## Mashups: The new breed of Web app

## An introduction to mashups

Skill Level: Introductory

Duane Merrill ([duane@duanemerrill.com](mailto:duane@duanemerrill.com))

Writer

Freelance 08 Aug 2006

Updated 24 Jul 2009

引用量： 132

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.

# Mashups：Web 应用程序新成员

Mashup 简介

## 简介

一种新型的基于 Web 的数据集成应用程序正在 Internet 上逐渐兴起。通常用术语 *mashup* 表示，它们的流行萌芽于对交互式用户参与和集成第三方数据的类似于科学怪人方式的重视。我们使用萌芽一词是有一定原因的；mashup Web 站点的特点就表现为它正在 Web 上扎根发芽，它们利用了从组织边界之外的数据源获取的内容和功能。

mashup 这种隐晦的数据集成定义当然不是非常严格。要深入了解什么是 mashup，就应该了解一下这个单词的起源：它源于流行音乐，mashup 是从两首不同的歌曲（通常属于不同的流派）中混合演唱和乐器的音轨而构成的一首新歌。与那些 “bastard pop” 歌曲类似，mashup 也是内容的一种不常见的创新组合（通常都源自于无关的数据源），这都是人工进行合成的（而不是通过计算机来合成的）。

那么，mashup 看起来到底是什么样子呢？ChicagoCrime.org 的 Web 站点上有非常直观的例子，它解释了*地图 mashup* 到底是什么。最初广泛流行起来的 mashup 之一是一个 Web 站点，它将芝加哥警局在线数据库中的犯罪记录与 Google Maps 上的地图复合在一起。用户可以与 mashup 站点进行交互，例如告诉它在图形界面上显示一个包含图钉的地图，图钉展示南加州最近所有入室抢劫案件的详细信息。这种概念和呈现方式非常简单，犯罪和地图数据复合之后提供的可视化的功能非常强大。

在 [Mashup 流派](https://www.ibm.com/developerworks/cn/xml/x-mashups.html#genres) 中，我们探索了流行的 mashup，包括地图 mashup。[相关技术](https://www.ibm.com/developerworks/cn/xml/x-mashups.html#related) 简要介绍了与 mashup 的构建和操作有关的技术前景。[技术挑战](https://www.ibm.com/developerworks/cn/xml/x-mashups.html#techchallenges) 和 [社会挑战](https://www.ibm.com/developerworks/cn/xml/x-mashups.html#socialchallenges) 分别介绍了影响 mashup 的主要技术挑战和社会挑战。

## Mashup 类型

在本节中，我们将简要介绍对出名的 mashup 类型进行的一些调查。

### 地图 mashup

在这个阶段的信息技术中，人们搜集大量有关事物和行为的数据，二者都常常具有位置注释信息。所有这些包含位置数据的不同数据集均可利用地图通过令人惊奇的图形化方式呈现出来。mashup 蓬勃发展的一种主要动力就是 Google 公开了自己的 Google Maps API。这仿佛打开了一道大门，让 Web 开发人员（包括爱好者、修补程序开发人员和其他一些人）可以在地图中包含所有类型的数据（从原子弹灾难到波士顿的 CowParade 奶牛都可以）。为了不落于人后，Microsoft（Virtual Earth）、Yahoo（Yahoo Maps）和 AOL（MapQuest）也很快相继公开了自己的 API。

### 视频和图像 mashup

图像主机和社交网络站点（例如 Flickr 使用自己的 API 来共享图像）的兴起导致出现了很多有趣的 mashup。由于内容提供者拥有与其保存的图像相关的元数据（例如谁拍的照片，照片的内容是什么，在何时何地拍摄的等等），mashup 的设计者可以将这些照片和其他与元数据相关的信息放到一起。例如，mashup 可以对歌曲或诗词进行分析，从而将相关照片拼接在一起，或者基于相同的照片元数据（标题、时间戳或其他元数据）显示社交网络图。另外一个例子可能以一个 Web 站点（例如 CNN 之类的新闻站点）作为输入，并在新闻中通过照片匹配而将照片中的内容以文字的形式呈现出来。

### 搜索和购物 mashup

搜索和购物 mashup 在 mashup 这个术语出现之前就已经存在很长时间了。在 Web API 出现之前，有相当多的购物工具，例如 BizRate、PriceGrabber、MySimon 和 Google 的 Froogle，都使用了 B2B 技术或屏幕抓取的方式来累计相关的价格数据。为了促进 mashup 和其他有趣的 Web 应用程序的发展，诸如 eBay 和 Amazon 之类的消费网站已经为通过编程访问自己的内容而发布了自己的 API。

### 新闻 mashup

新闻源（例如纽约时报、BBC 或路透社）已从 2002 年起使用 RSS 和 Atom 之类的联合技术来发布各个主题的新闻提要。以联合技术为基础的 mashup 可以聚集一名用户的提要，并将其通过 Web 呈现出来，创建个性化的报纸，从而满足读者独特的兴趣。Diggdot.us 正是这样的一个例子，它合并了 Digg.com、Slashdot.org 和 Del.icio.us 上与技术有关的内容。

## 相关技术

本节概要介绍了可以促进 mashup 发展的技术。有关这些技术的更多信息，请参阅本文最后的 [参考资料](https://www.ibm.com/developerworks/cn/xml/x-mashups.html#artrelatedtopics)。

### 架构

mashup 程序从架构上是由 3 个不同的部分组成的，它们在逻辑上和物理上都是相互脱离的（可能由网络和组织边界分隔）：API/内容提供者、mashup 站点和客户机的 Web 浏览器。

* API/内容提供者。它们是（有时是未知的）正在进行融合的内容的提供者。在 ChicagoCrime.org mashup 的例子中，提供者是 Google 和芝加哥警察局。为了方便数据的检索，提供者通常会将自己的内容通过 Web 协议对外提供（例如 REST、Web 服务和 RSS/Atom，稍后将加以介绍）。然而，很多有趣的潜在数据源可能并没有方便地对外提供 API。从诸如 Wikipedia、TV Guide 和所有政府和公共领域的 Web 站点上提取内容的 mashup 都是通过一种称为*屏幕抓取（screen scraping）* 的技术实现的。 在这种情况中，屏幕抓取就意味着使用一种工具从内容提供者那里提取信息的过程，这个工具会尝试对提供者的专为阅读而设计的页面进行分析。
* mashup 站点。即 mashup 所在的地方。非常有趣的是，这不过是因为这里是 mashup 逻辑所在的地方，而不是执行这些逻辑的地方。从一方面来说，mashup 可以直接使用服务器端动态内容生成技术（例如 Java servlets、CGI、PHP 或 ASP）实现为类似传统 Web 应用程序。

另外，合并内容可以直接在客户机的浏览器中通过客户机端脚本（即 JavaScript）或 applet 生成。这种客户机端的逻辑通常都是直接在 mashup 的 Web 页面中嵌入的代码与这些 Web 页面引用的脚本 API 库或 applet（由内容提供者提供）的组合。mashup 使用的这种方法可以称为*胖 Internet 应用程序*（RIA），这意味着它们是以交互式用户体验为导向的。（胖 Internet 应用程序具有 “Web 2.0” 的一个特点，Web 2.0 是 WWW 的新一代服务。）客户机端进行数据集成的优点包括：对 mashup 服务器的所产生的负载较轻（数据可以直接从内容提供者那里传送过来）、具有更好无缝用户体验（页面可以请求对内容的一部分进行更新，而不用刷新整个页面）。Google Maps API 的设计就是为了通过浏览器端的 JavaScript 进行访问，这是客户机端技术的一个例子。

通常，mashup 都使用服务器和客户机端逻辑的组合来实现自己的数据集成。很多 mashup 应用程序都使用了直接由用户提供的数据，（至少）使一个数据集是本地的。另外，对多数据源的数据执行复杂查询（例如 “请显示在 Kevin Bacon 的电影中出演角色的男演员所购买的房产的平均价格”）所需要的计算是不可能在客户机的 Web 浏览器中执行的。

* 客户机的 Web 浏览器。这是以图形化的方式呈现应用程序的地方，也是用户交互发生的地方。正如上面介绍的一样，mashup 通常都使用客户机端的逻辑来构建合成内容。

### Ajax

关于 Ajax 究竟是否是一个缩写词（有人认为它表示 “Asynchronous JavaScript + XML”）还存在争论。不论如何，Ajax 都是一个 Web 应用模型，而不是一种特定的技术。它包括几种关注内容的异步加载和呈现的技术：

* XHTML 和用于确定呈现风格的 CSS
* 浏览器为动态显示和交互所提供的文档对象模型（DOM）API
* 异步数据交换，通常是 XML 数据
* 浏览器端的脚本，主要是 JavaScript

将这些技术结合在一起使用时，它们的目标是通过与内容服务器交换少量的数据为用户创造平滑、良好的 Web 体验，而不用在用户执行某些操作之后重新加载并重新呈现整个页面。我们可以使用各种 Ajax 工具包和库（例如 Sajax 或 Zimbra）为 mashup 构建 Ajax 引擎，这通常是使用 JavaScript 实现的。Google Maps API 包括一个专用的 Ajax 引擎，它对用户体验的影响着实强大：它的工作方式类似于一个真正的本地应用程序，其中没有滚动条可以操作，也没有移动按钮强制页面重新加载。

### Web 协议：SOAP 和 REST

SOAP 和 REST 都是与远程服务进行通信所使用的与平台无关的协议。作为面向服务的架构范式的一部分，客户机使用 SOAP 和 REST 与远程服务进行交互，而不用了解它们底层的平台实现：服务的功能完全是由它请求和收到的显影消息描述来实现的。

SOAP 是 Web 服务范式中的一种基本技术。最初它是 Simple Object Access Protocol 的缩写，现在代表 *Services-Oriented Access Protocol*（或直接缩写为 SOAP），这是因为它的重点已经从基于对象的系统转向消息交换的交互操作。SOAP 规范中有两个关键组件。第一个组件是使用 XML 消息格式进行平台无关的编码，第二个组件消息结构，包括消息头和消息体。消息头用来交换非特定于应用负载（消息体）的相关信息，例如认证信息。SOAP 消息体封装了应用程序特有的负载。Web 服务的 SOAP API 是由 WSDL 文档来描述的，它们本身都描述了一个服务对外提供哪些操作，它可以接受的消息格式（使用 XML Schema），以及如何对其进行寻址。SOAP 消息通常都是通过 HTTP 协议传送的，不过也可以通过其他方式传送（例如 JMS 或 e-mail）。

REST 是 *Representational State Transfer* 的缩写，这是一种只使用 HTTP 和 XML 进行基于 Web 通信的技术。它的简单性和缺少严格配置文件的特性使它与 SOAP 很好地隔离开来，并且吸引了大家广泛的兴趣。与我们在现代变成语言中可以找到的典型基于动词的接口不同（它们构成了各种方法，例如 getEmployee()、addEmployee()、listEmployees() 等）不同，REST 从根本上来说只支持几个操作（即 POST、GET、PUT、DELETE），这些操作适用于所有的消息。REST 强调信息本身，称为资源。例如，一个员工的资源记录是由 URI 标识的，这可以通过一个 GET 方法获得，并使用一个 PUT 操作进行更新，等等。使用这种方法，REST 就与文档文本风格的 SOAP 服务非常类似。

### 屏幕抓取

正如前面介绍的一样，缺乏内容提供者提供的 API 通常会强制要求 mashup 开发人员采取屏幕抓取的方式来提取自己希望集成的信息。*抓取（Scraping）* 是使用软件工具处理并分析最初为人们阅读而编写的内容，从而从中提取出可以通过编程进行使用和操作的信息的语义数据结构表示。有些 mashup 使用屏幕抓取技术来获取数据，特别是从公用领域提取数据。例如，房地产地图 mashup 就可以在制图供应商提供的地图上显示售价和租价，这些数据可能是从当地的记录办公室抓取来的 “comp” 数据。另外一个抓取数据的 mashup 项目是 XMLTV，这是一组汇聚了各地电视节目清单的工具集。

屏幕抓取通常被认为是一个不雅的解决方案，这是有一定的原因的。它有两个主要的固有缺点。第一个缺点在于，与使用接口的 API 不同，抓取在内容提供者和内容消费者之间没有明确的联系。抓取者必须围绕一个源内容模型设计自己的工具，并且希望提供者一直采用这种模型来呈现内容。Web 站点倾向于周期性地更新外观，以保持新颖和时髦，对于抓取者来说，这是一项非常头痛的维护任务，因为工具很可能会失效。

第二个问题是缺少成熟的可重用屏幕抓取工具包软件，通俗地说就称为 *scrAPI*。此类 API 和工具包的消亡很大程度上是由于每种抓取工具都有极为特定于应用程序的需求。这为开发人员带来了过多的开发工作，他们必须对内容进行反向工程处理、开发数据模型、分析并从提供者站点上汇集原始数据。

### 语义 Web 和 RDF

屏幕抓取不好的一面直接源自于一个事实：为阅读而创建的内容并不太适合机器自动处理。这促进了语义 Web 的诞生，它是现有 Web 的增强版本，在为人们设计的内容中增加了足够多的可供机器阅读的信息。在语义 Web 环境中，信息这个术语与数据有所差异；数据只有在传达了自己的含义（即数据可被理解）之后才会变成信息。语义 Web 的目标是创建 Web 基础设施，使用元数据对数据进行增强，从而使数据变得有意义，最终使数据变得适合进行自动化、集成、推理和重用。

被称为资源描述框架（RDF）的 W3C 系列规范就是服务于这个目的的技术，它用来建立描述数据的语义结构。XML 本身并不足以实现这种功能；它太过随意，我们可以使用很多方法进行编码来对相同的数据进行描述。RDF-Schema 补充了 RDF 的能力，提供了以机器可读的方式编码概念的功能。一旦可通过一种数据模型描述数据对象，RDF 就提供了通过主语－谓语－对象三元组（主语 S 与对象 O 具有关系 R）在数据对象之间构建关系的能力。数据模型与关系图之间的区别让我们可以进行存在式的构建，这是可以进行搜索和形式化推理的知识的层次化结构。例如，我们可以定义这样一个模型：“肉食动物” 是 “动物” 的一个子类，条件是它 “吃” 其他 “动物”；并创建两个实例：一个实例是印度豹和北极熊，并提供它们的生存环境；另外一个是瞪羚和企鹅，并提供它们的生存环境。假设我们将这些单独的模型实例集成在一起，就可以推论说印度豹可能会以瞪羚为食，但却不会吃企鹅。

RDF 数据在很多领域中都迅速得到了应用，包括社交网络应用程序（例如 FOAF —— Friend of a Friend）和联合（例如 RSS，接下来就会介绍）。另外，RDF 软件技术和组件都正在成熟到一定规模，尤其是在 RDF 查询语言（例如 RDQL 和 SPARQL）、编程框架和推理引擎（例如 Jena 和 Redland）领域。

### RSS 和 ATOM

RSS 是一系列基于 XML 的联合格式。在这种情况中，联合（syndication）是指一个发布内容的 Web 站点可以创建 RSS 文档并在 RSS 发布系统中注册自己的文档。支持 RSS 的客户机可以查看新内容，并通过适当的方式连接到这些内容上。RSS 已经被用来联合广泛的内容，从新闻到头条、CVS 或 WIKI 页面的修改日志、项目更新甚至诸如无线电节目之类的视听数据。版本 1.0 基于 RDF，但最新的 2.0 版本不以 RDF 为基础。

Atom 是一种更新但非常类似的联合协议。它是 Internet Engineering Task Force（IETF）提出的一项草案标准，人们希望通过 Atom 提供比 RSS 更好的元数据维护；提供更好、更为全面的文档，并结合构建通用数据表示的概念。

这些联合技术对于集成基于事件或更新驱动内容的 mashup 来说都非常有用，例如新闻和 weblog 聚集程序。

## 技术挑战

与其他数据集成领域一样，mashup 开发也充斥着许多亟待解决的技术挑战，随着 mashup 应用程序特性和功能的进一步丰富，这种挑战也变得更加严峻。本节简单介绍了一些挑战，其中有些挑战目前已经能够解决或缓解，而其他问题依然没有解决。

### 数据集成挑战：语义和数据的品质

品质调查显示，当今的企业 IT 首要关注的问题就是是企业虚拟组织中的数据集成。（在这种情况中，我们使用了 *虚拟组织（virtual organization）* 这个术语表示很多联合业务单元的组合，每个业务单元都包含在自己的管理域中。）与很多发现自己忙于集成传统数据源的企业 IT 管理人员一样（例如，创建可以反映当前业务状况的企业仪表板），mashup 开发人员需要面对类似源自于在异构数据集之间共享语义的挑战。因此，要了解 mashup 开发人员是如何为此作出准备，只需了解企业 IT 所面临的集成挑战。

例如，我们必须设计数据模型之间的转换系统。在将数据转换成通用的格式时、在映射不完整时（例如，一个数据源可能有一个模型，其中一个地址类型包含了一个国家字段，而另外一个模型中没有这个字段），我们必须进行一些合理的假设。尽管已经面临这些挑战，但是 mashup 开发人员可能并不是源数据模型领域的专家，因为这些模型可能是第三方的产品，这些合理的假设可能并不直观清晰，这更加剧了挑战的严峻性。

除了缺少数据和映射不完整之外，mashup 设计者可能会发现他们希望集成的数据并不适合进行机器自动化处理；这将带来很多净化工作。例如，执法逮捕记录可能不一致：记录中可能为名字使用了常用的缩写形式（例如，一条记录中使用的是“mkt sqr”，另外一条记录中使用的是“Market Square”），这使得关于等同性的自动推理变得非常困难，即使采用很好的启发式规则也很难实现。语义建模技术，例如 RDF，可以帮助简化对不同数据集之间自动进行推理所面临的问题，这些数据集是内嵌在数据存储介质中的。对于传统的数据源来说，通常需要投入大量人力物力，进行分析和数据净化工作，然后才能将其用于语义建模技术。

mashup 开发人员可能还必须面对 IT 集成管理人员不需要面对的一些问题，其中一个问题是数据污染。作为应用程序设计的一部分，很多 mashup 都要求公共用户提供输入。wiki 应用程序领域的研究表明，这是一把双刃剑：它可能非常强大，因为可以提供开放的贡献和最佳的数据革新，但这又会导致不一致、不正确或容易产生误导的数据项。后者可能会危及数据的可信度，最终降低 mashup 带来的价值。

mashup 开发人员需要面对的另外一种集成问题是由于获取数据必须采用屏幕抓取技术而引起的。正如上一节所讨论的一样，分析和获取工具以及数据模型都需要大量与反向工程相关的工作。在最理想的情况下，可以创建这些工具和模型，但依然存在一个问题：源站点如何呈现自己的内容，这可能会破坏集成过程，并导致 mashup 应用程序出错。