CHAPTER 1

Learning from Specific Mashups

Before you set out to build your own mashups, you'll study some specific examples in this chapter. Mashups combine content from more than one source into a new integrated whole. You can understand a specific mashup by answering a number of basic questions:

- What is being combined?
- Why are these elements being combined?
- Where is the remixing or 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?

This chapter will explore three major examples:

- Housingmaps.com
- The Google Maps in Flickr Greasemonkey script
- Jon Udell's LibraryLookup bookmarklet

In this chapter, I will analyze these three examples using the previous questions loosely as a framework. A close study of each of these mashups will be amply rewarded when you start creating your own mashups.

Looking for Patterns in Mashups

One pattern you will see repeated among mashups that link two web sites is the combination of three actions:

1. Data is extracted from a source web site.
2. This data is translated into a form meaningful to the destination web site.
3. The repackaged data is sent to the destination site.

Of course, the details differ among the mashups, but this general pattern holds true, as you will see in the three mashups presented in detail in this chapter. *Where* the remixing actually

happens differs in the three mashups you'll see in this chapter: in a separate application as in Housingmaps.com, in Flickr for the Google Maps in Flickr script, and in the browser without a change of interface as in the LibraryLookup bookmarklet.

Although you'll see this pattern of data extraction, translation, and redirection in the mashups covered in this chapter, you'll find other patterns in mashups as well. Chapter 9 will explore those other patterns in detail.

UNDERSTANDING THE TERMINOLOGY

Throughout the book, I use a number of related terms (*mashup*, *remix*, *recombine*, *data*, and *services*) to describe differing aspects of reusing intellectual and creative work to build derivative works. Of course, reuse—whether in the form of artistic appropriation, scholarly attribution, literary quotation and allusion, parody, or satire—has a long history throughout human intellectual, creative, and commercial endeavors. Some terms, such as *reuse* (as in *software reuse* or *code reuse*), have been in popular usage for a while. Others, such as *remix* and *mashup*, have more recently arisen in the context of discussions around Web 2.0 to apply to the combination of data from disparate sources, often via the use of XML and XML web services. In some ways, *mashups* has won out as the term to refer to web interfaces and applications that combine content into something new, whereas the term *remix* is generally about reusing media while still having broader usage (as in *remix culture*).

The boundary between *mashup* and *remix* is a bit fuzzy, though. *Mashup* and *remix* are terms that have their origins in popular music.[1] Roughly speaking, a *remix* is an alternate version of a song, while a *mashup* brings together elements of two or more songs. The term *mashup* has expanded recently to describe the combination of video from multiple sources in a new video.[2] At this point, I will say that if I wanted to make the parallels from popular music hold up for digital applications, I would use *remix* to describe scenarios that are about reusing or repackaging data without combining it with other content (for example, using the Flickr API to make a web page that has only Flickr images), and I would reserve *mashups* to refer to combinations of data from a variety of sources (for example, combining Flickr photos with photos from Picasa). But the lines are fuzzy and, in my opinion, not worth the effort to draw too carefully.

Broadly speaking, I focus in this book on software mashups, mostly but not exclusively on web mashups that are remixing data and services. By *data*, I mean any digital content, whether it is on a computer network, on your computer, or on any other device. By *services*, I roughly mean services as in service-oriented architecture and software as a service, meaning web services and any applications that can be reused.

Whereas mashups are strongly associated with Web 2.0, parallel developments going under such names as *composite applications* are occurring in enterprise computing and service-oriented architectures. Composite applications are also concerned with weaving together data and services, though they usually integrate corporate data and supply chains sitting behind firewalls instead of public APIs from Google and Amazon. Although mashups and composite applications share common techniques, they are driven by vastly different cultural factors.

This book focuses on personal information instead of information reuse from an enterprise perspective. Personal information is distinct for its heterogeneity, its connection to personal information management, the need for mass customizability, and the many permutations of hardware, software, and data derived from the unique needs of individuals. Nonetheless, if there are opportunities to draw upon synergies with enterprise Web 2.0 without going far afield, I will do so here.

1. http://en.wikipedia.org/wiki/Mashup_%28music%29 and http://en.wikipedia.org/wiki/Remix
2. http://en.wikipedia.org/wiki/Mashup_%28video%29

Housingmaps.com

When I explain mashups to others, I typically use the example of the web site Housingmaps.com, a mashup of Craigslist and Google Maps. Housingmaps.com is useful in ways that are quick and easy to understand, which invites repeated usage. It also requires no software beyond a modern web browser. Moreover, Housingmaps.com takes two already well-known web applications to create something new.

Figure 1-1 shows Housingmaps.com displaying a specific rental listing. Note the photos of the apartment and the links to Craigslist. All the data is drawn from Craigslist and then displayed in a Google map.

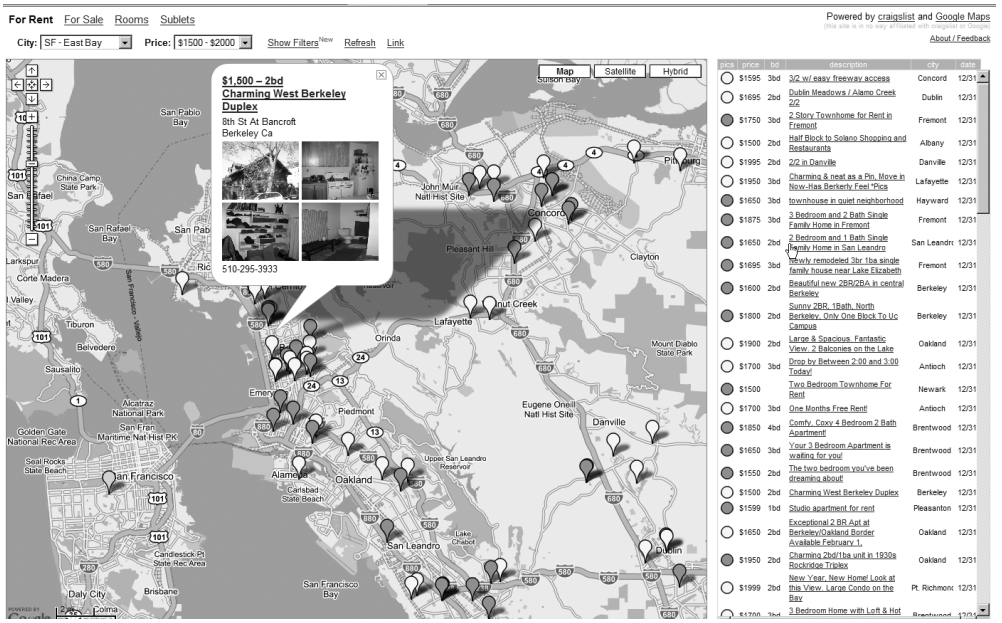

Figure 1-1. *Housingmaps.com*

What Is Being Combined?

Housingmaps.com takes the list of houses, apartments, and rooms that are for sale or rent from Craigslist and displays them on a Google map. Note that it was invented by neither Google nor Craigslist but by an individual programmer, Paul Rademacher, who, at the time of its invention, was working for neither Google nor Craigslist but who was later hired by Google.

Why Are the Constituent Elements Being Combined? What's the Problem Being Solved?

Craigslist provides links to Google Maps and Yahoo! Maps for any individual real estate listing, but it does not map the listings collectively. The single listing per map on the Craigslist interface makes it a challenge to mentally track the location of all the properties. Moreover, when looking for real estate, you often want to look at a narrowly defined neighborhood or find

houses with good access to transit. With Craigslist, you have to click many links and manually piece together a lot of maps to focus your search geographically.

Housingmaps.com addresses these challenge by letting you see on a Google map all the Craigslist apartments or houses in a specific area, not just an individual item. At Housingmaps.com, geographical location becomes the primary lens for looking for real estate, with a map as the central element of the user interface.

Where Is the Remixing Happening?

The remixing occurs on the server side on a web site (Housingmaps.com) that is distinct from both the source web site (Craigslist) and the destination application (Google Maps). Data is drawn from the source and transformed into a Google map, which is embedded in web pages at Housingmaps.com.

How Are These Elements Being Combined?

This question really breaks down into two questions:

- How does Housingmaps.com obtain the housing and rental data from Craigslist?
- How does Housingmaps.com create a Google map of that data?

A desirable, and increasingly common, method for mashups to obtain data from a web site is through a web site's publicly available application programming interface (API). An API is designed specifically to facilitate communication between programs, often including the exchange of data. (You will be introduced in detail to APIs in Chapters 6 and 7.)

At this time, Craigslist does not provide a public API but does provide RSS feeds. As I will discuss in Chapter 4, RSS feeds are used to *syndicate*, or transport, information from a web site to a program that *consumes* this information. The RSS feeds, however, do not provide enough detail to precisely position the listings on a map.

Consequently, Housingmaps.com *screen-scrapes* (or *crawls*) Craigslist; that is, Housingmaps.com retrieves and parses the HTML pages of Craigslist to obtain detailed information about each listing. The crawling is performed carefully so as to minimize the use of bandwidth. When you access Housingmaps.com, you are accessing not real-time data from Craigslist but rather the data that has been screen-scraped by Housingmaps.com.

■ **Note** Public APIs and RSS feeds are generally preferable to screen-scraping web sites. Screen-scraping, when poorly implemented, can overtax the data source. Always check that you are complying with the terms of service of the data source in how you use the data.

To display the real estate information on a Google map, the current version of Housingmaps.com uses the Google Maps API,[3] which is the official Google-sanctioned way of embedding Google maps in a non-Google-owned web page. (You will look in detail at the Google Maps API in various other places, particularly in Chapter 13.)

3. http://www.google.com/apis/maps/

It's interesting to go into a bit of history here to understand the emergence of the mashup phenomenon. When Housingmaps.com first showed up in April 2005, Rademacher was using Google Maps before it had any real API. He deciphered the original JavaScript of Google Maps and figured out how to incorporate Google Maps into Housingmaps.com. During the period between the release of Google Maps on February 8, 2005, and the publication of version 1 of the Google Maps API (on approximately June 29, 2005[4]), there was a period of intense "hacking" of Google Maps, described in the following way by members of the Google Maps team:[5]

> *For this and other reasons we were thrilled to see "hackers" have a go at Google Maps almost immediately after we launched the site back in early February. Literally within days, their blogs described the inner workings of our maps more accurately than our own design documents did, and soon the most amazing "hacks" started to appear: Philip Lindsay's Google Maps "stand-alone" mode, Paul Rademacher's [Housingmaps.com], and Chris Smoak's Busmonster, to mention a few.*

Comparable Mashups

Since the debut of Housingmaps.com, many other mashups—in fact, tens of thousands—have followed this pattern set of recasting data to make geographical location the organizing principle. These mashups cover an incredible range of topics and interests.[6]

Many other mashups involve extracting geocoded data (location information, often latitude and longitude) from one source to then place it on an online map (such as a Google map or Yahoo! map). I name two prominent examples here:

- Adrian Holovaty's Chicago crime map (`http://chicagocrime.org`), which is a database of crimes reported in Chicago fronted by a Google Map interface
- Weather Bonk, which is a mashup of weather data on a Google map (`http://www.weatherbonk.com/weather/about.jsp`)

Google Maps in Flickr

In the earlier days of Flickr (before August 2006), there was no built-in feature that allowed a user to show pictures on a map. The Google Maps in Flickr (GMiF) script was created to fill in that gap by letting you see a Flickr photo on a Google map. Today, even with Flickr's built-in map of geotagged photos, which uses Yahoo! Maps technology, GMiF remains a valuable mashup. GMiF allows users to use a Google map, which many prefer over Yahoo! Maps, to display their photos. Moreover, GMiF also integrates Google Earth, a feature not currently built into Flickr. GMiF provides an excellent case study of how you can extend an application such as Flickr to fit user preferences.

---

4. `http://benmetcalfe.com/blog/index.php/2005/06/29/google-make-map-api-available-finally/`
5. *Google Maps Hacks* by Rich Gibson and Erle Schuyler (O'Reilly Media, 2006)
6. See `http://googlemapsmania.blogspot.com/` for many new mashups based on Google Maps that appear every day.

What Is Being Combined?

GMiF (`http://webdev.yuan.cc/gmif/`) brings together Flickr pictures, Google Maps, and Google Earth within the Firefox browser via a Greasemonkey script. I'll break this down for you:

- Flickr (`http://flickr.com`) is a popular photo-sharing site.
- Google Maps (`http://maps.google.com/`) is an online mapping system.
- Google Earth (`http://earth.google.com/`) is a desktop "magic-carpet" interface that lets you pan and zoom around the globe.
- The Firefox web browser (`http://www.mozilla.com/firefox/`) is an open source web browser. Notable among its features is its extension/add-on architecture, which allows developers to add functionality to the browser.
- The Greasemonkey extension (`http://www.greasespot.net/`) is a Firefox extension that "allows users to install scripts that make on-the-fly changes to specific web pages. As the Greasemonkey scripts are persistent, the changes made to the web pages are executed every time the page is opened, making them effectively permanent for the user running the script."[7] Greasemonkey scripts allow you—as the user of that web site and not as the author of the web site—to make customizations, all within the web browser.

HOW TO INSTALL THE GMIF SCRIPT

To run the GMiF Greasemonkey script, you must use the Firefox web browser in conjunction with the Greasemonkey add-on and the GMiF script.

Here's how you install GMiF:

1. If you do not already have Firefox installed on your computer, go to `http://getfirefox.com`, hit the Download Firefox button, and follow the instructions to install it.
2. Now you need to install the Greasemonkey add-on for Firefox, so go to the following URL: `https://addons.mozilla.org/en-US/firefox/addon/748`.
3. Click the Install Now button. Restart the browser to activate the Greasemonkey add-on.
4. Now you need to install the GMiF Greasemonkey script, so go to the following URL: `http://webdev.yuan.cc/gmif/`.
5. Click the "Download GM user script: flickr.gmap.user.js (latest version)" link. Click Install when you are asked whether to install the script.

Why Are the Constituent Elements Being Combined? What's the Problem Being Solved?

GMiF is a Greasemonkey script that allows you as a user to display a Flickr picture on a Google map or in Google Earth at the geographic location associated with that picture. GMiF was written

7. `http://en.wikipedia.org/wiki/Greasemonkey`, accessed on January 1, 2007, as `http://en.wikipedia.org/w/index.php?title=Greasemonkey&oldid=97588087`

to support geotagging in Flickr. *Geotagging*, in the context of Flickr, is the process of associating a location with a given photo, which is typically but not necessarily the location where the photo was taken.

Until geotagging was officially integrated into Flickr with the use of Yahoo! Maps in August 2006,[8] there was no direct way to associate geocoding (location information) with any given picture. Rev Dan Catt catalyzed the mass-geotagging phenomenon by suggesting that Flickr users shoehorn the latitude and longitude information into the tags associated with a photo. Many people took up the practice. The GMiF Greasemonkey script uses that geocoding for a photo.

Let's take a look at how GMiF works. Consider one of my own photos, shown in Figure 1-2 (also available at `http://flickr.com/photos/raymondyee/18389540/`). Notice two things:

- This photo has associated geotagging information (for example, `geo:lat=37.8721`, `geo:lon=-122.257704`, and the tag `geotagged`).
- Note the presence of the rightmost GMap button above the photo. This button is the result of the GMiF script, which inserts the GMap button. In other words, if you do not have the GMiF Greasemonkey script installed, you won't see this button.

Figure 1-2. *The Flickr photo "Campanile in fog" (*`http://flickr.com/photos/raymondyee/18389540/`*) with associated geocoding embedded in the tags. (Reproduced with permission of Yahoo! Inc. ® 2007 by Yahoo! Inc. YAHOO! and the YAHOO! logo are trademarks of Yahoo! Inc.)*

8. `http://blog.flickr.com/flickrblog/2006/08/great_shot_wher.html` and `http://blog.flickr.com/flickrblog/2006/08/geotagging_one_.html`

Clicking the GMap button opens a Google map embedded in the Flickr web page, with a pin indicating the location of the picture in question (as shown in Figure 1-3). Note the red pin indicating the location of the photo. The blue pins correlate to other geotagged photos. The map also has a thumbnail of the photo in the upper-right corner.

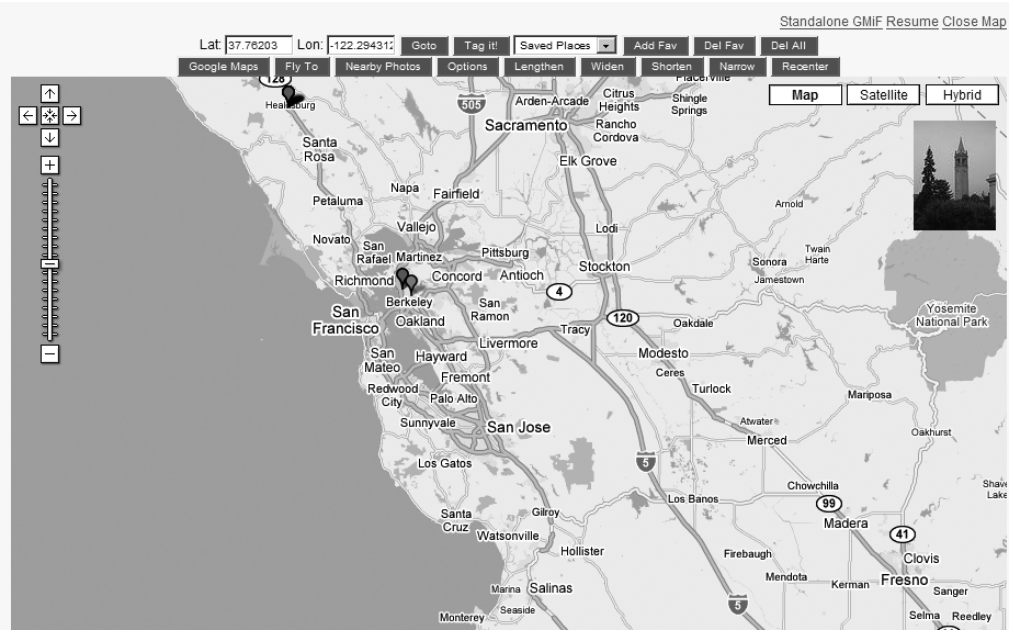

Figure 1-3. *Clicking the GMap button opens a Google map in the browser. (Reproduced with permission of Yahoo! Inc. ® 2007 by Yahoo! Inc. YAHOO! and the YAHOO! logo are trademarks of Yahoo! Inc.)*

Clicking the pin opens a callout with a picture and options of other things to do with the picture. Note how the latitude and longitude listed correspond to the information in the `geo:lat` and `geo:lon` tags, respectively (as shown in Figure 1-4).

Figure 1-4. *Clicking the red pin opens a balloon containing the photo and further geotagging functionality offered by GMiF. (Reproduced with permission of Yahoo! Inc. ® 2007 by Yahoo! Inc. YAHOO! and the YAHOO! logo are trademarks of Yahoo! Inc.)*

Among the GMiF functions is integration with Google Earth. If you hit the Fly To button, you will be presented with a file to download. If you have Google Earth installed and it is configured in the default fashion, downloading the file launches Google Earth, and you will be "flown" to the location of the Flickr photo (as shown in Figure 1-5).

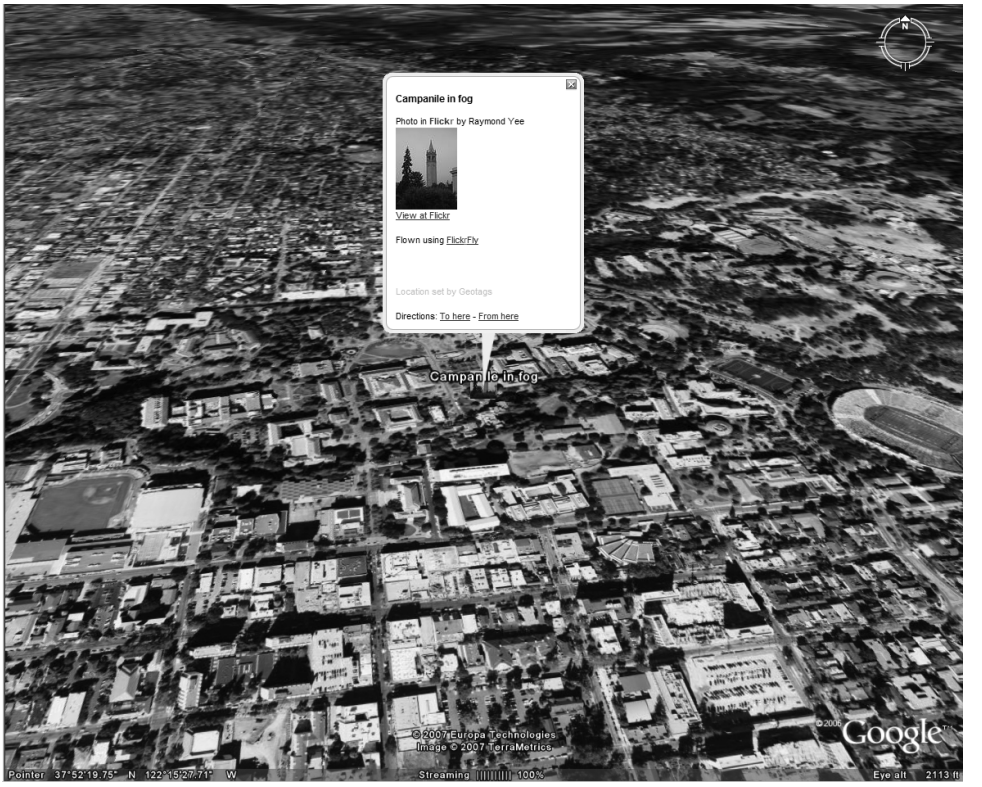

Figure 1-5. *Clicking the GMiF Fly To button launches Google Earth, which then displays the photo at the latitude and longitude associated with the photo.*

How Are These Elements Being Combined?

The GMiF Greasemonkey script rewrites the HTML of the Flickr page to insert the GMap button (this rewriting of the HTML DOM is akin to looking in the HTML source for where the Flickr buttons are and inserting HTML code for the button). Furthermore, JavaScript code is added to embed a Google map in the Flickr page, when you (as the user) click the GMap button.

Integration happens in the context of Flickr web page, loaded in the user's browser. Note how powerful this is: you don't have to go to another application to see the picture on a Google map because you get to use a slightly modified version of Flickr. These modifications do not require the intervention of Flickr itself. Hence, there is room for a lot of customization.

■ **Note** Of course, there are potential pitfalls with GMiF. GMiF, as with all Greasemonkey scripts, can cease to function if the HTML and JavaScript source of the underlying web page the script operates on changes. Also, with enough Greasemonkey scripts at play, instead of having a strong communal experience of Flickr, users have many different fragmented understandings of the interface. I think these benefits of being able to radically customize your interaction with a web site by actually changing your own version of the interface are worth dealing with these risks.

How is the integration of GMiF with Google Earth created? The downloaded file is a KML file. KML is a dialect of XML, which is the closest thing we have to a *lingua franca* for exchanging data. The KML file contains the latitude and longitude associated with the picture and a URL of the picture. KML is used to exchange geographic type data that is understood by Google Earth. In other words, GMiF takes information from one source (the URL of the picture and the latitude and longitude of the picture embedded in tags from Flickr) and translates that information into a form that is understood by the destination application, namely, KML for Google Earth. Once you translate that information, you still need to get the information to the destination; in this case, the transport happens through the formation and downloading of the KML file.

Admittedly, GMiF is a bit "hackerish," requiring the installation of the Firefox web browser (which does not come by default on Windows or Mac OS X), the Greasemonkey extension, and the GMiF script. But I bring this up here to talk about the lengths to which people are willing to go to experiment with their tools to combine technologies.

Comparable Mashups

Mappr (`http://www.mappr.com/`), "an interactive environment for exploring place based on the photos people take," is a mashup of Flickr and a Flash-based map.

FORMAL VS. INFORMAL APIS AND INTEGRATION MECHANISMS

I mentioned in previous sections how the proliferation of formal integration mechanisms in the form of APIs and XML feeds, for instance, are giving rise to many more mashups and remixed possibilities. It's important to note that you want to depend on not only these formal mechanisms but also on informal mechanisms. Hence in this book, I'll teach you how to look for both formal and informal mechanisms. The example mashups I describe here use both. I hope to convey to you how to look for those informal hooks.

LibraryLookup Bookmarklet

Let's say you find a book at an online bookstore (for example, Amazon), but instead of buying the book, you want to borrow it from your local library.

Jon Udell's LibraryLookup bookmarklet[9] makes it easy to jump from the Amazon page to the corresponding catalog entry in your local library catalog—via the simple click of a button. To accomplish the same task without LibraryLookup, you might instead manually re-enter your search in your local library catalog, which is a tedious task if you have to do it for many books.

LibraryLookup is a *bookmarklet*, which is "a small JavaScript program that can be stored as a URL within a bookmark in most popular web browsers or within hyperlinks on a web page."[10] A bookmarklet does not require the Greasemonkey extension in Firefox and works in web browsers other than Firefox. LibraryLookup is, in a manner of speaking, a mashup of online bookstores and library catalogs.

---

9. `http://weblog.infoworld.com/udell/stories/2002/12/11/librarylookup.html`
10. `http://en.wikipedia.org/wiki/Bookmarklets`, accessed as `http://en.wikipedia.org/w/index.php?title=Bookmarklet&oldid=96304211`

LibraryLookup is less flashy than previous examples; it's also not server side, like Housingmaps.com. It is client side like the GMiF script, but not in exactly the same way. But it shows another way to create browser-based integration.

Let's look at how LibraryLookup works from the user's point of view. To use the LibraryLookup bookmarklet, you need to do the following:

1. Configure a bookmarklet for the library of your choice.
2. Invoke that bookmarklet when you arrive on a web page for the book you want to look up in your library.

Configuring a LibraryLookup Bookmarklet

Go to the LibraryLookup Bookmarklet Generator at the following URL:

`http://weblog.infoworld.com/udell/stories/2002/12/11/librarylookupGenerator.html`

Now enter the base URL and library name, and select the catalog vendor corresponding to your library. Consider, for example, the Berkeley Public Library (BPL). In comparing the BPL OPAC to the examples of vendor online public access catalogs (OPACs) provided by Udell, you can determine that the BPL OPAC is an instance of an Innovative system. When you type in the base URL for the BPL OPAC (`http://www.berkeley-public.org`) and the name of the library (Berkeley Public Library), select Innovative for the vendor (as shown in Figure 1-6), and then hit Submit, you get a bookmarklet that you can then drag to your browser toolbar. The source of the bookmarklet is as follows:

```
javascript:var%20re=/([\/-]|is[bs]n=)(\d{7,9}[\dX])/i;
if(re.test(location.href)==true){var%20isbn=RegExp.$2;
void(win=window.open('http://www.berkeley-public.org'+'/search/i='+isbn,
'LibraryLookup','scrollbars=1,resizable=1,location=1,width=575,height=500'))}
```

■ **Note** If your library is not one of the vendors listed by Udell, it is not difficult to take these templates and make them work for libraries with slightly changed systems.

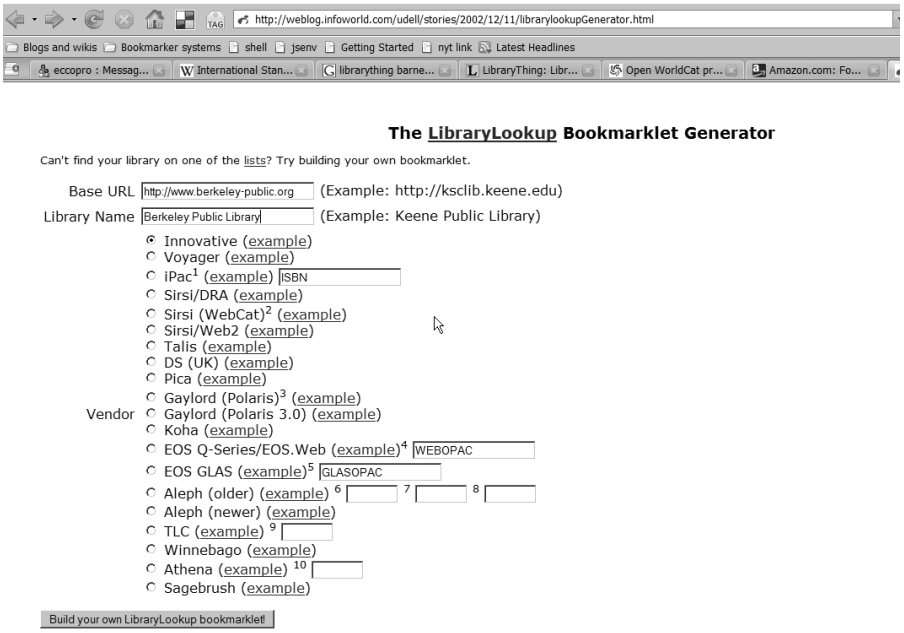

Figure 1-6. *The LibraryLookup Bookmarklet Generator with parameters for the BPL*

Invoking the LibraryLookup Bookmarklet

Let's see this bookmarklet in action. Here I use the LibraryLookup bookmarklet for the BPL, applied to the book *Foundations of Ajax*, which is published by Apress with an ISBN-10 of 1590595823. If you go to the corresponding Amazon page (`http://www.amazon.com/Foundations-Ajax-Foundation-Ryan-Asleson/dp/1590595823/`) and hit the BPL Library-Lookup bookmarklet, you would see a window pop up showing the book in the BPL (see Figure 1-7).

Figure 1-7. *Invoking the LibraryLookup bookmarklet to look up* Foundations of Ajax *at the BPL. (Software copyright Innovative Interfaces, Inc. All rights reserved.)*

How Does This Mashup Work?

The LibraryLookup bookmarklet looks for an ISBN (or ISSN) in the URL of the book-related site to identify the book you want to find. The bookmarklet does the following:

1. It extracts an ISBN from the URL of the library catalog.
2. It repackages the ISBN in a new URL to look up that book in your library catalog.

How Can This Mashup Be Extended?

This bookmarklet has some limitations. If you want to query multiple libraries in your area, you might find it tedious to create the bookmarklet for each of these libraries. One approach is to modify the bookmarklet to send ISBNs to the OCLC Open WorldCat catalog. Here's the corresponding bookmarklet:

```
javascript:var%20re=/([\/-]|is[bs]n=)(\d{7,9}[\dX])/i;
if(re.test(location.href)==true){var%20isbn=RegExp.$2;
void(win=window.open('http://worldcatlibraries.org/wcpa'+'/isbn/'+isbn,
'LibraryLookup','scrollbars=1,resizable=1,width=575,height=500'))}
```

There is a deeper limitation of the LibraryLookup bookmarklet, which you can see through the following example. If you use the BPL bookmarklet to see whether Czesław Miłosz's *New and Collected Poems: 1931–2001* is in the library by first looking it up at Amazon and finding a paperback version at `http://www.amazon.com/exec/obidos/ASIN/0060514485` and then invoking the bookmarklet to arrive at `http://library.berkeley-public.org/search/i=0060514485`, you might be surprised to not turn up the book in question, especially since the Nobel Prize winning poet spent the last 40 years of his life in Berkeley. It turns out that there are indeed copies of Miłosz's book in the BPL, but they are a different edition with a different ISBN (006019667X). See the following URL:

```
http://library.berkeley-public.org/search/i=006019667X
```

Different editions of a work have different ISBNs. Furthermore, it's not obvious how to derive the ISBN of related editions.

In recognizing that the LibraryLookup bookmarklet, by using an ISBN to uniquely identify a work, is not able to recognize various editions of a book, Udell has taken a number of different approaches to overcome this limitation, all of which use the OCLC xISBN service, a web service that returns a list of ISBNs that are associated with a submitted ISBN:[11]

- The first is a Greasemonkey script that works on an Amazon page for a book. The script first checks whether Udell's local library has a book with the same ISBN as the Amazon book in question. If not, the script then queries the local library for any of the ISBNs associated with the book, a listed generated by the xISBN service.[12]
- The second extension is a port of the Greasemonkey script (which is tied to Firefox) to something that works in Internet Explorer.[13]

Udell has also worked on another type of mashup between Amazon and a local library: a service that checks your Amazon wish list in order to receive notifications about availability in a Keene, NH library (Udell's local libraries).[14] This service awaits generalizations for multiple OPACs and multiple libraries.

---

11. `http://www.worldcat.org/affiliate/webservices/xisbn/app.jsp`
12. `http://weblog.infoworld.com/udell/2006/01/30.html`
13. `http://blog.jonudell.net/2007/04/23/greasemonkeying-with-ie/`
14. `http://elmcity.info/services`

Comparable Mashups

BookBurro, in the form of either a Firefox extension or a Greasemonkey script, displays the price of a corresponding book as a pop-up window.[15]

LibraryThing is "an online service to help people catalog their books easily."[16] It is much more than a typical mashup but has elements that are mashup-like—including the thingISBN API[17] described in the following way:

> *Today I'm releasing thingISBN, LibraryThing's "answer" to xISBN. Under the hood, xISBN is a test of FRBR, a highly developed, well-thought-out way for librarians to model bibliographic relationships. By contrast, thingISBN is based on LibraryThing's "everyone a librarian" idea of bibliographic modeling. Users "combine" works as they see fit. If they make a mistake, other users can "separate" them. It's a less nuanced and more chaotic way of doing things but can yield some useful results.*

William Denton has been experimenting with both xISBN and thingISBN, showing that it might be better to use both services rather than just one.[18]

Tracking Other Mashups

Of course, many mashups exist other than the ones I have highlighted in this chapter. You can always learn more by studying other examples of mashups.

In studying mashups, you will find one web site that is a particularly useful resource: `http://programmableweb.com`. This site is created and managed by John Musser.

I will be referring to Programmableweb.com throughout the book but want to highlight some specific parts here that will help you keep up with mashups:

- The Programmableweb.com blog is a narrative of the latest developments in the world of mashups and APIs.[19]
- The Mashup Dashboard provides an overview of the mashups in the Programmableweb.com database, which as of August 2007, covers more than 2,220 mashups.[20]

Summary

In this chapter, you studied three major examples of mashups: Housingmaps.com, Google Maps in Flickr, and the LibraryLookup bookmarklet. I chose these examples to illustrate some commonalities and differences you will find among mashups. By posing a number of analytic

---

15. `http://bookburro.org/`
16. `http://www.librarything.com/about.php`
17. `http://www.librarything.com/thingology/2006/06/introducing-thingisbn_14.php`
18. `http://www.frbr.org/categories/librarything/`
19. `http://blog.programmableweb.com/`
20. `http://www.programmableweb.com/mashups`

questions (What is being combined? Why are these elements being combined? Where is the remixing or recombination happening? How are they being combined, in terms of the interface and behind the scenes in the technical machinery? How can the mashup be extended?), you saw a repeated pattern:

1. Data is extracted from a source web site.
2. This data is translated into a form meaningful to the destination web site.
3. The repackaged data is sent to the destination site.

There are important differences among the various mashups, specifically in where the integration happens and what is being integrated. For instance, Housingmaps.com is a server-side application, whereas the mashing up of GMiF and LibraryLookup occurs within the browser.

Now that you have a sense of how mashups are constructed and what they are used for, you'll turn now to a study of the individual services and sources of data that can be recombined.