JISC

Advanced Software Development for Web Applications

David Plans Casal
Director of Research
Luminas Internet Applications

This report was peer reviewed by:

Mark H. Butler
Research Scientist, Digital Media Laboratory
HP Laboratories, Bristol.

Andrea Rota
Learning Technologist (PDP), Centre for Excellence in Learning Technology
Goldsmiths College, University of London.

Contents

1. INTRODUCTION

The Web has changed dramatically over the last decade. What was once a rudimentary system for distributing static content consisting of simple text and images has become a platform for highly sophisticated, distributed, enterprise-level applications. In Higher Education Web applications are used for a wide range of content delivery, information processing and administrative transactions. Typical examples include Virtual Learning Environments (VLEs), student portals run by individual institutions, the JORUM repository for learning and teaching materials, and the Research Councils' Electronic Grant Services (Je-S).

In order for Web servers to reliably deliver sophisticated applications to Web browsers a range of enterprise-level systems built on two key technologies, Java™-based J2EE and Microsoft .Net, have emerged. A number of Web application frameworks have been introduced to work with these technologies and these frameworks facilitate development by providing a skeletal support structure made of reusable software components to solve common Web application problems. Although such frameworks have been in use for a number of years they are constantly evolving; this report reviews their current state-of-the-art and discusses, in depth, two such frameworks: Ruby on Rails and Cocoon. It is argued that adoption of Web frameworks can reduce the development learning curve, reduce project costs, and, in the long run, offer a potential indicator towards future systems that may allow non-programmers to build applications. The report speculates that these easy-to-use development systems may, in the future, be able to work with emerging technologies and approaches being developed in the Semantic Web arena.

2. WHAT ARE WEB FRAMEWORKS? A BACKGROUND.

2.1 From static to dynamic

The earliest Web servers were designed to publish static content consisting of HTML files of simple text and graphics that were delivered by a simple 'pull' mechanism (i.e. the browser requests and displays the content items that constitute each Web 'page') using HTTP processing. Over time, the desire to create interactive content that performed like a desktop application created a demand for sophisticated, two-way communication between resources such as databases and their users (dynamic content). This led to the development of a class of software known as 'Web applications'.

2.1.1 What is a Web application?

A Web application is a software application that delivers its functionality to a user from a Web server, through a network such as the World Wide Web or an intranet. The user views and manipulates the application through a Web browser. This was initially achieved using *scripts* – small programs that could perform HTTP processing (they accepted HTTP requests, did some processing on the server and then sent back a response) (Knight and Dai, 2002). This development was facilitated by the creation of a Common Gateway Interface standard (CGI), which standardised how Web servers interact and invoke scripts that could carry out such tasks.

These scripts were commonly written in Unix-style shell scripts, compiled languages such as C, or interpreted languages such as Perl (a very flexible language particularly suited to text manipulation and handling basic Web-based interactions). However, the CGI approach had limitations. There were performance issues. It was difficult to create interaction across multiple pages, since each page often interacted with a separate script and there was no simple way to keep track of the state of the system from page to page. Also, the scripts often had to perform the task of writing out the page presentation as well as the dynamic content. While this meant that it was fairly easy to create simple applications, it became difficult and costly to maintain and update more complex sites.

2.2 Moving on to enterprise systems: server and client side scripting

In response to the problems with CGI a number of competing server and client-based technologies were developed in the late 1990s. Very simply, these technologies can be divided into client-based solutions, which endeavoured to extend the functionality of the browser, and server-based solutions, which incorporated the functionality of the CGI script onto the Web server, rather than inside the page (Doyle and Lopes, 2005). These technologies included the following:

- Web Server Extensions such as Apache Modules API and Microsoft ISAPI
- Server-side systems such as Java Servlets
- Browser client extensions such as Java Applets and client-side scripting with Javascript or VBScript
- Tagging systems in which tags are placed in HTML pages e.g. Coldfusion
- Server-side scripting systems such as Active Server Pages (ASP), PHP and Java Server Pages (JSP) in which additional code within the HTML files is processed on the server

Such developments began to coalesce to a certain extent around a basic model of dynamic Web interaction, known, in Java circles at least, as JSP model 1 (Cavaness, 2005). However, these developments, whilst solving immediate problems and allowing rapid development of fairly complex applications, tended to lend themselves to or even encourage bad software practices such as directly coding access to a database (Knight and Dai, 2002). For an in-depth discussion of some of the issues facing Web application developers and how they differ from traditional software engineering see Pressman (2000a).

In an effort to combat these problems there were attempts to incorporate some element of layering of complex applications i.e. a process of trying to separate key areas of the application (content, presentation, logic) and to provide boundaries (sometimes also known as Separation of Concerns) and to build in some of the techniques of object-oriented programming. This idea emerged in parallel with the increasing adoption of the Web for mainstream activities and its use for the delivery of traditional enterprise systems that were used for everyday business and commercial transactions. There was also a corresponding increase in the need to provide such enterprise-level systems that could scale and function securely. This meant facilitating the delivery of existing multi-tiered enterprise and business systems over the Web, using the browser as a client. This required sophisticated application servers[1] that could operate in a tiered or layered architectural structure in which the Web server, the presentation (View), and the business software and databases are separated (Doyle and Lopes, 2005). The requirements for such enterprise-level Web systems are defined in detail in Doyle and Lopes. The demand for technologies that could use these approaches led to the development of two competing application server solutions: Java 2 Enterprise Edition (J2EE) and Microsoft's .NET technologies.

The complexities of developing systems for this kind of multi-tiered architecture led to developments in software engineering based around the use of frameworks.

2.3 Frameworks

A framework is "a reusable, 'semi-complete' application that can be specialised to produce custom applications" (Fayad and Schmidt, 1997). Frameworks provide a skeletal support structure made of software components upon which new software applications can be quickly built and organised. Reusability, whereby generic components can be used in new applications, is a key characteristic.

The concept of frameworks has been used for many years in some application domains, for example, Graphical User Interface (GUI) design. They were also used in distributed object systems, which used CORBA and DCOM frameworks. The idea of frameworks is closely allied to that of software design patterns, a representation of recurring solutions to common software problems. Both techniques seek to capture the previous good work of software developers and allow re-adoption. There are a number of

[1] An application server is software present on a central computer server that handles the execution of applications and often works closely in tandem with a Web server and a database. The Web server handles the actual distribution of Web-based content via HTTP. For more information on application servers, see Pressman (2000b).

issues regarding the use of frameworks, for example the learning curve for new staff, and these are explored in more detail in Fayad and Schmidt.

The development of frameworks within the Web development community has led to a number of 'Web application frameworks' which follow the same principle in that they allow the developer to build a Web application upon the existing components of the framework, but deliver the functionality through a Web browser.

2.4 State of the Art in Web Application Frameworks

There are literally hundreds of such frameworks, but a small number stand out in particular. This is partly because of their widespread popularity, but also because they have absorbed lessons from the important Model-View-Controller pattern (see side panel on MVC). A number of these can be classed as fully MVC-compliant Web application frameworks.

In the following discussion we outline some of the popular frameworks which operate within the Java Virtual Machine (JVM) and J2EE environment. In contrast, a number of competing technologies also exist in the Microsoft domain including .NET and ASP.NET both of which operate within the company's Common Language Runtime (CLR) environment. ASP.NET is an upgraded version of ASP that supports Web Forms, which provide a page-driven object model for Web application programming that is similar to the Tapestry object model (Doyle and Lopes, 2005). There is considerable debate about the merits or otherwise of each of these camps and for discussions and comparison see, as a few examples amongst many, Fowler et al. (2004), Williams (2003), Miller (2003) and Woo (2003). The remainder of this discussion will focus on the Java and Open Source-oriented frameworks.

Some of the better known Java-based frameworks include JSF, Struts, Tapestry, Cocoon and SpringMVC. It is quite common for different frameworks to be used in combination, with each one providing functionality for a different layer of MVC tiered architecture. For example, using Struts for the View (presentation) level, Spring for Control and integrating with a database via Hibernate (Eagle, 2004). The following table outlines the main frameworks used within the Java and Open Source environments.

Table 1: Some common Web frameworks

Framework	*URL*	*Full MVC layer?*	*Characteristics and Comments*
Struts	http://struts.apache.org/	MVC	Apache, Open Source project Stable and mature *De facto* standard, widely used and large user base Great deal of documentation, books and articles Java-based, extends Java Servlet API to allow MVC Integrates with choice of MVC Model layers e.g. Hibernate for database access Operation-centric Uses Front Controller (Fowler et al., 2004)
Spring	http://www.springframework.org	MVC	Flexible, layered framework using XML configuration Open Source, but non-standard A medium level of documentation (several articles, eight or so books) J2EE framework that aims to make J2EE easier to use Seen as a replacement technology for Enterprise Java Beans (EJB) Can act as middle tier with other MVC frameworks such as Struts, Tapestry Based on Dependency Injection pattern
Tapestry	http://jakarta.apache.org/tapestry/	View (User Interface) framework	Apache, Open Source project Builds on Java Servlet API Small level of documentation (several articles and 2 books) Component-centric Takes an object-based approach to frameworks, dividing an application into pages, each made up of Tapestry components Blends well into HTML files Integrates with middleware frameworks such as Spring Limited documentation
Java™ Server Faces (JSF)	https://javaserverfaces.dev.java.net/	View (User Interface) framework	Comparatively new and immature Considered fast for development and extendable Developed through Java Community Process Based closely on J2EE standard Can interoperate with Struts Front Controller design pattern

Cocoon	http://cocoon.apache.org/	MVC (partial)	Apache Open Source project Component based Built around concept of 'Separation of Concerns' – a component pipeline rather than architectural tiers XML-based framework Strong publishing focus No direct link to Java component model e.g. EJB, which is a potential issue for enterprise-level developers A medium level of documentation
Ruby on Rails	www.rubyonrails.org	MVC	Open Source Full stack, object-oriented Web framework Non-Java (written in the 'Ruby' programming language) Provides a persistence framework for Database access As a new framework, Ruby on Rails has only a small amount of documentation to date but documentation and informal support is growing quickly

Struts is an Apache Software Foundation (ASF) framework for Java-built Web applications that uses an MVC-style framework (Davis, 2001). It is considered somewhat of a standard (in terms of the number of developers that use it and the large number of job descriptions that require it), and there is a great deal of information and documentation available for it. However, its lack of project activity may indicate that it is being superseded, or that other MVC framework implementations are taking over.

SpringMVC, another MVC-based framework, is a good candidate for application migration from Struts-build applications, as it is an extremely flexible framework that aims to provide a simpler development alternative to frameworks such as Struts. While its flexibility means it can couple with existing View technologies such as JSP, it demands a great deal of XML configuration, and asks that developers make all the choices (from writing every View to choosing a Controller).

Jakarta Tapestry, also from the ASF, takes a more object-based approach to frameworks, with pages built from HTML templates and small components of Java code.

Java™ Server Faces (JSF), from Sun, is an Application Programming Interface (API) and standardised tag library that interacts between Web pages and small components of code on the server ('beans'). JSF was only finalized as a standard in May 2004 so is comparatively new and immature, but it is relatively fast to develop with and has the benefit of being perceived as an industry standard, having been through the Java Community Process. It also proposes a method of development tied more inherently to application components instead of ‘pages’, as used by Struts.

Apache Cocoon, whilst less widely adopted than frameworks such as Struts or JSF, contains some of the most interesting ideas as far as 'pluggable' component-based application design is concerned. It includes many of the features of the MVC-based frameworks, coupled with a clear Separation of Concerns through the use of XML and XSLT, and centralised URL management through the concept of a sitemap.

Model, View, Controller

A well known model for software architecture in which there is a separation of the fundamental components of an application is the Model, View, Controller (MVC) pattern (see figure 1).

> “Model-View-Controller (MVC) is a software architecture that separates an application's data model, user interface, and control logic into three distinct components so that modifications to the view component can be made with minimal impact to the data model component.” (Wikipedia).

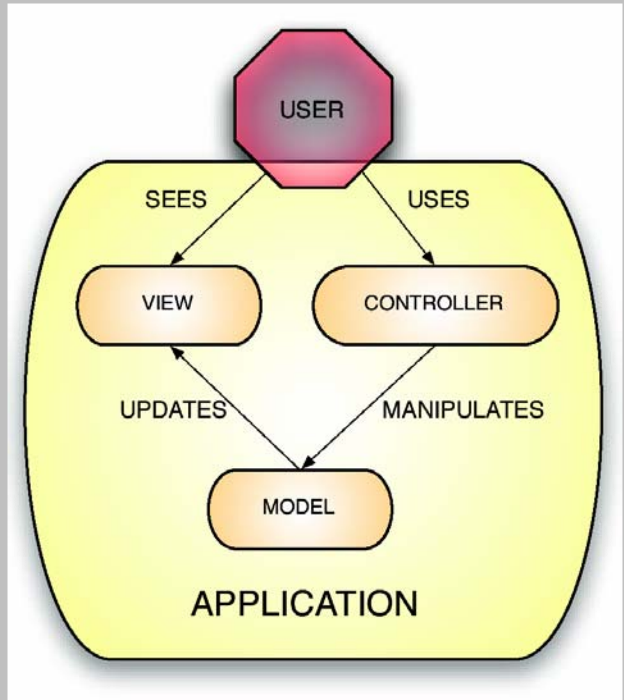

Figure 1: The MVC model from a user’s point of view.

In contrast to the complexity of the Java-based frameworks, **Ruby on Rails** aims to simplify the development process by allowing itself (as a framework) less flexibility than others in terms of choices, relying instead on the principle of convention over configuration (for more information on flexibility and scalability see section 4.2.4). It is claimed that this relies, not on XML configuration files, but on a set of programming conventions based on information already in place within the application code and the attendant databases (Kay, 2005). Rails does not use the Java language but instead relies on Ruby, a Japanese programming language created in the mid-1990s and which has recently attracted a lot of attention in Europe and the US. Ruby aims to provide the developer with a language based on the principle of 'least surprise', where what a user might guess is the way to make the language do something, is usually correct. Both these principles have allowed Rails to provide a framework with which applications can be developed very quickly.

In order to build an application using an MVC architecture, a developer first needs to be able to define the Model, which is the domain-specific representation of the world the application addresses. In a library application, for example, this would include things such as a user's last name, a book, a book ID, or the concept of 'lending'. Secondly, the developer must define the View, which renders the model into a user interface which can be used; finally the developer must define a Controller, which will act on the model and change information within it and within the View in response to user actions.

For example, a user selecting a book and reserving it would prompt the Controller to prompt the Model to act on a 'lending' function and store details in a database (such as the book's ID and a lending period), and the Controller would then change the View (which can be seen through the browser) to tell the user that they can now pick the book up at a particular lending desk.

The MVC pattern is a good idea, because it allows developers to write applications that are easier to maintain and more sustainable in the long run, as the Model and Controller are decoupled from the View. This means that, in the case of the Web, a framework that follows the MVC architectural pattern well, will allow for the View to be handled in total separation from the 'meat' of the application, and this allows for much less duplication of effort (or coupling), which means the application’s code is easier to write and maintain.

3. ISSUES AROUND THE USE OF WEB APPLICATION FRAMEWORKS IN HIGHER EDUCATION

Even though the remit of an HE project is usually to explore an academic subject with the aid of a Web application, the complexities involved in modern Web application development mean that many projects have suffered from delays, budget overruns, and even failure to meet their objectives.

There is no doubt that the use of frameworks provides considerable benefits in terms of the time and effort put into developing high quality systems (Morisio et al., 2002). However, there are a number of issues surrounding the practicalities of using Web application frameworks in development projects for Higher Education. These issues include: learning curve, complexity and resourcing, limits to scalability, and interoperability. Choosing a framework is difficult and a number of factors need to be considered, including the developer user base, the availability of support and documentation, the maturity and likely longevity of the framework, and the cost.

3.1 Learning Curve

The Web application building process often involves a steep learning curve for the typical developer (Morisio et al., 2002; Srinivasan, 1999), depending on the complexity and ambition of the project and the developer's previous experience. In addition to requiring knowledge of the frameworks, those based on Java and XML require a developer to either have deep pre-existing knowledge of both the Java language and XML, or be prepared to invest in learning them. Whilst application development productivity improves over time from one project to the next (as the developer gets used to the framework), in HE the initial learning curve can come as a considerable and undesirable expense and divert attention from the project’s underlying academic aims.

3.2 Complexity and resource

A developer working in higher education today will be faced with a choice: building using a Web application framework from the Java/XML or .NET domain, or using a scripting language such as PHP. As we have already mentioned, the complexity of some frameworks carries a learning curve penalty, although this varies between frameworks. However, they do provide enterprise-level functionality and libraries, easier interoperability with portals or digital repositories and scalability. A scripting solution may seem on the face of it to offer more rapid development, but in the long run it is possible that there may be serious issues with scaling the technology as a project develops and grows.

3.3 Interoperability

Sophisticated frameworks based on technologies such as J2EE provide strong interoperability through Web Services technologies such as SOAP and component technologies such CORBA and COM (Westkämper, 2004). This can be an important requirement within educational environments where the ability to work with existing infrastructure such as portals, VLEs, data repositories and e-learning frameworks is important.

3.4 A potential solution

It is clear that developers face difficult choices. What is required is a solution whereby a minimally competent project worker should be able to create a skeleton Web application to express the functional needs of her project, without the need to invest large amounts of time and money. We will therefore focus on two frameworks, which, in the author's opinion, address the MVC pattern in a way that is more suitable for developers without a high level of programming and software development skills. These are Cocoon, a mature Apache Web framework, and Ruby on Rails, an emergent Web framework with the remit to simplify the development process. As we shall see, these frameworks take differing approaches to the problem. In Cocoon's case, a reliance on XML configuration enables the developer to interact only (in most cases) with well-structured, understandable representations of the applications problem domain and user interface. In Ruby on Rails' case, because it offers an almost 'magic' approach to MVC by giving the developer a toolset which has been pre-established and provides the functionality needed to implement most MVC-based applications, the developer is able to almost 'guess' at the problem domain and enlist the framework to help her implement its functionality. Both Cocoon and Rails are examples of attempts to build frameworks that seek to free people from the role of deep technical programming and towards a more humanistic role as builders and thinkers, without the tools getting in the way.

In our examination of Cocoon and Rails, we will look at how both frameworks have attempted this, and what both approaches provide from a developer's point of view.

4. APACHE COCOON: XML CONFIGURATION

Cocoon, a project within the Apache Software Foundation, started as a Web publishing system - a direct result of having to find a solution to the content publishing and management problem. It is now an example of a more general Web framework based around XML, although its capacity and remit are not limited to applications that are used through the Web.

In 1998, Stefano Mazzocchi needed to find a way to automate the process of updating the java.apache.org Website for the Apache Software Foundation (Mazzocchi, 2002). In particular, he needed to update all the documentation files for each project contained within that website dynamically, so avoiding manual processing. However, since quite a number of developers were involved, there were inconsistencies. They needed to separate content (the actual documentation) from style (the resulting HTML which displayed the content to browsers). Stefano wrote a Servlet, which would use the then emerging Open standard XSL (before it became XSLT) to achieve the needed transformations.

4.1 Apache Cocoon architecture overview

4.1.1 Separation of Concerns and the contract pyramid

Apache Cocoon uses the concept of Separation of Concerns (SoC), a software engineering approach whereby complex systems are broken down into discrete features. It was first introduced as an idea in Dijkstra's (1974) 'On the role of scientific thought'. In Cocoon, SoC seeks to achieve independence between the different contractual agreements in a project: data ingestors, designers, archivists, and software engineers. Separation of Concerns is achieved by using the ‘pyramid of contracts’, which outlines four major areas of concern, and the five contracts between them (see figure 2). The key is that there is no contract between Logic and Style, which means that Cocoon achieves a clean separation between programmers and graphic design staff.

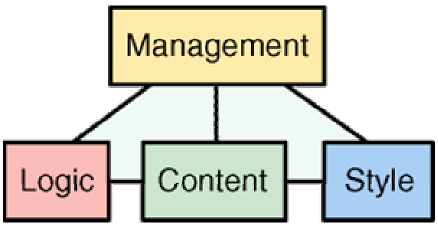

Figure 2: The Separation of Concerns principle.

This allows project managers to seal different work groups into separate areas, so that programmers and designers can influence but not hamper each other’s work.

4.1.2 Component pipelines

Underlying this general concept, there is Apache Cocoon’s component pipeline model in which an XML document is pushed through a series of stages in a pipeline. Every pipeline begins with a Generator, continues with zero or more Transformers and ends with a Serializer (see figure 3). The pipeline passes SAX events (Simple API for XML) (SAX, 2005) from one end of the pipeline to the other (see figure 4). SAX is a system for processing XML documents in a stream, as they are read, which avoids the need to store the entire document in memory before processing can begin and is therefore more efficient (Becker et al., 2003).

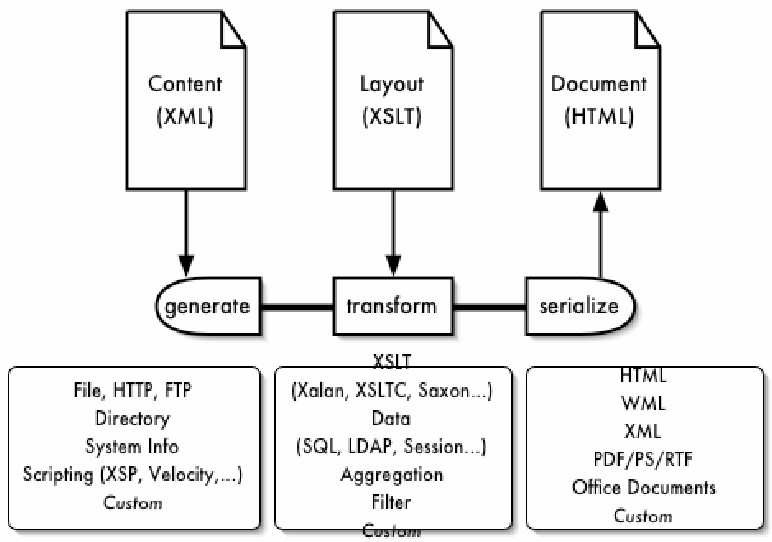

Figure 3: An Apache Cocoon component pipeline.

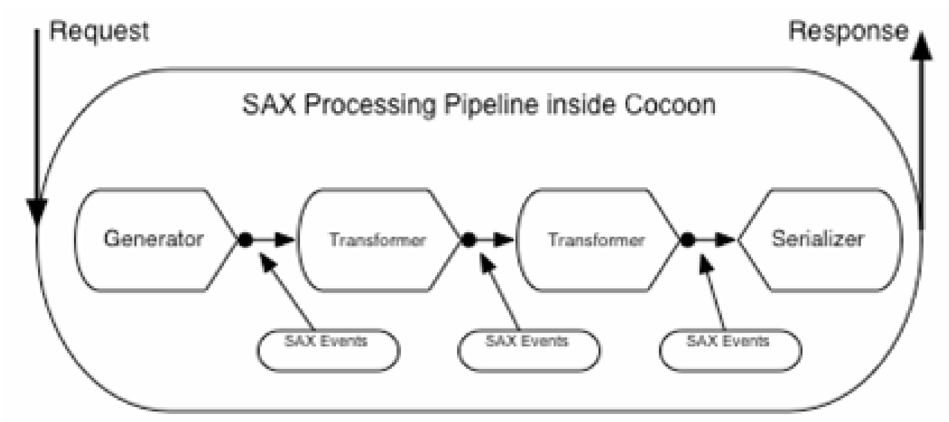

Figure 4: A typical SAX processing pipeline.

In every pipeline, the process an XML document goes through begins with a component called a Generator, which delivers SAX events down the pipeline, based, for example, on its parsing of an XML file (in the case of the simplest of Generators, the FileGenerator). After that, there are Transformers and Serializers. The simplest Transformer applies an XSLT to an XML file, and the simplest Serializer simply returns a human-readable XML file. Currently, there are Serializers available for generating a range of display formats such as HTML, XML, PDF, VRML, WAP, WML and MIDI. As Apache Cocoon is distributed under an Apache Software License, it is possible for developers to create their own serializers to handle specific requirements.

4.1.3 The sitemap

The Apache Cocoon sitemap essentially tells a Web application how a particular site will work. It accomplishes this by matching potential URLs to resources. Uniform Resource Identifier (URI) patterns help an application to identify pipelines, so that when a human or machine user of a site's resource requests 'http://www.resource.ac.uk/pottery/grey_pot/' the application can match the URI to a particular pipeline and fetch the resource that the sitemap indicates it belongs to, initiating a SAX event process which culminates in the user being given appropriate content, in the appropriate format. This is very important and ultimately useful when the site must cater for requests from other systems (e.g. archives, content providers, aggregators), and user interface requests according to the requirements of different devices (e.g. different views of content depending on browser settings, device detection). Cocoon delivers the best possible scenario in a multi-channelled way:

```
<map:pipeline>

   <map:match pattern="grey_pot.html">
    <map:generate src="grey_pot-page.xml"/>
    <map:transform src="stylesheets/simple-page2html.xsl"/>
    <map:serialize type="html"/>
   </map:match>

   <map:match pattern="grey_pot.wml">
    <map:generate src="grey_pot-page.xml"/>
    <map:transform src="stylesheets/simple-page2wml.xsl"/>
    <map:serialize type="wap"/>
   </map:match>

   <map:match pattern="grey_pot.pdf">
    <map:generate src="grey_pot-page.xml"/>
    <map:transform src="stylesheets/simple-page2fo.xsl"/>
    <map:serialize type="fo2pdf"/>
   </map:match>

</map:pipeline>
```

The first block outputs HTML from an XML document, and is directed at normal desktop computer browsers. The second outputs WML markup from the same XML document, and is directed at mobile device browsers. The third creates a PDF document on the fly, again from the same XML document.

The application, using this sitemap, will deliver the requested resource as an HTML page if requested by a browser, a WML page for a mobile device, a VoiceXML for a voice client, as a JPG rendition of a Scalable Vector Graphics file, or as a printable PDF file. All these results are obtained from a single point of departure, our original XML file:

```
<?xml version="1.0"?>
<page>
 <title>A Grey Pot</title>
 <content>
  <para>This is a page about a Grey Pot, from the Pottery resource.</para>
 </content>
</page>
```

In essence, for every server resource (like 'grey_pot.pdf') that is requested, the sitemap matches it to its XML file, which is parsed and then processed by an appropriate XSLT stylesheet. A Serializer is then called to produce the output needed (see figure 3).

4.2 In a nutshell

Apache Cocoon offers a developer with a certain level of knowledge (e.g. what XML is, why it can be used to express such things as Sitemaps) an extremely powerful framework within which to create XML applications. There is, however, some debate about the level of programming knowledge a potential developer would need to have. Ronald Schmeizer has noted that “It is possible to compose together components into a pipeline without any required programming” (Kerner, 2004), although it can be argued that declarations in XML are a form of programming.

Using Cocoon, one can address Web publishing in a uniquely powerful way: by using XML configuration files, a developer can express her idea of the application (its functionality and shape) without straying from readable text. In fact, there have been research projects to develop automatic systems that take design diagrams or Unified Modelling Language descriptions and automatically generate Cocoon-based Web applications (Kraus and Koch, 2002; Carrico et al., 2003).

As a framework based on Open standards which comes from a robust, mature and extremely successful Open Source community, Cocoon is at the forefront of current thinking in the provision of tools for managing and publishing content through Web applications. It is hard to think of a research field whose data and objectives could not easily be met by it; one way to look at its flexibility is to examine the many sources of data it can natively handle and transform (see figure 5). Because of its implementation of functional components into Blocks, Cocoon attains the ‘Lego™’ principle well (Mazzocchi, 2002), and points to future scenarios where framework functionality will become less dependent on its sources, and more easily distributable and reusable.

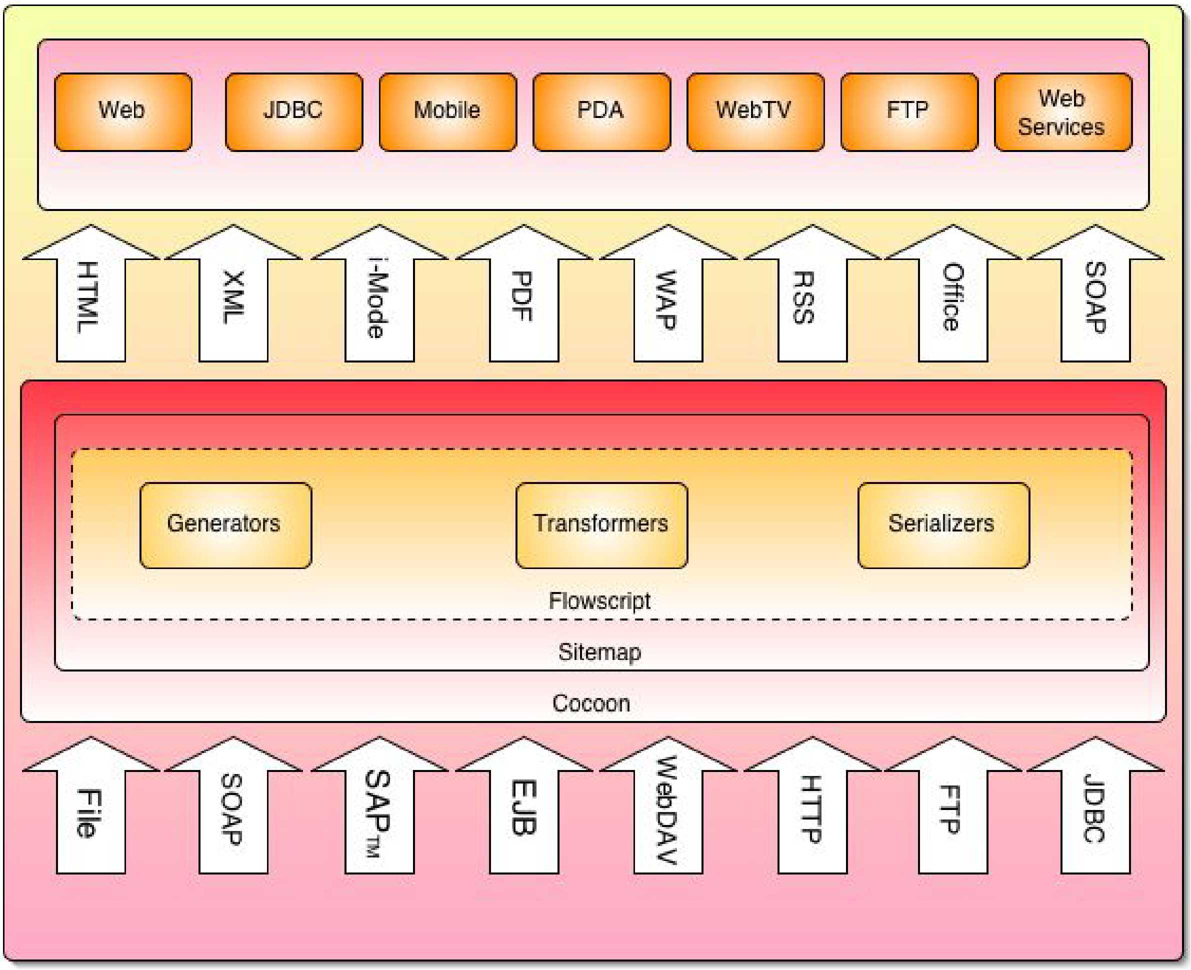

Figure 5: Cocoon seen from a data sources and outputs view.

5. RUBY ON RAILS: NO SURPRISES, PLEASE

Ruby on Rails, an Open Source Web framework built using the Ruby programming language (Ruby, 2005), sets out to make it easier to develop, deploy, and maintain Web applications (Mertz, 2005). Rails supports a full stack version of the MVC model (i.e. it covers all layers including the database access) and provides the scaffolding for building Web applications. In 'Agile Web Development with Rails' (Thomas and Hansson, 2005), Rails is presented as a framework which follows the MVC architectural model because it helps developers structure their applications more cleanly, and does so while encouraging developers to follow agile software development principles, such as sustainability and practicality, over major pre-emptive planning and requirements gathering. Hibbs (2005) argues that, in terms of Web application development, Ruby on Rails is five to ten times faster than comparable Java frameworks. See also the 'Ruby on Rails' chapter in Tate (2005). For a comparison with more traditional J2EE environments (using Struts) see Rustad (2005).

Ruby, the language with which Rails is built, is a very important part of its sophistication and power. Ruby is a pure, dynamic, completely object-oriented scripting language, developed by Yukihiro Matsumoto (Tate, 2005; Drake, 2001). It is interpreted rather than compiled and has a simple syntax that was partially inspired by languages such as Eiffel and Ada, but also includes features from Java, Python, Lisp and Smalltalk.

It is possible to argue that much of the design principles behind Rails and Ruby itself have more of a psychological benefit than a technical one, as their intention is not necessarily to implement design patterns such as MVC or features such as object orientation better than other frameworks or programming languages, but to simply create more efficient working environments for developers.

One of the ideas Rails follows as a result of its Ruby inheritance, is that of 'least surprise' or 'least astonishment' (Raymond, 2003). This is an ergonomic principle that can be applied to user interfaces, programming languages, or in general, to any toolset or application in which a 'user' is one of the factors of interaction. This principle states that, when two elements of an interface conflict or are ambiguous, its behaviour should be that which will least surprise the human user or programmer at the time the conflict arises, because the least surprising behaviour will usually be the correct one.

Below, we will look at the basic elements of the Rails framework from the point of view of a developer about to build a typical Web application. A diagram of the Rails framework can be seen at: http://wiki.rubyonrails.com/rails/pages/UnderstandingRailsMVC.

5.1 How Rails works

Ruby and its principle of least surprise give Rails a very real advantage: it makes more use of the language itself, and less use (in comparison to Cocoon or other Java/XML frameworks) of configuration files. In a way, this makes it easier for a developer to immediately understand the gist of an application. By way of example, a typical definition of a real 'class'[2] inside a Rails application would look like:

```
class  Project < ActiveRecord::Base
        belongs_to                      :portfolio
        has_one                         :project_manager
        has_many                        :milestones
        has_and_belongs_to_many         :categories
        validates_presence_of           :name, :description
        validates_acceptance_of         :non_disclosure_agreement
        validates_uniqueness_of         :key
end
```

[2] the term 'class' is used here to describe a representation that models in code what its real life equivalent would be or do.

While the immediate function of some of this code will not be readily apparent, a great deal of information is expressed there, in a readable form, which is perfectly understandable in terms of its remit. This class tells Rails' ActiveRecord (the part of Rails which will talk to the application's database) that it will need to contain projects as part of people's portfolios; that all projects should have a project manager; that projects will probably have more than one milestone, etc. This class will also ensure that when a project is started, people have to agree to a non-disclosure agreement.

With a way to model the shape of an application's world as direct as this, Rails does offer a developer of even minimal programming skill (albeit probably knowledgeable in HTML) a way to start working and seeing results right away. Frameworks like Cocoon potentially offer a way to do this with no programming whatsoever, since the developer has only to interact with XML configuration files. However, the ease with which Ruby presents its functionality is very powerful, and one could argue, easier to deal with than XML configuration. At this point, it almost becomes a matter of taste rather than comparative difficulty; it becomes a matter of choosing between Convention and Configuration. Below, we look at how Rails made this choice.

5.1.1 DRY and Convention over Configuration

As outlined in 'Agile with Rails' (Thomas and Hansson, 2005), the design of the Rails framework was influenced by two key principles: DRY (Don't Repeat Yourself, originally discussed in Venners, 2003) and Convention over Configuration. DRY is essentially about avoiding duplication: a developer should only have to express a piece of functionality once, and only in one place. Convention over Configuration is essentially achieved by establishing sensible defaults for each basic aspect of an application. Thus, if the developer follows the established conventions, she will be able to build an application rather quickly and using a relatively small amount of code. However, it should also be noted that convention can be overwritten, providing increased flexibility for more advanced users.

Together with the principles, Rails gives the developer a set of code generation tools which can help her quickly build the skeleton of a Web application, creating a set of folders and files which immediately present her with a working environment, and a set of best practices which follow the MVC pattern.

5.1.2 Routing (versus sitemapping)

We have seen in Cocoon how a sitemap gives the developer a way to structure the pathways so that the application will deliver its functionality and content. In Rails, incoming requests are sent to a 'router'. The router handles the request and eventually identifies an Action to perform (Actions reside in the Controller part of MVC). This Action may cause other Actions to spring up, or may interact with the database, eventually preparing a View which will be presented to the user through the browser (see figure 6).

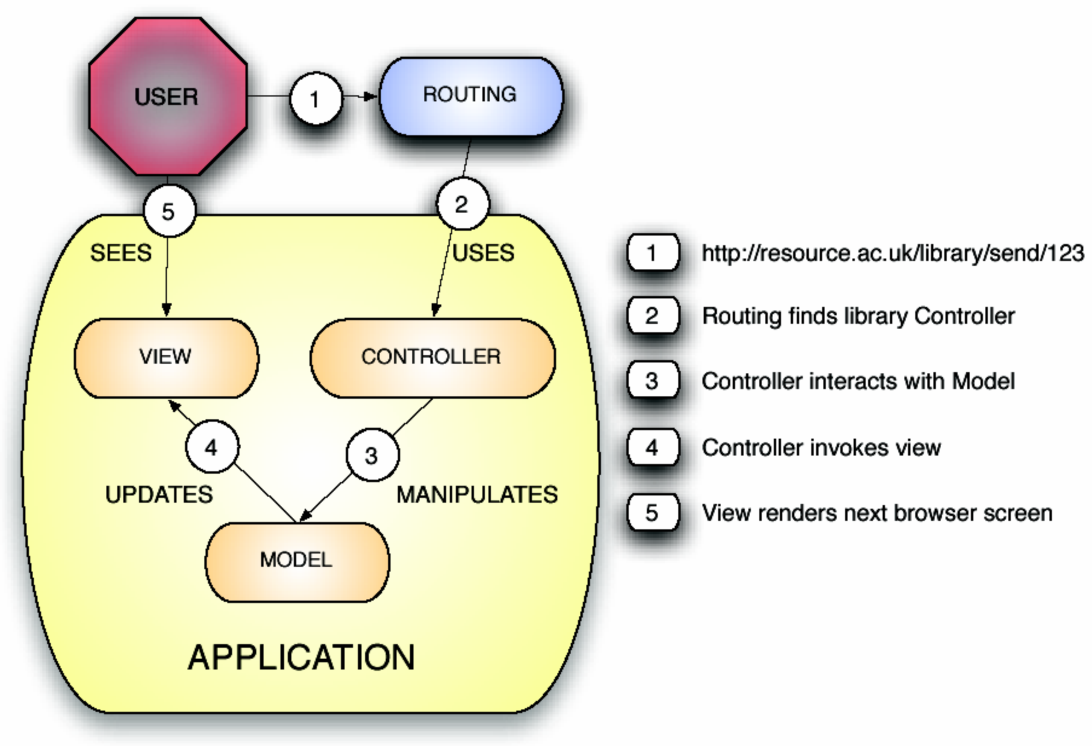

Figure 6: MVC in a Rails application.

5.2 In a nutshell

While the type of project within FE/HE that uses large bodies of data in XML formats might consider Cocoon as a more suitable option, Rails offers a fast solution to the developer who needs to get up and running with a typical, database-driven Web application. However, Rails is a new framework and in a sense the jury is still out on its uptake in the development community. While it implements the MVC pattern well, it does however mix concerns somewhat, in that a developer will be asked to embed logic into the View, in .rhtml files which contain not only HTML but also Ruby code. Dave Thomas (Thomas and Hansson, 2005) defends this possible deviation from MVC:

> 'By embedding code in the view we risk adding logic that should be in the model or the controller. This complaint is largely groundless: views contained active code even in the original MVC architectures. Maintaining a clean separation of concerns is part of the job of the developer.'

Finally, there has been some resistance from the Java community and there remain questions about scalability and enterprise-readiness (Tate, 2005). However, Rails offers developers with the need to rapidly create applications a way to get really far very quickly, and to adapt to change in a timely manner. This, in turn, means that the interaction between developers and project managers in trying to fulfil academic objectives can become more agile and less costly.

6. LOOKING AHEAD - SEMANTIC COMPONENT DISTRIBUTION

6.1 The future of frameworks – reuse and automation

In this report, we have discussed the general issues surrounding Web application frameworks and investigated two example frameworks in some depth. Such frameworks can greatly assist the development of applications by providing a skeleton architecture of *reusable* solutions to common Web application issues. As a software engineering technique they represent one of a class of reuse solutions that also includes individual software components[3] and design patterns. Although the reuse of individual software *components* is seen as the original vision of reusable software, the use of frameworks is seen as increasingly important. Frameworks can provide 'a reusable *context* for components' [italics added] (Johnson, 1997, p. 2) in which different components can co-operate. Developments in frameworks have reduced the amount of programming expertise needed to create sophisticated Web applications.

In the longer term, it may be possible to envisage systems that will provide much more automation in the development process, to the point where they become almost semi-automated. Looking even further into the future we could even imagine a scenario where a developer with a non-expert skillset could build and develop applications using sophisticated tools that require no programming knowledge and no detailed understanding of XML to be able to configure them. Such a tool might allow the user to simply define the user story or cases and then use an application framework to automatically build the application. The tool, guided by the developer, would interact with existing frameworks of code and components, software agents and Web Services to create an application (Griss and Pour, 2001). For example, within the HE environment we could envisage a process in which software agents eventually crawl resources such as JISC's Resource Discovery Network (RDN) to gather needed components, while being informed by interoperability agents from CETIS about possible conflicts. Following this process, interoperability rules pertaining to their use and make-up, including DRM descriptions, copyright alerts, possible dependencies and FOIA rules would then be gathered.

Such a semi-automatic development scenario is some way off and one in which non-programmers can automatically build systems is even further away. Software reuse has been widely studied and explored for a number of years but is clear that there remains significant problems and barriers (Garlan et al., 1995; Schmidt, 1999). However, the concepts and architectures that could potentially enable such a process to occur are still being actively researched. Work, for example, has been done in the past on the automatic generation of Web applications from design diagrams written in UML (Kraus and Koch, 2002) and research has been undertaken on the use of the Semantic Web for software reuse and component retrieval (Yao and Etzkorn, 2004).

6.2 Semantic Web

The vision of the Semantic Web as originally expressed by Tim Berners-Lee was of a world where, instead of people trawling laboriously through information on the Web and negotiating with each other directly to carry out routine tasks such as scheduling appointments, finding documents and locating services, the Web itself could do the hard work for them. The vision is that this can be done by providing sufficient context *about* resources on the Web and also providing the tools to use this context so that machines (or 'software agents' – programs working on behalf of people) can find the right things and make decisions. Much of this is envisaged to be achieved by 'imbuing the Web itself with *meaning*' (Matthews, 2005, page 1) through semantic description. The provision of such semantic descriptions of software components, services and objects will assist the automation of the Web application building process in that, ultimately 'users should be able to describe their needs or goals in a convenient fashion, and automats should be able to identify, invoke, compose and monitor services to accomplish the user's goals' (Peer and Vukovic, 2004, page 1).

[3] A component is a unit of software that encapsulates a certain software function

The Semantic Web as currently constructed does not yet allow non-programmers to automatically develop Web applications, and there are certain aspects of the Semantic Web, such as Description Logics, that are challenging even to hardened coders (see, for example, Rector et al., 2004). However, it does provide new ways of reusing existing, distributed code components, agents and modules, because such entities can be *well described*. As the description of entities is part of the makeup of the Semantic Web, the agents could use these descriptions as long as the tags embedded in the Web pages that advertise the components follow a schema that the agent recognises. The agent will then be able to 'harvest' these components and offer them as a choice to the user. As an indication of the potential for this kind of work researchers are beginning to look at ways in which Web application components (such as Cocoon Blocks, or Rails Helpers) could possibly be distributed using Semantic Web technologies.

6.3 The role of RDF

One potential advantage of Semantic Web technologies is that existing Web applications that process XML data contain a lot of information about the data and how it relates to the real-world. This information is hard coded into the application, but if made more 'visible', and available as data, it would be valuable for the reuse of these existing applications. This could be achieved by the use of the Resource Description Framework (RDF), a data modelling language that is used to define metadata. It provides a formalized data model and makes it possible for some of this information to be represented as data, using schemas or ontologies, and presented to the user (Bizer et al., 2005).

This has started to happen, with browsers such as Firefox already making their components describe themselves with RDF (see: http://www.mozilla.org/rdf/doc/) and it is also possible to foresee that projects like Cocoon will at some point in the future make use of RDF to describe their Blocks. The link could therefore be made between Firefox as a browser (an information and content discovery tool) and software components described in a common, resource description language, in that, at some point, one should be able to use the browser to gather application components and perhaps even to use them *in situ* to build a service.

However, there are a number of problems with creating well-described tags for pieces of information including software services and components, but a discussion of these problems is beyond the scope of this report. However, we include a brief introduction to two approaches that are being attempted by the Simile Project at the Massachusetts Institute of Technology (MIT) that attempts to address some of these difficulties.

6.3.1 An early example of RDF 'scrapers': the PiggyBank project

PiggyBank is an extension of Firefox that turns it into a Semantic Web browser. It extracts information from existing Web pages and stores the data in RDF format in a ‘bank’. If a resource already contains RDF-described information, PiggyBank will be able to retrieve it; if it doesn’t (as is most common with the majority of current website resources), it will deploy screen scrapers, a piece of XSLT or Javascript code which will harvest information that PiggyBank can use. Once data has been extracted it can be viewed through PiggyBank's own interface and it can be tagged with additional information by the user. Tagged items in the banks can be shared between users. The main bank can be found at http://simile.mit.edu/piggy-bank/.

The system therefore facilitates sophisticated manipulation of Web-based content and data. As the project pages describe:

> 'Having extracted the 'pure' information and stored it on your computer, Piggy Bank can now apply its own user interface to let you browse through that information independent of the original Web sites. For example, Piggy Bank can call upon Google Maps to display geographical information even if the original Web sites do not offer cartographic views of their data.' (http://simile.mit.edu/piggy-bank/introduction.html).

6.3.2 Shared RDF data collected by individuals: Semantic Bank

Whilst research continues into agents that can interact with the Semantic Web in the ways outlined in the previous section, another mechanism is being explored – the use of 'people power' and the emerging folksonomy[4] culture. In these systems, ordinary people tag content with descriptions and metadata and then share these within social communities. These systems represent 'serendipitous categorization' (Simile project website, http://simile.mit.edu/semantic-bank/) and a distributed classification system. They don't pretend to offer the precision and lack of ambiguity of a controlled vocabulary domain but, instead, take the approach that people may willingly re-order an unstructured data mass and that this can produce a more usable and efficient solution (Mazzocchi, 2005). Popular examples of this approach include del.icio.us and Flickr.

CONCLUSION

The Web has moved far from its roots as a simple text and image distribution system, and developing modern, sophisticated Web applications can be a difficult and expensive process. As we have seen, the use of such systems in Higher Education has tremendous benefits for users of Web-based systems but requires projects to use developers with high-levels of Web development experience and advanced knowledge of technologies such as Java and scripting languages. The introduction of Web frameworks somewhat helped this problem by providing coding skeletons upon which systems could be built, and this report has outlined two such frameworks, which, it is argued, offer considerable advantages to HE developers. There is, however, a very considerable way to go before non-programmers can specify and develop their own Web applications in a simple manner. Such systems have been imagined for a number of years with various proposals for generating code directly from design diagrams and use cases.

This report has argued that parallel developments in the Semantic Web may offer a way forward for the future development of such systems. The annotation of application components and objects with metadata regarding their function and purpose could help application builders with the automatic location of such components. Such annotation, either automatically by 'scrapers' or through the emerging folksonomy culture, is being explored. Recent developments in this area, such as the Simile project, will provide indicators of the viability of these approaches.

[4] A combination of the words folk (or folks) and taxonomy, the term folksonomy has been attributed to Thomas Vander Wal. Please see http://en.wikipedia.org/wiki/Folksonomy

References

BECKER, O., BROWN, P., CIMPRICH, P. 2003. *An introduction to streaming transformations for XML.* **XML.COM** website, http://www.xml.com, February 26. O'Reilly. Available at: http://www.xml.com/pub/a/2003/02/26/stx.html [Last accessed: 16/12/05].

BIZER, C., LEE, R., PIETRIGA, E. 2005. *Fresnel – a Web browser- independent presentation vocabulary for RDF*. **Proceedings of the 4th International Semantic Web conference [**forthcoming]. Springer. Also available online at: http://simile.mit.edu/repository/fresnel/trunk/docs/papers/wsISWC/fresnel.pdf [last accessed 10/12/05].

CARRIÇO, L., LOPES R., RODRIGUES, M., DIAS, A., ANTUNES, P. 2003. *Making XML from Hypermedia Models*. **Proceedings of the IADIS International Conference on WWW/Internet 2003**. International Association for Development of the Information Society. Also available online at: http://www.di.fc.ul.pt/~paa/papers/icwi-03.pdf [last accessed 12/12/05].

CAVANESS, C. 2005. *What is Struts*. **OnJava.com website,** http://www.onjava.com, 11th Feb. O'Reilly. [Last accessed 01/12/05].

DAVIS, M. 2001. *Struts, an Open Source MVC implementation*. **DeveloperWorks website,** http://www-128.ibm.com/developerworks/ , 1st Feb. IBM. Available online at: http://www-128.ibm.com/developerworks/java/library/j-struts/ [last accessed 02/12/05].

DIJKSTRA, E. 1982. *On the role of scientific thought*. **Selected Writings on Computing: A Personal Perspective**. Springer-Verlag. Also available online at: http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html [last accessed: 15/12/05].

DOYLE, B., LOPES, C. 2005. *Survey of Technologies for Web Application Development* [forthcoming]. ACM Press. Available online at: http://www.isr.uci.edu/~lopes/teaching/ics123F05/notes/web_programming_survey.pdf [last accessed 12/12/05].

DRAKE, J. 2001. *Programming in the Ruby language*. **DeveloperWorks website,** http://www-128.ibm.com/developerworks/, 1st July. IBM. Available online at: http://www-128.ibm.com/developerworks/linux/library/l-ruby1.html [last accessed 14/12/05].

EAGLE, M. 2004. *Wiring your Web Apps with Open Source Java*. **OnJava.com website,** http://www.onjava.com. O'Reilly. Available at: http://www.onjava.com/pub/a/onjava/2004/04/07/wiringWebapps.html [last accessed 10/12/05].

FAYAD, M., SCHMIDT, D. 1997. *Guest editorial*. **Communications of the ACM**: **Special issue on Object Oriented Application Frameworks,** vol. 40, no.10, October 1997. ACM Press. Also available online at: http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html [last accessed 05/12/05].

FOWLER, M., BOX, D., HEJLSBERG, A., KNIGHT, A., HIGH, R., AND CRUPI, J. 2004. *The great J2EE vs. microsoft.NET shootout*. **Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '04).** ACM Press.

GARLAN, D., ALLEN, R., OCKERBLOOM, J. 1995. *Architectural Mismatch: Why reuse is so hard*. **IEEE Software**, vol. 12, iss. 6, pp. 17–28. IEEE.

GRISS, M., POUR, G. 2001. *Accelerating Development with Agent Components*. **IEEE Computer**, vol. 34, iss. 5, pp. 37-43. IEEE.

HIBBS, C. 2005. *Rolling with Ruby on Rails.* **OnLamp.com website,** 20th Jan. O'Reilly. Available online at: http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html [last accessed 12/12/05].

JOHNSON, R. 1997. *Frameworks = components + patterns*. **Communications of the ACM**. October 1997, vol. 40, no. 10. ACM Press.

KAY, R. 2005. *Ruby on Rails.* **Computer World,** 28th November. IDG. Available online at: http://www.computerworld.com/printthis/2005/0,4814,106458,00.html [last accessed:15/12/05].

KERNER, S. 2004. *A new Cocoon from Apache*. **InternetNews website**, http://www.internetnews.com/, 27th May. Available at: http://www.internetnews.com/dev-news/article.php/3360611 [last accessed 07/12/05].

KNIGHT A., DAI, N. 2002. *Objects and the Web*. **IEEE Software**. March/April 2002, pp.51-59. IEEE.

KRAUS, A., KOCH, N. 2002. *Generation of Web applications from UML Models using an XML Publishing framework.* In: EHRIG, H., KRAMER, B., ERTAS, A. (eds) **6th World Conference on Integrated Design and Process (IDPT 2002)**. Integrated Design and Process Technology.

MATTHEWS, B. 2005. *Semantic Web Technologies*. **JISC Technology and Standards Watch.** April 2005. JISC.

MAZZOCCHI, S. 2002. *Introducing Cocoon 2.0.* **XML.com website,** http://www.xml.com, February 26. O'Reilly. Available at: http://www.xml.com/pub/a/2002/02/13/cocoon2.html [last accessed 12/12/05].

MAZZOCCHI, S. 2005. *Data First vs. Structure First*. **Stefano's Linotype** (blogg**)**, July 28. Available at: http://www.betaversion.org/~stefano/linotype/ [last accessed 14/12/05].

MERTZ, D. 2005. *Fast-track your Web apps with Ruby on Rails.* DeveloperWorks website, http://www-128.ibm.com/developerworks/, 14 Jun. IBM. Available at: http://www-128.ibm.com/developerworks/linux/library/l-rubyrails/ [last accessed 04/12/05].

MILLER, G. 2003. *.Net vs J2EE*. **Communications of the ACM,** June 2003, vol. 46, no. 6. ACM Press.

MORISIO, M., ROMANO, D., STAMELOS, I. 2002. *Quality, Productivity and Learning in Framework-Based Development: an Exploratory case study*. **IEEE Transactions on Software Engineering**, September 2002, vol. 28, iss. 9, pp. 876–888. IEEE.

PEER, J., AND VUKOVIC, M. 2004. *A Proposal for a Semantic Web Service Description Format.* In: ZHANG, L. J., JECKLE, M. (eds) 2004. **Proceedings of the European Conference on Web Services 2004 (ECOWS '04).** pp. 285–289. Springer-Verlag.

PRESSMAN, R. 2000a. *Web Engineering*. **Software engineering, a practitioner's approach**. 5th edn. McGraw-Hill: Maidenhead, UK. pp. 813-839.

PRESSMAN, R. 2000b. *Client and Server e-commerce software engineering*. **Software engineering, a practitioner's approach**. 5th edn. McGraw-Hill: Maidenhead, UK. pp. 764-812.

RAYMOND, E. *Interfaces*. **The Art of Unix Programming.** Addison Wesley.

RECTOR, A., DRUMMOND, N., HORRIDGE, M., ROGERS, J., KNUBLAUCH, H., STEVENS, R., WANG, H., WROE, C. 2004. **OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns.** *Available at:* http://www.co-ode.org/resources/papers/ekaw2004.pdf [last accessed 19/12/05].

RUBY, 2005. Ruby Language website, http://www.ruby-lang.org/en/. [Last accessed 17/12/05].

RUSTAD, A. 2005. *Ruby on Rails and J2EE: Is there room for both*? **DeveloperWorks website,** http://www-128.ibm.com/developerworks/, 12th July. IBM. Available online at: http://www-128.ibm.com/developerworks/linux/library/wa-rubyonrails/?ca=dgr-lnxw16RubyAndJ2EE [last accessed 15/12/05].

SAX, 2005. Sax project website, www.saxproject.org. [Last accessed:16/12/05].

SCHMIDT, D. 1999. Why Software Reuse has Failed and How to Make it Work for You. C++ Report, January 1999. Now only available at: http://www.cs.wustl.edu/~schnidt/resuse-lessons.html [last accessed 16/12/05].

SRINIVASAN S. 1999. *Design Patterns in Object-Oriented Frameworks*. **Computer.** February 1999, vol. 32, iss. 2, pp. 24–32. IEEE Computer Society Press.

TATE, B. 2005. **Beyond Java**. O'Reilly Books.

THOMAS, D., HANSSON, D. H. 2005. **Agile Web Development with Rails.** Pragmatic Bookshelf.

VENNERS, B. 2003. *Orthogonality and DRY principles*. **Artima Developer website**, http://www.artima.com/index.jsp, March 10th. Available at: http://www.artima.com/intv/dry.html [last accessed 05/12/05].

WESTKÄMPER, T. 2004. **Architectural models of J2EE Web tier frameworks**. Master's Thesis. University of Tampere Department of Computer Science. Also available online at: http://www.cs.uta.fi/research/ theses/masters/Westkamper_Timo.pdf [last accessed 14/12/05].

WILLIAMS, J. 2003. *The web services debate J2EE vs. .Net.* **Communications of the ACM,** June 2003, vol. 46, no. 6, pp. 59-63. ACM Press.

WOO, J. 2003. *The Comparison of J2EE and .NET for e-Business*. **The Technical Report (hipic-10292003) of High-performance Information Computing Center**. California State University. Available online at: www.calstatela.edu/faculty/jwoo5/ publications/hipic-techreport-102912003.pdf [last accessed 17/12/05].

YAO, H., ETZKORN, L. 2004. *Towards a semantic-based approach for software reusable component classification and retrieval.* **Proceedings of the 42nd ACM annual Southeast regional conference**. ACM Press.

About the Author

David Casal works as a Director of Research for an Open Source services company called Luminas, and is usually found there trying to facilitate people's transition into Open Source practices.

David's other interests tend to focus on music (his undergraduate focus) and new compositional paradigms including improvisation using elements of computing. His PhD is on hold while he hunts for the perfect programming language with which to accurately represent 'Leggierissimo'.