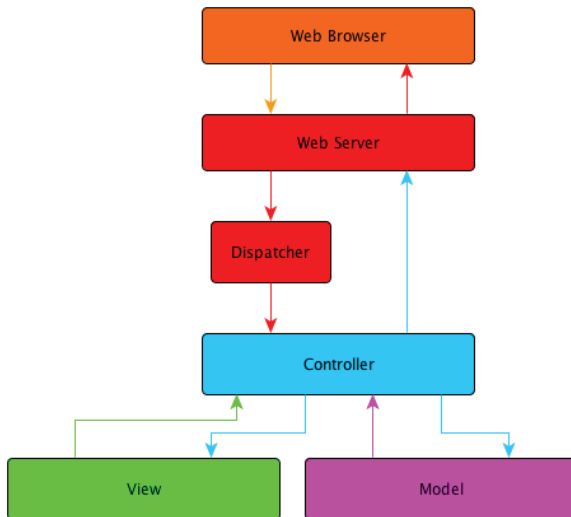


1. METHODOLOGIES

1.1 MODEL VIEW CONTROLLER (MVC)



Model-View-Controller (MVC) is a classic design pattern often used by applications that need the ability to maintain multiple views of the same data. The MVC pattern hinges on a clean separation of objects into one of three categories — models for maintaining data, views for displaying all or a portion of the data, and controllers for handling events that affect the model or view(s).

Because of this separation, multiple views and controllers can interface with the same model. Even new types of views and controllers that never existed before can interface with a model without forcing a change in the model design. (eNode, 2002)

1.2 ORTHOGONALITY

The basic idea of orthogonality is that things that are not related conceptually should not be related in the system. Parts of the architecture that really have nothing to do with the other, such as the database and the UI, should not need to be changed together. A change to one should not cause a change to the other. (Venners, 2003)

Successful use of the pattern isolates business logic from user interface considerations, resulting in

an application where it is easier to modify either the visual appearance of the application or the underlying business rules without affecting the other.

It is common to split an application into separate layers that run on different computers: presentation (UI), domain logic, and data access. In MVC the presentation layer is further separated into view and controller. (Wikipedia, Reenskaug, 1979)

1.3 "THE DRY PRINCIPLE" (DON'T REPEAT YOURSELF)

DRY says that every piece of system knowledge should have one authoritative, unambiguous representation. Every piece of knowledge in the development of something should have a single representation. A system's knowledge is far broader than just its code. It refers to database schemas, test plans, the build system, even documentation.

Given all this knowledge, why should you find one way to represent each feature? The obvious answer is, if you have more than one way to express the same thing, at some point the two or three different representations will most likely fall out of step with each other. Even if they don't, you're guaranteeing yourself the headache of maintaining them in parallel whenever a change occurs. And change will occur. DRY is important if you want flexible and maintainable software. (Venners, 2003)

1.4 AGILE SOFTWARE DEVELOPMENT

Agile Software Development refers to a group of software development methodologies that are based on similar principles. Agile methodologies generally promote: A project management process that encourages frequent inspection and adaptation; a leadership philosophy that encourages team work, self-organization and accountability; a set of engineering best practices that allow for rapid delivery of high-quality software; and a business approach that aligns development with customer needs and company goals. (Wikipedia, Hunt/Thomas, 1999)

1.5 AGILE MANIFESTO ([HTTP://WWW.AGILEMANIFESTO.ORG/](http://www.agilemanifesto.org/))

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.

- Customer collaboration over contract negotiation.
- Responding to change over following a plan, that is, while there is value in the items on the right, we value the items on the left more.
- Customer satisfaction by rapid, continuous delivery of useful software.
- Working software is delivered frequently (weeks rather than months).
- Working software is the principal measure of progress.
- Even late changes in requirements are welcomed.
- Close, daily cooperation between business people and developers.
- Face-to-face conversation is the best form of communication (Co-location).
- Projects are built around motivated individuals, who should be trusted.
- Continuous attention to technical excellence and good design.
- Simplicity.
- Self-organizing teams.
- Regular adaptation to changing circumstances.

2.0 OPEN API'S AND MODELS FOR WEB 2.0 TECHNOLOGIES

The historic trend for web-aware enterprise applications was to use expensive application servers, proprietary systems and cumbersome programming languages such as Microsoft .NET (ASP, VB) and Sun Microsystems Java (J2EE, Struts). With the Web 2.0 revolution creating fully functional, stable, lightweight, cost effective and scalable enterprise applications is possible using open-source frameworks, open standards and widely popular programming languages. Open protocols such as SOAP utilizing messaging patterns such as Remote Procedure Calls (RPC) provide an XML messaging framework for WSDL and other XML datasets are the core foundation of most modern web services applications. As Web 2.0 applications grow and scale it is important that certain development principles are followed and maintained using architectural software models such as Representational State Transfer (REST) for

distributed media and resources, Rapid Development Methodology (RDM) where changes to the system are made available for use without building and deployment cycles, restarts or reloading of key components, Model View Controller (MVC) which isolates business logic from the user interface allowing for (code) changes in one layer without the need to (rewrite code) change other layers and reduction of coupling through the DRY principle and orthogonality which eliminates propagation; modified code or structure in one layer should not cause changes in other layers of the application.

2.1 XML: EXTENSIBLE MARKUP LANGUAGE

XML is a general-purpose specification for creating custom markup languages. DOM is an interface-oriented Application Programming Interface that allows for navigation of the entire document as if it were a tree of "Node" objects representing the document's contents. A DOM document can be created by a parser, or can be generated manually by users (with limitations). Data types in DOM Nodes are abstract; implementations provide their own programming language-specific bindings. Pull parsing treats the document as a series of items, which are read in sequence using the iterated design pattern. This allows for writing of recursive-descent parsers in which the structure of the code performing the parsing mirrors the structure of the XML being parsed, and intermediate parsed results can be used and accessed as local variables within the methods performing the parsing, or passed down (as method parameters) into lower-level methods, or returned (as method return values) to higher-level methods.

Another form of XML Processing API is data binding, where XML data is made available as a custom, strongly typed programming language data structure, in contrast to the interface-oriented DOM. XML supports Unicode, allowing almost any information in any written human language to be communicated. It can represent common computer science data structures: records, lists and trees.

XMLs self-documenting format describes structure and field names as well as specific values. The strict syntax and parsing requirements make the necessary parsing algorithms extremely simple, efficient, and consistent. XML is heavily used as a format for

document storage and processing, both online and offline. It is based on international standards. It can be updated incrementally. It allows validation using schema languages such as XSD and Schematron, which makes effective unit-testing, firewalls, acceptance testing, contractual specification and software construction easier. The hierarchical structure is suitable for most (but not all) types of documents. It is platform-independent, thus relatively immune to changes in technology. Forward and backward compatibility are relatively easy to maintain despite changes in DTD or Schema. (Wikipedia)

2.2 REST (REPRESENTATIONAL STATE TRANSFER)

Representational State Transfer is a style of software architecture for distributed hypermedia systems such as the World Wide Web. REST refers to a collection of network architecture principles, which outline how resources are defined and addressed.

REST provides improved response time and reduced server load due to its support for the caching of representations. It improves server scalability by reducing the need to maintain session state. This means that different servers can be used to handle different requests in a session. It requires less client-side software to be written than other approaches, because a single browser can access any application and any resource. REST depends less on vendor software and mechanisms, which layer additional messaging frameworks on top of HTTP and provides equivalent functionality when compared to alternative approaches to communication. REST does not require a separate resource discovery mechanism, due to the use of hyperlinks in representations.

REST provides better long-term compatibility and evolvability characteristics than RPC. This is due to: The capability of document types such as HTML to evolve without breaking backwards- or forwards-compatibility. The ability of resources to add support for new content types as they are defined without dropping or reducing support for older content types. (Wikipedia, Tilkov, 2007/2008)

2.3 RESTFUL WEB SERVICES CHARACTERISTICS

Characteristics of RESTful Web Services include:

- Client-Server: a pull-based interaction style: consuming components pull representations.
- Stateless: each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- Cache: to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.
- Uniform interface: all resources are accessed with a generic interface (e.g., HTTP GET, POST, PUT, DELETE).
- Named resources - the system is comprised of resources, which are named using a URL.
- Interconnected resource representations - the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.
- Layered components - intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc. (Costello, 2002)

2.4 SOAP

SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the web services protocol stack providing a basic messaging framework upon which abstract layers can be built. Using SOAP over HTTP allows for easier communication through proxies and firewalls than previous remote execution technology. SOAP is versatile enough to allow for the use of different transport protocols. The standard stacks use HTTP as a transport protocol, but other protocols are also usable (e.g., SMTP). SOAP is platform independent and language independent. (Wikipedia)

2.5 WSDL: WEB SERVICE DESCRIPTION LANGUAGE

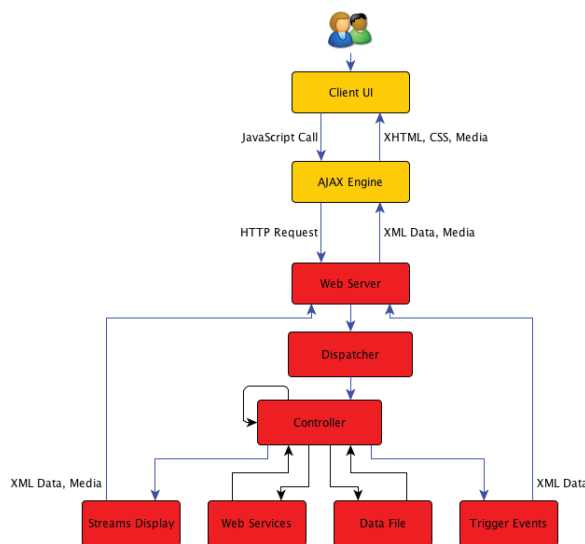
WSDL is an XML format for describing network services as a set of endpoints operating on messages

containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Any special data types used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the functions listed in the WSDL. (Christensen/Curbera/Meredith/Weerawarana, 2001)

3.0 AJAX: PRESENTATION TIER (UI)

3.1 EXAMPLE AJAX PRESENTATION LAYER DATAFLOW



- A typical AJAX application utilizes the Presentation Layer (UI) to create dynamic interaction within a Internet Browser without the need to reload elements.
- AJAX utilizes JavaScript and other programming languages to create a stateful and dynamic application
- AJAX allows for the display of Rich Media and interaction with the content without having to reload the browser
- Content is requested and delivered using HTTP requests
- All the processing of data is done on the Application Level leaving the Presentation Layer to just display content.

3.2 AJAX OVERVIEW

AJAX (asynchronous JavaScript and XML), or AJAX, is a group of interrelated web development techniques used for creating interactive web applications or rich Internet applications. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page. Data is retrieved using the XMLHttpRequest object or through the use of Remote Scripting in browsers that do not support it. XHTML and CSS for presentation the Document Object Model for dynamic display of and interaction with data XML and XSLT for the interchange and manipulation of data, respectively the XMLHttpRequest object for asynchronous communication. JavaScript is used to bring these technologies together.

In many cases, the pages on a website consist of much content that is common between them. Using traditional methods, that content would have to be reloaded on every request. However, using Ajax, a web application can request only the content that needs to be updated, thus drastically reducing bandwidth usage and load time. The use of asynchronous requests allows the client's Web browser UI to be more interactive and to respond quickly to inputs, and sections of pages can also be reloaded individually. Users may perceive the application to be faster or more responsive, even if the application has not changed on the server side.

The use of Ajax can reduce connections to the

server, since scripts and style sheets only have to be requested once. An Ajax framework is a framework that helps to develop web applications that use Ajax, a collection of technologies used to build dynamic web pages on the client side. Data is read from the server or sent to the server by JavaScript requests. However, some processing at the server side may be required to handle requests, such as finding and storing the data. This is accomplished more easily with the use of a framework dedicated to process Ajax requests. The goal of the framework is to provide the Ajax engine described below and associated server and client-side functions. (Wikipedia, Garrett, 2005, Merrill, 2006)

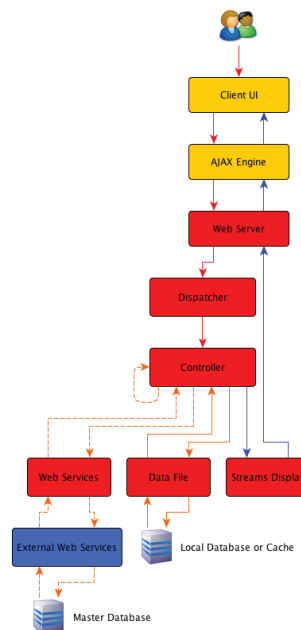
4.0 RUBY ON RAILS: APPLICATION TIER

4.1 EXAMPLE APPLICATION LAYER DATA FLOW

- In an example Web 2.0 Application utilizing AJAX and Ruby on Rails, a user enters a record through the Client UI.
- After the record is complete the user clicks submit.
- This action sends a HTTP request through the AJAX interface engine to the web server.
- The content of the note is formatted to the proper XML specification and is passed along.
- The web server dispatches the request to the Ruby on Rails application server.
- The Rails controller processes the request and a standard routine is executed for updating a record.
- The Rails controller then updates the record in the Local Cache Storage and a notification of the CRUD is returned to the controller.
- The Rails controller also executes a routine to the streams display, which updates the record via the web server and the browser AJAX engine notifying the user that the record has been updated and saved.
- A secondary background process is executed in the Rails controller that

uses REST / SOAP to connect to other external web services to update the master record in a master database.

- Updating the Local Cache Store's record first and displaying the record to the browser reduces latency as the secondary update to the external database can be processed as needed in scheduled batches or over a slower Internet connection.



4.2 RUBY ON RAILS OVERVIEW

Ruby on Rails (RoR) uses Model View Controller architecture implementing DRY principles and orthogonality utilizing separation of presentation (view and controller), domain logic, and data access. RoR is an open-source object-oriented programming language based on Ruby. The core of Rails provides a full set of development and deployment tools and supports the meta-programming method of scaffolding for building database software applications. A simple web server (WEBrick) and build system (Rake) are included with Rails. By including these common tools with the Rails system, a basic development environment is in effect provided with all versions of the software. Ruby on Rails is separated into various packages, namely ActiveRecord, ActiveSupport, ActionPack, ActiveSupport and

ActionMailer. Apart from standard packages, developers can make plugins to extend existing packages.

Additionally the Rails scaffolding provides a framework for unit and functionality testing including mock object and database modifications that are processed out-of-the-container as not to impact a deployed application. One of the strongest features of the Rails framework is its extensive support for SOAP XML messaging, RESTful web services and JavaScript (AJAX). Rails initially utilized SOAP for web services but now primarily uses RESTful web services.

AJAX is integrated into the RoR framework providing the Prototype and script.aculo.us libraries for fast and flexible development. Many other languages like Java are cumbersome to program and don't fully support AJAX natively. A key component to any Web 2.0 applications is database support and connectivity. RoR provides all the need components and libraries to support all major databases both open-source and proprietary.

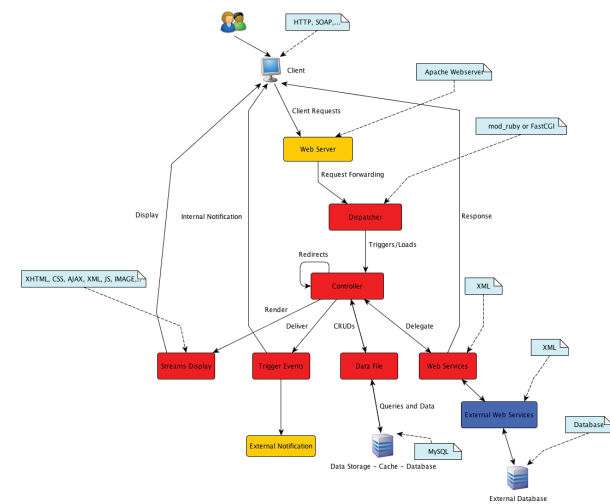
The benefits of using the RoR framework as opposed to other standard frameworks for scalable enterprise applications are many. RoR is a single stack framework providing and including components for the data access layer to the presentation layer. RoR is designed to be a self-contained horizontal and vertical thread. Each thread is independent and does not share data or resources with other Rail threads or other services such as database and web server processes. Rails is scalable with cheap and inexpensive hardware and can utilize network distributed memory cache applications such as memcached scaling every layer of the application framework.

RoR follows a zero configuration model requiring only several directives for per database and/or web service routing without the problems of coupling between application layers and environments. RoR is intended to emphasize Convention over Configuration (CoC), and the agile programming principle of Don't repeat yourself (DRY). "Convention over Configuration" means a developer only needs to specify unconventional aspects of the application. Built into the RoR framework is the concept of deployment environments. RoR provides three environments for production, testing and development. These environments are ruled by strict separation as to not affect

each other including their respective database backend. Switching between environments can be done easily with a master switch. New environments can be added with little configuration.

Ruby provides a reduced code footprint over .NET and Java applications. Some developers have noted a reduction of code that is 6 to 10 times less than the standard Java application. This decrease in code allows for rapid prototyping and development without being held back by thousands of lines of code. This also reduces the time needed for new team members to understand the code base allowing for a reduced learning curve of the software system. (Wikipedia, Bradley, 2006, Hunt/Thomas, 1999, Raymond, 2003)

5.0 MYSQL: DATABASE TIER



5.1 EXAMPLE LOCAL CACHE STORE DATAFLOW

- A user requests a record via the Presentation Layer through the AJAX engine.
- The web server directs the request to the Application Layer (Ruby on Rails).
- The Rails controller executes the request and sends a request to the Local Cache Store.

- The Local Cache Store responds with two possible messages
- If the record is not currently in the Local Cache Store because the record is new or that the record has not been active for a set period of time then the Rails controller must request the record through the external web services to the master databases. The controller then processes the record into and writes it to the Local Cache Store.
- If the record is in the Local Cache Store, the Rails controller receives the record.
- When the record is received by the Rails controller the record is sent to the streams display to be displayed by the AJAX engine in the Presentation Layer.
- When a update to the record occurs, the Local Cache Store record is updated first prior to updating the master record via external web services to the master database.

5.3 MYSQL OVERVIEW

MySQL is a relational database management system (RDBMS). It runs as a server providing multi-user access to a number of databases. MySQL is enterprise ready and offers many features including:

- A broad subset of ANSI SQL 99, as well as extensions.
- Cross-platform support.
- Stored procedures, triggers, cursors and updatable Views.
- True VARCHAR support
- X/Open XA distributed transaction processing (DTP) support; two phase commit as part of this, using Oracle's InnoDB engine.
- Independent storage engines (MyISAM for read speed, InnoDB for transactions and referential integrity, MySQL Archive for storing historical data in little space).
- Transactions with the InnoDB, BDB and Cluster storage engines; savepoints with InnoDB.
- SSL support.

- Query caching.
 - Sub-SELECTs (i.e. nested SELECTs).
 - Replication with one master per slave, many slaves per master, no automatic support for multiple masters per slave.
 - Full-text indexing and searching using MyISAM engine. Embedded database library.
 - Partial Unicode support (UTF-8 sequences longer than 3 bytes are not supported; UCS-2 encoded strings are also limited to the BMP).
 - ACID compliance using the InnoDB, BDB and Cluster engines.
 - Shared-nothing clustering through MySQL Cluster.
- (Wikipedia, MySQL)

REFERENCES

- Bradley, Rick. *Evaluation: moving from Java to Ruby on Rails for the CenterNet rewrite*. 2006. http://rewrite.rickbradley.com/pages/moving_to_rails.html
- Burbeck, Steve. *Applications Programming in Smalltalk-80: How to use Model-View-Controller*. 1987,1992. <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>
- Christensen, Erik. Curbera, Francisco. Meredith, Greg. Weerawarana, Sanjiva. *Web Services Description Language (WSDL) 1.1*. 15 March 2001 <http://www.w3.org/TR/wsdl>
- Costello, Roger L.. *Building Web Services the REST Way*. 2002. <http://www.xfront.com/REST-Web-Services.html>
- eNode, Inc. *Markup Language*. 2002. <http://www.enode.com/x/markup/tutorial/mvc.html>
- Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- Fitzpatrick, Brad. *Distributed Caching with Memcached*. 1 August 2004. <http://www.linuxjournal.com/article/7451>
- Fowler, Martin. *Inversion of Control Containers and the Dependency Injection pattern*. 23 January 2004. <http://www.martinfowler.com/articles/injection.html>
- Fowler, Martin. *Using an Agile Software Process with Offshore Development*. 08 July 2006. <http://www.martinfowler.com/articles/agileOffshore.html>
- Fowler, Martin. *The New Methodology (Agile)*. 13 December 2005. <http://martinfowler.com/articles/newMethodology.html>
- Garrett, Jesse James. *Ajax: A New Approach to Web Applications*. 18 February 2005. <http://www.adaptive-path.com/ideas/essays/archives/000385.php>
- Hunt, Andrew, Thomas, Andrew. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, October 1999
- Merrill, Christopher. *Performance Impacts on AJAX Development. Using AJAX to Improve Bandwidth Performance of Web Applications*. 15 January 2006. <http://www.webperformanceinc.com/library/reports/AjaxBandwidth/>
- Wikipedia. *Model-View-Controller*. http://en.wikipedia.org/wiki/Model_view_controller
- Wikipedia. (DRY) *Don't Repeat Yourself*. http://en.wikipedia.org/wiki/Don%27t_repeat_yourself
- Wikipedia. *Agile Software Development*. http://en.wikipedia.org/wiki/Agile_software
- Wikipedia. *Ruby on Rails*. http://en.wikipedia.org/wiki/Ruby_on_rails
- Wikipedia. *AJAX*. <http://en.wikipedia.org/wiki/AJAX>
- Wikipedia. *Virtual Database EMR*. http://en.wikipedia.org/wiki/Virtual_Database_EMR
- Pautasso, Cesare. Zimmermann, Olaf. Leymann, Frank. *RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision*. 2008. <http://www.jopera.org/files/www2008-restws-pautasso-zimmermann-leymann.pdf>; <http://www.jopera.org/docs/publications/2008/restws>
- Raymond, Eric. *The Art of Unix Programming*. Addison-Wesley, October 2003
- Riehle, Dirk. *A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming: How Methodologies May Learn From Each Other Appeared in Extreme Programming Explained*. G. Succi and M. Marchesi, ed. Boston. Addison-Wesley. 2001
- Reenskaug, Trygve. *Models-Views-Controllers*.

PIIMRESEARCH

WEB 2.0 SYSTEM ARCHITECTURE GUIDELINES
(OVERVIEW AND SOURCE DOCUMENTATION)
BENJAMIN BACON, PIIM, THE NEW SCHOOL
PUBLICATION DATE: OCTOBER 30, 2008

THE NEW SCHOOL

PARSONS INSTITUTE
FOR INFORMATION MAPPING

68 5th Avenue
Room 200
New York, NY 10011

T: 212 229 6825
F: 212 414 4031
piim.newschool.edu

10 December 1979. <http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf>

Stephens, Matt. Rosenberg, Doug. *Extreme Programming Refactored. The case against XP*. Apress. September 2008.

Tilkov, Stefan. *A Brief Introduction to REST*. 10 December 2007. <http://www.infoq.com/articles/rest-introduction>

Tilkov, Stefan. *Addressing Doubts about REST*. 13 March 2008. <http://www.infoq.com/articles/tilkov-rest-doubts>

Venners, Bill. *Don't Live with Broken Windows: A Conversation with Andy Hunt and Dave Thomas, Part I*. 3 March 2003. <http://www.artima.com/intv/fixit.html>

Venners, Bill. *Orthogonality and the DRY Principle: A Conversation with Andy Hunt and Dave Thomas, Part II*. 10 March 2003. <http://www.artima.com/intv/dry.html>

