CHAPTER 9

Dissecting mashups and remixes

Previous chapters have laid out how to work with individual applications and individual APIs and widgets. In the next two chapters, we turn to studying how to combine two or several services together. This chapter dissects several specific examples of mashups in detail and also draws out general mashup design patterns to look for when looking at a range of mashups. The examples selected for analysis illustrate some of main mashup patterns, recombine data in a novel and more or less seamless way, but at the same time have underlying mechanisms that are reasonably easy to understand and decipher. This chapter teaches ways to analyze mashups to think about how it must be put together. Assuming the background knowledge of Parts I & II, what else do you need to know?

WHAT ARE DESIGN PATTERNS?

In computer science, "design patterns" are commonly used and reusable ways of doing things. According to the Wikipedia: "A **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations."[1]

End Sidebar

Techniques to use to discern design patterns in Mashups

* One can get some practice at recognizing the constituent APIs by looking at programmableweb.com which lists mashups and the APIs they use. It's helpful to have a working knowledge of the individual data sources and APIs that can be mixed so when you see them juxtaposed with other pieces, the individual service is still recognizable to the person analyzing the mashup. (Note that a handful of services get used over and over again so it won't take looking at many mashups before one sees certain services over and over again -- at least at this relative early stage.)
* Deciphering the URL language (how URLs are structured in an application), akin to what we did in Chapter 2.
* Looking for and then taking apart Firefox extensions, greasemonkey scripts.

[1] http://en.wikipedia.org/wiki/Design_pattern_(computer_science) accessed

* Analyze the templates used by service composition frameworks such as Proto Software,[2] openkapow,[3] and QEDWiki.[4]

In discerning design patterns, one might ask:

* what genres are coming together? (e.g., a popular one: map + data)
* are there straightforward connections of various APIs or is there some transformation of data happening?
* Is a database being built and data accumulated?
* Is there caching of data?
* is the remixed info being offered up again to permit further recombination?

Example: GMiF Greasemonkey script

We return to the Google Maps in Flickr (GMiF) Greasemonkey script (`http://webdev.yuan.cc/gmif/`) (which mashes up Flickr, Google maps, Google Earth in the Firefox browser via Greasemonkey). In Chapter 1, we did a high-level analysis of GMiF but not a detailed technical analysis, which is the purpose of this section.

Technical Prerequisites for the study

1. Make sure you have the Greasmonkey extension installed in Firefox. [5]

2. Make sure you have the the GMiF script installed. (The latest version of as writing is 4.0). Click on `http://webdev.yuan.cc/gmif/flickr.gmap.user.js` and you will be prompted to install the script.

3. To make sure that GMiF is working, go to `http://flickr.com/photos/raymondyee/47854736/` Do you see a Gmap icon above the picture? If so, hit it.

Recap of GMiF from the end-user point of view

To recap what we learned from Chapter 1 as an end-user of GMiF:

* The changes in Flickr interface that the GMiF script causes: the extra button (labelled GMap) in the Flickr button bar for a photo
* When you click on the GMap button, an embedded Google Map shows up in the Flickr page.

[2] http://www.protosw.com/

[3] http://openkapow.com/

[4] http://services.alphaworks.ibm.com/qedwiki/

[5] https://addons.mozilla.org/en-US/firefox/addon/748

* If the GMiF script is able to calculate the lat titude and longitude of the photo, a marker with the photo is displayed on the map. Where is the balloon located? Notice that the latitude and longitude correspond to the locations in the `geo:lon` and `geo:lat` tags or the Exif header of the photo.
* If you then hit the "Fly-to" button, you are presented with a file named flickrfly 47854736.kml , a Google Earth KML file, which you can then have snt to Google Earth. The result is the photo showing up in Google Earth attached to the place that corresponds to the latitude and longitude of the photo.

Figuring out how GMiF 4.0 works: Overall Approach

We can study the code behind GMiF, most of which is visible to users who want to study the code. The vast majority of the functionality is contained in the following files:

* http://webdev.yuan.cc/gmif/flickr.gmap.user.js (the Greasemonkey script)
* http://webdev.yuan.cc/gmif/gmif_v4.0.js
* http://webdev.yuan.cc/gmif/maps2.js

We can figure out that these three files are the JavaScript files that come from http://webdev.yuan.cc through a study of http://webdev.yuan.cc/gmif/flickr.gmap.user.js, which is the base file for the GMiF Greasemonkey script and tracing what other files are loaded -- either by reading the code or by running a debugging inteface such as the Firebug extension's network monintoring feature.[6]

The goal in this section is not decipher or document it comprehensively but to analyze a specific slice of functionality, with the goal of elucidating some design patterns at work. The slice under examination is the functionality I listed above. GMiF provides functionality other than what I detail. For instance: "You can save geotags, add description or comment with only one click. Furthermore, GMiF is a geotagged photo viewer. You can browse all your geotagged photos and show nearby photos geotagged by others."[7]

We can boil down an analysis of this aspect of GMiF into a series of questions:

1. How is the GMiF button created and inserted into the Flickr interface , alongwith the proper event handlers?
2. How is the latitude and longituede of the photo currently read?
3. How can we change the code to integrate contemporary Flickr geotagging functionality?

[6] http://www.getfirebug.com/net.html

[7] http://webdev.yuan.cc/gmif/

How is the GMiF button created and inserted into the Flickr interface, alongwith the proper event handlers?

When you read `http://webdev.yuan.cc/gmif/flickr.gmap.user.js`, you will see that much of the functionality of GMiF is located in `http://webdev.yuan.cc/gmif/gmif_v4.0.js` -- which is loaded into the Flickr page as a script:

```
var js = document.createElement("script");
js.language = "javascript";
js.src = "http://webdev.yuan.cc/gmif/gmif_v4.0.js";
document.body.appendChild(js);
```

When we turn to look at looking at http://webdev.yuan.cc/gmif/gmif_v4.0.js, we see two variables: `GMiF.icon_mout` and `GMiF.icon_mover` -- they hold references to the icon and the mouseover version of the icon.

```
GMiF.icon_mout =
'data:image/gif;base64,R0lGODlhLwAYAKIAAMPDw76+vre3t9/f352dnb+/v39/f////ywAAAAALwAYA
AADUHi63P4wykmrvTjrzbv/mSGKYDiOpYUqayqRjOFS8HzVJOvC+dFjuJXMx2rIhkRgzTdsFpm7BfJ2wj2PO
qRW2Uo6oceu7TUGls/otHrNbmsSADs=';
```

Note that instead of an external URL that you might be used to seeing to represent a picture (e.g., http://photos16.flickr.com/22099286_cbf504ef03_o.gif which is an older version of the GMiF code), CK Yuan, makes use the of the data: URI scheme "defined in IETF standard RFC 2397, is an URI scheme that allows inclusion of small data items inline, as if they were being referenced to as an external resource.."[8] This protocol is supported in Firefox.

CONFIRMATION OF BASE64 ENCODING IN THE URL

We can write a bit of Python code to confirm that the encoded image is actually the GMap button.

```
f = file(r'd:\test.gif',"wb")
import base64
d =
base64.b64decode(r'R0lGODlhLwAYAKIAAMPDw76+vre3t9/f352dnb+/v39/f////ywAAAAALwAYA
AADUHi63P4wykmrvTjrzbv/mSGKYDiOpYUqayqRjOFS8HzVJOvC+dFjuJXMx2rIhkRgzTdsFpm7BfJ2w
```

```
GMiF.init()
```

So let's look at GMiF.init() to find out where the icon gets created, event handlers attached to it, and then gets appended to the Flickr interface. In studying the code, you will see that GMiF.init() is a long function, (which runs from line 52 to 933 of gmif_v4.0.js) -- so let's not get lost into a lot of other stuff. We can do a search for icon_mout to see what references there are. Line 133 has a reference -- let's look in that region of code:

```
var gmap;
if( _gi('gmap')) gmap = _gi('gmap');
else gmap = _ce("a");
gmap.style.margin = '0px';
var gimg = _ce("img");
//      var gname = _ce("a");

gimg.src = GMiF.icon_mout;
gimg.onmouseover = function() { gimg.src = GMiF.icon_mover; }
gimg.onmouseout = function() { gimg.src = GMiF.icon_mout; }
gmap.appendChild(gimg);
gmap.href = '#gmap';
```

Noting the functionality of the helper functions _gi and _ce defined in lines 8 and 9:

```
function _gi(e) { return document.getElementById(e); }
function _ce(e) { return document.createElement(e); }
```

we see that this section of code creates a gmap object (an 'a' document node) and gimg is an image -- the icon -- with mouseover functions that alternate between the two versions of the icon.

Where does this icon actually get inserted into the Flickr interface? What's the event handler attached for clicking on the Gmap icon? Go to line 928 to 932:

```
gmap.onclick = showmap;
GMiF.showmap = showmap;
if( GMiF.standalone ) {
    GMiF.loopHandle = setInterval("GMiF.showmap()", 50) ;
} else _gi('button_bar').appendChild(gmap);
```

The code looks for the div element with id button_bar and tacks the GMap icon as a last icon. Note that CK Yuan made a succesfull attempt to integrate the icon as seamlessly into the existing user interface of Flickr (a good integration strategy).

Using Firebug Extension and the JavaScript Shell to Analyze GMiF

We'll come back later to `showmap`, a complicated function, to study how the map is built. Let's see how we can use Firebug and the Javascript shell to confirm my analysis thus far of GMiF.

1. Go to `http://flickr.com/photos/raymondyee/47854736/` with the GMiF Greasemonkey script installed and enabled.

2. Turn on the Firebug window. In the HTML tab, you'll see that after the page is fully loaded, that two script elements with the src attribute values of `http://webdev.yuan.cc/gmif/gmif_v4.0.js` and `http://webdev.yuan.cc/gmif/maps2.js` are located as some of the last child elements of the body element.. That's the work of the GMiF Greasemonkey script.

3. You should also see the GMap icon. Use the Inspect function of Firebug and click on the GMap icon. Right-click on the corresponding source

```
<img
src="data:image/gif;base64,R0lGODlhLwAYAKIAAMPDw76+vre3t9/f352dnb+/v39/f////ywAAAAAL
wAYAAADUHi63P4wykmrvTjrzbv/mSGKYDiOpYUqayqRjOFS8HzVJOvC+dFjuJXMx2rIhkRgzTdsFpm7BfJ2w
j2POqRW2Uo6oceu7TUGls/otHrNbmsSADs="/>
```

and select the "Copy XPath" menu item to copy

```
/html/body/div[3]/table/tbody/tr/td/div/a/img
```

to the clipboard.

4. You can verify that the XPath expression corresponds to the icon. Invoke the JavaScript Shell to reference the icon via the XPath with the following code:

```
r = document.evaluate("/html/body/div[3]/table/tbody/tr/td/div/a/img",
document,null,XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,null)
r.snapshotItem(0).onmouseover
```

results in

```
function () { gimg.src = GMiF.icon_mover; }
```

You can see that we have the right link because of the corresponding `onmouseover` handler.

If we look at the parent element of the icon (the a element), we can use Firebug to get the corresponding XPath

```
/html/body/div[3]/table/tbody/tr/td/div/a
```

We can invoke the click handler and arrive at the result as literally clicked the Gmap icon:

```
a = document.evaluate("/html/body/div[3]/table/tbody/tr/td/div/a",
document,null,XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,null)
a.snapshotItem(0).onclick()
```

This series of interactions with GMiF using Firebug and the JavaScript Shell confirms our understanding of how the GMap icon is constructed and inserted into the Flickr interface -- and how clicking it embeds a Google Map.

How is the latitude and longituede of the photo currently read?

Now let's figure out how the actual map is constructed, specifically how the lat/lon info is extracted. The `showmap` function runs from lines 450 to 927. While this section does not provide a definitive analysis of how the latitude and longitude of the photo is read, I will point out a number of clues into the functionality:

* Flickr creates a global variable `global_pictures` that holds the picture or pictures of the current page.

GMiF makes use of `unsafewindow`. As Mark Pilgrim writes:" It's called `unsafeWindow` for a reason: its properties and methods could be redefined by the page to do virtually anything. You should never call methods on `unsafeWindow` unless you completely trust the remote page not to mess with you. You should only ever use it as a parameter to call functions defined by the original page, or to watch `window` properties, as shown in the next section."[9]

---

* In GMiF.init in gmif_v4.0.js lines 73-81:

```
var tmp =new Array()
var rawAtags = GMiF.photo.tags_rawA.concat(tmp);
while( rawAtags && rawAtags.length > 0 ) {
    tag = rawAtags.pop();
    splits = tag.split('=')
    if( splits[0] == 'geo:lat' ) GMiF.lat = GMiF.convCoords(splits[1]);
    if( splits[0] == 'geo:lon' || splits[0] == 'geo:long' ) GMiF.lon =
GMiF.convCoords(splits[1]);
    if( splits[0] == 'geo:datum' && ( splits[1] == 'TOKYO' || splits[1] == 'Tokyo'
|| splits[1] == 'tokyo' ) ) GMiF.datum = 'TOKYO';
}
```

we see the use of the raw tags to search for the latitude and longitude. If you go to http://flickr.com/photos/raymondyee/47854736/ after GMiF is loaded and use the JavasScript Shell, you can verify that GMiF.lat and GMiF.lon do actually hold the latitude and longitude (derived from the geo:lat and geo:lon tags). If, instead, you go to `http://flickr.com/photos/raymondyee/429267823/` (in which the lat/lon is not in the tags but in the Flickr geo database), GMiF.lat and GMiF.lon will be (0,-180) respectively.

* If the tags do not contain the lat/lon info, GMiF looks into the Exif headers. See lines 812-817:

```
if( (GMiF.lat=='' || GMiF.lon=='') && ( GMiF.exif.lat != undefined && GMiF.exif.lon
!= undefined )) {
    GMiF.lat = GMiF.exif.lat;
    GMiF.lon = GMiF.exif.lon;
```

[9] http://www.oreillynet.com/pub/a/network/2005/11/01/avoid-common-greasemonkey-pitfalls.html?page=last

```
    GMiF.zoom = 8;
    marked = true;
}
```

* If you have a photo that does not have the geotagged tag, a Google Map is embedded upon your clicking the GMap icon -- but no marker is created.

How can we change the code to integrate contemporary Flickr geotagging functionality?

Version 4.0 of GMiF does not read geocoding information that is located in either the tags or the exif headers. Hence , GMiF 4.0 does not work for the new-style in-house Flickr geotagging., which depends on specialized storage of geolocation information./

How to fix this problem? A number of ways come to mind:

1. Modify GMiF to call the Flickr `flickr.photos.geo.getLocation` method (http://www.flickr.com/services/api/flickr.photos.geo.getLocation.html).

2. Write a script to explicitly write out into triple tags the geolocation so that this information is available to GMiF 4.0

3. Modify GMiF to use the lat/lon info embedded in a meta tag in the head element of the Flickr page. Here's a bit of JavaScript and XPath to extract that info:

```
m = document.evaluate("//meta[@name='geo.position']",document, null,
XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,null);
m.snapshotItem(0).content;)
```

Method #3 is probably the easiest way, although method #2 means not having to change the GMiF script.

Archiving/Changing/Supplementing the functionality of v 4.0 of GMiF

Let's archive a few pieces of code from CK Yuan's site first to preserve the code and then to use as basis to rewrite What are the pieces we need to archive?

* the Greasemonkey script itself (`http://webdev.yuan.cc/gmif/flickr.gmap.user.js`) and the other scripts or graphics upon which the main GMiF script depends
* http://webdev.yuan.cc/gmif/gmif_v4.0.js
* http://webdev.yuan.cc/gmif/maps2.js

http://webdev.yuan.cc/gmif/flickr.gmap.user.js loads http://webdev.yuan.cc/gmif/gmif_v4.0.js which in turn loads http://webdev.yuan.cc/gmif/maps2.js Let's see whether we have surmised the dependencies on webdev.yuan.cc properly by copying the three files over to mashupguide.net, changing the references and then testing whether the script works.

http://webdev.yuan.cc/gmif/flickr.gmap.user.js ->
    http://examples.mashupguide.net/ch09/gmif/flickr.gmap.mod.user.js

http://webdev.yuan.cc/gmif/gmif_v4.0.js ->
http://examples.mashupguide.net/ch09/gmif/gmif_v4.0.js

http://webdev.yuan.cc/gmif/maps2.js ->
http://examples.mashupguide.net/ch09/gmif/maps2.js

You then modify the files to point to the ones on mashupguide.net.

```
flickr.gmap.mod.user.js


2c2
< // @name          Google Maps in Flickr
---
> // @name          MODIFIED Google Maps in Flickr
4c4
< // @description   GMiF - Display Google Maps in flickr
---
> // @description   GMiF - MODIFIED Display Google Maps in flickr
124c124,125
<      js.src = "http://webdev.yuan.cc/gmif/gmif_v4.0.js";
---
>      //js.src = "http://webdev.yuan.cc/gmif/gmif_v4.0.js";
>      js.src = "http://examples.mashupguide.net/ch09/gmif/gmif_v4.0.js";
```

```
$ diff original/gmif_v4.0.js gmif_v4.0.js
118c118,119
<        js.src = "http://webdev.yuan.cc/gmif/maps2.js";
---
>        //js.src = "http://webdev.yuan.cc/gmif/maps2.js";
>        js.src = "http://examples.mashupguide.net/ch09/gmif/maps2.js";
```

Doing so still leaves some depdencies in gmif_v4.0.js:

* GMiF.LfVrUrl = 'http://webdev.yuan.cc/lfvr' + GMiF.photo.ownersUrl;
* js.src = 'http://webdev.yuan.cc/gmif/xml_proxy.php?' + req.url;

I don't know the significance of not ridding of this dependency is yet.

Example: HousingMaps.com

Let's look at housingmaps.com as a second example to analyze. Here are some noteworthy points:

* Housingmaps is a server-side application, and no source has been published. Hence, we cannot consult any source code to verify what we surmise might be happening in the backend.

* The functionality of the system that is being combined: craiglist + Google Maps. Some questions to guide an analysis: How much of the Craigslist information is actually captured? How much latency is there between data on Craigslist and when it shows up on housingmaps.com?
* It's helpful to work with a specific subset of the data to see whether we can get any patterns. Consider rentals in the SF East Bay that go from $1500-$2000. You can get a map at

```
http://www.housingmaps.com/?c=sf_eby&t=apa&p=1500_2000
```

When we look at the table, at the time of writing, I see only listings for April 22, 2007 and April 21, 2007. Is there a filter for the last two days? How would we gather the corresponding information from craigslist?

```
http://sfbay.craigslist.org/search/apa/eby?query=&minAsk=1500&maxAsk=2000&bedrooms=&
neighborhood=
```

lists as of 4/22/07 7:11p 1119 listings -- a lot more than what housingmaps.com shows. Let's try to look at only the last two days. For the last two days, I see 205 listings. How many listings are in housingmaps.com? There are at least two ways to figure the answer out.

* One is to count the number of rows in the data display:

```
document.getElementById("listingspanel").getElementsByTagName('tr').length
```

returned 99 for me.

* The other way is to use Firebug to look at the names of JavaScript objects and method names. However, they seem to beobfuscated. It's not easy to see the internal data structures. However, if you watch xmlhttpresponse activities (XHR) with Firebug, you can see the loading of the following URL:

```
http://www.housingmaps.com/listings/c_apa_sf_eby_1500_2000.txt
```

You can count the number of entries in that list. I also got 99.

What have we learned so far? That there is probably some server-side script that generates listings periodically and filters down by date the listings -- you'renot going to see all the current Craigslists postings on the map.

Example: Chicagocrime.org

Chicagocrime.org is another famous take-data-from-one-source-and-place-the-data-on-a-map site. The About page is worth studying to get a sense of the issues involved in creating map-based mashups.[10]

* chicagocrimes.org accumulates data over time.
* "Finally, because Citizen ICAM removes crime data after 90 days and chicagocrime.org doesn't, there is a chance that any crime report on chicagocrime.org older than 90 days is no longer accurate. "

[10] http://www.chicagocrime.org/about/

* explicit mention of the use of screen-scraping.

Data Mining Programmableweb.com for Patterns

So far, we have done close analyses of specific mashups. We turn now to a macro-analysis of mashups. That is, we would look at the broadest range of mashups to look for design patterns that cross many examples.

In this section, I propose a method for doing such an analysis using ProgrammableWeb, probably the single best compilation available of mashups and corresponding APIs in use on the public web. There are some patterns that are immediately obvious from a study of the site; I say immediately obvious because John Musser, its creator has surfaced these elements in the interface. Let me point out some of the data about mashups:

* You can get an overview of the mashup world, newly registered ones, what's popular at the Mashup Dashboard.[11]
* "mapping" is the most popular tag associated with mashups, followed by "photo"
* The Web 2.0 Mashup Matrix displays mashups by their use of every combination of 2 APIs in the ProgrammableWeb database. [12]

In addition, to what is obvious in the data, I would like to pose more questions that should be dertivable from what is in ProgrammableWeb.com:

* How many APIs are used by the mashups? That is, what's the distribution -- how many use 1, 2, 3, etc. APIs.
* What's the most common pair of APIs being used? Most common threesome?
* Is there any correlation between the popularity of an API and the popularity of mashups that use that API?
* Are there broader correlations among usage patterns of APIs if we cluster them by categories? Are mashups likely to use more than one API in the same category or across categories?

As of the writing of this book, there is no formal API to programmableweb.com -- so answering these and aliied questions require some screen-scraping. We'll see in the next section

Mashup design patterns in service composition frameworks

This section analyze the templates used by service composition frameworks such as Proto Software,[13] openkapow,[14] dapper[15] and Coghead[16] and QEDWiki[17] in search for general

[11] http://www.programmableweb.com/mashups

[12] http://www.programmableweb.com/matrix

[13] http://www.protosw.com/

[14] http://openkapow.com/

patterns at work in mashups. Service composition frameworks are sytems that make it easier to recombine data and services, often through graphical user interfaces and semi-automated tools to parse data.I'll be examing service composition frameworks in greater detail in Chapter 11

Dapper

You can use Dapper to screen-scrape websites to create an ad hoc API for the website. It is not hard to extract something like the total number of mashups at programmableweb.com. You are asked to surf to a page and add sample similar pages. Dapper then looks for common patterns and asks you to highlight fields that you want extracted. This data is then presented in a variety of formats of your choice (including XML, JSON, etc.)

```
http://www.dapper.net/dapp-howto-use.php?dappName=TotalnumberofmashupsatProgrammableWebcom
```

On the other hand, I found it difficult to extract tables, such the most popular APIs by mashup count.

Another dapper to try to build is one that converts the SF Gate restaurant listings to a map based on the parameters you choose for location and food type: http://sfgate.com/cgi-bin/listings/restaurants/basic?cuisine=&loc=51&term=&Go.x=28&Go.y=13&Go=Search&Submit=Search

Yahoo Pipes

The central pattern at work is to exploit as much as possible the huge number of RSS/Atom feeds out there, all of which have structured data. A GUI environment is given to support recomposing these feeds. There are filters.

Proto

Proto is a desktop mashup creator. You can get an overview at http://www.protosw.com/products/intro-movie . It runs in Win32 Here are some features:

* There is a GUI environment with components (inputs and outputs). The GUI reminds me of Yahoo Pipes.

* VBA interface (COM scripting?)
* It's easy to drop in widgets like Yahoo maps Flash module combined with a spreadsheet table.

[15] http://www.dapper.net/

[16] http://www.coghead.com/

[17] http://services.alphaworks.ibm.com/qedwiki/

* There is a webservices component, though it expects WSDL.
* Lots of stuff geared to the Win32 desktop business environment: Excel reader and writer, ODBC connectors

The fact that the demo is putting data from Outlook on a map is an indication of the popularity of map-based mashups.

QEDWiki

QEDWiki is "a browser-based assembly canvas used to create simple mash-ups." It looks pretty sophisticated but after a half of hour of work on it, I still couldn't get anything to work.

Common Themes Among these systems

* GUI to support screen-scraping of pages.
* Lots of widgets, some scripting

Conclusions

In this chapter, we have looked at several mashups in detail (namely, GMiF and housingmaps.com) as well as mashups in the aggregate. We have looked for design patterns in emerging service composition frameworks.

Below are notes that may be worked into the text

Miscellaneous Observations

* I'd like to learn more UML and see how to apply UML to describe patterns in the mashups.
* I'd like to be more systematic in applying analytic questions: Where is the remixing happening? What is the container? Client vs server-side mashups
* Versioning of scripts is important to allow for enchancements but also changes in the underlying dependencies of a mashup. There's also a tradeoff of having explicit versions -- there should be an easy-to-use mechanism for updating mashups. (I'm thinking of GMiF as an example)
* In analyzing GMiF, it took me a while to understand that global_pictures is a Flickr variable -- not obvious. The use of private flickr variables obviously makes these types of scripts fragile. Somehow, I thought that GMiF would use the API (with its public methods) more than using private variables.
* It's a good design pattern to blend in with the existing interface.

* the combination of reading code and using tools like Firebug and the JavaScript Shell is a good way to do reverse engineering and unerstanding the code.
* map vs non-map mashups is probably not a bad first cut at a classification scheme.
* Why maps? Maps are a dominant metaphor -- easy to understand, every piece of data has a literal physical place and therefor a place in the metaphor
* When it comes to maps, we can ask the question of what type of data are in play: dots, lines, regions, and associated items (written descriptions, pictures). This brings up the issues of data sources and how they get converted. A close study of chicagocrimes.org would help here.
* some diagrams would help A -> B vs A+ B+ C+ D -> E.