Mashups: The new breed of Web app

An introduction to mashups

Skill Level: Introductory

Duane Merrill (duane@duanemerrill.com)
Writer
Freelance

08 Aug 2006

Updated 24 Jul 2009

Mashups are an exciting genre of interactive Web applications that draw upon content retrieved from external data sources to create entirely new and innovative services. They are a hallmark of the second generation of Web applications informally known as Web 2.0. This introductory article explores what it means to be a mashup, the different classes of popular mashups constructed today, and the enabling technologies that mashup developers leverage to create their applications. Additionally, you'll see many of the emerging technical and social challenges that mashup developers face.

Introduction

A new breed of Web-based data integration applications is sprouting up all across the Internet. Colloquially termed *mashups*, their popularity stems from the emphasis on interactive user participation and the monster-of-Frankenstein-like manner in which they aggregate and stitch together third-party data. The sprouting metaphor is a reasonable one; a mashup Web site is characterized by the way in which it spreads roots across the Web, drawing upon content and functionality retrieved from data sources that lay outside of its organizational boundaries.

This vague data-integration definition of a mashup certainly isn't a rigorous one. A good insight as to what makes a mashup is to look at the etymology of the term: it

was borrowed from the pop music scene, where a mashup is a new song that is mixed from the vocal and instrumental tracks from two different source songs (usually belonging to different genres). Like these "bastard pop" songs, a mashup is an unusual or innovative composition of content (often from unrelated data sources), made for human (rather than computerized) consumption.

So, what might a mashup look like? The ChicagoCrime.org Web site is a great intuitive example of what's called a *mapping mashup*. One of the first mashups to gain widespread popularity in the press, the Web site mashes crime data from the Chicago Police Department's online database with cartography from Google Maps. Users can interact with the mashup site, such as instructing it to graphically display a map containing pushpins that reveal the details of all recent burglary crimes in South Chicago. The concept and the presentation are simple, and the composition of crime and map data is visually powerful.

In Mashup genres, you'll survey the popular genres of mashups, including mapping mashups. Related technologies overviews the technology landscape that relates to the construction and operation of mashups. Technical challenges and Social challenges present the eminent technical and social challenges, respectively, affecting mashups.

Mashup genres

In this section, I give a brief survey of the prominent mashup genres.

Mapping mashups

In this age of information technology, humans are collecting a prodigious amount of data about things and activities, both of which are wont to be annotated with locations. All of these diverse data sets that contain location data are just screaming to be presented graphically using maps. One of the big catalysts for the advent of mashups was Google's introduction of its Google Maps API. This opened the floodgates, allowing Web developers (plus hobbyists, tinkerers, and others) to mash all sorts of data (everything from nuclear disasters to Boston's CowParade cows) onto maps. Not to be left out, APIs from Microsoft (Virtual Earth), Yahoo (Yahoo Maps), and AOL (MapQuest) shortly followed.

Video and photo mashups

The emergence of photo hosting and social networking sites like Flickr with APIs that expose photo sharing has led to a variety of interesting mashups. Because these content providers have metadata associated with the images they host (such as who took the picture, what it is a picture of, where and when it was taken, and more), mashup designers can mash photos with other information that can be associated with the metadata. For example, a mashup might analyze song or poetry lyrics and

create a mosaic or collage of relevant photos, or display social networking graphs based upon common photo metadata (subject, timestamp, and other metadata.). Yet another example might take as input a Web site (such as a news site like CNN) and render the text in photos by matching tagged photos to words from the news.

Search and Shopping mashups

Search and shopping mashups have existed long before the term mashup was coined. Before the days of Web APIs, comparative shopping tools such as BizRate, PriceGrabber, MySimon, and Google's Froogle used combinations of business-to-business (b2b) technologies or screen-scraping to aggregate comparative price data. To facilitate mashups and other interesting Web applications, consumer marketplaces such as eBay and Amazon have released APIs for programmatically accessing their content.

News mashups

News sources (such as the New York Times, the BBC, or Reuters) have used syndication technologies like RSS and Atom (described in the next section) since 2002 to disseminate news feeds related to various topics. Syndication feed mashups can aggregate a user's feeds and present them over the Web, creating a personalized newspaper that caters to the reader's particular interests. An example is Diggdot.us, which combines feeds from the techie-oriented news sources Digg.com, Slashdot.org, and Del.icio.us.

Related technologies

This section gives an overview of the technologies that are facilitating the development of mashups. For further information about any of these technologies, consult Resources at the end of this article.

The architecture

A mashup application is architecturally comprised of three different participants that are logically and physically disjoint (they are likely separated by both network and organizational boundaries): API/content providers, the mashup site, and the client's Web browser.

- The API/content providers. These are the (sometimes unwitting) providers of the content being mashed. In the ChicagoCrime.org mashup example, the providers are Google and the Chicago Police Department. To facilitate data retrieval, providers often expose their content through Web-protocols such as REST, Web Services, and RSS/Atom (described below). However, many interesting potential data-sources do not (yet) conveniently expose APIs. Mashups that extract content from sites like

Wikipedia, TV Guide, and virtually all government and public domain Web sites do so by a technique known as *screen scraping*. In this context, screen scraping denotes the process by which a tool attempts to extract information from the content provider by attempting to parse the provider's Web pages, which were originally intended for human consumption.

- The mashup site. This is where the mashup is hosted. Interestingly enough, just because this is where the mashup logic resides, it is not necessarily where it is executed. On one hand, mashups can be implemented similarly to traditional Web applications using server-side dynamic content generation technologies like Java servlets, CGI, PHP or ASP.

  Alternatively, mashed content can be generated directly within the client's browser through client-side scripting (that is, JavaScript) or applets. This client-side logic is often the combination of code directly embedded in the mashup's Web pages as well as scripting API libraries or applets (furnished by the content providers) referenced by these Web pages. Mashups using this approach can be termed *rich internet applications* (RIAs), meaning that they are very oriented towards the interactive user-experience. (Rich internet applications are one hallmark of what's now being termed "Web 2.0", the next generation of services available on the World Wide Web.) The benefits of client-side mashing include less overhead on behalf of the mashup server (data can be retrieved directly from the content provider) and a more seamless user-experience (pages can request updates for portions of their content without having to refresh the entire page). The Google Maps API is intended for access through browser-side JavaScript, and is an example of client-side technology.

  Often mashups use a combination of both server and client-side logic to achieve their data aggregation. Many mashup applications use data that is supplied directly to them by their user base, making (at least) one of the data sets local. Additionally, performing complex queries on multiple-sourced data (such as "Show me the average purchase price for real estate bought by actors who have co-starred in movies with Kevin Bacon") requires computation that would be infeasible to perform within the client's Web browser.

- The client's Web browser. This is where the application is rendered graphically and where user interaction takes place. As described above, mashups often use client-side logic to assemble and compose the mashed content.

Ajax

There is some dispute over whether the term Ajax is an acronym or not (some would

have it represent "Asynchronous JavaScript + XML"). Regardless, Ajax is a Web application model rather than a specific technology. It comprises several technologies focused around the asynchronous loading and presentation of content:

- XHTML and CSS for style presentation
- The Document Object Model (DOM) API exposed by the browser for dynamic display and interaction
- Asynchronous data exchange, typically of XML data
- Browser-side scripting, primarily JavaScript

When used together, the goal of these technologies is to create a smooth, cohesive Web experience for the user by exchanging small amounts of data with the content servers rather than reload and re-render the entire page after some user action. You can construct Ajax engines for mashups from various Ajax toolkits and libraries (such as Sajax or Zimbra), usually implemented in JavaScript. The Google Maps API includes a proprietary Ajax engine, and the effect it has on the user experience is powerful: it behaves like a truly local application in that there are no scrollbars to manipulate or translation arrows that force page reloads.

Web protocols: SOAP and REST

Both SOAP and REST are platform neutral protocols for communicating with remote services. As part of the service-oriented architecture paradigm, clients can use SOAP and REST to interact with remote services without knowledge of their underlying platform implementation: the functionality of a service is completely conveyed by the description of the messages that it requests and responds with.

SOAP is a fundamental technology of the Web Services paradigm. Originally an acronym for Simple Object Access Protocol, SOAP has been re-termed *Services-Oriented Access Protocol* (or just SOAP) because its focus has shifted from object-based systems towards the interoperability of message exchange. There are two key components of the SOAP specification. The first is the use of an XML message format for platform-agnostic encoding, and the second is the message structure, which consists of a header and a body. The header is used to exchange contextual information that is not specific to the application payload (the body), such as authentication information. The SOAP message body encapsulates the application-specific payload. SOAP APIs for Web services are described by WSDL documents, which themselves describe what operations a service exposes, the format for the messages that it accepts (using XML Schema), and how to address it. SOAP messages are typically conveyed over HTTP transport, although other transports (such as JMS or e-mail) are equally viable.

REST is an acronym for *Representational State Transfer*, a technique of Web-based communication using just HTTP and XML. Its simplicity and lack of rigorous profiles

set it apart from SOAP and lend to its attractiveness. Unlike the typical verb-based interfaces that you find in modern programming languages (which are composed of diverse methods such as `getEmployee()`, `addEmployee()`, `listEmployees()`, and more), REST fundamentally supports only a few operations (that is `POST`, `GET`, `PUT`, `DELETE`) that are applicable to all pieces of information. The emphasis in REST is on the pieces of information themselves, called resources. For example, a resource record for an employee is identified by a URI, retrieved through a `GET` operation, updated by a `PUT` operation, and so on. In this way, REST is similar to the document-literal style of SOAP services.

Screen scraping

As mentioned earlier, lack of APIs from content providers often force mashup developers to resort to screen scraping in order to retrieve the information they seek to mash. *Scraping* is the process of using software tools to parse and analyze content that was originally written for human consumption in order to extract semantic data structures representative of that information that can be used and manipulated programmatically. A handful of mashups use screen scraping technology for data acquisition, especially when pulling data from the public sectors. For example, real-estate mapping mashups can mash for-sale or rental listings with maps from a cartography provider with scraped "comp" data obtained from the county records office. Another mashup project that scrapes data is XMLTV, a collection of tools that aggregates TV listings from all over the world.

Screen scraping is often considered an inelegant solution, and for good reasons. It has two primary inherent drawbacks. The first is that, unlike APIs with interfaces, scraping has no specific programmatic contract between content-provider and content-consumer. Scrapers must design their tools around a model of the source content and hope that the provider consistently adheres to this model of presentation. Web sites have a tendency to overhaul their look-and-feel periodically to remain fresh and stylish, which imparts severe maintenance headaches on behalf of the scrapers because their tools are likely to fail.

The second issue is the lack of sophisticated, re-usable screen-scraping toolkit software, colloquially known as *scrAPIs*. The dearth of such APIs and toolkits is largely due to the extremely application-specific needs of each individual scraping tool. This leads to large development overheads as designers are forced to reverse-engineer content, develop data models, parse, and aggregate raw data from the provider's site.

Semantic Web and RDF

The inelegant aspects of screen scraping are directly traceable to the fact that content created for human consumption does not make good content for automated machine consumption. Enter the Semantic Web, which is the vision that the existing Web can be augmented to supplement the content designed for humans with

equivalent machine-readable information. In the context of the Semantic Web, the term information is different from data; data becomes information when it conveys meaning (that is, it is understandable). The Semantic Web has the goal of creating Web infrastructure that augments data with metadata to give it meaning, thus making it suitable for automation, integration, reasoning, and re-use.

The W3C family of specifications collectively known as the Resource Description Framework (RDF) serves this purpose of providing methodologies to establish syntactic structures that describe data. XML in itself is not sufficient; it is too arbitrary in that you can code it in many ways to describe the same piece of data. RDF-Schema adds to RDF's ability to encode concepts in a machine-readable way. Once data objects can be described in a data model, RDF provides for the construction of relationships between data objects through subject-predicate-object triples ("subject S has relationship R with object O"). The combination of data model and graph of relationships allows for the creation of ontologies, which are hierarchical structures of knowledge that can be searched and formally reasoned about. For example, you might define a model in which a "carnivore-type" as a subclass of "animal-type" with the constraint that it "eats" other "animal-type", and create two instances of it: one populated with data concerning cheetahs and polar bears and their habitats, another concerning gazelles and penguins and their respective habitats. Inference engines might then "mash" these separate model instances and reason that cheetahs might prey on gazelles but not penguins.

RDF data is quickly finding adoption in a variety of domains, including social networking applications (such as FOAF -- Friend of a Friend) and syndication (such as RSS, which I describe next). In addition, RDF software technology and components are beginning to reach a level of maturity, especially in the areas of RDF query languages (such as RDQL and SPARQL) and programmatic frameworks and inference engines (such as Jena and Redland).

RSS and ATOM

RSS is a family of XML-based syndication formats. In this context, syndication implies that a Web site that wants to distribute content creates an RSS document and registers the document with an RSS publisher. An RSS-enabled client can then check the publisher's feed for new content and react to it in an appropriate manner. RSS has been adopted to syndicate a wide variety of content, ranging from news articles and headlines, changelogs for CVS checkins or wiki pages, project updates, and even audiovisual data such as radio programs. Version 1.0 is RDF-based, but the most recent, version 2.0, is not.

Atom is a newer, but similar, syndication protocol. It is a proposed standard at the Internet Engineering Task Force (IETF) and seeks to maintain better metadata than RSS, provide better and more rigorous documentation, and incorporates the notion of constructs for common data representation.

These syndication technologies are great for mashups that aggregate event-based or update-driven content, such as news and weblog aggregators.

Technical Challenges

Like any other data integration domain, mashup development is replete with technical challenges that need to be addressed, especially as mashup applications become more feature- and functionality-rich. This section touches on a handful of these challenges, some of which you can address and mitigate, while others are open issues.

Data Integration Challenges: Semantic Meaning and Data Quality

Qualitative surveys suggest that the number one enterprise IT concern today is data integration within the enterprise virtual organization. (In this context, I use the term *virtual organization* to mean a composition of federated business units, each contained within its own administrative domain.) Like many enterprise IT managers who find themselves up to the task of integrating legacy data sources (for example, to create corporate dashboards that reflect current business conditions), mashup developers are faced with the analogous challenges of deriving shared semantic meaning between heterogeneous data sets. Therefore, to get an idea for what mashup developers have in store,you need look no further than the storied integration challenges faced by enterprise IT.

For example, translation systems between data models must be designed. When converting data into common forms, reasonable assumptions often have to be made when the mapping is not a complete one (for example, one data source might have a model in which an address-type contains a country-field, whereas another does not). Already challenging, this is exacerbated by the fact that the mashup developers might not be domain experts on the source data models because the models are third-party to them, and these reasonable assumptions might not be intuitive or clear.

In addition to missing data or incomplete mappings, the mashup designer might discover that the data they wish to integrate is not suitable for machine automation; that it needs cleansing. For example, law enforcement arrest records might be entered inconsistently, using common abbreviations for names (such as "mkt sqr" in one record and "Market Square" in another), making automated reasoning about equality difficult, even with good heuristics. Semantic modeling technologies, such as RDF, can help ease the problem of automatic reasoning between different data sets, provided that it is built-in to the data-store. Legacy data sources are likely to require much human effort in terms of analysis and data cleansing before they can be availed to semantic modeling technologies.

Mashup developers might also have to contend with several issues that IT

integration managers might not, one of which is data pollution. As part of their application design, many mashups solicit public user input. As evidenced in the wiki application domain, this is a double-edged blade: it can be quite powerful because it enables open contribution and best-of-breed data evolution, yet it can be subject to inconsistent, incorrect, or intentionally misleading data entry. The latter can cast doubts on data trustworthiness, which can ultimately compromise the value provided by the mashup.

Another host of integration issues facing mashup developers arise when screen scraping techniques must be used for data acquisition. As discussed in the previous section, deriving parsing and acquisition tools and data models requires significant reverse-engineering effort. Even in the best case where these tools and models can be created, all it takes is a re-factoring of how the source site presents its content (or mothballing and abandonment) to break the integration process, and cause mashup application failure.

Component Challenges

The Ajax model of Web development can provide a much richer and more seamless user experience than the traditional full-page-refresh, but it poses some difficulties as well. At its fundamentals, Ajax entails using the browser's client-side scripting capabilities in conjunction with its DOM to achieve a method of content delivery that was not entirely envisioned by the browser's designers. (Perhaps this hack-like nature of Ajax lends to its appeal.) However, this subjects Ajax-based applications to the same browser compatibility issues that have plagued Web designers ever since Microsoft created Internet Explorer. For example, Ajax engines make use of an `XMLHttpRequst` object to exchange data asynchronously with remote servers. In Internet Explorer 6, this object is implemented with ActiveX rather than native JavaScript, which requires that ActiveX be enabled.

A more fundamental requirement is that Ajax requires that JavaScript be enabled within the user's browser. This might be a reasonable assumption for the majority of the population, but there are certainly users who use browsers or automated tools that either do not support JavaScript or do not have it enabled. One such set of tools are the robots, spiders, and Web crawlers that aggregate information for Internet and intranet search engines. Without graceful degradation, Ajax-based mashup applications might find themselves missing out on both a minority user base as well as search engine visibility.

The use of JavaScript to asynchronously update content within the page can also create user interface issues. Because content is no longer necessarily linked to the URL in the browser's address bar, users might not experience the functionality that they normally expect when they use the browser's BACK button, or the BOOKMARK feature. And, although Ajax can reduce latency by requesting incremental content updates, poor designs can actually hinder the user experience, such as when the granularity of update is small enough that the quantity and overhead of updates

saturate the available resources. Also, take care to support the user (for example, with visual feedback such as progress bars) while the interface loads or content is updated.

As with any distributed, cross-domain application, mashup developers and content providers alike will also need to address security concerns. The notion of identity can prove to be a sticky subject, as the traditional Web is primarily built for anonymous access. Single-signon is a desirable feature, but there are a multitude of competing technologies (ranging from Microsoft Passport to the Liberty Alliance), thus creating disjointed identity namespaces that you must integrate as well. Content providers are likely to employ authentication and authorization schemes (which require the notion of secure identity or securely identifiable attributes) in their APIs to enforce business models that involve paid subscriptions or sensitive data. Sensitive data is also likely to require confidentiality (that is, encryption), and you must take care when you mash it with other sources to not put it at risk. Identity will also be crucial for auditing and regulatory compliance. Additionally, with data integration happening both on the server and client-side, identity and credential delegation from the user to the mashup service might become a requirement.

Social Challenges

In addition to the technical challenges described in the previous section, social issues have (or will) surface as mashups become more popular.

One of the biggest social issues facing mashup developers is the tradeoff between the protection of intellectual property and consumer privacy versus fair-use and the free flow of information. Unwitting content providers (targets of screen scraping), and even content providers who expose APIs to facilitate data retrieval might determine that their content is being used in a manner that they do not approve of. For a good review of Web aggregation and regulations, see Resources.

The mashup Web application genre is still in its infancy, with hobbyist developers who produce many mashups in their spare time. These developers might not be cognizant of (or concerned with) issues such as security. Additionally, content providers are only beginning to see the value in providing APIs for machine-based content access, and many do not consider them a core business focus. This combination can yield poor software quality, as priorities such as testing and quality assurance take the backseat to proof-of-concept and innovation. The community as a whole will have to work together to assemble open standards and reusable toolkits in order to facilitate mature software development processes.

Before mashups can make the transition from cool toys to sophisticated applications, much work will have to go into distilling robust standards, protocols, models, and toolkits. For this to happen, major software development industry leaders, content providers, and entrepreneurs will have to find value in mashups, which means viable

business models. API providers will need to determine whether or not to charge for their content, and if so, how (for example, by subscription or by per-use). Perhaps they will provide varying levels of quality-of-service. Some marketplace providers, such as eBay or Amazon, might find that the free use of their APIs increases product movement. Mashup developers might look for an ad-based revenue model, or perhaps build interesting mashup applications with the goal of being acquired.

Summary

Tutorials in the "The ultimate mashup -- Web services and the semantic Web" series

- Part 1: Use and combine Web services
- Part 2: Manage a mashup data cache
- Part 3: Understand RDF and RDFs
- Part 4: Create an ontology
- Part 5: Change out Web services
- Part 6: Give the user control

Mashups are certainly an exciting new genre of Web applications. The combination of data modeling technologies stemming from the Semantic Web domain and the maturation of loosely-coupled, service-oriented, platform-agnostic communication protocols is finally providing the infrastructure needed to start developing applications that can leverage and integrate the massive amount of information that is available on the Web. As mashup applications gain higher visibility, it will be interesting to see how the genre impacts social issues such as fair-use and intellectual property as well as other application domains that integrate data across organizational boundaries, such as grid computing and business-to-business workflow management.

For a deeper-dive into mashup development, stay tuned for the launching of a new series of tutorials on developerWorks that will teach you how to construct your own mashups. In fact, the series will even teach you how to use Semantic Web technology and ontologies to enable others to create their own mashups.

Resources

Learn

- Programmable Web: Stay up to date with the latest on mashups and the new Web 2.0 APIs.
- Considering Ajax, Part 1: Cut through the hype (Chris Laffra, developerWorks, May 2006): Consider this set of discussion points for every developer before you use Ajax techniques for a Web site.
- Ajax page: Visit this page sponsored by the Mozilla Development Center.
- The Interplay of Web Aggregation and Regulations (LawTech): Be sure to read this good review of Web aggregation and regulations (PDF file).
- DB2 and open source: Put yourself on the map with Google Maps API, DB2/Informix, and PHP on Linux (Marty Lurie and Aron Y. Lurie, developerWorks, March 2006): Create an easy-to-use map with your data on it.
- Building Web service applications with the Google API (Nicholas Chase, developerWorks, May 2002): Learn to embed Google search results and other information in your Java applications in this tutorial.
- The ultimate mashup -- Web services and the semantic Web tutorial series: Take the all the tutorials in this series and create a custom mashup.
- Second Generation Web Services: Read this XML.com article for coverage of the REST architecture.
- REST and the Real World: Read more on REST from XML.com.
- The W3C Semantic Web Activity site: Read about the Semantic Web.
- W3C RDF Activity: Visit this site for the latest on Resource Description Framework.
- Introduction to Jena: Use RDF models in your Java applications with the Jena Semantic Web Framework (Philip McCarthy, developerWorks, June 2004): Find out how to use the Jena Semantic Web Toolkit to exploit RDF data models in your Java applications.
- What is RSS?: From XML.com, learn about this syndication format for news, content, and personal weblogs.
- Atom Overview: Read about the XML-based Web content and metadata syndication format and application-level protocol from AtomEnabled.org.
- IBM XML 1.1 certification: Find out how you can become an IBM Certified Developer in XML 1.1 and related technologies.
- XML: See developerWorks XML Zone for a wide range of technical articles and

tips, tutorials, standards, and IBM Redbooks.

- developerWorks technical events and webcasts: Stay current with technology in these sessions.

Get products and technologies

- W3C SOAP Specification: Get the latest version.
- Scraping with style: scrAPI toolkit for Ruby: Try this technology for your mashups.

Discuss

- XML zone discussion forums: Participate in any of several XML-centered forums.

About the author

Duane Merrill
Duane Merrill has developed grid computing and distributed data integration platforms for over five years. He has been a contributor to the Legion Project at the University of Virginia and a core developer for the Avaki Corporation's distributed enterprise information integration product Avaki. He is currently obtaining his Ph.D in Computer Science at the University of Virginia.