by third parties, and used for advertising purposes.

- **Lack of security controls:**

  A vast majority of these connected devices will have less protection and will be prone to cyber-attacks. Some of the technologies that IT systems are accustomed with, like operating systems, firmware, and patches will not be available on these devices. IoT devices typically lack anti-malware and anti-virus protection, and don’t have IT teams to monitor and track security issues. As a result, new attacks might evolve to compromise the individual device or gain access to the enterprise network.

- **Security risks to the enterprise network:**

  No matter what network segmentation strategy an enterprise might establish, there will be some security gaps within the system that will allow IoT to intersect with enterprise network. These points of intersection will be highly vulnerable to attack.

6. IoT Criticism and Controversies

While technologists tout the Internet of Things as one more step toward a better world, scholars and social observers have some reservations and doubts about approaching ubiquitous computing revolution. Peter-Paul Verbeek, a professor of philosophy of technology at the University of Twente, Netherlands, writes that technology already influences our moral decision making, which in turns affects human agency, privacy and autonomy. He cautions against viewing technology merely as a human tool and advocates instead to consider it as an active agent.

A different criticism is that the Internet of Things is being developed rapidly without appropriate consideration of the profound security challenges involved and the regulatory changes that might be necessary. In particular,

as the Internet of Things spreads widely, cyber-attacks are likely to become an increasingly physical (rather than simply virtual) threat.

The U.S. National Intelligence Council in an unclassified report maintains that it would be hard to deny "access to networks of sensors and remotely-controlled objects by enemies of the United States, criminals, and mischief makers. An open market for aggregated sensor data could serve the interests of commerce and security no less than it helps criminals and spies identify vulnerable targets. Thus, massively parallel sensor fusion may undermine social cohesion if it proves to be fundamentally incompatible with Fourth-Amendment guarantees against unreasonable search. "In general, the intelligence community views Internet of Things as a rich source of data. Given widespread recognition of the evolving nature of the design and management of the Internet of Things, sustainable and secure deployment of Internet of Things solutions must design for "anarchic scalability."

Application of the concept of anarchic scalability can be extended to physical systems (i.e. controlled real-world objects), by virtue of those systems being designed to account for uncertain management futures. This "hard anarchic scalability" thus provides a pathway forward to fully realize the potential of Internet of Things solutions by selectively constraining physical systems to allow for all management regimes without risking physical failure.

Justin Brookman, of the Center for Democracy and Technology, expressed concern regarding the impact of IoT on consumer privacy, saying that "There are some people in the commercial space who say, ‘Oh, big data — well, let’s collect everything, keep it around forever, we’ll pay for somebody to think about security later." The question is whether we want to have some sort of policy framework in place to limit that."

7. Summary

While IoT devices can serve as energy-conservation equipment, it is important to keep in mind that everyday good habits can bring the same benefits. Practical, fundamental considerations such as these are often overlooked by marketers eager to induce consumers to purchase IoT items that may never have been needed in the first place.

8. References

Bibliography

[JIAHENG2013] Jiaheng Lu. *An Introduction to XML Query Processing and Keyword Search*. 2013. 1. 3642345549. 300. Web page .

[CISCO_IBSG_IoT] Dave Evans. *The Internet of Things How the Next Evolution of the Internet Is Changing Everything*. 2011. 11.

[CISCO_IoT] CISCO. *Internet of Things* . Web page .

[TECHTARGET_IoT] TECHTARGET. *Internet of Things* . Web page .

[spideroak_IoT] spideroak. *Impact of Internet of things on enterprise security*. Web page .

[Songdo] *Songdo International Business District*. Web page .