



Bringing Ruby to the Enterprise: Is Ruby Ready?

Publish Date: 2/15/06

Publish Team: MomentumSI SOA Practice

Overview

The Ruby language is generating a great deal of buzz in the software community these days. Developers are becoming interested in Ruby for a variety of reasons, whether because of the promise of increased productivity, the power of the language itself, or simply its ease of use. At the same time, many who are new to Ruby wonder if it is capable enough to be used in enterprise software development.

This paper is written for developers and IT staff who are considering using Ruby in an enterprise environment. It gives a broad overview of the features of the language from a developer's perspective, and highlights the key advantages of choosing Ruby in the enterprise.

Introduction to Ruby

Ruby is a modern object-oriented language that shares many of the features of Smalltalk, Perl and Python, while being incredibly flexible. One of its strengths is its very simple syntax, which makes it highly readable and maintainable. Ruby is also widely considered a language that is fun to program. Much of this is related to the idea that Ruby follows "The Principle of Least Surprise". That is, the language was designed to be very natural, to minimize inconsistencies (e.g. everything is an object), and to be concise. These features contribute to its ease of use, and productivity enhancement over other platforms. Ruby also has good support for testing, an important facet of agile development methods.

While interest in Ruby has only taken off recently, it is a very mature language that has been around for over ten years. It is highly portable, and available on a wide range of operating system platforms.

The Case for Enterprise Ruby

Enterprise software can be defined as software that processes large amounts of business critical data. Even a short-term enterprise software outage can cost a business large sums of money. As a result, enterprise grade software is usually seen in terms of non-functional requirements such as scalability, availability, and reliability.

Applications in an enterprise are often initially created in isolation, whether due to business urgency, expedience, or the difficulty of coordination between business units. As a result, an enterprise can find itself with lots of silos of data that need processing and correlation. These heterogeneous, distributed systems often need to be integrated later in their lifecycle, though they may not have been designed with integration in mind.

The recent success of the Ruby platform in the rapid development of database-backed websites has brought the Ruby language into the spotlight. It has also brought to the forefront the question of whether Ruby is suitable for enterprise development.

Though the Ruby language has been around for a long time, it has seen limited adoption in enterprise environments. Many still consider it to be a leading-edge technology, one that lacks support for some common enterprise integration technologies. There is also a lack of “best practice patterns” for implementing an enterprise solution with Ruby.

In spite of this, there are many reasons why it would be desirable to use Ruby for enterprise software development. Ruby began as a scripting language, and as such, it is well suited to acting as “glue code” in integrating applications. Scripting languages got their start as a way to rapidly and flexibly coordinate tasks between processes. It follows that Ruby should be capable as a technology for integrating components and services in the enterprise.

Ruby is also a good platform for data manipulation, an important element of enterprise software. Many of Ruby’s features in that area build on the features in Perl, a language with excellent support for generating reports. XML data manipulation is another area where Ruby shines.

Productivity also tends to be higher with Ruby than traditional enterprise platforms. Dynamic languages like Ruby can be good tools for rapid prototyping. With a scripting language, the development process follows a fast write-run-test cycle that is a natural fit for the iterative approach of agile development methods. Ruby code also tends to be far more concise than similar code developed in other languages. At the same time, the intent of the code is typically clearer. Faster development and easier maintenance are features that directly impact the bottom line.

Ruby is a platform that can be used on any application tier. The Ruby on Rails web application framework provides a robust servlet container on the presentation tier. For pure server applications, object-relational mapping tools are available to layer on top of drivers for database access. Ruby has frameworks to produce and consume web services. Also, there are options available for creating messaging solutions for enterprise application integration.

A Guide to the Ruby Platform

A programming language is nothing without a platform of supporting frameworks and libraries, and Ruby is no exception. This section provides an overview of Ruby frameworks for building enterprise web applications and web services, and enterprise applications that require database and application integration support.

Ruby on Rails is a full-stack web application framework that prefers convention over configuration files, placing ease of use for common situations ahead of the need for ultimate flexibility for all situations. Conventions increase productivity by freeing the developer from having to spell out every intention explicitly in configuration files.

A primary selling point of Rails is the claim that its users gain a large productivity increase over standard Java in the development of web applications. This is attributed both to the simplicity of using the Rails framework, as well as the vastly reduced line-of-code count between Ruby and Java code when performing identical functions. Rails is also more productive to use than other

frameworks because its templating language is itself Ruby, which obviates the need to learn a specialized framework language.

Scalability with Rails can be addressed in two main ways. First, since Rails can run in multiple processes on a single machine via FastCGI, fronted by a web server like Apache, it scales fairly linearly with increases of computing power on a single machine. Rails also scales well across machines using the standard web server infrastructure techniques of keeping the application servers stateless, and using load balancers between the web server and app server tiers.

Rails comes bundled with ActiveRecord, an Object-Relational Mapping (ORM) library, for database access. ActiveRecord is also available for use outside of Rails. ActiveRecord's strength is in mapping database schemas that are either created from scratch, or are similar to the kinds of schema that it works best with. In these instances, ActiveRecord is among the simplest ORMs to use. It also includes a powerful data validation system, and impressive support for automating database schema migrations.

There are several options for database access where legacy database schemas are too complex to be handled by ActiveRecord. The first is a mature ORM called Lofcadio. One of its design goals is to offer strong support for legacy schemas. Another option is to fall back from ORMs to direct database connections. Ruby includes a database abstraction layer called DBI, which is similar to JDBC in Java, and allows for database independence. Of course, another option would be to use a database driver directly. Note that ActiveRecord allows access to these other layers, so variations from ActiveRecord's preferred style of schema design can be handled without losing ActiveRecord's benefits.

Note that for working with Rails, the presentation tier does not require either ActiveRecord, or any database for that matter. Rails can easily be used as a front-end to a service-oriented architecture (SOA), or any external service-based API that Ruby can connect to. In this tiered approach, there is a clean separation between the Rails presentation layer, and the service and data layers. The different layers can be developed by different teams, using different technologies.

Ruby offers a number of options for integration with enterprise services. Ruby can be bridged to existing JMS messaging systems that support STOMP (Streaming Text Orientated Messaging Protocol). STOMP is an open messaging protocol that provides publish/subscribe services, as well as receipt and transaction services. Ruby has good SOAP support, and Ruby classes can be generated from WSDL. Ruby also has a very complete XML-RPC library.

Ruby also has excellent base HTTP client support for custom HTTP messaging integration. It offers the WEBrick embedded web server/servlet container for message exchange. WEBrick can also be easily integrated with existing CGI scripts, and provides HTTPS/SSL support.

There are several areas where direct Ruby support for enterprise technologies is lacking. There is no standard Ruby service comparable to JMX for application management. There is also no support for a standard authentication/authorization service. Distributed transactions are not available through any Ruby transaction manager. Also, there is no real CORBA support in

Ruby. Finally, internationalization support is not unified like it is in Java. Although there are libraries available to bridge some of the gaps, custom i18n development may be required, depending on the application.

In these areas where Ruby lacks native integration support, there are Ruby-Java bridges available for direct integration to existing Java code, or for bridging to services better provided by Java.

Comparing Enterprise Java and Ruby

As an aid determining the enterprise capabilities of Ruby, the following matrix compares the enterprise features of Java and Ruby. Note that many of the Ruby frameworks are still maturing. While they still provide valuable functionality, they may not be as full-featured as the corresponding Java framework.

Enterprise Features	Java	Ruby
Web applications	Struts, Tapestry and JSF are major Java web frameworks. Java provides flexibility in the choice of web framework, flexibility in the use of supporting frameworks (like ORM), and flexibility in the use of each framework, at the cost of added complexity. AJAX support is ad-hoc.	<p>Ruby on Rails dominates the Ruby web app framework market. Rails is a full-stack framework, including a database access layer. Rails is less flexible than other solutions, as it targets simplicity.</p> <p>Nitro is a web framework, currently in beta, that allows for greater flexibility than Rails. Apps can be written in the MVC style of Rails and major Java frameworks, or in the server pages style of ASP/JSP.</p> <p>Both frameworks have strong AJAX support, integrating with the Prototype AJAX library.</p>
Scalability and Availability	Enterprise app servers provide clustering and fail-over services for properly written apps, at the cost of added complexity. Alternately, stateless Java apps can be scaled with load balancers just like LAMP (Linux, Apache, MySql, Perl/Python/PHP) apps.	State of the art for Rails apps is to follow the LAMP pattern. Keep the application servers stateless, and push the state to the database. Then use load balancers between tiers, to add/remove any number of app servers. Rails provides simplified clustering via a high-performance distributed memory cache.

Enterprise Features	Java	Ruby
Object/Relational mapping	Hibernate is the most popular Java ORM, with the flexibility to map to most database designs.	ActiveRecord is the Rails ORM solution. It follows the Active Record pattern of mapping an object to a database table row. This is a simpler design, and better suited for mapping to schemas that are similar to the domain model. Lafcadio is an alternative ORM, with more sophisticated mapping support for legacy schemas.
Database connectivity	Enterprise app servers provide sophisticated connection handling, including automatic recovery of leaked connections and connection reservation handling.	ActiveRecord provides simple connection pooling and dropped connection restarting.
Transaction support	Containers like Spring provide lightweight transaction services. App servers supporting JTA provide distributed transaction support.	Standard transaction support. No support for distributed transactions, though ActiveRecord does support persisting to multiple databases.
Dependency Injection	Spring is the standard dependency injection container for Java.	Needle is a dependency injection container for Ruby. While DI is not as necessary in Ruby due to its dynamic nature, Needle offers service management, and AOP-like advice.
Packaging and Deployment	J2EE has the WAR file and EAR file standards.	Rails has a unified packaging and deployment framework called Capistrano, with features beyond WARs and EARs like automated parallel deployment to multiple servers, and easy rollback.

Enterprise Features	Java	Ruby
Enterprise Application Integration	<p>A variety of JMS messaging products, like ActiveMQ are available, as are ESBs like ServiceMix.</p> <p>Most any legacy app can be bridged using the Java Connector Architecture.</p>	<p>Ruby can be bridged to JMS implementations like ActiveMQ (and thus ESBs like ServiceMix) that support STOMP (Streaming Text Orientated Messaging Protocol).</p> <p>Ruby Reliable Messaging is a Ruby-only messaging service.</p> <p>Ruby lacks the kind of legacy app connectivity provided by the Java Connector Architecture.</p>
Web services	J2EE provides many APIs for web services support.	Rails has good support for simple web services via ActionWebService and other libraries. XML/RPC, SOAP 1.1, WSDL, REST
XML documents	Fully supported.	Support for parsing and building, and XPath expressions. Lacks good support for XSLT and schema validation.
Distributed objects	Support for CORBA and Java RMI.	Ruby has an RMI-like facility for distributed objects called Distributed Ruby. CORBA support is poor.
Management	Application management via JMX. A variety of SNMP stacks available.	No standard app management API or framework. Simple SNMP library available.
Directory services	JNDI for Java services and LDAP for others.	LDAP support from a beta library.
Build systems	Ant and Maven provide full project support, and continuous integration tools like CruiseControl are available.	Rake provides Ant-like functionality. RubyGems provides automated library download support. DamageControl provides continuous integration.
Internationalization	Excellent support.	While Ruby supports Unicode encodings internally, its own string libraries do not. There are libraries that augment i18n string support, and many libraries for specific tasks support i18n, but there is not unified support.
Authentication	Support via JAAS, support for Kerberos.	No standard authentication libraries, or standard single sign-on support.

Enterprise Features	Java	Ruby
Platform support	Runs on a wide variety of platforms, OSes, etc.	Runs best on Unix systems, potential issues on Windows platforms.

MomentumSI Recommendations

It can be a challenge to decide whether Ruby makes sense for a particular enterprise project. MomentumSI offers guidance on using Ruby on projects broken down according to various enterprise application categories:

- **Mission critical / non-stop processing:** Ruby is not ready for an environment that includes integration with transaction monitors, or CICS/IMS interfaces.
- **Complex transactional:** The suitability of Ruby in a complex transactional environment depends on the nature of the transactions. While Ruby does not offer native XA support natively, it can be integrated into transactional messaging systems. This opens up the possibility of using Ruby for enterprise application integration.
- **SOA / Web services:** Ruby can be used as both a producer and a consumer of SOAP-based web services, enabling its use in service-oriented architectures.
- **Data intensive:** Ruby has been proven out in data intensive applications. Examples include space shuttle simulations done by NASA, and meteorological number crunching done by NOAA.
- **Web applications:** Ruby on Rails is a productivity leader in the area of database-backed web applications, with the ability to scale well, and with excellent developer support.
- **Workgroup software:** Software in the enterprise targeted at workgroups usually requires rapid prototyping, which is a Ruby strength.

Conclusion

MomentumSI recommends that those businesses looking for a competitive advantage consider using Ruby in an enterprise environment. While Ruby is not ready to be used in every aspect of the enterprise, it can fit into many areas of an application strategy. By doing so, a business can reap the productivity gains from the agility that the Ruby platform offers.