CHAPTER 2

Behind The Scenes: Some Technical Details on Mashups

API: the Secret of the Mashup[1]

According to the description on the ProgrammableWeb site[2], a mashup involves the combination of data from two or more Web sources to build new applications and services. To create a mashup it is therefore essential to be able to gather structured information from websites (or perhaps to extract the available content and then convert it in a structured format). The most effective and desirable access mechanism to a Web service is its Application Programming Interface (API). APIs are "a set of functions, procedures or classes that an operating system, library or service provides to support requests made by computer programs"[3]. They reveal the underlying logic on which a service is built, its key resources, and the functions amenable to be performed from outside the site. In other words, they allow a computer program to access and manipulate data on a Web service the same way that a website interface lets the human user surf and dive into its content.

APIs can be of different types: each site chooses how to expose its content to the outside, so there may be great differences between API providers - or even within the same provider for its different services. Naturally, in order to be helpful to a programmer, the API must be fully documented, and even better also freely accessible. (or "Naturally publicly documenting APIs and offering them available free of charge makes them easier for programmers to use."?)

Why should a website provide a public API, exposing its valuable information for free online? Because by explaining in detail how its application logic works, it puts developers and users in a position to re-use data in ways the provider itself might never have thought possible. Allowing users to intensively manipulate the contents of a website or to invoke its services from within a third-party client is also an excellent way to test the application: when hundreds (or even thousands) of people begin to develop Web services on top of a site, the site is put to the test and any bug in the code is likely to be discovered and fixed.

If a site has developed a public API, it should advertise its functions to as many people as possible, so information about the API should not be difficult to find on the site. Those seeking API information are likely to look in the "developer" section of the website, in the site map or in the Help or FAQ pages, while those offering APIs should also make sure that theirs are listed in the online sites that gather information about APIs (discussed later in this chapter).

A very important aspect to take into consideration when using an API is its license or the so-called Terms of Service (TOS), which are the conditions set out by the API provider. There may be limitations on the type of usage that developers should make of an API: in most cases public APIs are free for non-commercial use, while for commercial use the provider may charge a fee or not allow the use at all. Additionally there are often limits applying to the rate and bandwidth usage of the client's requests.

It is also worth noting that APIs evolve over the time and that the version may change, so it is necessary to check the documentation online from time to time. Another issue to pay attention to is the copyright status on contents being used in the mashup: items published online may not be legally available for re-use by others, or possibly your application might be asked to meet specific requirements to obtain permissions for re-publication or re-use. For these reasons, pay careful attention to the license under which information has been published and the conditions under which it may be used by third parties. Whenever a specific license is lacking, apply the laws in force in your country.

In addition to the application logic that allows developers to know what resources are made available and what operations can be performed on them, another important aspect of an API is its communication protocols: how, from a technical point of view, does one invoke the programming interface for the information exchange to take place? What are the syntax requirements for the API requests? Also what format will the provider use to package information to be sent back to the client making the request? What transport channels must or may be used for the most rapid and secure content exchange? To better understand and answer these questions, we need to take a quick glance at the world of Web service and its main features.

Web Services Architectures

The W3C defines a Web service as “a software system designed to support interoperable machine-to-machine interaction over a network."[4] Web services are a technology that enables information and communication exchange between different applications. Understanding how they work is clearly a fundamental requirement for correct use of programming interfaces and the creation of effective mashups.

Web services are based on a conceptual model that has a *service provider*, which is an application that makes certain information available or that provides the capability to perform certain operations, and a *service consumer*, which will make use of the information or the services. The service consumer (e.g., the website in which the mashup takes place) issues a request to a service provider (e.g., the website that provides the API). If this request is issued correctly, the service provider sends back a response formatted according to the service’s rules (Web services require compliance only with their stated rules and are independent of any hardware or software platform on which they are implemented, thus ensuring a high level of interoperability).

In practice, the transport protocol and the language most commonly used are HTTP and XML. If your program will acquire information, data or services from a website through its application programming interface, you need to know the particular communication requirements for that site and how to format the request according to those requirements. You will then send your request over the HTTP protocol choosing one of its methods: either GET (the one employed in the everyday navigation on the Internet), POST (used mainly by the service consumer to send information to the service provider), PUT or DELETE (for operations to be performed on the server)[5].

Frameworks in the Web service architecture may be different depending on the technology and protocols employed: the most widely used are REpresentational State Transfer (REST)[6] and SOAP[7], which lay the foundations, respectively, of resource-oriented and service-oriented architecture[8].

REST is the simplest and thus by far the most used protocol in the creation of mashups. As we shall see in the *An Example with Yahoo! Answers* section, REST consists of a simple to implement interface based on a resource-oriented architecture. REST assumes a service user that requests information (or triggers operations) from a specific application. The request employs a URL containing the API parameters, and is transmitted using one of the methods (e.g. GET) of the HTTP Protocol[9]. Note that REST architecture requires operations to be invoked by the HTTP method, not from within the URL sent over the network, provided that the URL only carries the scoping information (that is, the requested resources). This means that, in a pure REST style, if you simply ask for information from a Web service, you should use the GET HTTP method, while if you ask the service to perform any operation on its resources, your request should take a different method, either POST, PUT or DELETE[10].

When the query is sent over the network, the service provider sends back a response with the information formatted in a language (typically in XML, although other response formats like JavaScript Object Notation (JSON), are being increasingly adopted). The client can then use the response as an input for other operations or can render it graphically in a Web page. As you can see, the REST Web service's request-response logic is the same that is used on the Web when a human user navigates through the various online venues. The only differences are that on one hand the transactions are activated through the API calls rather than URLs sent via browser, and on the other hand that the response format is intended for computer - rather than human - consumption[11].

SOAP is another Web service style that has developed alongside of REST; it relies on international standards and protocols[12] and has been adopted primarily in the enterprise world.

SOAP uses HTTP as the transport protocol for exchanging information, but it requires that both the requests sent by the service consumer and the answers returned by the service provider be wrapped in an XML envelope. The provider itself describes the Web service through specific XML schemas which are then published online so that the consumer application can conform to it.

Most Web 2.0 sites use REST interface for their APIs: This makes sense when you consider that one of the Web 2.0 goals is to lower the technical barriers that in the past have prevented the average user from active participation in online information production. It is easy to issue a request by adding parameters to a URL and sending it to a server from a browser or an HTML form. It is even easier if the URL takes the GET method. With little or no knowledge of programming, building a rich and compelling mashup is possible today thanks to REST.

SOAP is definitely more complex to implement and, according to the REST fans (the so-called RESTafarians), it fails to promote the ease of use and effectiveness inherent in the World Wide Web concept. It is, however, important to understand the SOAP architectural style, both because many sites use only SOAP, and because SOAP is more standardized and refined than REST. You might also find SOAP to be a requirement for business-oriented development projects.

Where to Find APIs Online

As you first approach the world of mashups, it is very useful to have access to the experience of other users, as well as to learn which websites are most open to information re-use. A resource of great utility to consider for this type of training is ProgrammableWeb. This site has the most comprehensive list of mashups[13]; it also collects news from the Web, trends from the industry landscape, technical information, guides and references. On ProgrammableWeb, mashups can be browsed by category, popularity, date, and tags. Every mashup has an associated description of its functionalities, along with the APIs used, tags, author information and links to related mashups. In addition, ProgrammableWeb users can vote on the mashups, add mashups to their favorites list, and follow up on new applications using the same API, via RSS feeds.

As Raymond Yee explains in his excellent book "Pro Web 2.0 mashup"[14], a useful way to develop the creative mashup ideas is to study the applications already created by other users and ask yourself questions like:

“What is being combined?

Why are these elements being combined?

Where is the remixing or the recombination happening?

How are various elements being combined (that is, first in the interface but also behind the scenes in the technical machinery)?

How can the mashup be extended?”[15]

These questions and their answers provide a useful grid with which to analyse any mashup found on ProgrammableWeb and help you thoroughly understand the dynamics and problems involved in a mashup.

You can also learn quite a bit by studying the message conveyed by the components that enable the mashup, in particular the APIs - the real workhorses of the mashups. On this site each API has an associated analytical fact sheets that gives a general description, tags, latest news, a list of mashups that use it, which protocols are implemented (Web services styles like REST and SOAP, and data formats), functionalities (for example different types of methods to be invoked), security models (authentication, SSL etc.), support (offered both from the vendor and the community worldwide) and the signup requirements and licensing terms of service. Often there are also guides on how to use the API, including feedback from those who have used it.

The information provided on ProgrammableWeb and its how-to documents are a very valuable resource, giving an idea of the range of sites that allow you to exploit their services. Moreover, beginning mashup developers will appreciate the simplicity of the site’s contents and the ability to search the database also for code libraries and development tools.

The ProgrammableWeb is a general directory and among the several thousand mashups described, very few are library-oriented. Thanks to the efforts of the Library 2.0 movement and to the release of information management software that are more modular and standards-compliant, in recent times library-related mashups and public APIs are gaining a certain public attention. The *Library Software Manifesto*[16] calls for free access to APIs for libraries that have purchased a library system. It is often only because of such public access interfaces that all of the data - bibliographic or not - stored in library databases can be exploited beyond the set of uses provided by vendors (here we talk of legal and permitted uses!). Although libraries, as organizations that deal primarily with information, may seem to be logical players in the world of data and Web services, in the past there has not been much awareness neither about the importance of open APIs nor about the potential they could have for libraries' goals. For example, there are not many APIs for online catalogs and, apart from the queries allowed by Z39.50 protocol, it is difficult, or even impossible, to create a mashup like the one that John Blyberg shows in chapter ... without having an API that supports it. Often librarians do not know if the software in use in the library has an API and vendors do not volunteer this type of information.

This makes all the more important the effort of the *Mashed Library* social network[17], which has collected a list of library-oriented APIs and Web services that were then incorporated into a permanent list on TechEssence[18]. Among the APIs listed on the TechEssence blog there are many that interact with OPACs as well as others relating to software for digital resource management made available by some of the major players in the information technology scene, such as the OCLC consortium, the e-commerce website Amazon, the library-related products vendor Talis, and the social bookmarking service LibraryThing. We should also mention the JISC Information Environment Service Registry (IESR)[19], a machine readable registry of electronic resources, including APIs, with access information and documentation, and the Search and Retrieve via URL (SRU) Web service, developed by the Library of Congress as an evolution of Z39.50 protocol[20].

It is time to foster bottom-up initiatives in the library and information science arena and to harness librarians' creativity to make more flexible and up-to date use of the library software. It is time to extensively exploit integrated library systems in a more user-oriented manner, shaping the ways different sources of digital content expose their data to make information suitable to be used by third-party application like a blog, a Web calendar, or an RSS feed. This way the users themselves will be able to create applications that neither vendors nor librarians might ever imagine.

If, after reading this book, you are wanting to get your hands dirty with mashup development, you can take either the ProgrammableWeb or the TechEssence list as a starting point for discovering API magic.

Mashups without APIs

So far we have described application programming interfaces and their styles of communication, but what happens if a site does not have structured interfaces through which other services can retrieve the needed information? While there are many sites that are equipped with APIs, not all make them publicly available to third-party applications; in some cases you can use the API only by paying a fee, while in others APIs are kept secret by companies or organizations because they are considered strategic for the commercial success of a product or service. There are also websites that do not offer any programming interfaces with which to interact. In these cases it is more difficult to utilize the data, nonetheless there could be good chances to create a mashup by tapping into the sources provided.

Raymond Yee, in his book, suggests that you first study the specific URL language used by the site. This is useful even if a public API is present, but becomes crucial in a case where there is no API and you have to discover the underlying information architecture adopted by a site on your own. The way in which Web addresses are built may provide clues about whether and how information is structured into URL-identifiable resources, perhaps even categorized, tagged or organized in some way so that a computer program may operate with them.

One reliable source of consistently formatted data that allows for re-use in Web services is a site's feed.[21] Feeds are nothing more than small chunks of information formatted in XML or in one of its dialects (such as RSS or ATOM) normally used by websites to disseminate updated content and read by users by means of aggregators or feedreaders. In fact feeds can be seen as a basic RESTful Web service, given that an XML-formatted response is invoked by a URL request over HTTP.

Utilizing the file that underlies the typical orange icon indicating the presence of a feed, the developer can parse the information and use it as input for another application or render it directly in a Web page. The presence of content import/export features or the use of Web development techniques like Ajax[22], are other hints that the site has data that is processable by an external program.

If a site does not provide either an intelligible URL language nor feeds nor other gateways to access its data, then you must rely solely on its Web interface using the technique known as "screen scraping." [23] Through this mechanism, information intended exclusively for human consumption is extracted from the Web page and sent as input to a computer program. The screen-scraper program then acts on the content presentation layer of the Web pages and uses its HTML tags as hooks to identify the desired information resources.

Naturally these hacking techniques produce results that are often fleeting and unreliable: if a site doesn't allow for an API, there may be solid reasons - including legal ones - why it does not make its data available for third party use. You must analyze carefully whether screen scraping can be used in a particular situation, even if it represents the only opportunity you have to obtain some piece of information.

An Example with Yahoo! Answers

It's time now to move from theory to practice and see how a real Web service requests data from an online source and processes it to make it usable in a mashup. Given that mashups generally combine data from two or more information resources, for each source in your mashup you will need to understand its particular inputs and outputs, as we illustrate here with Yahoo! Answers.

There are many websites that make their APIs available to users: search engines like Google, which provides the opportunity to query its index; media sharing sites like Flickr and YouTube, from which you can draw pictures and movies; aggregators such as Technorati and Feedreader, from which to get user-generated content and feeds, and many others. Yahoo! itself offers a wide variety of APIs and Web services that interact with its search engine as well as with many others of its online services such as maps, music, financial information, social bookmarking services and so on. In the Yahoo! Developer Network (YDN)[24] you can find all of the necessary information about the application interfaces and other related technologies, along with guides and video-tutorials, Web design libraries[25], a mashup gallery[26], reports on hacking initiatives, and support from both the company's team and the developer community. Out of all of these, the service we have chosen as an example for this chapter is Yahoo! Answers[27].

In its own words, Yahoo! Answers is "an online community where anyone can ask and answer questions on any topic. Yahoo! Answers connects people to the information they're seeking with those who know it"[28]. Consulting the YDN Answers section[29] we discover that APIs allowing access to questions and answers posted by Yahoo! Answers' users is available for public use. The API uses a REST interface so we know this means we need to:

- build a URL with parameters specified by the API
- and send it using one of the required HTTP methods.

If you want to use the Yahoo! Answers API, you have to request an Application ID[30]. This procedure is often required by sites hosting open APIs, because this way the API provider (that is, the service provider) can track the number and types of the requests and the bandwidth each application makes use of, in addition to the characteristics of the client requesting the content.

There are four types of queries provided by the API: *questionSearch*, *getByCategory*, *getQuestion*, and *getByUser*. These reply with, respectively: Questions that match your query's argument; questions from a certain category; details of all the answers given to a question; Yahoo! Answers questions from a specific user. The API page also has a form so that you can immediately try your queries and verify the answers returned by the service provider. The only HTTP method available to the API is GET, therefore you can only read information from the site, not upload, edit or delete data on the server.

The first of the queries, *questionSearch*, "finds open, resolved, or up-for-vote questions that include your search terms." To access searches you must build a request that retrieves questions using the parameters provided by the API.

The target URL to which the request must be sent is:

http://answers.yahooapis.com/AnswersService/V1/questionSearch

where *http://answers.yahooapis.com/* is the API hostname, and *AnswersService* the service. The acronym *V1*, located just after it, refers to the API version number and *questionSearch* represents the query type we have chosen.

Some of the query arguments that you can employ for this kind of request are:

query - search terms (required)

category_id - search only in the specified category ID or IDs (IDs may be seen in the request URLs when browsing Yahoo! Answers categories)

region - filter based on country (e.g., us: United States; uk: United Kingdom; it: Italy and so on)

sort - sorting order of result set

- relevance: By relevance (default)
- date_desc: By date (newest first)
- date_asc: By date (oldest first)

(omit for default "relevance")

appid - the application ID (required)

output - defines the output for the call. Accepted values are "xml", "json", "php", and "rss" (omit for default "xml")[31]

For example, if you would like to get questions asked by users in the United States regarding "solar energy" in the *Green Living* sub-category, and have the results sorted by date with the most recent questions first, you send:

GET http://answers.yahooapis.com/AnswersService/V1/questionSearch?appid=MyYahooAppId&query=solar%20energy&category_id=2115500307&region=us&sort=date_desc[32]

(note that you must substitute MyYahooAppId with your application ID and that arguments and values must be URL-encoded.[33])

The number *2115500307* in *category_id* is the identifier for the *Environment/Green Living* category as shown in the URL of the corresponding Yahoo! Answers page. By default the maximum number of results is ten and the default response format is XML. Because we have not specified either of those parameters, the defaults will be used.

You can test this URL by pasting it in the location bar of your browser.[34] The result is an XML-formatted response containing the requested information, plus a set of further parameters like the *id* and the *nick name* of the user who asked the question, the number of the answers provided for each question and - if they exist - the preferred ones among them. The XML code may be parsed and sent as input for a third-party application[35] or rendered in an HTML page for displaying the results in your preferred style.

It is also possible to get a feed-style response choosing RSS as the output format (*&output=rss*). In this case the URL resulting from the API call is an RSS feed address that can be used by a feedreader to get notified over the time when new questions appear on Yahoo! Answers that meet your criteria[36].

The JSON output format returns a serialized JavaScript, a less complex format than the XML that is provided by default. JSON can be used in combination with the callback function provided among the API parameters to solve the cross-site security issues like the Same Origin Policy that you are likely to encounter if you write your application in a client-side scripting language like JavaScript[37]. Lastly, remember that this Web service limits you to 5,000 queries per IP address per day[38] and that you are asked to agree to the Yahoo! API Terms of Use and Terms of Service.[39] In addition, websites and applications using this Yahoo! Web service must display the attribution "Powered by Yahoo! Answers".

Mashup Editors

A significant advantage of the current level of attention around mashups is the fact that there are more and more tools and services being developed that make building a mashup increasingly simple, even for the average Internet user. Some of these entirely eliminate the need for programming, hiding such technical details behind the interface. Some big Web companies have begun working on editors for mashups, that is, applications that allow you to combine information instantiating simple commands in a visual user interface. As Nicole Engard explains in chapter , Yahoo! with its Pipes[40] has revolutionized the mashup editor.

Other providers have code editors that are aimed at the more sophisticated developer, such as the Google Mashup Editor (GME)[41]. This tool is a development framework that allows you to draw and remix information from external RSS and Atom feeds or Google services such as Maps or Base to create Ajax-based applications. GME includes an editor, a feed browser (which loads the feeds for handling directly into the interface without forcing open a window in the browser), and a sandbox for testing the application on the fly prior to publishing it. GME offers syntax highlighting, tag-completion, error checking and contextual reference, plus a subversion repository for accessing previous versions of the mashup and sharing coding tasks with other developers. Each file can manage XHTML, JavaScript, and CSS as well as GME modules, which are simple tags that are used to instantiate elements or functions. Once the page is published, these code modules are processed by the GME engine and transformed into JavaScript. For the mashup interface you can choose from a series of graphic themes created by GME or load your own CSS. GME uses the Atom

Publishing Protocol[42] as the format for representing information; all the data that are displayed, read or modified within GME are converted to this format. At the time of writing GME is still in beta, and users who would like to try it must request to become part of the beta team. Mashups created with the GME can be published and hosted on Google Mashup Editor website.

Microsoft, too, is taking part in the mashup trend and has enhanced its offer with a Web editor for mashups called Popfly[43]. This program, which requires the use of the Silverlight plug-in, allows the creation of mashups "without writing a line of code", thanks to a graphic editor where you can drag and drop 3-dimensional components (the *blocks*) from a sidebar containing offerings of *Images and videos*, *News and RSS*, *Social networks* and others, plus a basic *Tools* set. This latter provides the operations that can be applied to the other modules to such as merge, sort, or list the content. Sophisticated users can switch to the advanced view that displays the JavaScript code that lies behind each module, which they can then edit directly. Finally it is also possible to customize your application with HTML and to test the running mashup in the debug console.

Thanks to the interconnection of the Popfly editor with Web 2.0 social sites you can easily create mashups like slideshows of your favorite pictures and podcast and video players. Popfly also offers a space for hosting both Web pages and applications developed with the mashup editor.

Among other visual mashup editors that deserve a mention are OpenKapow[44] and Dapper[45], as well as more business-oriented tools like Coghead[46], JackBe Presto Platform[47] and Serena[48] software.

In general, however, for those who are taking their first steps on the mashup path and have limited development skills, it is better to start with one of the first editors described here (Yahoo! Pipes, Google Mashup Editor, Microsoft Popfly). These are fully documented, and have many examples and quite a few pre-built modules that one can exploit. They also have active online communities that can provide support for beginners and advanced users alike.

[1]I would like to thank Karen Coyle and Raymond Yee for their generous feedback and support.

[2]www.programmableweb.com/faq#Q1.

[3]Wikipedia contributors, "Application programming interface," *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/wiki/API. In this chapter we will talk exclusively about Web APIs.

[4]Web Services Architecture Working Group, "Web Services architecture," *W3C*, http://www.w3.org/TR/ws-arch/#whatis. Marshall Breeding provides a comprehensive layout of Web services and Web services for libraries in his "Web services and the Service-Oriented Architecture", *Library Technology Report* 42, no. May/June (2006).

[5]Borrowing from database parlance, HTTP methods invoked by the client interface to be performed on the server are often referred to as *CRUD* (Create, Read, Update, Delete) (http://en.wikipedia.org/wiki/Create,_read,_update_and_delete).

[6]REST architectural principles were conceptualized for the first time by Roy T. Fielding in chapter 5th of his PhD thesis "Architectural styles and the design of network-based software architectures" (University of California, Irvine, 2000) (www.ics.uci.edu/~fielding/pubs/dissertation/top.htm).

[7]SOAP originally stood for 'Simple Object Access Protocol', but this acronym is no longer used.

[8]*In their bible of REST, RESTful Web services*. Farnham: O'Reilly, 2007, Sam Ruby and Leonard Richardson outline with clarity the differences between Web services styles and advantages of resource-oriented over service-oriented architecture. Wikipedia offers a good explanation of the basic concepts of REST architecture: "An important concept in REST is the existence of resources (sources of specific information), each of which is referenced with a global identifier (e.g., a URI in HTTP). In order to manipulate these resources, *components* of the network (clients and servers) communicate via a standardized interface (e.g., HTTP) and exchange *representations* of these resources (the actual documents conveying the information). (...) [A]n application can interact with a resource by knowing two things: the identifier of the resource, and the action required (...) The application does, however, need to understand the format of the information (*representation*) returned, which is typically an HTML, XML or JSON document of some kind, although it may be an image, plain text, or any other content" (http://en.wikipedia.org/wiki/Representational_State_Transfer).

[9]In fact HTTP protocol identifies resources by URI (Uniform Resource Identifier) rather than URL (Uniform Resource Locator). For simplicity of the exposition, here the two are considered synonyms (see Wikipedia's HTTP entry: http://en.wikipedia.org/wiki/HTTP).

[10]The differences among the pure REST and styles like REST-RPC are well explained in the mentioned book *RESTful Web services*. Although most Web services claim to be RESTful, they often expose data in an hybrid manner (for instance they put API methods inside the URL rather than utilize HTTP protocol). See also Duncan Cragg's blog posts: *STREST (Service-Trampled REST) will break Web 2.0* (http://duncan-cragg.org/blog/post/strest-service-trampled-rest-will-break-web-20/) and *The REST dialogues* (http://duncan-cragg.org/blog/tag/dialogue/).

[11]"Every Web application -every website- is a service. You can harness this power for programmable applications if you work with the Web instead of against it, if you don't bury its unique power under layers of abstraction", Sam Ruby and Leonard Richardson. *RESTful Web services* (cit.), p. xiii.

[12]While REST can be considered a *de facto* standard, SOAP has been developed by W3C (www.w3.org/TR/soap/) and OAIS (www.oasis-open.org/specs/).

[13]At the time of writing (September, 2008) ProgrammableWeb enlists more than 3,300 mashups.

[14]Raymond Yee, *Pro Web 2.0 mashups: remixing data and Web services* (Berkeley, CA : Apress, 2008). Yee's book has indeed inspired much of this chapter content. A practical demonstration of his approach is in the live talk he did at the Library of Congress on May, 30th, 2008, available online at this URL www.loc.gov/today/cyberlc/feature_wdesc.php?

rec=4346. Deserve a visit also Yee's blog (http://blog.mashupguide.net/) and the wiki of his workshop at CARL Information Technology Interest Group Meeting on July, 25th, 2008 (http://raymondyee.net/wiki/MashupTheLibrary20080725).

[15]*Pro Web 2.0 mashups: remixing data and Web services* (cit.), p. 3.

[16]The *Library Software Manifesto* (http://techessence.info/manifesto) was posted on *TechEssence* blog on November, 12th, 2007 and received, at the time of writing, more than 10,000 reads. Blog author Roy Tennant explains at the beginning of the post, that "[t]his is offered in an attempt to rationalize the relationship between libraries and library systems vendors, which is presently unhealthy".

[17]Mashed Library (http://mashedlibrary.ning.com/) is a Ning-based social network created in July 2008 by Owen Stephens and sponsored by UKOLN which gathers ideas and proposals to take place an "event at which we try to do interesting stuff with library technology and/or data".

[18]http://techessence.info/apis.

[19]http://iesr.ac.uk/.

[20]SRU and SRU via HTTP SOAP (the former Search and Retrieve Web service) (www.loc.gov/standards/sru/index.html) are XML-focused protocols for search and retrieve information based, respectively, on REST and SOAP interfaces.

[21]http://en.wikipedia.org/wiki/Web_feed.

[22]Ajax is a set of technologies comprising JavaScript and XML for creating rich and interactive Web applications (http://en.wikipedia.org/wiki/AJAX).

[23]Wikipedia defines screen scraping as "a technique in which a computer program extracts data from the display output of another program. The program doing the scraping is called a screen scraper. The key element that distinguishes screen scraping from regular parsing is that the output being scraped was intended for final display to a human user, rather than as input to another program, and is therefore usually neither documented nor structured for convenient parsing. (http://en.wikipedia.org/wiki/Screen_scraping).

[24]http://developer.yahoo.com/.

[25]The Yahoo! User Interface (YUI) Library (http://developer.yahoo.com/yui/) provides DHTML and JavaScript code and CSS samples for building interactive, usable and accessible Web applications. The YUI Library has been released as open source and free for all uses.

[26]On the Yahoo! Gallery website (http://gallery.yahoo.com/) users can upload and browse mashups and plug-ins created with Yahoo! APIs and Web services.

[27]http://answers.yahoo.com/.

[28]http://help.yahoo.com/l/us/yahoo/answers/overview/overview-55778.html. Questions stay open for some days to let other users reply. When a question is answered, the requester can choose the best answer or let Yahoo! Answers community vote on it.

[29]http://developer.yahoo.com/answers/.

[30]An Application ID is a string that identifies your Web application as the source of a request. When you make the API call, you must pass in the ID. You can subscribe for an Application ID from this page: http://developer.yahoo.com/wsregapp/ (you must be logged into Yahoo! to get to the page).

[31]The arguments and their explanations are quoted from *questionSearch* page (http://developer.yahoo.com/answers/V1/questionSearch.html).

[32]See Yahoo! Developer Network to get information about how to builds REST URLs (http://developer.yahoo.com/search/rest.html).

[33] URL encoding (or Percent encoding) is a technique used to convert some special characters of a URL in a valid format. In this case the space between words in compounds terms like “solar energy”, has been encoded to *%20*. See the Wikipedia page (http://en.wikipedia.org/wiki/Percent-encoding) and a list of encoded characters (www.w3schools.com/TAGS/ref_urlencode.asp).

[34]You can call the API from within a Web service written in a server-side programming language like Perl and Php or from an Ajax client. You can also try the API functionalities sending the request trough the command-line program Curl or trough the Firefox add-on Poster. Yahoo! Developer Network offers a Software Development Kit (SDK) (http://developer.yahoo.com/download/) with code libraries available for public use, to implement Web services in diverse programming languages.

[35]For example, a consortium of libraries on a geographical scale would cooperate in reference services, developing a mashup that draw within a unique interface questions about a certain topic from the online reference service’s users and from the local Yahoo! ones.

[36]If your goal is only to keep informed of newly posted items, you can also simply using Yahoo! Answers human interface, that lets you save an RSS feed for search’s results.

[37]With the Same Origin Policy, most browsers don't allow a JavaScript client to get information from a source different from the one which is making the request (that is, from within you application you can’t load content from another server, like, for example, the Yahoo! Answers'one). For more information see Wikipedia’s entry at http://en.wikipedia.org/wiki/Same_origin_policy.

[38]See information on rate limiting (http://developer.yahoo.com/search/rate.html) and Yahoo! Web Services Usage Policy (http://developer.yahoo.com/usagePolicy/) to get informed about acceptable uses.

[39]Available at these URLs: http://info.yahoo.com/legal/us/yahoo/api/api-2140.html and http://info.yahoo.com/legal/us/yahoo/utos/utos-173.html.

[40]http://pipes.yahoo.com/pipes/.

[41]http://code.google.com/gme/.

[42]See Internet Engineering Task Force (IETF)’s RFC 5023 (http://tools.ietf.org/html/rfc5023).

[43]www.popfly.com/.

[44]http://openkapow.com/.

[45]www.dapper.net/.

[46]www.coghead.com/.

[47]www.jackbe.com/products/index.php.

[48]www.serena.com/mashups/index.html.

References

(links accessed 10th September, 2008)

Arkin, Assan. “Scraping with style: scrAPI toolkit for Ruby”. Labnotes, posted on July 11th, 2006, http://blog.labnotes.org/2006/07/11/scraping-with-style-scrapi-toolkit-for-ruby/

Bloch, Joshua. “How to design a good API and why it matters”. http://www.slideshare.net/guestbe92f4/how-to-design-a-good-a-p-i-and-why-it-matters-g-o-o-g-l-e

Campbell, Ryan. “How to add an API to your Web service”. ParticleTree, posted on August 31th, 2006, http://particletree.com/features/how-to-add-an-api-to-your-web-service/

CARL North Information Technology Program Archives. “Mashup the library. A workshop on mashup technology and the art of remixing library and information resources”. http://carl-acrl.org/ig/carlitn/archives.html

Fox, Pamela, “Web 2.0 mashups: how people can tap into the 'Grid' for fun and profit”. Talk presented at Open Grid Forum, January 29th-February 2nd, 2007. http://wwww.slideshare.net/wuzziwug/web-20-mashups-how-people-can-tap-into-the-grid-for-fun-profit-20924/

Garrett, Jesse J. “Ajax: a new approach to Web applications”. Adaptive Path, February 18th, 2005. http://www.adaptivepath.com/ideas/essays/archives/000385.php

Gregorio, Joe. “How to create a REST protocol”. XML.com. http://www.xml.com/pub/a/2004/12/01/restful-web.html

He, Hao. “Implementing REST Web services: Best practices and guidelines”. XML.com. http://www.xml.com/pub/a/2004/08/11/rest.html

Heilmann, Christian, *Beginning JavaScript with DOM scripting and Ajax: from novice to professional*, Berkeley, CA : Apress, 2006

Herren, John. “Introduction to mashup development”. Talk presented at the 4th Mashup

Camp, Mountain View, CA, July 16th-19th, 2007. http://www.slideshare.net/jhherren/mashup-university-4-intro-to-mashups

Levitt, Jason. “JSON and the dynamic script tag: easy, XML-less Web services for JavaScript”. XML.com. http://www.xml.com/pub/a/2005/12/21/json-dynamic-script-tag.html

OASIS. “SOA reference model”. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

Schnell, Eric. “Mashups and Web services” in *Library 2.0 and beyond: innovative technologies and tomorrow's user*, edited by Nancy Courtney, 63-74. Westport, Conn. : Libraries Unlimited, 2007

Theurer, Dan, “Web services + JSON = dump your proxy”. Dan Theurer blog, posted on December 15th, 2005. http://www.theurer.cc/blog/2005/12/15/web-services-json-dump-your-proxy/

Udell, Jon. “The beauty of REST”. XML.com. http://www.xml.com/pub/a/2004/03/17/udell.html

Udell, Jon. “Library Lookup Project”. http://weblog.infoworld.com/udell/stories/2002/12/11//librarylookup.html

Yee, Rayomond. “Semantic Search the US Library of Congress”. ProgrammableWeb blog, posted on April, 29th, 2008. http://blog.programmableweb.com/2008/04/29/semantic-search-the-us-library-of-congress/

Zakas, Nicholas C., Jeremy Mc Peak, and Joe Fawcett, *Professional Ajax*, 2nd edition, Indianapolis, IN : Wiley Technical Pub., 2007

Further References

Curl. http://curl.haxx.se/
Google Code. APIs and developer tools. http://code.google.com/more/
MashupCamp, University and Wiki. www.mashupcamp.com
Poster add-on. https://addons.mozilla.org/en-US/firefox/addon/2691

ProgrammableWeb. Mashup feed. www.mashupfeed.com/

REST wiki. http://rest.blueoxen.net/cgi-bin/wiki.pl

W3 Schools. SOAP Tutorial. http://www.w3schools.com/soap/

Biography

Bonaria Biancu works for the Library of the University of Milano-Bicocca, in Milan (Italy), involved in managing the website, the digital library and the University's open archive. She is the author of the blog "The Geek Librarian" and the social network "Biblioteca 2.0". She holds courses about Web 2.0 and Library 2.0 and has attended to numerous conferences and seminars. E-mail: bonariabiancu@gmail.com