MASHUPS

Strategies for the Modern Enterprise

J. JEFFREY HANSON

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Hanson, J. Jeffrey.
Mashups : strategies for the modern enterprise / J. Jeffrey Hanson.
p. cm.
Includes index.
ISBN 978-0-321-59181-4 (pbk. : alk. paper) 1. Software engineering. 2. Mashups (World Wide Web) 3. Web site development. I. Title.

QA76.758.H363 2009
006.7—dc22

2009004655

Copyright © 2009 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-59181-4
ISBN-10: 0-321-59181-X
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, May 2009

Editor-in-Chief
Mark L. Taub

Acquisitions Editor
Trina MacDonald

Editorial Assistant
Olivia Basegio

Development Editor
Michael Thurston

Managing Editor
John Fuller

Full-Service Production Manager
Julie B. Nahil

Copy Editor
Geneil Breeze

Indexer
Michael Loo

Proofreader
Linda Begley

Cover Designer
Chuti Prasertsith

Compositor
Rob Mauhar

Preface

In this book I introduce you to a trend in software engineering that will permeate several areas of software development for many years to come. This trend is referred to as "mashups," and in this book I discuss the concepts of mashup implementations as they affect the enterprise.

The discussions and projects started while writing this book will continue to be a work in progress as the mashup landscape evolves.

I chose the topic of mashups for this book because I am excited to see the things that are being done with mashup development and to see this excitement starting to crop up in the enterprise as organizations begin to adopt a mashup development mindset. Companies such as JackBe, IBM, Microsoft, Yahoo!, and others have developed powerful mashup tools and environments that are beginning to show the power mashups can offer to an enterprise.

I have been privileged for a couple of decades to see many attempts to create an environment in which existing UI artifacts and processes can be reused with little or no programming intervention. Mashup development using semantic technologies, reusable markup-based UI artifacts, and metadata-enabled data formats has reached the point at which powerful applications and services can be constructed with existing code, data, and UI components using very little programming intervention.

Overview of This Book

This book discusses implementation strategies, frameworks, and code samples for enterprise mashups. The term "mashup" originated from the music industry to define the technique of producing a new song by mixing together two or more existing songs. The term has been adopted by the software development industry to define applications created by mixing user-interface artifacts, processes, and/or content from multiple sources, typically using high-level web programming languages such as HTML, JavaScript, and others.

A mashup infrastructure enables a development model that can produce new pages, applications, and services rapidly with very little additional work. The use and reuse of semantic web technologies, user interface artifacts, and loosely coupled services provide a powerful domain for mashup application development.

Mashups are being created at an almost unprecedented rate for different consumer and social environments. This trend is starting to spill over into the enterprise domain due to the power and speed with which development teams can create new services, applications, and data transformations by exploiting the agile and dynamic environment of mashup infrastructures. Some of the more popular, publicly accessible mashups include HousingMaps, TwitterVision, Big Contacts, Weather Bank, and others.

As mashups begin to migrate to the enterprise, more sophisticated programming languages and constructs become involved. Lower-level concepts also become involved including data mediation and transformations, interprocess communications, single sign-on, governance, and compliance to name a few.

This book discusses how developers can create new user interfaces by reusing existing UI artifacts using high-level web page markup and scripting languages such as HTML and JavaScript. Also discussed is the ability that a mashup infrastructure gives to developers to integrate disparate data using semantically rich data formats.

The ideas presented in this book are focused on implementation strategies using such technologies as XML, Java, JavaScript, JSON, RDF, HTML, RSS, and others. The discussions presented in this book look at programming and scripting languages in a generic sense—that is, I do not attempt to address mashup implementations across all popular frameworks. For example, I do not delve into how mashups can be implemented using Spring, JSF, Struts, and so on. However, whenever possible, I do mention some of the more prevalent frameworks that can be used to address a specific need.

It is my hope that a reader of this book will gain a good understanding of the main concepts of mashup development, particularly as applied to the enterprise. I present code examples and actual working mashups. I seek to provide a reader with a running start into mashup development for the enterprise.

Target Audience for This Book

This book is intended for use by software/web developers as well as by managers, executives, and others seriously interested in concepts and strategies surrounding mashups and enterprise mashups. The book strives to serve as an instructive, reliable introduction to the enterprise mashup arena. I hope that the book will answer many of the questions that might be asked by those seeking to gain a good foundation for discovering the world of mashup development. This book also describes solid business reasons for choosing enterprise mashups: speed of implementation, quick results, and rapid value-add.

To get the most use of this book, it is advisable that you briefly introduce yourself to HTML, JavaScript, XML, Java, and the basics of the HTTP protocol. However, many of the abstract concepts of mashups and mashup development can be garnered from this book without the need of programming skills.

About JSF, Spring, Hibernate, and Other Java Frameworks

The goal of this book is to present the concepts and techniques for designing and building enterprise mashups and enterprise mashup infrastructures. Concepts and techniques for mashups and mashup infrastructures are topics broad enough to discuss without attempting to weave in the specifics of multiple frameworks such as Spring, JSF, Hibernate, and Struts. Also, many of the concepts and techniques for mashups are currently realized using web page markup and scripting languages such as HTML and JavaScript. However, where warranted, I highlighted some frameworks that fill a specific niche for tasks such as transforming data, authentication, manipulating XML, and providing kernel functionality.

Frameworks such as Spring, JSF, Hibernate, Struts, EJB3, JPA, and others are very powerful. For example:

- Spring provides libraries and frameworks for building rich web applications, integrating with BlazeDS, building document-driven web services, securing enterprise applications, supporting Java modularity, integrating with external systems, building model-view-controller (MVC) applications, and others. Spring also includes a popular MVC framework.
- JavaServer Faces (JSF) is a set of APIs and a custom JSP tag library for building user interfaces for web applications, managing UI-component state, handling UI events, validating input parameters, defining page navigation, and other tasks.
- Struts is an MVC framework that extends the Java servlet API to allow developers to build sophisticated web flows using the Java programming language.
- Hibernate is an object-relational mapping (ORM) framework that allows developers to map a Java-based object model to relational database tables.

These frameworks are ubiquitous and, of course, useful, and one might expect to find a discussion of each of them in a book such as this. However,

they are not mashup-oriented in nature at this point and would, therefore, require the reader to have an in-depth knowledge of each framework to engage the reader in a coherent discussion as they relate to mashups. With this in mind, I have chosen to keep my discussions of mashups and the Java programming language as generic as possible. A detailed discussion of building enterprise mashups using frameworks such as these deserves a complete book for each. Some enterprises purposely disallow JavaScript in user interfaces. It is a good idea to explore JSF tools, such as JBoss Rich Faces, which includes artifacts that intelligently manage JavaScript availability.

Introduction

The term "mashup" originated with the technique of producing a new song by mixing together two or more existing songs. This was most notably referred to in the context of hip-hop music. Mashup applications are composed of multiple user interface components or artifacts and/or content from multiple data sources. In the context of software engineering, the term "mashup" defines the result of combining existing user interface artifacts, processes, services, and/or data to create new web pages, applications, processes, and data sets.

Very rapid implementations of new functionality are afforded by mashups via the use of semantic web technologies, reusable user interface artifacts, and loosely coupled services. Many different consumer and social spaces use mashups. However, enterprises are beginning to reap the benefits afforded by a mashup environment. Mashups create an extremely agile and dynamic design and implementation environment within the enterprise realm allowing users with limited technical skills to develop powerful and useful applications and services.

In a mashup environment, users can create new user interfaces by reusing existing UI artifacts using high-level scripting languages such as HTML and JavaScript. Mashups also enable users to integrate disparate data very quickly and easily using semantically rich data formats that don't require complex programming and middleware technologies. Services and processes are beginning to be integrated with similar speed and ease using loosely coupled techniques inherited from the lessons learned from service-oriented architecture (SOA) solutions.

Web 1.0 to Web 2.0 to Web 3.0

Technologies surrounding the presence of an organization or user have taken two significant steps over time, transitioning from what is sometimes referred to as "Web 1.0" to what has become known as "Web 2.0." Web 1.0 began with the first HTML-based browsers and, even though it still lingers in many web sites, the Web 1.0 model has evolved rapidly towards a Web 2.0 model.

Web 1.0 delivered content in a static manner using HTML markup and simple HTML forms. Applications written to a Web 1.0 model typically responded to HTTP requests with entire web page updates created from data pulled from relational tables and content management systems using standard web application programming languages such as Perl, C, C++, Java, and others.

Web 2.0, fueled by ubiquitous access to broadband, originated soon after the turn of the century and is in some form in most up-to-date web sites and web applications today. Web 2.0 moves the online experience away from static content delivery towards a model based on interactive participation, blogging and RSS feeds, search and tagging, AJAX and partial-page updates, collaboration and social networking, wikis, online bookmarking and content sharing, and so on. Web 2.0 turned the Internet into a true application platform. Technologies surrounding Web 2.0 have led to the enablement of shareable and embeddable UI artifacts such as widgets, dynamic JavaScript, videos, and HTML snippets.

Web 3.0 is a term that describes the next online evolutionary trend following the Web 2.0 model. The model for Web 3.0 is emerging as a transformation to a decentralized and adaptable framework across virtually any type of connected entity including web browsers, desktops, handheld devices, and proprietary firmware. Content and functionality are delivered in the Web 3.0 model via on-demand software as a service (SaaS), cloud computing, open APIs, standard web-based protocols, and semantically rich data formats. Content is secured using open, decentralized, security protocols and standards. Web 3.0 is moving organizations away from proprietary, closed systems to a model that encourages sharing, collaboration, and reuse.

To many, mashups are becoming synonymous with Web 3.0. Mashups are the embodiment of true open, reusable, web-based components and data. This concept will certainly change the way organizations do business and yield a flood of activity towards enterprise mashups.

Overview of Mashup Technologies

Technologies used today to produce a mashup application include HTML snippets, widgets, dynamic JavaScript, AJAX, and semantic-web formats. Content for a mashup is retrieved from internal systems as well as third-party web sites. Protocols and data formats include HTTP, HTTPS, XML, SOAP, RDF, JSON, and others.

Mashups are created ad hoc most of the time. However, in the enterprise realm, mashup applications must take into consideration such things as privacy, authentication, governance, compliance, and other business-related constraints.

Mashups can combine data from disparate data sources, existing UI artifacts, and/or existing software processes or services. The specific design for a

mashup depends on whether the mashup will be visual or nonvisual. In many cases an enterprise mashup solution will be a combination of data, UI artifacts, and software processes. The solution might be a combination of nonvisual and visual efforts.

Ultimately, a mashup application exploits existing data, UI artifacts, and software processes to create new applications and services that might also be exploited as components for other mashup efforts. This propagation of reusable components or modules is creating a revolutionary atmosphere where underlying programming frameworks and languages are irrelevant and higher-level scripting languages, semantics, and UI components are emerging as the primary application enablers.

Figure I.1 illustrates a shopping-specific mashup application that combines data gathered from a geocoding site, a wholesaler site, a local database, and a social networking site. The data is then mashed together inside a browser web page to form the new application.

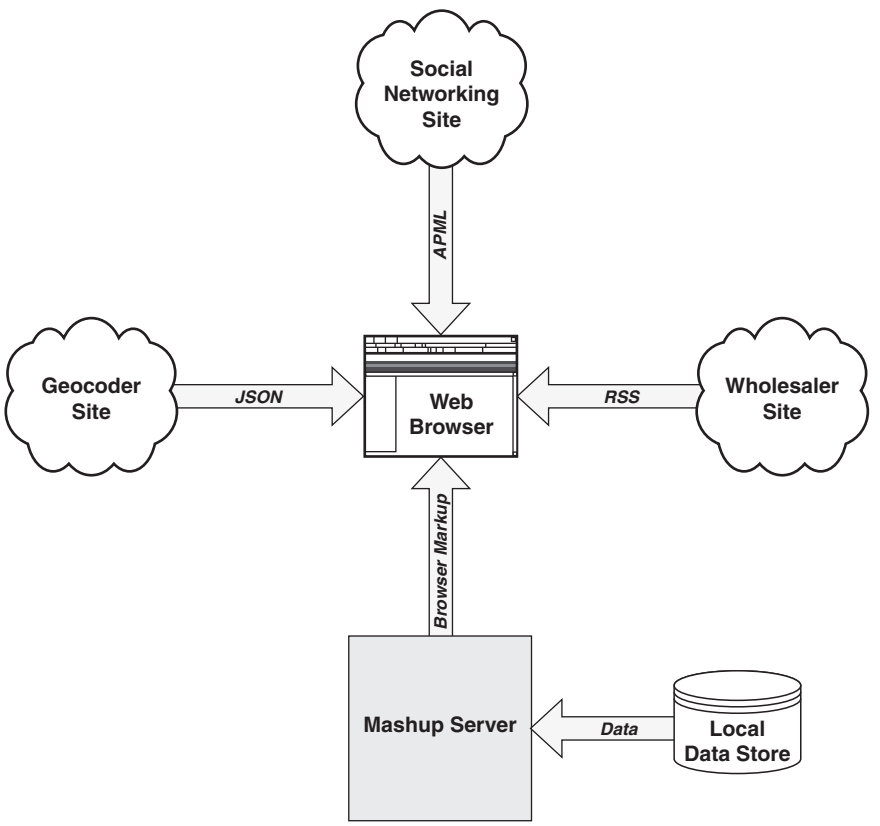

Figure I.1 *A shopping-specific mashup application combining data within a browser web page*

As shown in Figure I.1 data for a mashup can be retrieved from a number of different locations and combined within a browser web page to create a new application or interface. Building a mashup in this scenario typically uses JavaScript processing and DOM manipulation within the browser page to create the user interface for the new application.

Mashups can be created using traditional programming languages outside the browser. Figure I.2 illustrates a shopping-specific mashup application that combines data gathered from a geocoding site, a wholesaler site, a local database, and a social networking site. The data is then mashed together inside a mashup server to create the new application.

As shown in Figure I.2 data for a mashup can be retrieved from a number of different locations and combined within a mashup server to create a new application or interface. Building a mashup in this scenario typically uses traditional programming languages such as Java, PHP, Python, C#, Perl, Ruby, and C++ to integrate the data. The user interface for the new application is created using traditional web frameworks such as JSP, ASP, Struts, and others.

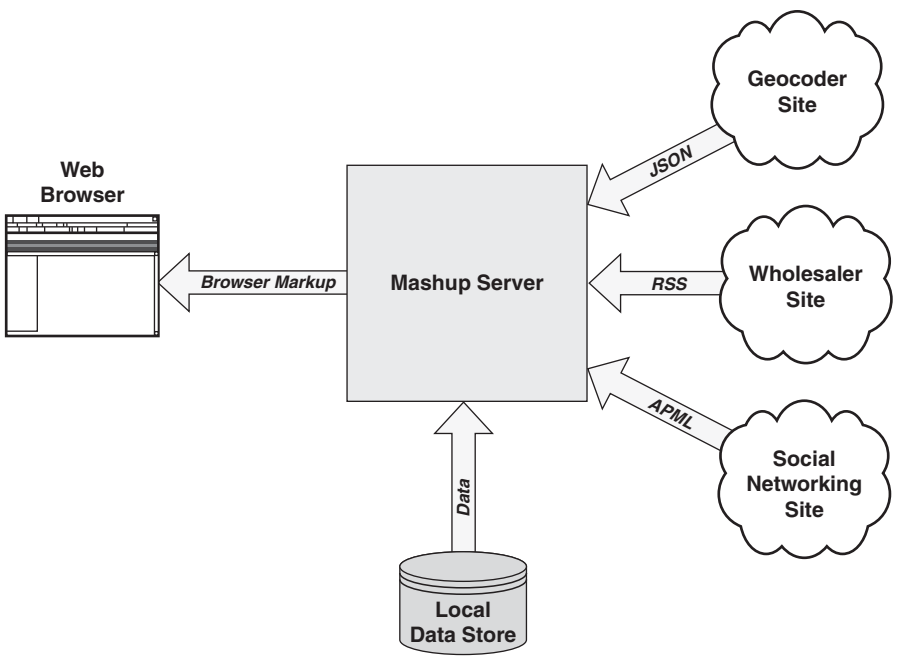

Figure I.2 *A shopping-specific mashup application combining data within a mashup server*

Enterprise Mashup Technological Domains

Mashup domains depend on what is to be "mashed" together. Generally, three high-level categories of items can be mashed together—user interface artifacts (presentation), data, and/or application functionality (processes). This might include HTML snippets, on-demand JavaScript to access an external API, web service APIs from one of your corporate servers, RSS feeds, and/or other data to be mixed and mashed within the application or pages. The implementation style, techniques, and technologies used for a given mashup depend on this determination. Once the items are determined, a development team can proceed with applying languages, processes, and methodologies to the application at hand.

Technologies used to create a mashup also depend on the sources from which the mashup items will be accessed, the talents a development staff needs to build the mashup, and the services that need to be created or accessed to retrieve the necessary artifacts for the mashup.

Mashups rely on the ability to mix loosely coupled artifacts within a given technological domain. The requirements of the application determine what artifacts (UI, data, and/or functionality) are needed to build the mashup.

From a high-level perspective, the technological domain as applied to mashups can be viewed as presentation-oriented, data-oriented, and process-oriented. Different languages, methodologies, and programming techniques apply to each technological domain.

As shown in Figure I.3, mashup categories can be divided according to presentation artifacts, data, and application functionality/processes.

Certain preparations must be made to design and implement a mashup that is ready for the enterprise. Primary areas of concern are requirements and constraints, security, governance, stability, performance, data, implementation, and testing. Certain aspects of each area of concern are unique to the environment of enterprise mashups in respect to other enterprise software disciplines.

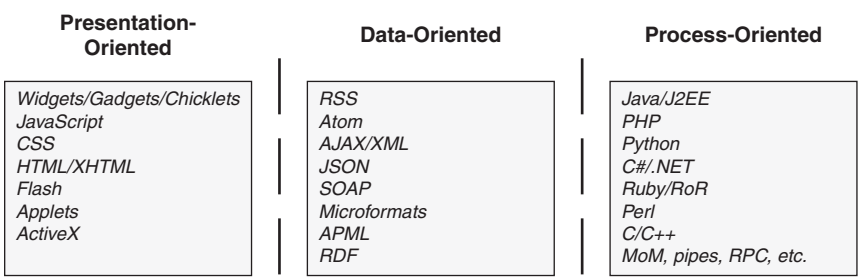

Figure I.3 *Three primary mashup technological domains*

The requirements and constraints for an enterprise mashup must integrate with the existing policies and practices of the environment for a company or enterprise. Identifying the requirements and constraints for a mashup is an evolutionary process, since the environment is bound to be affected by the mashup community with which it will interact. However, there are some basic requirements and constraints of a mashup that can be identified and addressed. As the mashup evolves these items will evolve or be replaced.

Preparing for enterprise mashup implementations involves a thorough understanding of the processes and IT landscape unique to a given company and/or industry. Techniques and technologies used for one company or industry are generally transferable to another; however, some aspects are unique and must not be overlooked. In fact, these very points of uniqueness typically create the most value for an enterprise mashup. Effectively exposing the distinctive facets of an organization is a primary goal of any online endeavor. Achieving this goal with a mashup infrastructure can add value in ways that are not even apparent until a given mashup community is exposed to the resources and artifacts you present. Once a community becomes actively involved with your mashup infrastructure, the infrastructure itself evolves as a result of the community's unified creativity. Therefore, it is very important for an organization to make sure it has certain foundational preparations in place in anticipation of this creative evolution.

Considerations Unique to the Enterprise Mashup Domain

An enterprise mashup must heed a more restrictive set of considerations such as compliance, standards, and security that public domain mashups are often free to ignore. In addition, a company or enterprise mashup must not expose certain types of intellectual property and/or protected information. This is similar to the issues that service-oriented organizations face when exposing service APIs to the web community. Enterprise mashups face the same issues as well as new issues related to data and UI artifacts that may be used by the mashup community.

If the promise of mashups is realized and a company or enterprise does experience a viral wave of activity and popularity due to its mashup environment, the company or enterprise must be ready to handle the surge of online activity. This is why preparatory efforts relating to security, performance, and scalability must be taken to ensure that your infrastructure will handle the surge. Every aspect of an IT infrastructure should be optimized and modularized to enable as much flexibility and, therefore, creativity as possible. The atmosphere of a

community-based creative mind can be created; not only in the presentation domain, but in the data domain and process domain as well if you take the proper steps to create an infrastructure that supports loosely coupled interactions throughout.

Implicit to a loosely coupled infrastructure hoping to benefit from a viral community is the importance of the ability of a company or enterprise to monitor and manage the infrastructure effectively. As viral effects take place, it is inevitable that bottlenecks in the infrastructure will be encountered even after painstaking efforts are taken in the design and implementation of the infrastructure. Therefore, it is vital that a potent monitoring and management framework be in place to identify bottlenecks quickly so that they might be rectified immediately.

As shown in Figure I.4, a typical loosely coupled enterprise architecture embodies a number of different frameworks from which data, services, processes, and user interface components emerge.

The need for an agile security model for an enterprise mashup cannot be emphasized enough. Since the mashup infrastructure will need to handle requests and invocations from a vast array of clients and environments, the mashup infrastructure must be ready to handle many types of identity management and access controls. Since it is impossible to know beforehand just how

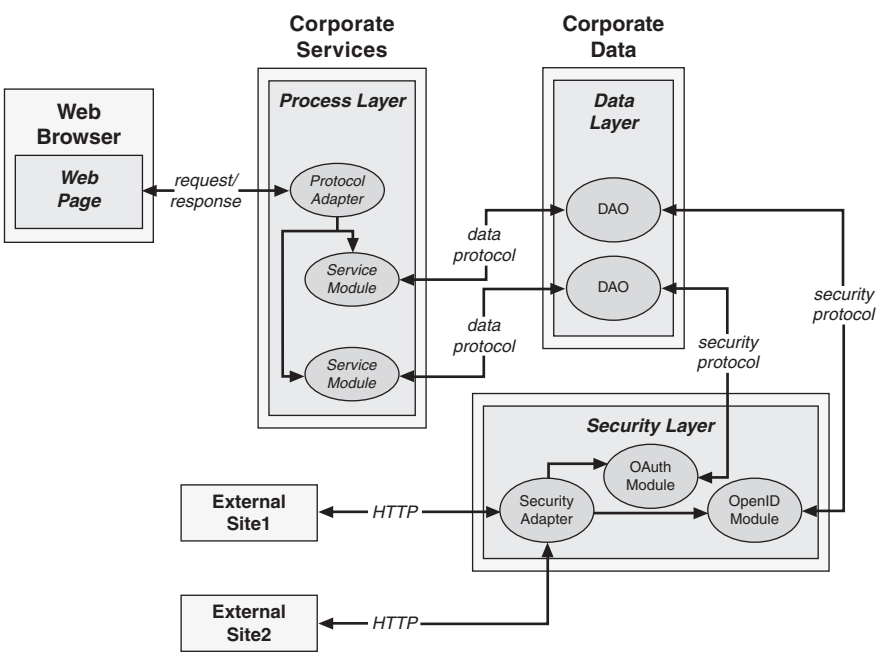

Figure I.4 *Typical loosely coupled enterprise architecture*

data modules, UI artifacts, and services will be used by the mashup community, an organization must have an infrastructure in place that allows it to control how its information is used without inhibiting creativity.

Being able to support an agile and viral environment is also very important when it comes to deploying enterprise mashups and the components of the mashups. Flexible deployment techniques and technologies must be used throughout the scope of the infrastructure to allow updates and enhancements to be deployed without affecting the mashup community's interactions with the mashup infrastructure. This includes activities related to editing and/or execution of the mashup artifacts.

Finally, the enterprise mashup and its artifacts must be tested. Testing enterprise mashups and mashup artifacts is one of the most important tasks a company or enterprise must address since the infrastructure and artifacts will be exposed to such a dynamic and vast community. Methods and techniques for testing mashups and mashup artifacts must be as agile and dynamic as the environment in which they will operate.

Solving Technological Problems

An enterprise mashup infrastructure must present solutions for a very agile and evolutionary environment. Data sources can change rapidly, services are added and changed at any given time, presentation technologies are constantly being integrated with the system, marketing and sales departments are eager to apply the potential facilitated by the easy UI generation model, and so on.

The dynamic nature of an enterprise mashup environment must be flexible and powerful enough to handle existing business operations as well as many new operations that arise out of the dynamic nature of the mashup development model.

An enterprise mashup infrastructure can be used to update, access, and integrate unstructured and structured data from sources of all kinds. An enterprise mashup infrastructure can apply structure to extracted data that was previously unstructured. Such is the case when structure is applied to an ordinary HTML page using screen-scraping techniques.

An enterprise mashup infrastructure presents views of existing resources and data to other applications where they are restructured and aggregated to form new composite views that may even create new semantic meaning for the composite data and, therefore, for the enterprise itself.

As shown in Figure I.5, an enterprise mashup infrastructure provides a number of different frameworks and layers from which data, services, processes, and user interface components are integrated.

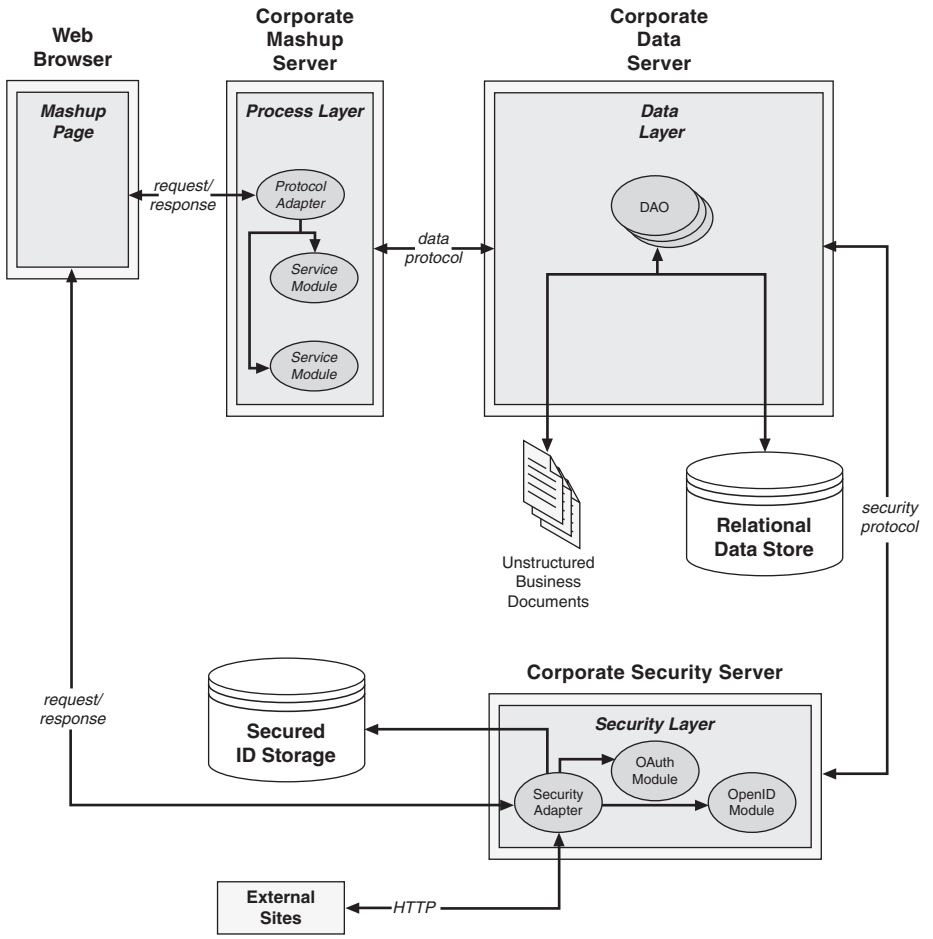

Figure I.5 *High-level view of an enterprise mashup infrastructure*

An enterprise mashup infrastructure helps to solve nonvisual integration problems as well as visually related problems. Nonvisual integration solutions enabled using the resource-oriented and semantic nature of an enterprise mashup infrastructure can be applied directly to specific business problems or indirectly through the orchestration and aggregation of the reusable components presented by the infrastructure.

Addressing mashups in a nonvisual sense relies on accurate and comprehensive organization and structure of information using semantically rich data formats to create an environment where content and data are easily discovered and reused.

Structuring Semantic Data

As with any enterprise application environment, enterprise mashup infrastructures must address some fundamental concerns such as information management, governance, and system administration to name a few. In addition to the typical enterprise application concerns, mashup infrastructures must address an environment that seeks to fulfill dynamic requirements and flexible solutions to business issues.

One of the biggest challenges facing an enterprise is that issue of managing and sharing data from disparate information sources. Legacy mechanisms for managing and sharing information typically kept data and metadata (data about the data) separated. Semantic techniques and technologies seek to bridge the gap between data and metadata to present a more effective means for applying meaning to information.

Choosing the fundamental format for data within your mashup infrastructure should be one of the first areas that you address. Mashup infrastructures derive much of their benefit from being able to apply and present semantic meaning to data and content. This enables consumers of the data and content to create aggregate components and content much more easily than traditional application environments.

Applying semantics to an aggregate repository of corporate information involves extending typical data stores and content sources to enable the information stored within unstructured documents and files with structured meaning, thereby giving the information sources features that enable both machines and humans with a greater ability to understand the information. Once effective semantic meaning has been applied to an information source, the data stored within can be discovered, aggregated, automated, augmented, and reused more effectively.

As shown in Figure I.6, an enterprise mashup infrastructure can provide components, modules, and frameworks to transform and enable data with semantic richness.

Figure I.6 illustrates some of the disparate sources from which corporate information is stored and how a mashup infrastructure might provide a solution for structuring the data from these sources with semantic meaning.

When determining a solution for building a semantic foundation, an organization should turn to formal specifications. These specifications currently include XML, the Resource Description Framework (RDF), the Web Ontology Language (OWL), RDF Schema (RDFS), microformats, and others.

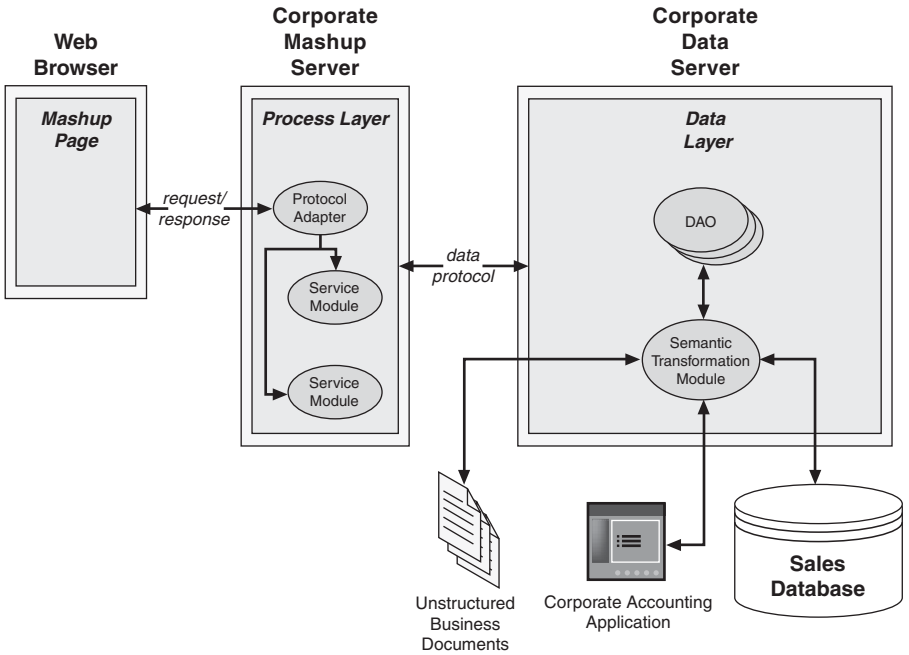

Figure I.6 *High-level view of semantically enabled enterprise mashup infrastructure*

Effective Design Patterns

Software design patterns present tested and proven blueprints for addressing recurring problems or situations that arise in many different design and development scenarios. By defining a design/development solution in terms of a pattern, problems can be solved without the need to rehash the same problem over and over in an attempt to provide a custom solution each time.

Using design patterns for software development is a concept that was borrowed from architecture as it applied to building homes, workplaces, and cities. The idea revolved around the concept that looking at problems abstractly presented common solutions to different architectural problems. The same concept was applied to software engineering and proved to work equally as well.

Design and implementation efforts of a mashup share many of the same development issues as traditional software engineering. Therefore, many of the same techniques and methodologies that provide successful results to traditional software paradigms work equally as well with mashup development. Software patterns are one of the most widely used methodologies in traditional software engineering and are also strongly suggested as a mechanism for addressing mashup design and development scenarios.

Since mashups address many different, dynamic scenarios and technologies, finding any sort of common ground on which to base design and implementation decisions can be a great help to software practitioners.

Mashups are very data-intensive. Therefore patterns that define common solutions to the conversion or adaptation of different data formats offer a substantial benefit to developers. A pattern defining a common solution for enriching data as the data is transferred from one module to another offers significant benefits, as well.

Mashups seek to provide rich experiences for client-side users. Therefore, patterns defining common solutions applied to AJAX, JavaScript, XML, and CSS can provide benefits to UI developers.

The following is a list of some of the mashup activities for which patterns can offer useful design help:

- Semantic formats and patterns for data access and extraction
- Semantic formats and patterns for data transfer and reuse
- Patterns and methods for data presentation
- Patterns and methods for scheduling and observation
- Content reuse with clipping
- Data/content augmentation patterns for normalizing content
- Patterns and purposes for notifications and alerts

With many of the processes in a mashup running externally in the Internet cloud, it is extremely desirable to find common patterns that address issues such as scalability, security, and manageability within this nebulous environment.

Unique Security Constraints

A mashup development model is very open by definition. This openness introduces many new security risks; therefore, security must be a primary concern when developing a mashup infrastructure.

Traditional mechanisms such as firewalls and DMZs are not sufficient for the granularity of access that mashups require for UI artifacts and data. The mashup infrastructure itself must be prepared to deal with issues such as cross-site request forgery (CSRF), AJAX security weaknesses, cross-site scripting, and secure sign-on across multiple domains.

The fact that a mashup is a page or application built typically using data combined from more than one site, illustrates the manner in which security vulnerabilities can multiply quickly. As new invocations are added to access resources or to call service API, new security vulnerabilities become possible. In addition, external mashups can embed your components and UI artifacts, thereby combining your functionality and data with components and UI artifacts of unknown origin. These wide-open integration possibilities make it imperative to ensure that your data and functionality are not open to hacker attempts and other forms of intrusion.

The most common attack scenarios within a mashup environment are cross-site scripting, JSON hijacking, denial of service attacks, and cross-site request forgeries.

The intrinsic openness of a mashup environment and the inability to predict exactly how components of a mashup infrastructure will be used in the future implies the need to address security at every aspect of the development lifecycle. Therefore, security must be a primary part of a development team's code review and testing processes.

A mashup environment most likely uses components and UI artifacts developed externally. This means that testing external components must be included in a development team's testing process right alongside an organization's own components. External components should be tested individually and in aggregate with other components of a given mashup.

One of the most important steps for any organization is to institute best practices and mashup security policies based on standards established by industry, government, and compliance groups.

When instituting a security policy, an organization should note the following guidelines:

- Create a thorough security policy.
- Establish a proper authentication and authorization plan.
- Allow for flexibility.
- Employ message-level and transport-level security.
- Implement corporate standards for secure usage patterns.
- Support industry security standards.

Conceptual Layers of an Enterprise Mashup

A mashup infrastructure must expose and support programming entities that can be combined in a mashup. The infrastructure must also address the corresponding issues and solutions for each type of entity. This is modeled as three high-level categories of items: user interface artifacts (presentation), data (resources), and/or application functionality (processes). UI artifacts include such entities as HTML snippets, on-demand JavaScript, web service APIs, RSS feeds, and/or other sundry pieces of data. The implementation style, techniques, and technologies used for each category of mashup items present certain constraints and subtleties.

Content and UI artifacts used to build a mashup are gathered from a number of different sources including third-party sites exposing web service APIs, widgets, and on-demand JavaScript. RSS and Atom feeds are also common places from which mashup content is retrieved. Some tools are now exposing services that will glean content and information from any existing site using screen-scraping techniques.

The three-category architecture of mashup web applications is discussed next.

Presentation Layer

The presentation layer for a mashup can pull from a local service platform, publicly available APIs, RSS data feeds, dynamic JavaScript snippets, widgets, badges, and so on. The presentation layer uses technologies and techniques for viewing disparate data in a unified manner. This unified view integrates UI artifacts representing business documents, geocoded maps, RSS feeds, calendar gadgets, and others.

The presentation layer for an agile and powerful enterprise mashup application depends on a modular and flexible infrastructure. The foundation for an effective enterprise mashup infrastructure is typically structured around a multilayered platform. The layers for the mashup infrastructure can be implemented as interconnected modules that manage service registrations, service unregistrations, and service lifecycles.

Since mashups are based on principles of modularity and service-oriented concepts, a modular technology is warranted that combines aspects of these principles to define a dynamic service deployment framework facilitating remote management.

Data Layer

UI artifacts and processes for a mashup infrastructure rely on content and data from multiple sources. Content and data are modeled as resources. Resources can be retrieved using a REST (Representational State Transfer)-based invocation model. In other words, resources are created, retrieved, updated, and deleted using a simple syntax that relies on URIs to define the location of each resource.

A mashup infrastructure should provide a mashup data layer that can access data from multiple sources perhaps using a REST-based invocation model. The resources can then be serialized to a mashup application or page in different semantic formats.

The data layer for a mashup infrastructure combines data in one of two ways: client-side data integration or server-side data integration, as discussed next.

Client-Side Data Integration

In client-side data integration, data is retrieved from multiple sites and mixed together in a client-side application pane or web page typically using scripting techniques and languages such as JavaScript, AJAX, and DOM manipulation. In this type of mashup, data is returned from a site or server in the form of XML, RSS, JSON, Atom, and so on. Much of the data returned originates from data-oriented services sometimes referred to as Data-as-a-Service (DaaS). DaaS describes a data-oriented service API that can be called without relying on third-party processes or components between the service provider and the service consumer.

Figure I.7 illustrates data being integrated on the client in a mashup infrastructure.

Server-Side Data Integration

In server-side data integration, data is retrieved and mixed at the server using technologies such as Java, Python, Perl, and C++, among others. In this style, mashup data is retrieved from one or more sites/servers and used as values or configuration settings to create new data models within a server-side process.

Figure I.8 illustrates data being integrated on the server in a mashup infrastructure.

Process Layer

Processes in the mashup infrastructure can be encapsulated as independent services. Each service can be defined and deployed within the context of a module

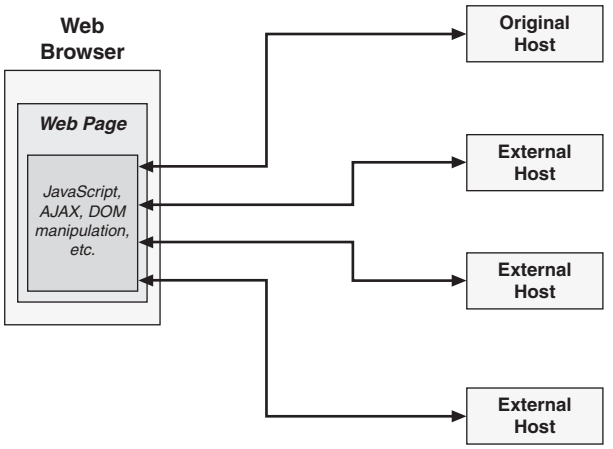

Figure I.7 *Client-side data integration*

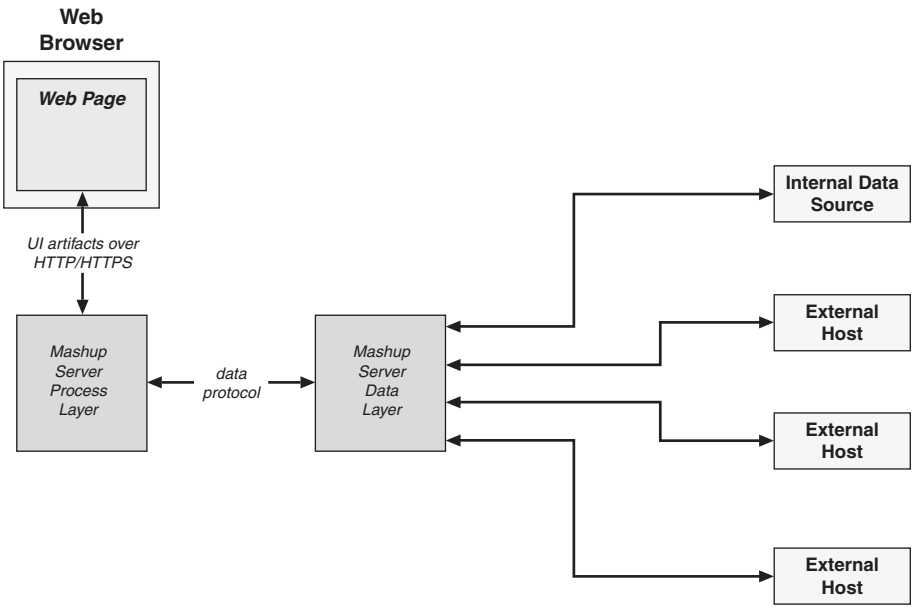

Figure I.8 *Server-side data integration*

managed by a service container. Service modules might consist of one or more services that are deployed automatically to the service container.

The process layer will combine functionality together in one or more aggregate processes using programming languages such as Java, PHP, Python, C++, and so on. Mashups built for enterprise applications or web applications can involve frameworks such as JEE, .NET, and Ruby on Rails.

In the process layer, functionality is combined using interprocess/interthread communication techniques such as shared memory, message queues/buses, and remote procedure calls (RPC), to name a few. The aggregate result of the process layer differs from the data layer in that the data layer derives new data models, whereas the process layer derives new processes and/or services.

Due to the number of disparate sources from which functionality and data are typically retrieved for an enterprise mashup, enterprises often employ a hybrid approach when building a mashup infrastructure.

Using REST Principles for Enterprise Mashups

Interactions within an enterprise mashup often involve the exchange of data from one or more hosts. Data is often referred to as a resource. Therefore, it is important to provide coherent interfaces for resources exposed by an enterprise mashup infrastructure. The resource-oriented architecture defined by Roy Thomas Fielding in his Representational State Transfer (REST) dissertation is a model used by many mashup frameworks and platforms.

REST is a model for interacting with resources using a common, finite set of methods. For the HTTP protocol, this is embodied in the standard methods `GET`, `POST`, `PUT`, `DELETE`, and sometimes `HEAD`. In a REST-based application or interaction, resources are identified by a URI. The response for REST-based invocation is referred to as a representation of the resource.

To help define interfaces for accessing resources in a mashup infrastructure, an understanding of how the interfaces will be accessed is needed. An examination of REST-based interactions across the HTTP request/response space can help to understand the interactions for services and resources in an enterprise mashup.

- **HTTP `GET`**—Retrieves resources identified by URIs (Universal Resource Identifiers), named in a consistent manner for an organization.
- **HTTP `POST`**—Creates a resource identified by the data contained in the request body. Therefore, to create the resource named myfeed.rss, the URI shown for the HTTP `GET` request would be used in an HTTP `POST` request along with the additional `POST` data needed to create the resource.
- **HTTP `PUT`**—Updates the resource identified by the request URI using the data contained in the request body.
- **HTTP `DELETE`**—Deletes the resource that is identified by the request URI.

In a REST model, content and data are modeled as resources. Resources are retrieved using a REST-based invocation model—that is, resources are created, retrieved, updated, and deleted using a simple syntax that relies on URIs to define the location of each resource.

The significance of REST is being made apparent by the vast number of web service API providers promoting REST as the invocation model for their APIs. Many of these same service providers are using semantically rich data formats such as RDF, RSS, and Atom as responses to service invocations.

With REST being supported by so many API providers, mashup infrastructures supporting REST-based invocations and semantic data formats will be highly adaptable to interactions with external hosts and service providers.

Emerging Mashup Standards

Standards for mashup development are just beginning to emerge. In the meantime, standards based on open data exchange, semantically rich content, and open security models are driving mashup implementations.

The following list shows some of the more prominent standards influencing mashup application development:

- **XML (eXtensible Markup Language)**—A general-purpose markup language for representing data that is easy for humans to read and understand. Data formatted as XML is easy to transform into other formats using tools that are available in nearly every programming language. XML supports schema-based validation and is supported by many formal enterprise data-format standards. XML supports internationalization explicitly and is platform and language independent and extensible.
- **XHTML (eXtensible HyperText Markup Language)**—A standard introduced in 2000 to form an integration of XML and HTML. XHTML embodies a web development language with a stricter set of constraints than traditional HTML.
- **OpenSocial API**—A unified API for building social applications with services and artifacts served from multiple sites. The OpenSocial API relies on standard JavaScript and HTML as the platform languages developers can use to create applications and services that interconnect common social connections. OpenSocial is being developed by an extensive community of development partners. This community partnership is leading to a platform that exposes a common framework by which sites can become

socially enabled. Some of the sites currently supporting OpenSocial include iGoogle, Friendster, LinkedIn, MySpace, Ning, Plaxo, Salesforce.com, and others.

- **The Portable Contacts specification—**Targeted at creating a standard, secure way to access address books and contact lists using web technologies. It seeks to do this by specifying an API defining authentication and access rules, along with a schema, and a common access model that a compliant site provides to contact list consumers. The Portable Contacts specification defines a language neutral and platform neutral protocol whereby contact list consumers can query contact lists, address books, profiles, and so on from providers. The protocol defined by the specification outlines constraints and requirements for consumers and providers of the specification. The specification enables a model that can be used to create an abstraction of almost any group of online friends or user profiles that can then be presented to consumers as a contact list formatted as XML or JSON data.

- **OpenSAM (Open Simple Application Mashups)—**A set of open, best practices and techniques for integrating software as a service (SaaS) applications into applications to enable simple connectivity between platforms and applications. OpenSAM is supported by a number of high-profile web application leaders such as EditGrid, Preezo, Jotlet, Caspio, and others.

- **Microformats—**An approach to formatting snippets of HTML and XHTML data to create standards for representing artifacts such as calendar events, tags, and people semantically in browser pages.

- **Data portability—**Introduced in 2007 by a concerted group of engineers and vendors to promote and enable the ability to share and manipulate data between heterogeneous systems. Data in this context refers to videos, photos, identity documents, and other forms of personal data.

- **RSS and Atom—**XML-based data formats for representing web feeds such as blogs and podcasts. RSS and Atom are ideally suited for representing data that can be categorized and described using channels, titles, items, and resource links. An RSS or Atom document contains descriptive information about a feed such as a summary, description, author, published date, and other items.

- **OPML (Outline Processor Markup Language)—**An XML dialect for semantically defining generic outlines. The most common use for OPML at this time is for exchanging lists of web feeds between feed-aggregator

services and applications. OPML defines an outline as a simple, hierarchical list of elements.

- **APML (Attention Profiling Markup Language)—**Seeks to facilitate the ability to share personal attention profiles so that interests might be easily shared between users. An Attention Profile is a type of inventory list of the topics and sources in which a user is interested. Each topic and/or source in the profile contains a value representing the user's level of interest. APML is represented as an XML document containing implicit interests, explicit interests, source rankings, and author rankings.

- **RDF (Resource Description Framework)—**A standard built on the notion that all resources are to be referenced using URIs. RDF also attempts to promote semantic meaning to data. This idea is central to the mashup environment, where data is a collection of loosely coupled resources. With respect to this knowledge, RDF makes a great fit as a universal data model for the data layer of your mashup infrastructure. RDF describes data as a graph of semantically related sets of resources. RDF describes data as subject-predicate-object triples, where a resource is the subject and the object shares some relation to the subject. The predicate uses properties in the form of links to describe the relationship between the subject and object. This interconnecting network of resources and links forms the graph of data that RDF seeks to define.

- **JSON (JavaScript Object Notation)—**A JavaScript data format that offers the advantage of easy accessibility and parsing from within a JavaScript environment. JSON supports a limited number of simple primitive types allowing complex data structures to be represented and consumed easily from standard programming languages.

- **OpenID—**A free service that allows users to access multiple secured sites with a single identity. Sites enabled to use OpenID present a form to a user where the user can enter a previously registered OpenID identifier such as jdoe.ids.example.com. The login form information is passed on to an OpenID client library where it is used to access the web page designated by the OpenID identifier—in this case, http://jdoe.ids.example.com. An HTML link tag containing a URL to the OpenID provider service is read from the web page. The site hosting the client library then establishes a shared secret with the OpenID provider service. The user is then prompted to enter a password or other credentials. The client library site then validates the credentials with the OpenID provider service using the shared secret.

- **OAuth—**A protocol for handling secure API authentication by invoking service invocations on behalf of users. OAuth-enabled sites direct users and associated OAuth request tokens to authorization URLs where the users log in and approve requests from the OAuth-enabled sites. OAuth uses a key, such as an OpenID identifier to enable authentication without passing around usernames and passwords.
- **WS-Security—**Specifies extensions to SOAP messaging to ensure message content integrity and message confidentiality using a variety of security models such as PKI, SSL, and Kerberos. The WS-Security specification is the result of work by the WSS Technical Committee to standardize the Web Service Security (WS-Security) Version 1.0 recommendation. The WS-Security specification defines message integrity, message confidentiality, and the ability to send security tokens as part of a message. These definitions are to be in combination with other web service standards, specifications, and protocols to support a variety of security models and technologies.

Solving Business Problems

Mashups are beginning to play a big role in the business domain. Some of the most prominent uses for mashups within the context of a business are emerging in the space of business process management and IT asset management. The reason for this lies in the ease in which mashups can be created along with the reduced investment needed to adopt the technology.

The nature of mashup development is to use existing technologies such as JavaScript, HTML, XML, and others. This reduces or eliminates the need for large investments in IT retooling and makes a significant difference on the bottom line of organizational staffing, especially in terms of the money saved on integration projects.

Since mashups can retrieve data and UI artifacts from multiple sources, they help to reduce the workload shouldered by a single organization. Mashups promote reuse by definition; therefore, they also reduce the workload for an organization by enforcing reuse within the organization.

Mashup efforts across the enterprise are creating applications and services that complement existing business activities such as business process management (BPM), IT asset management, enterprise information services (EIS), and software as a service (SaaS). How mashups complement business activities can be seen in the following trends:

- **BPM—**Mashups enable business experts to organize workflow and process management activities without relying on highly skilled IT resources, thereby allowing workflows to be modified as needed to meet business requirements.
- **EIS—**Properly architected mashup infrastructures provide semantically rich data layers that allow disparate data to be integrated from multiple sources with little or no help from skilled IT staff. In addition, intuitive user interfaces can be provided using mashup technologies to further simplify the complexities of data integration.
- **IT asset management—**Mashups enable business users with the power to wire UI artifacts, processes, and data together in a graphical manner. Furthermore, mashups allow the creation of applications using components linked to low-level IT functionality. IT asset management is exploiting this model to provide IT asset management components exposing such functionality as time-series data, resource monitoring, and device deployment, to name a few.
- **SaaS—**Mashups flourish in service-oriented and resource-oriented environments. As more businesses move towards a mashup model, on-demand services will become second nature and ubiquitous across the enterprise. This will lead tool vendors to drive SaaS as a prominent model for distributing value.

A mashup environment promotes reuse of existing data, UI artifacts, and software processes to create new applications and services, which might also be reused as components for other mashups. This model is creating a revolutionary atmosphere where underlying programming frameworks and languages are irrelevant, and higher-level scripting languages, semantics, and UI components are emerging as drivers for creating new application functionality.

Enterprises are exploiting the mashup revolution in many different ways to drive down the cost of IT resources and to increase the time-to-market for new business services.

Summary

Mashups allow rapid implementations of new functionality via the use of semantic web technologies, reusable user interface artifacts, and loosely coupled services. Mashups are used in many different consumer and social spaces.

However, enterprises are beginning to reap the benefits afforded by a mashup environment. Mashups are creating an extremely agile and dynamic design and implementation environment within the enterprise realm allowing users with limited technical skills to develop powerful and useful applications and services.

Mashups enable users to create new user interfaces by reusing existing UI artifacts using high-level scripting languages such as HTML and JavaScript. Mashups also enable users to integrate disparate data very quickly and easily using semantically rich data formats that don't require complex programming and middleware technologies. Services and processes are beginning to be integrated with similar speed and ease using loosely coupled techniques inherited from the lessons learned from service-oriented architecture (SOA) solutions.

This introduction discussed some of the high-level concepts surrounding enterprise mashups. In the following chapters I expand on these concepts to guide you through the design and implementation of mashups and mashup infrastructures.

Chapter 1

Mashup Styles, Techniques, and Technologies

To begin design work on a mashup, you must determine what is to be "mashed" together. Three high-level categories of items can be mashed together—user interface artifacts (presentation), data, and/or application functionality (processes). This might include HTML snippets, on-demand JavaScript to access an external API, web service APIs from one of your corporate servers, RSS feeds, and/or other data to be mixed and mashed within the application or pages. The implementation style, techniques, and technologies used for a given mashup depend on this determination. Once the items are determined, your development team can proceed with applying languages, processes, and methodologies to the application at hand.

In this chapter, I point out some of the most widely used styles, techniques, and technologies to build mashups for each of the three primary categories or items.

Determining the Technological Domain for a Mashup

Along with determining what is to be mashed, you must also determine the sources from which the mashup items will be accessed; the style, technologies, and techniques your staff needs to build the mashup; and what services you need to build or access to retrieve the necessary artifacts for your mashup.

Mashups rely on the ability to mix loosely coupled artifacts within a given technological domain. The requirements of the application determine what artifacts (UI, data, and/or functionality) will be needed to build the mashup.

From a high-level perspective, the technological domain as applied to mashups can be viewed as presentation-oriented, data-oriented, and process-oriented. Different languages, methodologies, and programming techniques apply to each technological domain.

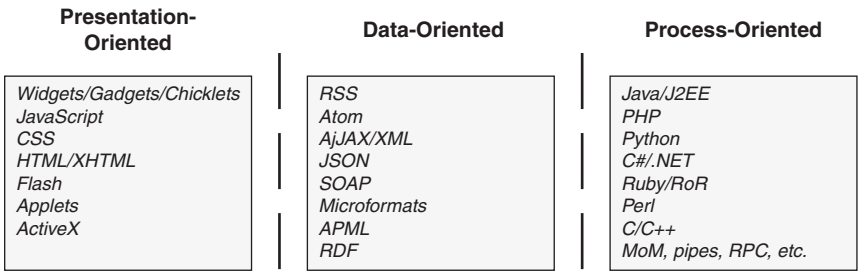

Figure 1.1 *Three primary mashup technological domains*

As shown in Figure 1.1, mashup categories can be divided according to presentation artifacts, data, and application functionality/processes. Each of these categories requires specific skills and programming technologies, as discussed next.

Presentation-Oriented

A presentation-oriented mashup mixes different user interface (UI) artifacts together to create a new application or page. Typically, this type of mashup aims to create an application or page displaying UI artifacts in a manner similar to a conventional portal. That is, each artifact is to be displayed in a separate small area on an application pane or page in a segregated manner in relation to the other artifacts. Very little or no interaction takes place between the UI artifacts in a presentation-oriented mashup.

Technologies used in a presentation-oriented mashup can include

- **Gadgets and widgets—**User interface components that can be placed on an application pane or page independent of other items on the application pane or page. Legacy definitions of widgets referred to them in a more fine-grained manner as applied to desktop components, for example, buttons, scrollbars, sliders, and toolbars. The definition has been expanded in relation to web pages and mashups to include components comprised of more complex and self-contained functionality such as a clock, calculator, weather information, and so on. Gadgets and widgets may or may not retrieve data from an external site.
- **On-demand JavaScript, JavaScript snippets, and badges—**Small sections of JavaScript code that can be inserted within an application pane or page to create user interface components. Typically, the JavaScript relies on interaction with a web service API that returns data and functionality used to build the user interface artifact.

- **CSS/HTML/XHTML—**Snippets that can be inserted to create segregated user interface components that can be reused without regard to the application domain.
- **Flash components/Java applets/ActiveX controls—**Self-contained user interface components that rely on proprietary container technologies such as a virtual machine or runtime that is embedded within the application pane or browser page.

A presentation-oriented mashup is usually the easiest and quickest type of mashup to build. Since there is little or no interaction between the mashed items, the mashup developer can simply worry about placing the items on the application pane or page in the desired location with the desired UI theme.

Data-Oriented

Data-oriented mashups (in-process or out-of-process) involve combining data from one or more externally hosted sites together in an application pane or web page typically using scripting techniques and languages such as JavaScript, JScript, and others. In this type of mashup, data is returned from a site or server in the form of XML, RSS, JSON, Atom, and so on. Much of the data returned originates from data-oriented services sometimes referred to as Data-as-a-Service (DaaS). DaaS describes a data-oriented service API that can be called without relying on third-party processes or components between the service provider and the service consumer.

In-Process Data-Oriented Mashups

In-process data-oriented mashups rely on data being mixed together using application or browser technologies such as JavaScript, AJAX, or the Document Object Model (DOM). In this style of mashup data is retrieved from one or more sites/servers and used as values or configuration settings to build user interface artifacts within an application process or browser page.

Out-of-Process Data-Oriented Mashups

Out-of-process data-oriented mashups rely on data being mixed together using technologies such as Java, Python, Perl, C++, XML, and XSLT, to name a few. In this style of mashup data is retrieved from one or more sites/servers and used as values or configuration settings to create new data models within a server-side process or separate client-side process.

Process-Oriented

Process-oriented mashups (out-of-process) involve combining functionality together in one or more external processes using programming languages such as Java, PHP, Python, and C++. Mashups built for enterprise applications or web applications can involve frameworks such as JEE, .NET, and Ruby on Rails.

In a process-oriented mashup, functionality is combined using interprocess/interthread communication techniques such as shared memory, message queues/buses, remote procedure calls (RPC), and so on. Even though data is exchanged between processes and threads in a process-oriented mashup, the end result differs from a data-oriented mashup in that the data-oriented mashup seeks to derive new data models, whereas a process-oriented mashup seeks to derive new processes and/or services.

More often than not, enterprises require a hybrid approach when building a mashup. This is due to the number of disparate sources from which functionality and data are retrieved such as geocoding sites, RSS feeds, and corporate data stores.

So, when preparing to build a mashup you must first determine what is to be mashed, the technological domain in which to build the mashup, and the sources from which the mashup artifacts will be accessed. With this information in hand you can then determine whether your development staff possess the skills to employ the techniques and technologies needed to build the mashup. The skills of your staff have a big impact on the decision of which mashup style you choose. The following section discusses the reasons for choosing different mashup styles and/or domains.

Choosing a Mashup Style

There are some clear reasons for picking one mashup style/domain over another. Depending on the goal of the mashup, you should weigh the pros and cons of each mashup style before beginning work.

Pros and Cons of Presentation-Oriented Mashups

Presentation-oriented mashups are popular because they are quick and easy to build. They rely primarily on data and UI artifacts retrieved from sites using service APIs, data feeds, and so on. This model is often referred to as a Software-as-a-Service (SaaS) model, although many of the artifacts used in this model are not technically services. Presentation-oriented mashups often require no preauthorization, no installation steps, and no other technologies than those found within any standard web browser.

It is easy to implement presentation-oriented mashups because you can usually use services, components, and script libraries that are publicly accessible without requiring you to install application platforms or tools. In this model, you can simply embed or include the scripting code or service call right in an HTML page as needed. Many publicly available scripting components today allow you to customize the look-and-feel of the UI artifacts that they generate.

Presentation-oriented mashups typically don't require any service coding or deployment; all service coding is provided by external processes/sites to be used by service consumers at will.

Performance is typically quite responsive with presentation-oriented mashups, since all requests are made directly to a service provider or script provider. This direct-access model eliminates any interactions with an intermediary process through which data is retrieved or functionality is derived. However, this also creates a direct coupling to the service or services that can eventually turn into broken links, partially drawn pages, and/or slowly drawn pages if one or more of the services fail or become nonperforming.

Relying on presentation-oriented techniques and technologies, enterprise mashups can reduce the load that might otherwise be shouldered by one or more corporate servers. Since requests are made directly between a service or data consumer and the provider, the use of presentation-oriented techniques and technologies in at least part of a mashup places the processing burden outside the corporate development domain.

One of the biggest challenges with presentation-oriented mashups is attempting to access services hosted on a site other than the site from which the original page was retrieved. Most standard web browsers enforce a sandbox security model in which a given web page is not allowed to access a site/service located external to the host in which the page originated. This is referred to by a number of names including the server-of-origin policy, the browser security sandbox, the same-origin policy, and the same-domain policy.

The browser security sandbox is in place as an attempt to secure sensitive information and ward off attacks from rogue scripts and components attempting to violate privacy and security. Figure 1.2 illustrates the browser security sandbox.

Many mashups employ the use of AJAX to communicate with a server and retrieve data. AJAX is a technology that uses a standard JavaScript object—`XMLHttpRequest`—to pass data from the JavaScript container (browser) to a web host. AJAX enables web pages to perform in a manner more similar to desktop applications, for example, less page refreshing and use of dynamic data updating. However, this dynamic communication model opens the door to malicious scripts; hence the need for the browser security sandbox. The browser security

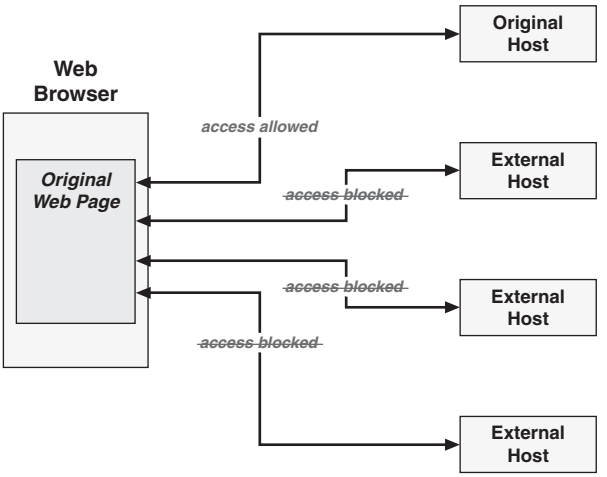

Figure 1.2 *The browser security sandbox*

sandbox only allows JavaScript code to use the `XMLHttpRequest` object to communicate with the host or domain from which the containing page originated. This restriction is a major constraint for building mashups. Therefore, alternative mechanisms are needed if a presentation-oriented mashup is to incorporate data and services from multiple sites. Alternatives have been developed, specifically the use of a technique known as "on-demand scripting" or "dynamic scripting." This technique, which I talk about in detail later, exploits the inclination of a browser to execute JavaScript as it is encountered in the process of interpreting an HTML page.

Pros and Cons of Data-Oriented Mashups

Data-oriented mashups present their own set of challenges and benefits. Primarily, a data-oriented mashup must deal with the intrinsic requirement to act as the data mediator and/or broker. The mashup in this model must take data from multiple sites and mix it together in a form that will be useful to the mashup application or page. As a data broker/integrator, the mashup must deal with multiple data formats and, possibly, communication protocols, such as office document formats, message queue exchange protocols, and HTTP.

In-Process

Mashing data together within a web browser can be a formidable task. Depending on the type of data, browser-based scripts and technologies are not particularly suited for data integration. Most data-mashing techniques performed within a browser involve manipulating XML-based or JSON-based

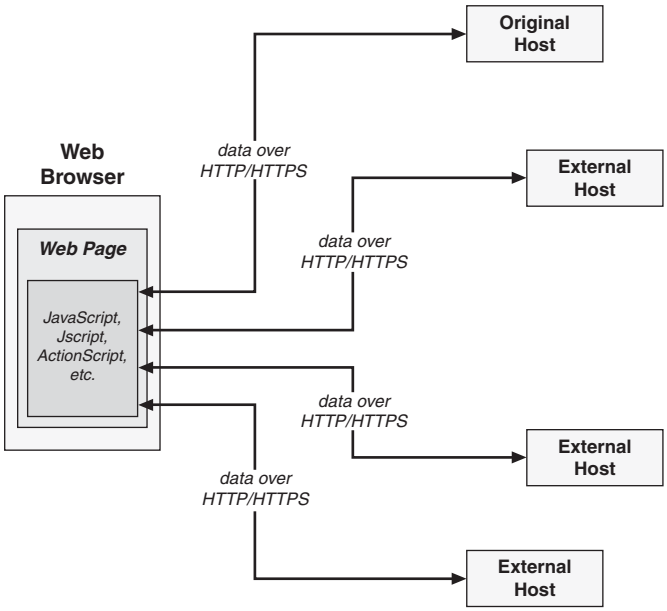

Figure 1.3 *Flow of data in an in-process mashup*

(http://json.org/) data using various JavaScript techniques and applying the results to the web page using DOM manipulation.

Figure 1.3 illustrates the flow of data in an in-process mashup. As illustrated, data is received from multiple sites/hosts and is processed using scripting technologies within a browser page.

Out-of-Process

Mashing data together outside a web browser is nothing new. Any web application or desktop application framework will provide some form of time-tested data integration technology. Frameworks for processing plain text, comma-separated text, XML, relational data, and so forth have been around for decades and optimized to process heterogeneous data very efficiently. New technologies built specifically for disparate data integration, such as enterprise service buses, are also available in this mashup model.

Figure 1.4 illustrates the flow of data in an out-of-process mashup. As illustrated in the figure, a mashup server receives data from multiple sites/hosts and integrates the data using languages, libraries, and frameworks well-suited for data parsing, assembly, mediation, and so on.

Mashing data out-of-process can typically handle many more data formats and apply more robust transformations and data-augmentation techniques than an in-process model.

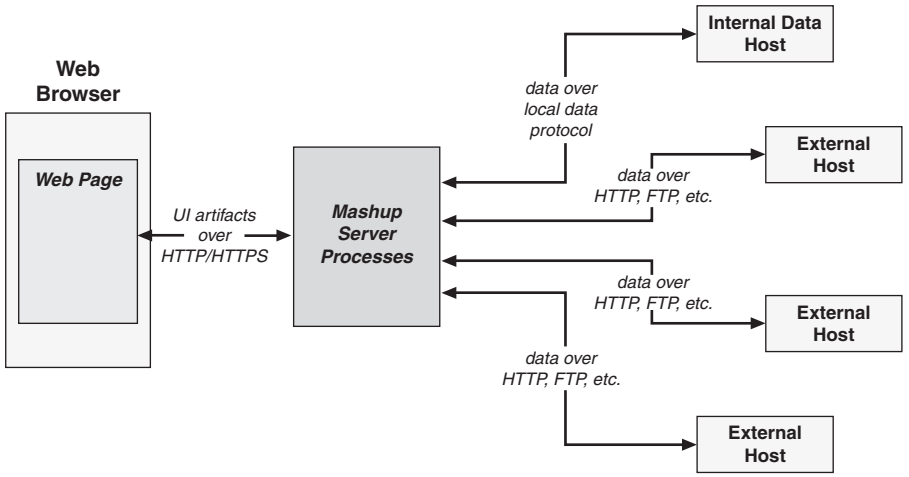

Figure 1.4 *Flow of data in an out-of-process mashup*

Out-of-process data mashing offers the advantage of evaluating the data payload before returning the result to the mashup client. In this model a mashup server can monitor a data feed for just the data updates and return the updates to the client rather than the entire payload.

Data caching at the mashup server and returning results directly to the client rather than invoking the data request on a remote host is also a benefit of mashing data out-of-process.

Pros and Cons of Process-Oriented Mashups

In a process-oriented mashup, service requests are transmitted from a mashup client to the mashup server in the same fashion as an out-of-process data-oriented mashup. So, from the perspective of the mashup client, the process is the same. However, from the perspective of the mashup server, things change.

A process-oriented mashup server deals with integration of data as well as integration of services and processes. Just as data can be retrieved from multiple different internal and external sources in a data-oriented mashup, services and processes can be invoked on multiple different internal and external service hosts and/or processes.

As illustrated in Figure 1.5, a process-oriented mashup server deals with integration of processes and services from many different internal and external processes and hosts. Since this situation exists in most standard web application or server-oriented environments, it is inevitably encountered in most mashup environments where any form of processing takes place outside the mashup client (browser). The pros and cons of this model are also shared with standard ser-

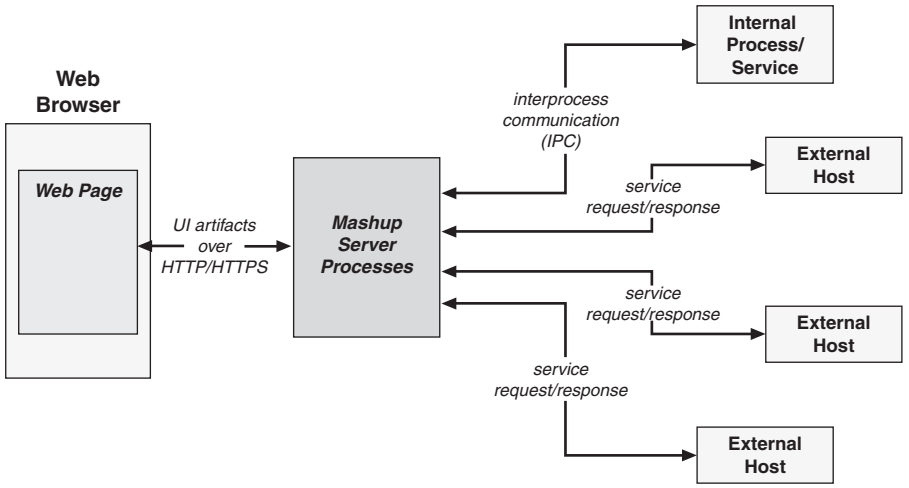

Figure 1.5 *Flow of services and processes in a process-oriented mashup*

vice-oriented server environments, including the intricacies of IPC, transaction management, and service availability.

Process-oriented mashups and out-of-process data-oriented mashups allow you to deal with shared security keys, tokens, credentials, and so on using many currently available technologies and frameworks such as SAML, OpenID, and OAuth. Shared security mechanisms are becoming available more and more from an in-process mashup domain, but server technologies still dominate. Handling data security at the server level also allows you to incorporate data-retrieval and security in the same step resulting in fewer hops from client to server.

Out-of-process, data-oriented mashups and process-oriented mashups allow data and services to be processed asynchronously, often resulting in a more efficient use of processing power and time. Browser-based concurrency is usually limited to far fewer calls than server-based concurrency.

Presentation-Oriented Mashup Techniques

When you work with a mashup in the presentation domain, you are constrained to an environment that has evolved in a hodge-podge way from processing simple text and graphics pages to one that is on the verge of offering as many or more features as a complex desktop application framework. This messy evolution has suffered because of the mismatch between the free-natured needs of the web environment and the restricted nature of the few primary

browser vendors. Nevertheless, standards and best practices are emerging that offer sanity to the mess. The following sections discuss some of the popular techniques and technologies being used to produce mashups in the presentation domain.

Mashing Presentation Artifacts

The easiest form of presentation-oriented mashup involves aggregation of UI artifacts within a web page in a portal-like manner—that is, completely segregated from each other using discrete areas within a single HTML page. In this model, UI artifacts such as gadgets, widgets, HTML snippets, JavaScript includes, and on-demand JavaScript are embedded within an HTML document using layout elements and techniques such as HTML tables and CSS positioning.

When mashing together UI artifacts in a web page using browser layout techniques, each artifact typically resides in its own separate area, as illustrated in Figure 1.6

As illustrated in Figure 1.6, a mashup using aggregated UI artifacts references one or more web sites from a web page and retrieves UI code that builds each artifact as a separate component in a separate area on the browser page.

Mashing Presentation Data

A browser page can also build and modify UI artifacts using data retrieved from multiple sources using such data formats as XML, RSS, Atom, and JSON. In this model, the data is retrieved from one or more sites and parsed by the

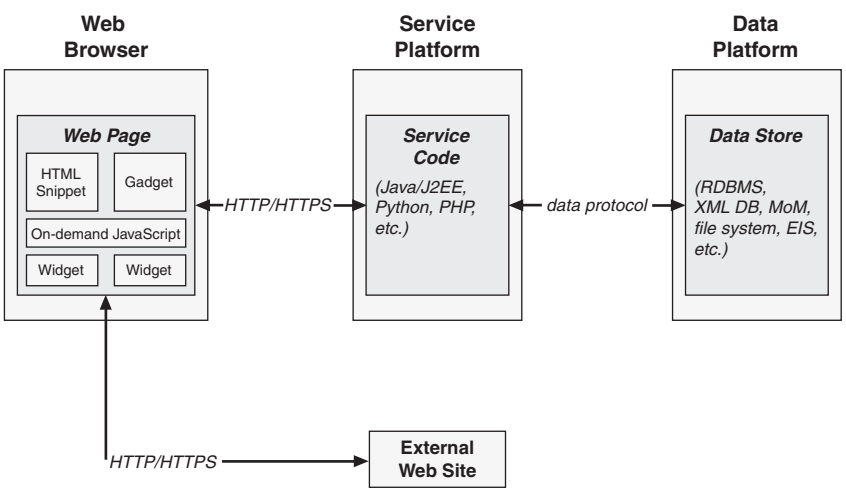

Figure 1.6 *Mashed presentation artifacts*

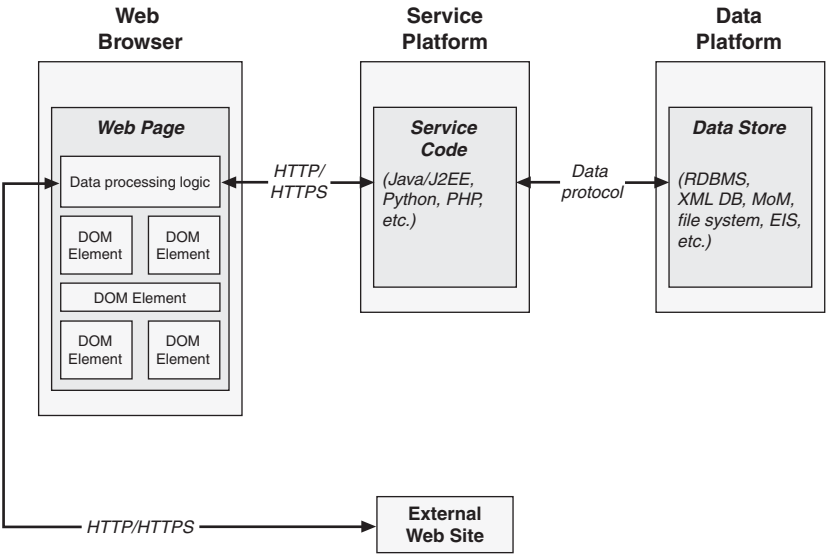

Figure 1.7 *Mashed presentation data*

browser, and UI artifacts are created or updated using scripting techniques such as DOM manipulation to alter the resulting HTML document.

Figure 1.7 illustrates the flow of data from service hosts and external sites to a mashup created by aggregation of data in the presentation domain (browser page).

This model is more complex than mashing together UI artifacts. In this model, the scripting code that processes the data must be robust enough to handle multiple data formats or restrict the page to accessing services that only support the formats supported by the page. However, since the scripting code will be parsing the data to a fine-grained level, the UI can be created and updated to create a more sophisticated user experience. This model offers more flexibility at the cost of additional complexity.

Using AJAX and the XMLHttpRequest Object

Asynchronous JavaScript and XML (AJAX) is a set of technologies and techniques used for handling data feeds using JavaScript and for creating dynamic web pages with a sophisticated look and feel. One feature of AJAX is the use of JavaScript and CSS techniques to update a UI artifact on a web page without refreshing the entire page. AJAX also features a JavaScript-based HTTP request/response framework that can be used within a web page.

The HTTP request/response framework provided by AJAX is enabled by a component known as the `XMLHttpRequest` object. This object is supported by most

browsers and offers the ability to pass data to an external host and receive data from an external host using the HTTP protocol. Requests can be made synchronously or asynchronously. Asynchronous AJAX requests are made in the background without affecting the web page from which the request is made.

The AJAX framework can send and receive virtually any data format. However, XML-based data and JSON data are the most frequent payloads used for various reasons discussed elsewhere in this chapter. Once data is received it is parsed and applied to the page, typically by manipulating the DOM.

Figure 1.8 illustrates the process through which data flows within a web page using the AJAX framework. As illustrated, when using AJAX, data is received by the `XMLHttpRequest` object. UI artifacts can then be created and modified using the data.

Document Object Model (DOM)

Every JavaScript-enabled web page is represented internally to a browser as an instance of the W3C Document Object Model (DOM). DOM is a platform-independent and language-neutral object model for representing XML-based documents that allows programs and scripts to dynamically access and update the content, structure, and style of a document.

The HTML DOM is part of the official DOM specification that defines a standard model for HTML documents. The HTML DOM facilitates accessing and manipulating HTML elements in a given web page. The HTML DOM presents web page as a node-based tree structure containing elements, attributes,

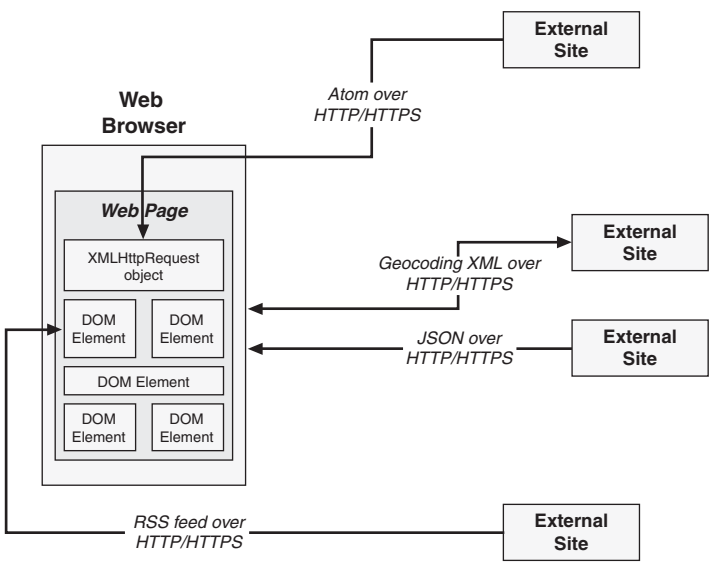

Figure 1.8 *Presentation data mashup using AJAX*

and text. Every element on a web page (for example, div, table, image, or paragraph) is accessible as a DOM node. JavaScript allows the manipulation of any DOM element on a page dynamically. This allows you to perform such operations as hiding elements, adding or removing elements, and altering their attributes (color, size, position, and so on).

Listing 1.1 presents a simple HTML document.

Listing 1.1 *Simple HTML Document*

```
<html>
<head>
<title>A simple HTML doc</title>
</head>
<body>
   <p>
   This is a simple HTML document.
   </p>

   <img id="image1"
        src="http://example.com/image1.png"
        width="250"
        height="350"/>
</body>
</html>
```

Listing 1.2 is an example of a JavaScript function that changes the width and height of the image (`<img>`) element with an id of `image1` in the preceding example:

Listing 1.2 *JavaScript Manipulation of DOM*

```
function changeImageSize()
{
   var anIMGElement = document.getElementById("image1");
   anIMGElement.width = "400";
   anIMGElement.height = "300";
}
```

As shown in Listing 1.2, the DOM can be accessed by the global "document" variable. With a reference to the document variable, you can traverse the nodes of the DOM to find any element by name or id.

Extensible Markup Language (XML)

XML (eXtensible Markup Language) is a specification and standard for creating self-describing markup languages. It is extensible in that it allows you to define your own elements. It is used extensively in data transformation and integration frameworks to facilitate the transfer and integration of structured data across disparate systems and applications. XML is used in many web-enabled environments as a document annotation standard and as a data serialization format.

Listing 1.3 is an example of a simple XML document.

Listing 1.3 *Simple XML Document*

```
<?xml version="1.0" encoding="UTF-8"?>
<contacts>
   <contact>
      <name>John Doe</name>
      <address1>123 anywhere st.</address1>
      <address2>Apt 456</address2>
      <city>Yourtown</city>
      <state>CA</state>
      <zip>12345</zip>
      <country>USA</country>
   </contact>
   <contact>
      <name>Jane Doe</name>
      <address1>456 S 789 W</address1>
      <address2>Suite 987</address2>
      <city>Mytown</city>
      <state>NY</state>
      <zip>54321</zip>
      <country>USA</country>
   </contact>
</contacts>
```

Presentation-oriented mashups consume XML and XML derivatives returned from service hosts and web sites. Once XML is received, it is parsed and applied to a given web page. As XML is parsed, DOM manipulation techniques are usually applied to update UI artifacts.

JavaScript Object Notation (JSON)

JSON (JavaScript Object Notation) is a simple, string-based, data-interchange format derived from JavaScript object literals. JSON is very easy for users to read and write and for JavaScript engines to parse. Strings, numbers, Booleans, arrays, and objects can all be represented using string literals in a JSON object.

Listing 1.4 is an example of a simple JSON object:

Listing 1.4 *JavaScript Object Notation (JSON) Object*

```
{
  'contacts':
  [{
    'name':'John Doe',
    'address1':'123 anywhere st.',
    'address2':'Apt 456',
    'city':'Yourtown',
    'state':'CA',
```

```
    'zip':'12345',
    'country':'USA'
  },
  {
    'name':'Jane Doe',
    'address1':'456 S 789 W',
    'address2':'Suite 987',
    'city':'Mytown',
    'state':'NY',
    'zip':'54321',
    'country':'USA'
  }]
}
```

Presentation-oriented mashups also consume JSON objects returned from service hosts and web sites. Once a JSON object is received, it must be parsed in order to apply to a given web page. Since JSON is pure JavaScript, it is easily parsed using standard JavaScript utilities. As with XML, DOM manipulation techniques are usually applied to update UI artifacts once a JSON object is parsed.

Sidestepping the Browser Security Sandbox

Perhaps the biggest challenge in doing a presentation-oriented mashup is contending with the browser security sandbox, which is in place to keep sensitive information secure. To protect against malicious scripts, most browsers only allow JavaScript to communicate with the host/server from which the page was loaded. If a mashup requires access to a service from an external host, there is no easy way to access it using the `XMLHttpRequest` object.

When an attempt is made to access a host/server external to the host/server from which a web page was loaded, an error similar to the following will be encountered:

```
Error: uncaught exception: Permission denied to call method XMLHttpRequest.open.
```

Therefore, a mechanism is needed through which services can be accessed without violating the browser security sandbox. JSONP provides one solution.

JSON with padding (JSONP) or remote JSON is an extension of JSON where the name of a JavaScript callback function is specified as an input parameter of a service call. JSONP allows for retrieving data from external hosts/servers. The technique used by JSONP is referred to as dynamic or on-demand scripting. Using this technique, you can communicate with any domain and in this way avoid the constraints of the browser security sandbox.

Listing 1.5 is an example of using on-demand JavaScript to retrieve data from an external site.

Listing 1.5 *On-Demand JavaScript*

```
function retrieveExternalData()
{
  var script = document.createElement("script");
  script.src =
    'http://www.example.com/aservice?output=json&
     callback=aJSFunction';
  script.type = 'text/javascript';
  document.body.appendChild(script);
}
```

Listing 1.5 illustrates how to dynamically add a <script> tag into a page by manipulating the DOM so that the page can load and call another web site. The <script> tag executes on-demand and makes the service request to the site specified. If the service response is in JSONP format the JavaScript interpreter converts the response object into a JavaScript object. When a script element is inserted into the DOM the JavaScript interpreter automatically evaluates the script. JSONP responses wrap JSON data in a JavaScript function call using the name of the callback parameter. The JavaScript function is then called and any objects defined in the JSONP response are passed.

The following is an example of an evaluated JSONP response:

```
aJSFunction({ "item 1":"value 1", "item 2":"value 2" });
```

As shown in the preceding line of code, the response returned from the service is formatted as a JSON object wrapped in a JavaScript function call with the callback parameter name. The script is evaluated and the JavaScript function is called, completing the service request/response interaction.

Data-Oriented Mashup Techniques

Data can be mashed together in-process or out-of-process. These two domains typically equate with a web browser and a remote server application, respectively. Many times data will be mashed in a hybrid approach using both in-process and out-of-process techniques.

This section discusses in depth some of the techniques used for both in-process and out-of-process data mashups.

Mashing Data In-Process

Mashing data in-process involves applying data integration techniques in the same process as the mashup page. This is typically accomplished with scripting

code such as JavaScript and JScript. However, proprietary component technologies such as Java applets and ActionScript can be used.

During the process of mashing data in this model, a request is made to a service and data is returned in the service response. The data is then parsed, processed, and applied to UI artifacts in the page. DOM manipulation is typically used to apply the processed data.

Mashing XML Data In-Process

All standard web browsers expose an XML-parser object to JavaScript that can be used to load and parse XML data. Each parser reads XML data from a string; therefore, a string response returned from a service call can be passed to the XML parser and processed as needed.

In Listing 1.6, an XML parser is created and used to parse the XML document string defined previously in Listing 1.3. As each contact item is encountered, a new paragraph element is created using DOM manipulation, and the element is added to the web page.

Listing 1.6 *Parsing XML Using JavaScript*

```
<script>
function parseXMLData(xmlString)
{
  var xmlDoc;

  if (document.implementation.createDocument)
  {
    // Create the Mozilla DOM parser
    var domParser = new DOMParser();
    // Create the XML document object
    xmlDoc = domParser.parseFromString(xmlString, "text/xml");
  }
  else if (window.ActiveXObject)
  {
    // Create the Microsoft DOM parser
    xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async = "false";
    // Create the XML document object
    xmlDoc.loadXML(xmlString);
  }

  // get root node
  var contactsNode = xmlDoc.getElementsByTagName('contacts')[0];

  // traverse the tree
  for (var i = 0; i < contactsNode.childNodes.length; i++)
```

```
  {
    var contactNode = contactsNode.childNodes.item(i);
    for (j = 0; j < contactNode.childNodes.length; j++)
    {
      var itemNode = contactNode.childNodes.item(j);
      if (itemNode.childNodes.length > 0)
      {
        var itemTextNode = itemNode.childNodes.item(0);
        var paraEl = document.createElement("p");
        var textEl =
          document.createTextNode(itemNode.nodeName + ":"
                                   + itemTextNode.data);
        paraEl.appendChild(textEl);
        var bodyEl =
          document.getElementsByTagName("body").item(0);
        bodyEl.appendChild(paraEl);
      }
    }
  }
}
</script>
```

In the preceding listing a DOM parser is instantiated and is accessed to get the root node. Each node is then traversed via its child nodes until the desired text data node is found. Note that there is a different parser used for Microsoft Internet Explorer and Mozilla.

Mashing JSON Data In-Process

Since JSON is pure JavaScript, a string containing JSON data can simply be evaluated to create a JavaScript object. Then the JavaScript object can be accessed using normal JavaScript. For example, suppose that you are working with a string containing the JSON data shown previously in Listing 1.4. You can pass that string to the JavaScript `eval` function as follows:

```
var jsonObj = eval("(" + jsonString + ")");
```

This creates a JavaScript object on which we can operate using standard JavaScript techniques. For example, you can access the name field for the first contact using the following snippet:

```
jsonObj.contacts[0].name
```

Typically, you know the structure of the data beforehand. However, you might not know the length of array data within the structure. In that case, the JSON object must be traverse and array data parsed dynamically.

In Listing 1.7, a string containing JSON data, as defined previously in Listing 1.4, is evaluated and parsed. As each array element is encountered, a new para-

graph element is created using DOM manipulation and the element is added to the web page:

Listing 1.7 *Processing JSON Using JavaScript*

```
function parseJSONData(jsonString)
{
  var jsonObj = eval("(" + jsonString + ")");

  for (var x in jsonObj)
  {
    // ignore properties  inherited from object
    if (jsonObj.hasOwnProperty(x))
    {
      if (jsonObj[x] instanceof Array)
      {
        // handle arrays
        for (var i = 0; i < jsonObj[x].length; i++)
        {
          var bodyEl =
            document.getElementsByTagName("body").item(0);

          // create name element
          var paraEl = document.createElement("p");
          var textEl = document.createTextNode("Name: "
                                        + jsonObj[x][i].name);
          paraEl.appendChild(textEl);
          bodyEl.appendChild(paraEl);

          // create address 1 element
          paraEl = document.createElement("p");
          textEl = document.createTextNode("Address 1: "
                                    + jsonObj[x][i].address1);
          paraEl.appendChild(textEl);
          bodyEl.appendChild(paraEl);

          // create address 2 element
          paraEl = document.createElement("p");
          textEl = document.createTextNode("Address 2: "
                                    + jsonObj[x][i].address2);
          paraEl.appendChild(textEl);
          bodyEl.appendChild(paraEl);

          // create city element
          paraEl = document.createElement("p");
          textEl = document.createTextNode("City: "
                                    + jsonObj[x][i].city);
          paraEl.appendChild(textEl);
          bodyEl.appendChild(paraEl);
```

```
            // create state element
            paraEl = document.createElement("p");
            textEl = document.createTextNode("State: "
                                    + jsonObj[x][i].state);
            paraEl.appendChild(textEl);
            bodyEl.appendChild(paraEl);

            // create zip element
            paraEl = document.createElement("p");
            textEl = document.createTextNode("Zip: "
                                    + jsonObj[x][i].zip);
            paraEl.appendChild(textEl);
            bodyEl.appendChild(paraEl);

            // create country element
            paraEl = document.createElement("p");
            textEl = document.createTextNode("Country: "
                                   + jsonObj[x][i].country);
            paraEl.appendChild(textEl);
            bodyEl.appendChild(paraEl);
          }
        }
      }
    }
}
```

With the ability to parse data returned from a service and to manipulate elements of the HTML DOM with the parsed data, you can dynamically create and modify UI artifacts to fit the needs of your mashup.

Mashing Data Out-of-Process

Mashing data out-of process involves applying data integration techniques in a separate process, typically on a remote host/server. In this mashup model, a remote software module receives requests from a client and takes the necessary steps to gather and transform the data needed to formulate a response.

Technologies and techniques in this approach overlap with enterprise data integration technologies and techniques, the scope of which is beyond this discussion. However, the following presents a high-level view of some of the more common approaches to enterprise data transformation and integration currently in use:

- **Brute-force data conversion—**This technique involves converting one data format to another using proprietary conversion tools or a custom byte-for-byte conversion program. Proprietary applications often offer an extension framework allowing third parties to build components or plug-ins that will convert one data format to another.

- **Data mapping—**Data mapping involves the creation and application of a map of data elements between two disparate data models—source and destination. The map is used by conversion programs to determine how an element from the source dataset applies to the destination dataset. Extensible Stylesheet Language Transformations (XSLT) is often used in this approach to convert XML data from one form to another.
- **Semantic mapping—**This approach uses a metadata registry containing synonyms for data elements that can be queried by conversion tools that use the synonyms as a guide for converting one data element to another.

Process-Oriented Mashup Techniques

A process-oriented mashup involves mashing together services and/or processes. Techniques used for this model range from simply combining method calls in an object to a complex, structured workflow system.

In Figure 1.9, object2 is interacting with a number of different services and processes. It is interacting with object1 using a standard method call and an external site using a service call over a web protocol. object2 is also interacting with an internal workflow system using asynchronous messaging. The results from these calls are then mashed together to formulate the response that will ultimately be returned to the mashup client (web page).

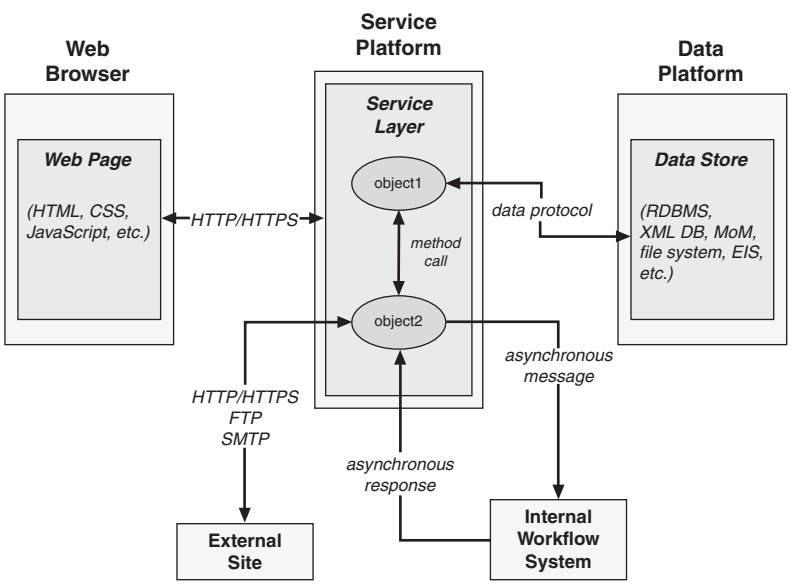

Figure 1.9 *Process-oriented mashup architecture*

Hybrid Mashups

In actuality, enterprise mashups usually involve techniques and technologies for each of the three mashup domains. Widgets, gadgets, or dynamic scripts will be retrieved using presentation-oriented techniques. Data will be retrieved using in-process data-oriented techniques. More data will be retrieved on the server using out-of-process techniques. Services and processes will be aggregated to formulate responses on the server that will be returned to the mashup client results from service API requests.

Figure 1.10 illustrates a typical enterprise mashup environment where data and services are accessed from a web page (in-process) and from the service platform (on the server, out-of-process). Presentation-oriented techniques, data-oriented techniques, and process-oriented techniques must all be employed to handle the needs for this environment.

The techniques and mashup domains discussed in this chapter are discussed further in subsequent chapters. For now, I demonstrate some of the easier techniques of a presentation-oriented mashup in a small example in the next section.

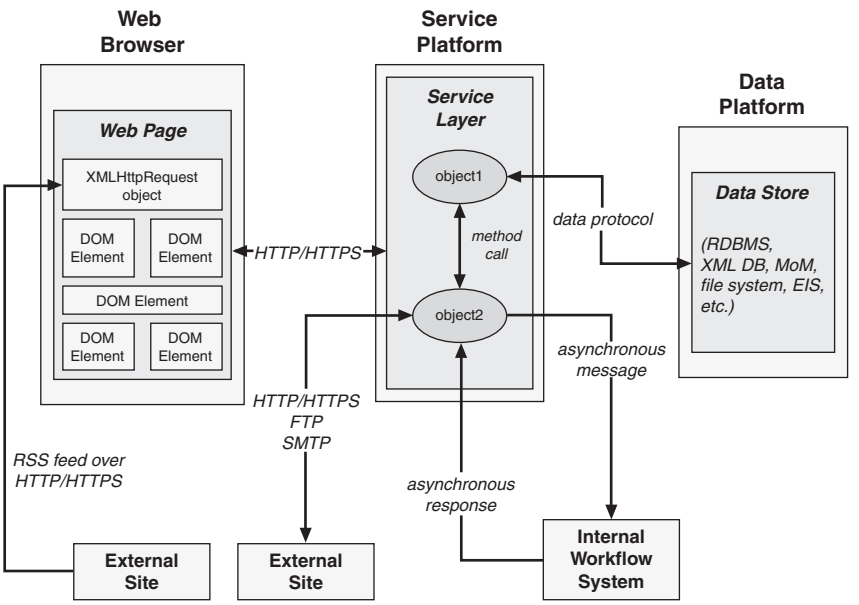

Figure 1.10 *Hybrid mashup architecture*

Implementing a Simple Mashup

This section demonstrates the application of concepts discussed in this chapter to create a simple presentation-oriented mashup. Apart from a local service platform, the sources used for the mashup are publicly available and provide either a service API, an RSS data feed, or a dynamic script feed. This section does not discuss issues such as application keys, security, and governance for this mashup; these topics are discussed in depth in later chapters.

For the sake of complete coverage of the mashup domains discussed, the mashup will make service calls and retrieve data from external sites as well as a local service platform that operates around the model illustrated in Figure 1.11.

Figure 1.11 illustrates a service platform that uses only a small number of primary components to process service and resource requests. The platform uses the Representational State Transfer (REST) approach (to be discussed later) for service and resource request/responses. The service platform provides access to services and uses a simple resource framework to create, retrieve, update, and delete resources.

The application (shown in Listing 1.8 and in Figure 1.12) allows users to view disparate data that might be available in a typical enterprise. The data is presented in a portal-like manner to simplify the layout and UI-management code. The application integrates a list of corporate documents, a map feed, an RSS

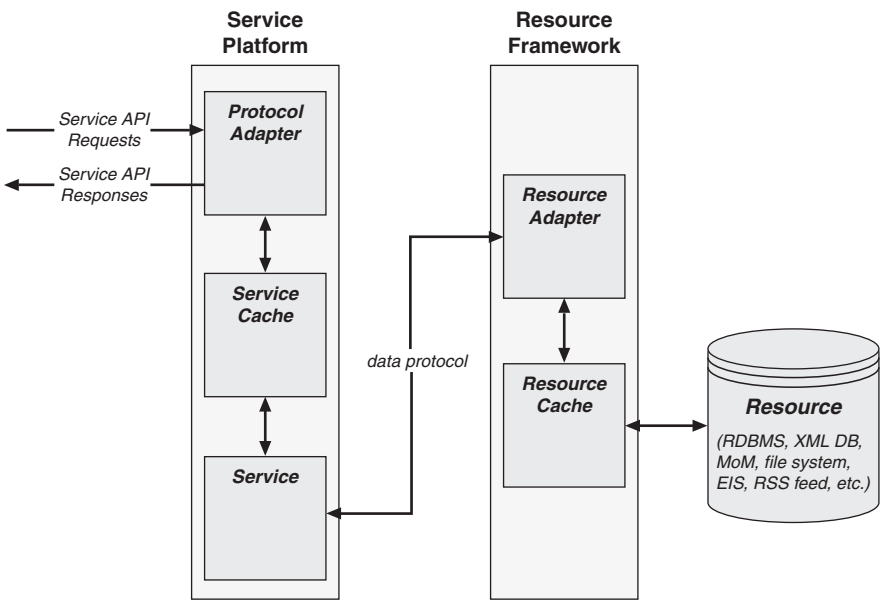

Figure 1.11 *Services platform architecture*

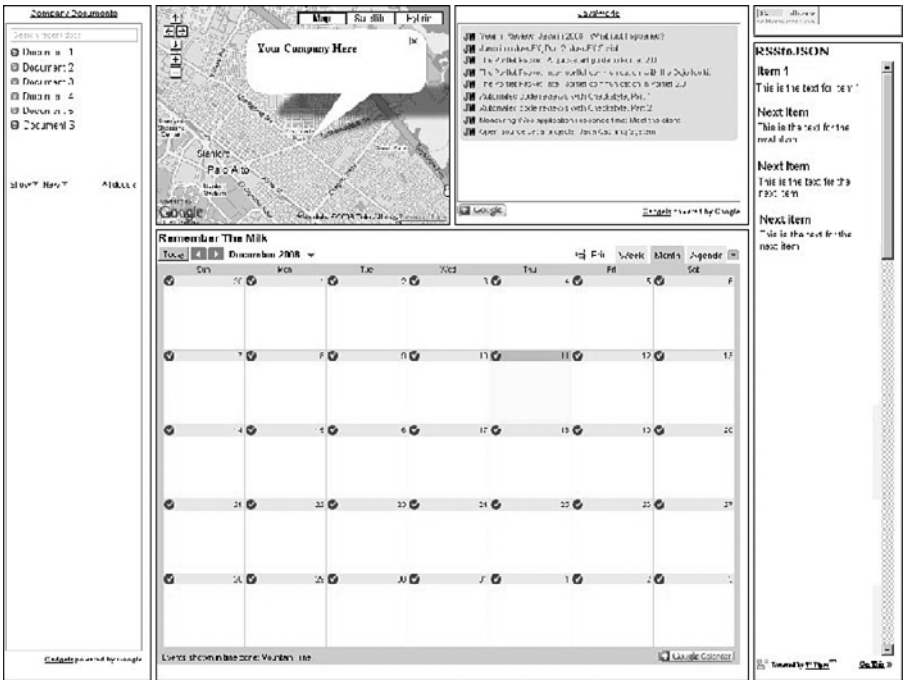

Figure 1.12 *Presentation-oriented mashup example*

feed, a calendar gadget, a Twitter counter chicklet, and a Twitter archive list delivered as RSS.

Listing 1.8 *Presentation-Oriented Mashup Using Aggregated UI Artifacts*

```
<html>
<head>
<!-- Include Google Maps Javascript Library -->
<script type="text/javascript" src="http://maps.google.com/maps?file=api&v=1& key=
ABQIAAAA01HpWF7mf2aW91RNaGDc7xTfGML3OZxtDDthfq-aZ1uFtrk9MRS_VWEizymnfki_h89lqU7A0ts2PA">
</script>

<script type="text/javascript">
```

Figure 1.12 illustrates the final result of the presentation-oriented mashup example discussed in this section.

The load function in Listing 1.9 retrieves a hard-coded Google map and applies it to the map div element in the HTML DOM.

Listing 1.9 *Applying a Google Map to a Web Page*

```
  function load()
  {
    if (GBrowserIsCompatible())
    {
```

```
      // create map component in div with the id = "map"
      var map = new GMap2(document.getElementById("map"));
      // create map components components
      map.addControl(new GSmallMapControl());
      map.addControl(new GMapTypeControl());
      // create center point when map is displayed
      map.setCenter(new GLatLng(37.4419, -122.1419), 13);
      map.openInfoWindow(map.getCenter(),
                         "<b>Your Company Here</b>");
      // re-open the info balloon if they close it
      var point = new GLatLng(37.395746, -121.952234);
      map.addOverlay(createMarker(point, 1));
    }
  }

  function createMarker(point, number)
  {
    var marker = new GMarker(point);
    // create clickable point with title for address
    GEvent.addListener(marker, "click", function()
    {
       marker.openInfoWindowHtml("<b>Your Company Here</b>");
    });
    return marker;
  }

  function retrieveExternalData()
  {
      var script = document.createElement("script");
      script.src =
        'http://www.example.com/mashups/someservice';
      script.type = 'text/javascript';
      document.body.appendChild(script);
  }

</script>
</head>
<body onload="load()">
```

In the example shown in Listing 1.10 a `div` element is added to contain a list of corporate documents retrieved from a Google Docs account using dynamic script.

Listing 1.10 *Dynamic Script to Show a List of Documents*

```
    <!-- div to hold documents -->
    <div id="docs"
         style="border-style:ridge; position: absolute;
            left: 10px; top: 10px; width:200px; height:930px">
    <script src="http://gmodules.com/ig/ifr?
            url=http://www.google.com/ig/modules/docs.xml
```

```
            &amp;up_numDocuments=9&amp;up_showLastEdit=1
            &amp;synd=open&amp;w=180&amp;h=860
            &amp;title=Company+Documents
            &amp;lang=en&amp;country=ALL
            &amp;border=%23ffffff%7C3px%2C1px+solid+%23999999
            &amp;output=js"></script>
    </div>
```

Listing 1.11 illustrates a `div` element that will hold the results from the Google map retrieval.

Listing 1.11 *Element to Contain a Google Map*

```
    <div id="map"
         style="border-style:ridge; position: absolute;
         left: 220px; top: 10px; width:400px; height:300px">
</div>
```

In Listing 1.12 a `div` element is added to contain an RSS feed using dynamic script.

Listing 1.12 *Element to Contain an RSS Feed*

```
    <!-- div to hold RSS feed -->
    <div id="feed"
         style="border-style:ridge; position: absolute;
         left: 630px; top: 10px; width:400px; height:300px">
    <script src="http://gmodules.com/ig/ifr?
      url=http://customrss.googlepages.com/customrss.xml
      &amp;up_rssurl=http%3A%2F%2Fwww.javaworld.com%2Findex.xml
      &amp;up_title=CustomRSS
      &amp;up_titleurl=http%3A%2F%2Fcustomrss.googlepages.com
      &amp;up_num_entries=10&amp;up_linkaction=showdescription
      &amp;up_background=E1E9C3&amp;up_border=CFC58E
      &amp;up_round=1&amp;up_fontfamily=Arial
      &amp;up_fontsize=8pt&amp;up_openfontsize=9pt
      &amp;up_itempadding=3px&amp;up_bullet=icon
      &amp;up_custicon=Overrides%20favicon.ico
      &amp;up_boxicon=1&amp;up_opacity=20
      &amp;up_itemlinkcolor=596F3E&amp;up_itemlinkweight=Normal
      &amp;up_itemlinkdecoration=None&amp;up_vlinkcolor=C7CFA8
      &amp;up_vlinkweight=Normal&amp;up_vlinkdecoration=None
      &amp;up_showdate=1&amp;up_datecolor=9F9F9F
      &amp;up_tcolor=1C57A9&amp;up_thighlight=FFF19D
      &amp;up_desclinkcolor=1B5790&amp;up_color=000000
      &amp;up_dback=FFFFFF&amp;up_dborder=DFCE6F
      &amp;up_desclinkweight=Bold&amp;up_desclinkdecoration=None
      &amp;synd=open&amp;w=380&amp;h=240&amp;title=JavaWorld
      &amp;border=%23ffffff%7C3px%2C1px+solid+%23999999
      &amp;output=js"></script>
    </div>
```

Listing 1.13 shows a div element to contain a Google calendar retrieved using a JavaScript badge.

Listing 1.13 *Google Calendar Element*

```
<!-- div to hold calendar -->
<div id="calendar"
     style="border-style:ridge; position: absolute;
     left: 220px; top: 320px; width:810px; height:620px">
<iframe src="http://www.google.com/calendar/embed?
             src=o78s3eqe3ov403cpuav2bje5ja9j1tp2%40
             import.calendar.google.com&ctz=America/Denver"
  style="border: 0"
  width="800" height="600" frameborder="0" scrolling="no">
  </iframe>
</div>
```

Shown in Listing 1.14 is a div element to contain the Twitter counter chicklet.

Listing 1.14 *A div Element to Contain a Twitter Chicklet*

```
<!-- div to hold the Twitter counter chicklet -->
<div id="chicklet"
     style="border-style:ridge; position: absolute;
     left: 1040px; top: 10px; width:200px; height:40px">
<a href="http://twittercounter.com/?username=jhanson583"
   title="TwitterCounter for @jhanson583"><img src="http://twittercounter.com/
counter/?username=jhanson583"
                                          width="88"
                                          height="26"
                                          style="border:none;"
                                          alt="TwitterCounter
for @jhanson583" /></a>
</div>
```

Listing 1.15 illustrates the div element to contain the Twitter RSS feed using dynamic script.

Listing 1.15 *Twitter RSS Feed div Element*

```
<!-- div to hold twitter feed -->
<div id="ufbadge"
     style="border-style:ridge; position: absolute;
     left: 1040px; top: 60px; width:200px; height:880px">
  <script src="http://pipes.yahoo.com/js/listbadge.js">
         {"pipe_id":"dq0Qhuqp3BG1psChjtzu1g",
          "_btype":"list",
          "pipe_params":{
             "urlRSS":"http:\/\/twitter.com
             \/statuses\/user_timeline\/10852552.rss"},
          "width":"190",
          "height":"870"}
```

```
    </script>
  </div>
```

In Listing 1.16 I add the div element to contain the local RSS feed using dynamic script.

Listing 1.16 *RSS Feed Element*

```
  <!-- div to hold local RSS feed -->
  <div id="localfeed"
       style="border-style:ridge; position: absolute;
       left: 10px; top: 950px; width:1210px; height:200px">
    <script src="http://localhost:8080/mashups/js/rssbadge.js">
           {"urlRSS":"http:\/\/localhost:8080
           \/mashups\/services\/feeds\/zurn.rss"}
</script>
  </div>
</body>
</html>
```

The mashup in the preceding example illustrates many different techniques and technologies as applied to a simple presentation-oriented mashup. In the next chapter I discuss the preparations that need to be made before building an enterprise mashup.

Summary

In this chapter, I discussed some of the styles, techniques, and technologies that are used to build mashups for each of the three primary mashup domains—presentation, data, and process.

I discuss how to determine the domain for your mashup by analyzing the artifacts and data that are to be mashed. The domain is determined by analyzing user interface artifacts (presentation), data, and/or application functionality (processes).

The implementation style, techniques, and technologies used for a given mashup depend on the domain of the mashup determined by the analysis of artifacts and data. The techniques and technologies also depend on where processing will occur—in-process or out-of-process.

Once the domain is determined and the sources for the artifacts and data are established, you can proceed to apply languages, processes, and methodologies to the task of designing and building the mashup.

Index

H

I

J

N

O

P

Q

R

S

T

U