Web Application Frameworks

by Gideon Sireling

August 2008

there are lies, damned lies, and comparative studies

Table of Contents

Introduction

History

HTML and the HTTP protocol were originally designed as a way of presenting, and hyperlinking, static documents (resources) on the Internet. A URL (Uniform Resource Location) such as www.example.com/foo/bar.html identifies a resource /foo/bar.html on host www.example.com. When dealing with static content, the resource path almost always corresponds to a relative path on the host's file system.

However, as the Web developed, users clamoured for a more dynamic, interactive experience. Rather than simply serve static documents, Web servers would now be expected to respond to users; serve them pages personalised to their preferences, and respond to information POSTed. To handle this, the original National Centre for Supercomputing Applications httpd server introduced the Common Gateway Interface. A URL such as www.example.com/prog.cgi would point not to a static document, but to an executable program. A standard interface would allow the program to read information about the HTTP request, and provide a response, which would be sent back to the client.

The Problems

CGI defines, as its name implies, an interface between programs (usually written in Perl or C) and the web server. It does not address the problems which web development presents over traditional desktop development:

- HTTP is a stateless protocol. In its typical implementation, there is is no way to preserve state between requests; a POST cannot set variables for use by a subsequent request.
- Websites are client/server, with a high latency (and unreliable connection) between the browser and web server. All the executable code and data resides on the server. (This is not strictly true; JavaScript, flash, and other technologies run on the client. However, in the real world, these are typically inappropriate to anything other than improving the user interface.)
- Users can move back through their browsing history at any time, or duplicate the page into a new window/tab, and follow a different path in each.
- Users may bookmark a web page for later viewing. This captures only the URL used to request it; any state being held by the server may be lost. These last two points effectively allow users to sprinkle code with arbitrary GOTOs and concurrency.

The Solutions

A variety of solutions exist to some or all of these problems, spanning the full range from simple CGI libraries to the colossal JEE. These Web Application Frameworks are the subject of this study.

Related Work

Introduction

There is an incredible paucity of existing comparisons between web application frameworks on different platforms. Each framework has the usual fan base extolling its virtues, but there only seem to be two serious studies. However, there are a number of comparative reviews of Java frameworks.

Better Web App Development

NASA's Jet Propulsion Lab produced an entertaining comparison entitled Better Web App. The comparison was based on a simple time logging system, built with four different frameworks; JEE, Zope, Django and TurboGears.

There are two severe problems with NASAs comparison which prevent it being useful in the context of this study.

- Their sample application is incredibly trivial - even more so than ours. There is no state, and no navigation - there is in fact only one page.
- Lines of code (particularly XML) are used as a highly significant metric, with little regard for how much effort these lines require.
- The comparison's conclusions are in favour of Zope, a content management system built on top of the Plone application framework. This is largely due to Zope's support for authentication and internationalisation, issues which we have specifically excluded.

The Infamous .NET Pet Shop

Sun Microsystems developed the Java Pet Store as a demonstration of how applications should be designed for J2EE. It uses the example of a functioning online pet store.

To demonstrate the alleged superiority of their .NET platforms, Microsoft developed the .NET Pet Shop. They ran benchmarks, showing significantly higher performance for .NET, with much less programming effort. The original comparison seems to have disappeared from its original URL, but a related document can be found on MSDN.

The comparison unleashed a storm of protest. (Unfortunately, most of the original URLs are no longer active, so citations are lacking.) The key complaints were:

- The comparison used an outdated system for the Java Pet Shop.
- Java is designed to be multi-platform, whilst .NET is Microsoft only. This means that the Java solution requires more abstraction to cover platform differences. Crucially, the Java solution emitted dynamic SQL from a data access layer, whereas the .NET contender used SQL Server stored procedures.
- Again, a naive line count is given undue significance.
- The Java Pet Store was never intended to be the most efficient solution for the given problem. It uses a simple example to demonstrate the architectural capabilities of J2EE, something only required in full by much more complex requirements. Microsoft designed the .NET Pet Store as the most efficient solution.

Clearly, this commercially motivated comparison cannot be used in the context of an objective study. (A response from Microsoft appeared in TechRepublic.)

Comparison of Java Web Frameworks

Comparison of Java Web Frameworks is a paper presented at the 2004 Borland Convention by Neal Ford. Ford begins by presenting Model 2, a Java implementation of MVC. He then presents the architectures of Struts, Turbine, Tapestry, WebWork, Velocity, and InternetBeans Express. Ford's paper is about architecture, with little attention is paid to the actual experience of developing a web application. Unfortunately, the frameworks reviewed are all considerably outdated, and the paper is now only of historical relevance.

Comparing Web Frameworks

Comparing Web Frameworks, by Matt Raible of Raible Designs.com, compares JavaServer Faces, Spring MVC, Stripes, Struts 2, Tapestry and Wicket. A brief list of pros and cons for each is given, followed by the sweet spots (taken from Java Web Framework Sweet Spots, see below). This is followed by a detailed evaluation, where based on the criteria of Ajax Support, Bookmarkability, Validation, Testability, Post and Redirect, Internationalization, Page Decoration, Community and Support, Tools, Marketability of Skills, and Job Count. Raible's presentation is very similar in spirit to this study, and has many more evaluation criteria, but there is no sample application.

An earlier version described the architecture of the tested frameworks, and gave some examples, but like Neal Ford's comparison, it is severely outdated and no longer relevant.

Java Web Framework Sweet Spots

Also of interest is Java Web Framework Sweet Spots, likewise by Matt Raible. This is not a comparison as such, but a series of questions sent to the authors of various frameworks. Although many of the frameworks have evolved considerably since the questionnaire was presented, the philosophies behind a framework to not change as fast as its implementation, and it still makes a very interesting read. The authors were asked about their frameworks' sweet spots, how they compare to others, and a variety of other questions.

Comparing Webapp Frameworks

Simon Brown wrote a series of blog posts on java.net comparing some Java frameworks, beginning with Comparing Webapp Frameworks. He uses plain JavaServer Pages, JavaServer Pages Standard Tag Library, Struts, Wicket, Stripes, and WebWork to implement a simple read-only blog. Like most of the Java comparisons, it is too outdated to be relevant to someone faced with the choice of what to use today. The blog application demonstrates an approach which has been superseded by REST.

Methodology

Web Applications

There are two distinct types of web application:

REST

Many applications are little more than a customised database front end. Users enter data, edit data, and extract data through reports. Whilst these applications can certainly be developed along general guidelines (see the following type), a relatively new application architecture is taking the web development world by storm. Application frameworks such as Ruby on Rails stress Representational State Transfer, typically abbreviated to REST, an the application architecture. A full description of REST can be found in the original paper by Roy Fielding. Each application resource (typically a database row or business object) is represented by a URL, e.g. www.example.com/people/3761. This resource is acted upon by the HTTP verbs; GET will retrieve the data, POST will update some attributes, DELETE will remove it, and PUT (e.g. to www.example.com/people/new) will create a new resource.

Stateful Web

Unfortunately, not all applications fall into this category. Some web sites follow a more traditional work flow, and must maintain state between requests. There are many different ways of achieving this, and the industry has not settled on any one as better than the others.

If an application can be RESTfully architectured, this is an ideal solution. There is no state to maintain, and HTTP provides a perfect interface. Furthermore, REST is cache and web farm friendly (although scalability issues are not part of this study). REST URLs are also clean - they clearly say what they mean. Bookmarking, back buttons and forking are no problem for a RESTful site. As this is a problem which has been adequately solved, it will not be considered further.

The Test

For the purposes of this study, a very simple application will be developed in a variety of frameworks:

1. In the first screen, the web site will ask the user for their forename, and whether they want to give their surname.
2. If the answer is positive, a second page will ask for their surname.
3. Finally, the friendly application will greet the user by name.

Each page (except the first) will have a means of navigating back to the previous page.

For basic security, the user cannot cause markup or script to be executed in any page.

Although this application is trivial, it has two of the critical elements which confound web application developers:

- Each page after the first depends on data gathered by previous pages.
- One of the pages is optional, depending on state. This becomes particularly awkward when navigating backwards, or by bookmark.

Cheating

There are a variety of ways in which we could "cheat", relying on the application's simplicity to take shortcuts. As an extreme example, we could put all the variables into the query string, e.g. simpleapp?first=John&last=Smith. In order to make a sensible comparison, we will use each framework as intended, and treat the application as if it was of moderate complexity.

What are we looking for?

The application should be quick and easy to develop. Ideally, the back button, tab/window duplication and bookmarks will work in an intuitive fashion. Generic error pages or timeouts are unacceptable.

What are we not looking for?

- Object Relational Mapping. Frameworks sometimes include an Object Relational Mapper, e.g. ActiveRecord in Ruby on Rails. Although highly valuable, and worthy of a study in their own right, ORMs are orthogonal to Web Applications. Frameworks do sometimes feature particularly good integration with their packaged ORM, but this would take the study out of scope.
- The line between a Web Application Framework and Content Management System can become blurred. This study will consider internationalisation and document management to be features of a CMS, and are not expected to appear in a Web Application Framework. (Of course, the presence of these features may be desirable in practice.)
- Authentication and user management straddle the line between WAF and CMS. Although such features are desirable, their absence is not considered to be a failing.
- Development time is an important metric; however, this is largely dependent on the developer's skills and experience. Any individual's skills and experience will vary widely across different platforms, and there is no clear way of standardising this. The best we can do is to detail all the steps required to build the application on a particular framework, which should give a reasonable (although unquantifiable) indication of required effort.
- Lines of code, and similar metrics, are popular measurements of development effort. This may be appropriate when comparing similar projects, but when the projects, and particularly the underlying platforms, differ, this metric becomes meaningless. One line of code (or function point) may require much more effort than another, particularly if a tool assisted in its generation.
- Scalability is a complex topic, difficult to test in the real world, and is beyond the scope of this study.

What else should we be looking for?

- Validation is an important framework feature. Values may be required or optional; they may have to fall within a particular range. Validation rules can be complex functions of multiple values. Validation must be performed on the data sent by the browser, but it is also desirable to validate data before it is submitted (using JavaScript). The varying and complex requirements for validation put a thorough evaluation beyond the scope of this study.
- POSTing values to a server requires a fresh page to be sent to the browser in response, even if only a small portion of the page needs updating, creating a poor user experience. The alternative, popularly known as AJAX (Asynchronous JavaScript and XML), uses XMLHttpRequest to transfer small of amounts of data between script running within the page and the server. AJAX is an important technology, and the support (if any) provided by a web framework should certainly be considered, but is beyond the scope of this study.

Meet the Frameworks

JavaServer Faces

Sun Microsystems released the Java platform in 2000. It consisted of three components; a language, a runtime, and standard libraries.

The Java language was intended to be a replacement for the ubiquitous C++. The language was drastically simplified, eliminating many of C++'s more complex features and inconsistencies. Furthermore, every data type except a handful of built in primitives is a class, and no methods can be created outside of a class, leading some to jest that OO stands for Object Obsessed. Low level operations and casting were severely curtailed to enhance the language's safety. Like C++, Java is statically typed.

Java was originally intended to be interpreted. Instead of being compiled to native machine instructions, it was compiled to bytecode for the Java Virtual Machine. Many JVMs were created for different hardware and operating systems, but all of them ran the same bytecode, making Java trivially cross-platform. To improve performance, JVMs now perform just-in-time compiling.

Installations of a Java Runtime Environment include, in addition to a JVM, the standard libraries; a huge collection of utility classes available to every Java program. There are in fact three versions of the Java platform:

- Java Micro Edition for portable end embedded computers.
- Java Standard Edition for desktops.
- Java Enterprise Edition for servers.

The language is the same in each case, but the standard libraries differ. This study examines the use of JEE's framework for web applications, JavaServer Faces.

Tapestry

Tapestry was the first attempt at simplifying JEE development. Howard Lewis Ship, working independently, wrote a new MVC framework based on pages and components. Previous versions of Tapestry used a lot of XML glue, but Tapestry 5, a complete rewrite, has dispensed with this nuisance.

Stripes

Stripes was born out of frustration with the sheer number of artefacts and configurations needed in existing frameworks. A "Hello World" page in a classic JEE framework requires a JSP page, a backing bean, an action bean, and a fairly incredible amount of XML configuration before anything will work.

An equivalent JSP in Stripes will generally display correctly without any further work. If an ActionBean is added in line with Stripe's conventions, it will be automatically wired up to the page without the need for any configuration. When conventions do not achieve the desired result, all configuration can be performed with annotations in the bean code.

RIFE

The developers at Uwyn ("Use What You Need") like Java, but they do not like the way that everyone else applies Java to the web. Thus, they wrote their own open source framework, RIFE. RIFE runs on any Java application server, but does not use any standard Java web frameworks. Instead, they have some very interesting - and very unique - ideas of their own.

Wicket

Whatever the problem, Wicket's solution is a component. Wicket models a web site as a set of components, and all state is persisted as component state. Like many of the newer frameworks, XML is eschewed. Literal HTML is held in templates, everything else is done in Java.

ASP.NET

Microsoft perceived the exploding popularity of cross-platform Java as a threat to their proprietary Windows operating system. To combat this threat, they developed Visual J++; an implementation of Java with extensions to make Windows programming easier. Sun sued Microsoft for breaking the Java standard (which violated their license), and the eventual settlement barred any further development of J++.

Instead, Microsoft set to work on a new platform. The Java Runtime environment was replaced with the Common Language Runtime, a JIT compiler for Common Intermediate Language bytecode. From the start, Microsoft had intended multiple languages to target the CLR. (Although this is certainly possible for the JVM, it had not been Sun's intention.) Nevertheless, the lingua franca of .NET is Microsoft's C#. Although marketed by Microsoft as an evolution of C++, it is actually an evolution of Java; a litigation-free form of J++.

Microsoft do not differentiate between desktop and server versions of the runtime package, although ASP.NET, the web server component, will only be installed if IIS (Microsoft's web server) is present. (The ASP portion of ASP.NET indicates that it succeeds Active Server Pages, an older Microsoft technology.)

ASP.NET MVC

Another popular movement now threatens Microsoft's dominance; the use of dynamic languages, such as Ruby and Python, and the burgeoning popularity of associated frameworks, such as Ruby on Rails and Django.

Microsoft's response was the ASP.NET Extensions project, a suite of technologies intended to bring the high productivity of these frameworks to their .NET developers. Chief amongst these is ASP.NET MVC, a model-view-controller framework to replace the problematic Web Form lifecycle. (Problems with the standard implementation of ASP.NET are discussed later.)

PHP

Whereas Java and C# are object orientated and statically typed, PHP (originally Personal Homepage Processor) is a dynamically typed procedural language, to which OO features have been recently added. PHP was created to script web servers, and the standard library has rich support for this. PHP is much simpler than JSF or ASP.NET, and is cross-platform.

JSF and ASP.NET could be considered descendants of C++ CGI programming, whereas PHP continues the Perl legacy. PHP is maintained by The PHP Group.

Ruby on Rails

Ruby on Rails is a relative newcomer to web programming, and represents the new paradigm of coupling dynamic programming languages with REST architectures. Ruby is a dynamically typed language in which everything is treated as an object, much like Smalltalk or JavaScript. Ruby's dynamic properties are extensively exploited by Rails, particularly in database persistence, where data access objects are created on the fly to match the schema.

The Rails framework aggressively promotes REST, but this study will investigate how well it performs outside of its comfort zone. Rails is maintained by 37signals.

Seaside

Alan Kay famously stated, "I invented the term Object-Oriented, and I can tell you I did not have C++ in mind." What, then, did he have in mind? Smalltalk, which he developed over various iterations from 1969 to 1980, when the modern Smalltalk-80 standard was published. (Although he expresses some concerns about Smalltalk as well.)

Seaside is very, very different. It is written in Smalltalk, a language rarely seen outside of educational establishments and research laboratories. (Seaside is actually written in Squeak, but it has been ported to many other Smalltalk dialects.) There are no HTML files. Instead, like Smalltalk itself, every part of the web application is a live object residing in the image. Components produce HTML programmatically.

The key feature of Seaside is its use of continuations. In almost every other web framework, the execution context which renders an HTML page dies after the response is sent to the browser. Upon receiving a request, particularly with component-based frameworks, the server may attempt to recreate the old execution. In contrast, Seaside saves the execution context as a continuation, which may live across many request-response cycles.

WASH

Haskell is a functional language, much loved by academics but rarely seen in practice. It is a language of few libraries and many papers.

WASH stands for Web Authoring System Haskell. Written by Peter Thiemann, it is formed of several modules:

- CGI
- HTML
- Mail
- DB
- WSP (WASH Server Pages)

The CGI module, used in this study, provides comprehensive support for web programming; a Haskell program calls the library functions to serve CGI requests.

Arc

Arc is the new kid on the Lisp block. Designed by Paul Graham, it has the modest goal of becoming the premier language for the next hundred years.

Arc is in an early development stage, and would not appear to be a suitable subject for a comparative study. However, the inventor has issued a challenge to do just that.

Commercial Usage

JEE, perhaps the first commercial framework, is still broadly perceived in the IT industry as the most enterprise grade framework; something which should be used for global financial systems and multinational conglomerates. JSF and Tapestry are both common implementation of JEE, although there are many others not investigated here, and custom code using JSP as the presentation layer is still common.

Coming later, ASP.NET has firmly entrenched itself amongst small and medium sized businesses, although it is edging up into larger enterprises, and taking market share from JEE in all sectors.

Even older than JEE, the free and open source PHP is wildly popular amongst hobbyists, and utilised extensively by smaller organisations. Ruby on Rails, also free and open source, is mostly used by hobbyists, although there is growing interest from small business. The remaining frameworks are rarely seen in actual use.

Framework Versions Used in this Study

- Java Enterprise Edition 5 (JavaServer Faces 1.2)
- Tapestry 5
- Stripes 1.4.3
- RIFE 1.6.1
- Wicket 1.3
- ASP.NET 3.5
- PHP 5.2.3
- Ruby on Rails 2
- Seaside 2.8
- WASH 2.12
- Arc Anarki, based on arc2

JavaServer Faces

Architecture

Java Enterprise Server

A web server which holds containers for JEE components.

Web Container

The Web Container contains JSP pages and servlets, as well as the necessary configuration files.

Java Servlet

A servlet is a Java class which enables the server to respond to particular requests.

JavaServer Pages

Rather than coding servlets as pure Java classes, a JSP page can be created. A JSP page contains static text, as would an HTML page, and dynamic content. The dynamic content may consist of delimited blocks of Java code, or tags. When a JSP page is first requested, the server compiles it into a servlet; it is the servlet which actually fills the request.

Custom Tags

Including Java code in a JSP page mixes logic with presentation, and does not bode well for code reuse. Instead, a library of Java classes confirming to documented conventions can be compiled into a tag library. The library functions are called from a JSP page with an XML-compliant tag syntax. A standard tag library is provided as part of the JSF platform.

JavaServer Faces

Although now a key pillar of high-level Java web development, JavaServer Faces were only added to the JSF specification in 2004, in response to Microsoft's ASP.NET. JavaServer Faces use the same syntax as tags, but are much more complex. They provide many services, including validation and navigation. JavaServer Face components can be bound to JavaBeans (Java classes confirming to documented conventions), which will synchronise their data with the component's properties, and can execute methods in response to component events.

faxes.config

This XML file controls all navigation, and configures backing beans.

State

A backing bean is a Java class which holds data and application logic for a JSF application. A backing bean can be scoped to the session, thereby holding data across page views. (Whether this bean is held in the server's memory, or written to the client as an ASP.NET style ViewState, is configurable for the application.)

Source Files

web/A.jsp

The JSP template for page A.

web/B.jsp

The JSP template for page B.

web/C.jsp

The JSP template for page C.

web/WEB-INF/faces-config.xml

The configuration file, containing the navigation XML and bean declaration.

src/app/Name.java

The backing bean.

Deployment

Deploying JSF applications is a complicated process, best handled by tools. The test application was run from within the IDE.

Tools

A number of high quality IDEs exist for Java. This application was developed with NetBeans, provided freely by Sun. NetBeans has helpful wizards for creating the initial directory structure, and which assist in editing some of the configuration files.

Sun also offers Java Studio Creator, an IDE for visual JavaServer Faces development. This would probably be a better choice for developing complex applications, but is a complexity overkill for this simple test.

Tapestry

Architecture

Page

Tapestry pages are valid XHTML templates. These are automatically wired up to classes of the same name in a pages package.

Component

The Tapestry XML namespace is used to indicate elements in the template which will be automatically created as components. Expansions (similar to JSP expressions) can also be used. (Pages are also implemented as components.)

State

State is generally persisted on the server. @Persist and @ApplicationState annotations indicate data which should be persisted across postbacks, or throughout the session, respectively. The persistence strategy can be customised as required.

Source Files

src/app/Name.java

The model.

src/app/pages/A.java

The backing bean for screen A.

src/app/pages/B.java

The backing bean for screen B.

src/app/pages/C.java

The backing bean for screen C.

web/A.tml

The template for screen A.

web/B.tml

The template for screen B.

web/C.tml

The template for screen C.

Deployment

Deployment is the same as for any other Java web application with custom servlets, and can be performed with standard JEE tools.

Tools

There is no specific tool support for Tapestry 5. However, as it is a Java web application, any standard Java tools can be used. This application was developed with NetBeans.

Stripes

Architecture

JavaServer Pages

By default, Stripes uses custom tags on JavaServer Pages. See the chapter on JSF for a description of this architecture.

Stripes

The Stripes servlet uses conventions, overridden by annotations, to match JSP pages with Java beans (ActionBeans). HTML widgets are represented by JSP tags, and navigation is handled by ActionBean methods returning Resolution objects.

State

By default, Stripes does not persist state. However, applying the @Wizard annotation to an ActionBean will trigger special treatment. State for wizard beans is stored on the page in hidden fields.

Source Files

web/app/A.jsp

The JSP template for page A.

web/app/B.jsp

The JSP template for page B.

web/app/C.jsp

The JSP template for page C.

src/app/App.java

The action bean.

src/app/Name.java

The model.

Deployment

Deployment is the same as for any other JSP application with custom servlets, and can be performed with standard JEE tools.

Tools

There is no specific tool support for Stripes. However, as it is a JSP application, any standard Java tools can be used. This application was developed with NetBeans.

RIFE

Architecture

Elements

The basic unit of a RIFE site is the element. An element represents one web page or component. An element is associated with a template and a Java class; if desired, the Java class can be used as a controller, with a data class serving as the model. Metadata can be defined in the data class, or in a separate class. Metadata includes validation constraints, and an ability to group the model's fields. The element can then define multiple forms in its template, associating each form with a group.

Flow Control

There are two methods for controlling flow in RIFE.

Between elements

- Exit links define endpoints where control may be passed to another element.
- Input links define endpoints from which data may be received.
- Flow links connect endpoints to elements.
- Data links define the data which flows between endpoints.

These links are normally defined in an XML configuration file, although they can all be set in Java.
Within elements

Continuations enable a block of Java code to pause while waiting for a page response, and if necessary, go back to a previous pause. This enables programmatic flow control in a fully imperative style. During pauses, the thread is not actually paused. Rather, a copy of the local state is created (a continuation), and this continuation is activated when the page is posted back to the server. If the program moves back to a previous state, the current continuation is abandoned, and a previous continuation is activated.

Templates

Whereas other templates contain the logic to build themselves, RIFE templates are more passive. They only contain marked fields/blocks and data binding declarations. The controller is responsible for filling the fields, and moving or hiding blocks.

State

RIFE provides configurable state storage to handle server-side data with data links. Continuations are also an excellent solution to maintaining server-side state. Configurable state storage generates a new ID for each request, which brings some of the same benefits as continuations; submitting a browser-cached copy of a previous page will use the old state data.

Source Files

web/Index.html

The application template.

web/participants.xml
Tells RIFE where to find the site's configuration.

web/rife.xml
The site configuration; only one element, which handles its own flow control.

src/app/Index.java
The business and UI logic.

src/app/Name.java
The model class.

src/app/NameMetaData.java
The metadata class, mapping model fields to template forms. Validation would go in here.

Deployment
Deployment is the same as for any other Java web application with custom servlets, and can be performed with standard JEE tools.

Tools
There is no specific tool support for RIFE. However, as it is a Java web application, any standard Java tools can be used. This application was developed with NetBeans.

Wicket

Architecture

Application

The application class, descending from WebApplication, is the site's starting point. getHomePage() returns the home WebPage component.

Component

Everything in Wicket is a component. A component can render itself as HTML, and persists its state (model) across page transitions.

Template

Components can be associated with an HTML template; the file name (preceding .html) must be the same as the component's class name. Elements with a wicket:id attribute are connected to child components.

Model

Any object implementing IModel can be the model for a wicket component.

State

Component state appears to be maintained through a combination of URL, hidden fields, and server state.

Source Files

app/Name.java

The model.

app/App.java

The application.

app/Index.java

A component representing the index page.

app/Index.html

The template for the index component.

app/Index$NameWizard.html

The wizard template.

app/Index$NameWizard$A.html

The template for screen A.

app/Index$NameWizard$B.html

The template for screen B.

app/Index$NameWizard$C.html

The template for screen C.

Deployment

Deployment is the same as for any other Java web application with custom servlets, and can be performed with standard JEE tools.

Tools

A variety of plug-ins are available to add Wicket support to popular Java IDEs. This application was developed with the NetBeans Wicket module.

ASP.NET

Architecture

Internet Information Services

IIS is Microsoft's web server for their Windows operating system.

ASP.NET Server

This is a handler for ASP requests, which integrates into the IIS pipeline.

Web Forms

A typical ASP.NET web page uses Web Form components, and is compiled by the server into an HTTP handler for the URL. Web Form components are invoked with an XML-compliant syntax. They maintain their state between postbacks, and are exposed to the code-behind as objects.

Code Behind

The declarative web form consists of HTML and component markup. It may be associated with a code-behind file, a partial .NET class which is merged with the form during compilation. The code-behind processes component events, and performs any other application logic.

State

There are two primary mechanisms for ASP.NET pages to store state.

ViewState

The ViewState is used by controls to persist any properties which have changed since the control was created. It can be disabled for the whole control, but not by specific properties. In ASP.NET 2, controls will emit essential data even when viewstate is disabled. ViewState is serialised to the page as a hidden input control.

SessionState

The server holds a dictionary of objects in memory for each session, called the SessionState. This is the usual way for application logic to persist data across requests.

Source Files

Default.aspx

This is the web form markup, a mixture of HTML and ASP.NET controls.

Default.aspx.cs

This is the code-behind file, containing all the application logic.

Deployment

All the code behind files are compiled into a .NET assembly. The markup pages are then uploaded to the web server, while the assembly goes into a /bin directory.

Tools

The default IDE for ASP.NET is Microsoft's Visual Studio. Visual Studio creates all the files and boilerplate, and has a WYSIWYG designer for the markup. The text editor provides auto-completion and formatting for markup and code. One menu option will compile and deploy the site.

ASP.NET MVC

Architecture

Internet Information Services

IIS is Microsoft's web server for their Windows operating system.

ASP.NET Server

This is a handler for ASP requests, which integrates into the IIS pipeline.

MVC Routing

The MVC framework intercepts incoming requests, and uses configurable routing to select the appropriate controller.

Model

A .NET representing a business object, with methods for the business logic. Database persistence is also managed by this class.

View

A variety of templating engines can be used, including Web Forms.

Controller

A class which maintains the model, processes client requests, and despatches the correct view.

State

The model is responsible for persisting itself, and the controller uses the model's methods to persist or retrieve data.

Additionally, the ASP.NET's standard session store is available.

Source Files

Controllers/AppController.cs

This is the controller for the App directory. By default, all requests of type /App/A will be handled by AppController.A.

Models/Name.cs

The model; a simple C# class.

Views/A.aspx

The template for page A.

Views/B.aspx

The template for page B.

Views/C.aspx

The template for page C.

Deployment

ASP.NET MVC sites are deployed in the same manner as regular ASP.NET.

Tools

ASP.NET MVC includes full Visual Studio support.

PHP

Architecture

The only layer is the PHP processor, which can be used by any web server as a handler for PHP pages. The global configuration file includes a list of libraries, which are available to all pages.

State

The server holds a dictionary of objects in memory for each session, accessible as the $_SESSION array.

Source Files

.htaccess

Configures PHP when running as an Apache module. An equivalent php.ini should be used under other scenarios.

A.php

The HTML page for step A.

fromA.php

PHP code to process the data submitted by A.html, and decide which page to move to next.

B.php

The HTML page for step B.

C.php

A mixture of HTML and PHP code to produce the final page.

Deployment

All the source files are simply copied to the web server.

Tools

Some third party tools of varying quality exist, but PHP is typically developed with a text editor and an FTP client.

Ruby on Rails

Architecture

Rails is a Ruby program; any server which can call the Ruby interpreter can run Rails.

Rails is based on a Model, View, Controller pattern.

Model

A class representing a business object, with methods for the business logic.

View

A template, with embedded Ruby code, which renders a view of the model.

Controller

A class which maintains the model, processes client requests, and despatches the correct view.

State

Rails encourages the use of a database for all data (through its ActiveRecord object relational mapper), but there is also a session hash available. As stated in the requirements, this application does not make any use of a database, so the model is held in the session.

Source Files

app/controllers/App_controller.rb

This is the controller for the App directory. By default, all requests of type /App/A will be handled by AppController.A, defined in App_controller.rb.

app/models/Name.rb

This is the model; a simple Ruby class defining the data object and access methods.

app/views/app/A.rhtml

The Ruby HTML template for page A.

app/views/app/B.rhtml

The Ruby HTML template for page B.

app/views/app/C.rhtml

The Ruby HTML template for page C.

Deployment

All the source files can simply be copied to the web server. For more complex scenarios, there are third party tools to handle deployment.

Tools

Rails comes with a number of console tools for generating the initial directory structure and boilerplate. The two used here were rails, which generates the directory structure and files for an empty site, and the default generator, to create a controller and view directory. All of the source files listed were written from scratch. Aptana was used as an IDE, but no features other than the text editor were utilised.

Seaside

Architecture

Seaside Server

Seaside commonly uses the Kom HTTP Server, running in a Smalltalk image, although other servers are possible.

Components

All HTML is output by components, which descend from WAComponent.

Tasks

Descendants of WATask define workflow. They call components to interact with the user.

The root of the website is a Component or a Task.

The Model

The model consists of ordinary Smalltalk classes, which can be held in the session, or instance variables of components.

State

The execution context is preserved across postbacks, so state is maintained automatically. A descendant of WASession can hold session-wide data if desired.

Source Files

Smalltalk applications do not exist as independent entities; all the code lives in an image as a set of live objects. Code can be shared by filing out, which produces a file ("goodie") that can be filed into another image. However, there are more sophisticated methods of sharing code, such as the Monticello package manager.

The source for the application is given as a file out, App.st, split by class for easier reading.

Deployment

Deploying a Smalltalk application consists of filing in code, or loading a package. Alternatively, a complete image may be copied to the server.

Tools

Smalltalk has excellent development tools built into the environment. This application was built with Ramon Leon's Squeak image.

WASH

Architecture

HTML

The output page and its elements are all created with Haskell data constructors. These are assembled within the HTML monad, which utilises Haskell's type system to ensure a valid document.

Alternatively, a pre-processor can be utilised to incorporate literal HTML into the Haskell source.

Widgets

Form widgets return a handle to the monad, which is initially type-tagged as invalid.

The CGI Monad

Callbacks can be attached to a form's submit button, or individual fields. When the client submits a form, the widget handle(s) are passed to the callback as a parameter, type-tagged as valid.

State

WASH maintains an interaction log of all the program's interactions in hidden fields. As Haskell is purely functional, the log can be replayed at any time to recover the current state.

Source Files

The entire source is a single Haskell program, Main.hs.

Deployment

The compiled executable is copied into the server's CGI directory.

Tools

There is no actively maintained IDE for Haskell; Emacs or some other generic programmer's editor is typically used with a batch compiler, such as GHC.

Arc

Architecture

HTML

An HTML library writes HTML tag trees to standard output.

Web Server

Request handlers are defined in a very similar way to functions. The web server dispatches requests to a matching handler, and redirects standard out to the response stream.

Closures

The web server can register a closure as the destination of a link or form. In the case of a form, the closure receives a hash table of the submitted values.

State

State is maintained by passing data to closures.

Source Files

The entire source is a single Arc script, App.arc.

Deployment

```
(load "App.arc")
```

Tools

Anarki comes with Arc modes for Emacs and Vim. An IDE for Arc, Arc Welder, is also under development.

Scoreboard

Ease of Development

How much effort was needed to develop the application?

JavaServer Faces

The backing bean and JSP pages were reasonably intuitive, if a little verbose. The configuration files, however, are a source of significant pain. There is a considerable amount of XML in proportion to the size of the application.

Tapestry

A clear and straightforward relationship between the code and intent, with highly informative error messages, makes development a breeze.

Stripes

Ensuring that the source files are in accordance with Stripes conventions was somewhat tedious and error prone. However, an experienced Stripes developer could presumably arrange this without too much difficulty. Once applied, the conventions did ensure fairly simple code.

RIFE

The XML approach requires a great deal of XML, which must be synchronised with the templates and Java code. Not a very attractive proposition, which is why the continuations approach was taken instead.

A considerable quantity of code is required to handle the wiring between the template and model classes, including a metadata class. The code to handle flow control is also rather verbose; altogether, it appears that excess XML has merely been replaced with excess Java. Worse, RIFE continuations have been hacked into a language which cannot properly support them. Although the implementation is ingenious, problems abound. The fullName property had to be moved out of the model class (Name.java), due to excessive difficulty in surviving cloning and continuing, and handled explicitly in the controller class (Rife.java). Handling nontrivial (although far from complex) flow control seems to be something of a black art. However, a wealth of examples are offered, which helps considerably.

Wicket

The everything-is-a-component approach requires some getting used to. The advantage is that, one you understand Wicket components and their models, you understand most of what there is to know about wicket. The disadvantage is that trying to model everything as a tree of Wicket components and models can lead to rather a lot of convoluted code, much of it unrelated to the problem domain. Another downside of Wicket is that all the components have to be created in Java. As can be seen in Index.java, this requires a great deal of code. Every other component-orientated framework creates the components from markup, a much less laborious approach.

The Wicket application actually cheats for flow control. WizardStep.Evaluate() is used to skip screen B; a more complex scenario would require a considerable amount of work to customise the WizardModel.

ASP.NET

This was fairly easy to develop. Only two files were needed, both of which were created by the IDE. Visual Studio's visual designers and excellent text editor made light work of a simple task.

ASP.NET MVC

Although requiring many more files and directories than regular ASP.NET, the Visual Studio project template creates an initial directory structure, and automates the process of adding new files.

PHP

Coding the PHP was a moderate effort. Although there is no complex configuration or boilerplate, there is very limited assistance from the framework; all the mapping and session storage must be hand-coded.

Ruby on Rails

Although all the Ruby source was written by hand, this was highly intuitive. One issue with the model is discussed in the next section.

Seaside

Smalltalk's environment is a joy to work in, although dynamic typing adversely effects the IDE's understanding of code. (For example, you cannot browse to an identifier's class, as this is not statically defined.)

The back and next buttons could have been hard coded into every screen, but Seaside components are so easy to create that it was almost trivial to write a quick-and-dirty wizard component.

WASH

This was a very straightforward program to develop - the single code file approaches the clarity of shell scripting.

Arc

Arc is in a very early development stage, hence many bugs and a rather rudimentary library. Presumably this will improve with time (and Anarki).

The application code itself is very straightforward, similar to WASH, but without the monad plumbing.

Bookmarking

What will happen if the user bookmarks the middle of the application, and returns to it next week?

JavaServer Faces

The server is liable to throw a null reference exception, as the backing bean has not been properly initialised.

Tapestry, ASP.NET MVC, Ruby on Rails

The application will commence from the bookmarked location, with default data.

Stripes

Every page appears with the URL app/App.action, so bookmarking and returning later will simply start the application afresh. Stripes is very forgiving of empty fields, so even if a user went directly to e.g. app/B.jsp, the application will function with default data.

RIFE, ASP.NET, WASH

There is only one URL/template, so bookmarking and returning later will simply start the application afresh.

Wicket

An internal error, as the action URL does not match the form data.

PHP

The application will resume from the middle, with missing data. This is liable to cause an error, when attempting to retrieve a missing POST or SESSION value.

Seaside

The URL contains the application entry point, and a session/continuation identifier. Attempting to return to an old URL will revert to the corresponding state if the continuation is still valid, otherwise the application will begin afresh.

Arc

"Unknown or expired link"

Browser Navigation

What is the effect of using the browser's back and forward buttons, then interacting with the displayed page?

JavaServer Faces

The application will behave as expected, using session data in the server's memory. Entering data, pressing the browser's back button, then the application's next button, will cause the old data to reappear. However, the server can be configured to serialise the session bean to the page as a hidden form field, in which case old data will not reappear.

Tapestry, RIFE, ASP.NET MVC, Ruby on Rails, Arc

The application will behave as expected, using session data in the server's memory. Entering data, pressing the browser's back button, then the application's next button, will cause the old data to reappear. (Although Arc uses closures, only the last closure for a session is kept.)

Stripes, ASP.NET, WASH

As all the state information is streamed to the page in a hidden field, these buttons will work as expected.

Wicket

Information about the application state is stored in hidden fields. Disaster will ensue if this does not remain synchronised with the URL and server data.

PHP

As long as the browser retrieves the page from its cache, the application will simply continue, overwriting existing session data. However, if the browser attempts to retrieve a new copy of the page out of sequence, this may cause an error, due to the expected POST data not being present.

Seaside

The application will behave as expected. Entering data, pressing the browser's back button, then the application's next button, will not cause old data to reappear.

Forking

What if the users forks the session into a second browser window?

JavaServer Faces

The backing bean will be shared between the two instances, because they have the same session identifier. Each session will overwrite the other's data. If the server is configured to serialise the session bean to the page, the sessions will be kept separate.

Tapestry, ASP.NET MVC, PHP, Ruby on Rails, Arc

The state will be shared; each session will overwrite the other's data.

Stripes, ASP.NET, WASH

As the state is held on the page, two independent sessions can proceed simultaneously.

RIFE

The continuations appear to be shared between the sessions; each will overwrite the other's data.

Wicket

The hidden fields, URL and server data will become desynchronised, causing a fatal error.

Seaside

The two sessions will work independently, using different continuations.

Discussion

Scalability

We said that scalability, i.e. how well a site can handle a large number of clients, is beyond the scope of this study. However, there is another type of scaling which we can consider - how well will the framework's architecture scale to large, complex web sites?

JavaServer Faces

Backing beans go a long way to cleanly separating concerns, but a problem arises with navigation. This is actually defined in two different places; complex navigation decisions are made by bean methods, while the actual navigation performed is defined in the application's configuration file. A complex site will probably have dedicated navigation beans, resulting in the navigation logic being needlessly divided between beans and the configuration file. Furthermore, the verbose syntax of the XML configuration would become difficult to manage as the site expands.

Tapestry, Stripes

A chief design goal of Stripes and Tapestry 5 was to avoid this configuration problem. There is no XML, and much of the configuration is by convention. As can be seen from the source, this goal has been achieved.

Although Stripes, like ASP.NET, stores "wizard" state on the page as hidden fields, the architecture does not tend to excessive state data.

RIFE

The XML links form a neat flow and data graph; with a suitable GUI, this might be manageable. However, it is difficult to see how the XML-serialised graph could be manually maintained for a large site. The situation is much better with continuations, where everything is handled as ordinary Java code.

Wicket

The component and model approach provides very neat encapsulation; large Wicket codebases are easy to manage.

ASP.NET

A design goal of ASP.NET was to make web programming "easy", like visual desktop programming. Hence, the framework will persist control values across POSTs, and events can be wired up, just like desktop GUI applications. Unfortunately, this abstraction is broken. HTTP bears little resemblance to the Windows message pump; user events will be queued up with builtin control events and framework events, then executed in a very complex and poorly documented sequence during POST. (Leon Andrianarivony lists 62 events in the ASP.NET page lifecycle.) Compounding the problem is that control values are not restored immediately, but during one of the framework events. This cascading complexity of interacting events can be very difficult to manage, and cause problems which are very hard to solve.

Additionally, the default behaviour of storing state in the page is only relevant when all the information the application needs is in the current or previous page's ViewState. (ASP.NET makes the state of a POSTing page available to the POSTee.) Anything more complex requires the Session or persistent storage, leaving the developer in the same position as with PHP. In particular, a large amount of information in the ViewState may cause the HTTP requests and responses to bloat to an unacceptable size. Once ViewState is turned off, the developer will also have to manually manage some control behaviours which would otherwise have been automatic.

ASP.NET MVC

The MVC framework has none of these problems. Additionally, strong typing alleviates the problems of Rail's implementation (see below).

PHP

As can be seen from the source, even this simple application has led to a mess of PHP code interspersed with HTML. Whilst there are no intrinsic problems with PHP that would complicate scaling, simply managing the code could become a nightmare.

Ruby on Rails

Rails presents a much prettier picture for the execution of complex site designs. The clean separation of model, view, and controller, makes for a very modular and easily manageable application. A sole concern is the model's data types, as discussed above.

Seaside

Seaside's components are an excellent form of composability; they enable large web applications to be managed with the same ease as a single tier system.

WASH

WASH programs are straightforward Haskell programs, and can be managed in the same way as any other large program.

Like PHP, WASH does not separate code from HTML. However, judicious use of Haskell functions to construct the HTML, along with Haskell's functional composition and excellent combinators, can somewhat alleviate this problem.

The interaction log is a source of concern. This grows at an alarming rate as the application proceeds, and could become a formidable barrier to complex, long-running programs. Furthermore, although performance has been specifically excluded, it must be pointed out that CGI is highly inefficient; the server must start a new process to serve every request. The use of Wash Server Pages is a possible alternative.

Arc

An Arc website is just another Arc program. As in WASH, all the usual techniques for managing large programs are equally applicable to large websites. Similarly, standard combinators can be used to structure the logic and presentation.

Components and Templates

The authors of some component orientated frameworks argue that, by managing state, persistent components bring easier development and better organised code. For example, clicking previous on screen B will fill in the fields on screen A with their correct values, without any specific coding. However, template orientated frameworks generally achieve the same benefits by binding a model class to template fields (except WASH and PHP, which strictly speaking isn't a framework - see below). It would seem that, in most common scenarios, components are an extra layer of complexity which brings little benefit.

Seaside's components, however, are of a different nature. Behaving like the persistent components of a desktop application, they bring the promise of component-based development to the web tier.

Designer Friendly Templates

Some frameworks are of the opinion that HTML templates should be produced by designers, which programmers will then tweak to work with their logic.

This would appear to be a mistake. Business analysts may assist with domain modelling, but do not specify a program's data structures. Similarly, a web designer can specify what a page should like, but should not be responsible for producing the actual HTML templates; this is the programmer's responsibility. Technically minded designers who feel left out can play with CSS to their hearts' content. (HAppS takes this to an extreme; a handler can output Haskell data types, which are automatically streamed into XML, then transformed for display with XSLT.)

Model-View-Controller Pattern

The model-view-controller pattern is an important separation of concerns. Briefly:

Model

Holds all the state information (which may include business logic)

View

Interacts with the user, displaying a particular view of the model and forwarding input to the controller

Controller

Processes user input, updates the model, and selects a view.

We will now investigate the extent to which the various frameworks support this pattern.

Model

The most striking difference between the different frameworks is how they handle the model.

JavaServer Faces, Tapestry, Stripes

Backing beans provide the data model, with automatic conversion between HTTP strings and the model's Java types.

RIFE

RIFE is very flexible; simple Java classes can be used as the model, with the meta data in a separate class, or these can be combined.

Wicket

Any Java classes can be used as the model.

ASP.NET

Although there is no model as such, ASP.NET controls are exposed as strongly typed properties to the code-behind. This works well in simple scenarios.

ASP.NET MVC, Seaside

Any classes can be used as the model.

PHP

There is no model; all data is manually copied into and out of session variables.

Ruby on Rails

The model is a set of classes in the models directory. With a little coding, ActiveRecord can easily keep the model synchronised with the view and returned data. However, studying app/models/Name.rb will reveal a problem. HTTP uses plain text, and the model fields are therefore all set to strings. ActiveRecord will automatically convert these to/from the database types when retrieving or persisting data, but this does not help the application logic. The solution used here was to write an explicit setter for converting enterSurname to a boolean.

The problem here is not dynamic typing; there is no type mismatch for a compiler to find. Rather, the problem is a lack of metadata. All the Java frameworks can recognise (through reflection) that a bound field is boolean, and perform the required conversions automatically. Ruby does not support metadata, so ActiveRecord cannot determine the desired conversion.

WASH, Arc

There is no automatic model. Instead, the session state is held in custom data types, which are passed as parameters to each CGI action/continuation. Widget data must be manually copied to and from the model at each stage.

View

JavaServer Faces, Stripes

Use JSP, a Java templating language.

Tapestry, ASP.NET

Elements in the templates indicate components, which are passed any attribute values by the runtime engine, and generate markup when requested. The runtime then replaces the component declaration with its output.

ASP.NET MVC

Although ASP.NET MVC uses Web Forms as the default view engine, it is feasible - and common - to use simple HTML helpers instead of components.

RIFE

RIFE has an interesting approach to views, with a logic-free template that is manipulated by the controller.

Wicket

Wicket uses HTML templates with attributes to indicate components; there is no other code in the HTML files.

PHP, Ruby on Rails

A free mixture of code and literal HTML.

Seaside

All the output is generated by components' renderContentOn: method.

WASH

Haskell functions (which may be created by the preprocessor from inline HTML templates) construct a well-formed document in the HTML monad.

Arc

The HTML library creates trees of HTML tags. It is worth noting that the quality of Arc's HTML is atrocious, but this may improve with time.

Controller

A true controller has access to an updated model (although it may have to perform this update itself), and then decides what to output. Hooks in a Page control are not a proper implementation of the MVC pattern, as the page (view) is selected and instantiated before the "controller" can take control.

JavaServer Faces

A suitable combination of faces.config, action attributes, and backing beans can provide a (convoluted) controller.

Tapestry

Tapestry uses a simple component architecture; everything is handled by events in the page's code.

Stripes

An ActionBean can serve as a controller of sorts; this is indeed what src/app/App.java does in the sample application. However, ActionBeans must be hard-wired to particular views (in the stripes:form beanclass attribute).

RIFE

RIFE's elements are controllers, where the display of templates in controlled programatically.

Wicket

Wicket is components all the way down, and all controller logic resides in component events.

ASP.NET

Not only does ASP.NET have no model, there is no controller. Microsoft recognise this, and suggest a variety of awkward workarounds, but a true controller can only be acheived by writing a custom HTTP handler.

ASP.NET MVC, Ruby on Rails

The MVC pattern is fully implemented.

PHP

Simply beginning a PHP page with code can make it a controller.

Seaside

The root component of the test application is an example of how a Seaside component can act as a controller. However, WATask is more suited to complex workflows.

WASH

Each CGI action is effectively a controller.

Arc

Each handler or continuation is effectively a controller.

PHP

Finally, it must be pointed out that this study has been a little harsh on PHP. Ruby on Rails is a framework for the Ruby language. Similarly, JSF and ASP.NET can be considered frameworks for Java and .NET, building on the basic libraries. Although there are many third party frameworks for PHP (there is even a PHP on Rails), we have only considered the raw language with its standard libraries. This is because the other languages considered are general purpose; PHP and its libraries are designed to power web sites. No particular web framework for PHP has become popular, probably because they use PHP as just another scripting language, and other common scripting languages are more suited.

Conclusions

Industry Heavyweights

As discussed in Commercial Usage, as a broad generalisation, big enterprises use JEE, medium businesses use ASP.NET, while smaller organisation and individuals tend to use PHP. However, an examination of the Internet's busiest sites gives some surprising results. As ranked by Alexa, these are Google and Yahoo!. Both of these sites provide a multitude of popular Internet services, primarily search. Each handles a high proportion of the Internet's total traffic, serving billions of requests daily. How do they maintain operations at such a gargantuan scale? Yahoo! uses PHP, whilst Google prefers Python, a scripting language similar to Ruby. Why is this?

Semantic Gap

An important quality of code is the clarity of intent. For example, consider two methods for reading a text file into a string:

Java

```
FileInputStream file = new FileInputStream("filename");
byte[] buffer = new byte[file.available()];
file.read(buffer);
file.close();
String text = new String(buffer);
```

C#

```
string text = File.ReadAllText("filename");
```

Amongst other advantages, it is immediately obvious what the C# code is intended to do, whereas understanding the intent of the Java code requires executing it in the reader's head or recognising the pattern from past experience. This is because one operation (read a text file) maps to one method (File.ReadAllText) in C#, which is the ideal semantic mapping. In Java, five statements are required, of which only the shortest (file.read) actually corresponds to the desired operation.

JEE is notorious for requiring reams of complex code/XML to achieve simple requirements. ASP.NET works well for simple applications, such as those built in MSDN articles and trade show demonstrations. In more complex scenarios, such as real-word business requirements, managing the page lifecycle becomes the main focus of development effort. In contrast, the frameworks for dynamic languages tend to result in considerably tighter mapping from intent to code. Dynamic languages do run slower than Java and C#, but as the companies running the busiest Internet sites have discovered, the savings is programmer time are much greater than the expense in machine time.

Statically Typed Productivity

Can a statically typed language achieve the productivity of its dynamic brethren? This is the question to which ASP.NET MVC would like to answer "yes". The conclusion of this study is that it can; time will tell whether this is indeed the case.

Sources

JavaServer Faces

web/A.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>A</title>
  </head>
  <body>
    <f:view>
      <h:form id="form">
        <table>
          <tr>
            <td>Forename:</td>
            <td><h:inputText id="forename" value="#{Name.forename}" /></td>
          </tr>
          <tr>
            <td>Enter a Surname?</td>
            <td><h:selectBooleanCheckbox id="enterSurname"
value="#{Name.enterSurname}" /></td>
          </tr>
        </table>
        <h:commandButton action="#{Name.FromA}" value="Next" />
      </h:form>
    </f:view>
  </body>
</html>
```

web/B.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>B</title>
  </head>
  <body>
    <f:view>
      <h:form id="form">
        Surname: <h:inputText id="surname" value="#{Name.surname}" />
        <h:commandButton action="A" value="Previous" />
        <h:commandButton action="C" value="Next" />
      </h:form>
    </f:view>
  </body>
</html>
```

web/C.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>C</title>
  </head>
  <body>
    <f:view>
      <h:form id="form">
        <p>Hello <h:outputLabel value="#{Name.fullName}" /></p>
        <h:commandButton action="#{Name.FromC}" value="Previous" />
      </h:form>
    </f:view>
  </body>
</html>
```

web/WEB-INF/faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="1.2" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
  <navigation-rule>
    <from-view-id>/A.jsp</from-view-id>
    <navigation-case>
      <from-outcome>B</from-outcome>
      <to-view-id>/B.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>C</from-outcome>
      <to-view-id>/C.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
  <navigation-rule>
    <from-view-id>/B.jsp</from-view-id>
    <navigation-case>
      <from-outcome>C</from-outcome>
      <to-view-id>/C.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>A</from-outcome>
      <to-view-id>/A.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
  <navigation-rule>
    <from-view-id>/C.jsp</from-view-id>
    <navigation-case>
      <from-outcome>B</from-outcome>
      <to-view-id>/B.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>A</from-outcome>
      <to-view-id>/A.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <managed-bean>
    <managed-bean-name>Name</managed-bean-name>
    <managed-bean-class>app.Name</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

src/app/Name.java

```
package app;

public class Name {

  private String forename, surname;
  private boolean enterSurname;

  //Forename
  public String getForename() {
    return this.forename;
  }

  public void setForename(String forename) {
    this.forename = forename;
  }

  //EnterSurname
  public boolean getEnterSurname() {
    return this.enterSurname;
  }

  public void setEnterSurname(boolean enterSurname) {
    this.enterSurname = enterSurname;
  }

  //Surname
  public String getSurname() {
    return this.surname;
  }

  public void setSurname(String surname) {
    this.surname = surname;
  }

  //FullName
  public String getFullName() {
    String fullName = this.forename;
    if (this.enterSurname) fullName += " " + this.surname;
    return fullName;
  }

  //Navigate from A
  public String FromA() {
    return this.enterSurname ? "B" : "C";
  }

  //Navigate back from C
  public String FromC() {
    return this.enterSurname ? "B" : "A";
  }
}
```

Tapestry

src/app/Name.java

```
package app;

public class Name {

    private String forename, surname;
    private boolean enterSurname;

    //Forename
    public String getForename() {
        return this.forename;
    }
    public void setForename(String forename) {
        this.forename = forename;
    }

    //EnterSurname
    public Boolean getEnterSurname() {
        return this.enterSurname;
    }
    public void setEnterSurname(Boolean enterSurname) {
        this.enterSurname = enterSurname;
    }

    //Surname
    public String getSurname() {
        return this.surname;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getFullName() {
        String fullName = this.getForename();
        if (this.getEnterSurname()) fullName += " " + this.getSurname();
        return fullName;
    }
}
```

src/app/pages/A.java

```
package app.pages;

import org.apache.tapestry.annotations.ApplicationState;
import app.Name;

public class A {

        @ApplicationState
        private Name name;

    public Name getName() {
        return this.name;
    }

    public void setName(Name name) {
        this.name = name;
    }

    Object onAction() {
        return this.name.getEnterSurname() ? "B" : "C";
    }
}
```

src/app/pages/B.java

```
package app.pages;

import org.apache.tapestry.annotations.ApplicationState;
import app.Name;

public class B {

    private String next;

    @ApplicationState
    private Name name;

    public Name getName() {
        return this.name;
    }

    public void setName(Name name) {
        this.name = name;
    }

    //navigation, see http://wiki.apache.org/tapestry/Tapestry5HowToUseForms,
multiple submits
    void onSelectedFromA() {
        this.next = "A";
    }

    void onSelectedFromC() {
        this.next = "C";
    }

    Object onSuccess() {
       return this.next;
    }
}
```

src/app/pages/C.java

```
package app.pages;

import org.apache.tapestry.annotations.ApplicationState;
import app.Name;

public class C {

        @ApplicationState
        private Name name;

    public Name getName() {
        return this.name;
    }

    public void setName(Name name) {
        this.name = name;
    }

    Object onAction() {
        return this.name.getEnterSurname() ? "B" : "A";
    }
}
```

web/A.tml

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
  <head>
    <title>A</title>
  </head>
  <body>
      <t:form>
        <table>
          <tr>
            <td>Forename:</td>
            <td><t:textField t:value="name.forename"/></td>
          </tr>
          <tr>
            <td>Enter a Surname?</td>
            <td><t:checkbox t:value="name.enterSurname"/></td>
          </tr>
        </table>
        <t:submit value="Next"/>
      </t:form>
  </body>
</html>
```

web/B.tml

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
  <head>
    <title>B</title>
  </head>
  <body>
      <t:form>
        Surname: <t:textField value="name.surname" />
        <t:submit t:id="A" value="Previous"/>
        <t:submit t:id="C" value="Next"/>
      </t:form>
  </body>
</html>
```

web/C.tml

```
<html xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
  <head>
    <title>C</title>
  </head>
  <body>
    <p>Hello ${name.fullName}</p>
    <t:actionlink>Previous</t:actionlink>
  </body>
</html>
```

Stripes

web/app/A.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ taglib prefix="stripes" uri="http://stripes.sourceforge.net/stripes.tld"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>A</title>
  </head>
  <body>
      <stripes:form beanclass="app.App">
        <table>
          <tr>
            <td>Forename:</td>
            <td><stripes:text name="name.forename"/></td>
          </tr>
          <tr>
            <td>Enter a Surname?</td>
            <td><stripes:checkbox name="name.enterSurname"/></td>
          </tr>
        </table>
        <stripes:submit name="FromA" value="Next"/>
      </stripes:form>
  </body>
</html>
```

web/app/B.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ taglib prefix="stripes" uri="http://stripes.sourceforge.net/stripes.tld"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>B</title>
  </head>
  <body>
      <stripes:form beanclass="app.App">
        Surname: <stripes:text name="name.surname" />
        <stripes:submit name="A" value="Previous"/>
        <stripes:submit name="C" value="Next"/>
      </stripes:form>
  </body>
</html>
```

web/app/C.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="stripes" uri="http://stripes.sourceforge.net/stripes.tld"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>C</title>
  </head>
  <body>
      <stripes:form beanclass="app.App">
        <p>Hello <c:out value="${actionBean.name.fullName}"/></p>
        <stripes:submit name="FromC" value="Previous"/>
      </stripes:form>
  </body>
</html>
```

src/app/App.java

```
package app;

import net.sourceforge.stripes.action.*;

@Wizard
public class App implements ActionBean {

    //ActionBean Context
    ActionBeanContext context;
    public ActionBeanContext getContext() {
        return context;
    }
    public void setContext(ActionBeanContext context) {
        this.context = context;
    }

    //Name
    Name name;
        public Name getName() {
                return this.name;
        }
        public void setName(Name name) {
                this.name = name;
        }


    //Navigate from A
    public Resolution FromA() {
        return new ForwardResolution(this.name.getEnterSurname() ? "B.jsp" :
"C.jsp");
    }

    //Navigate back from C
    public Resolution FromC() {
        return new ForwardResolution(this.name.getEnterSurname() ? "B.jsp" :
"A.jsp");
    }

    //Navigate to A
    public Resolution A() {
        return new ForwardResolution("A.jsp");
    }
    //Navigate to C
    public Resolution C() {
        return new ForwardResolution("C.jsp");
    }
}
```

src/app/Name.java

```
package app;

public class Name {

    String forename, surname;
    boolean enterSurname;

    //Forename
    public String getForename() {
        return this.forename;
    }

    public void setForename(String forename) {
        this.forename = forename;
    }

    //EnterSurname
    public boolean getEnterSurname() {
        return this.enterSurname;
    }

    public void setEnterSurname(boolean enterSurname) {
        this.enterSurname = enterSurname;
    }

    //Surname
    public String getSurname() {
        return this.surname;
    }

    public void setSurname(String surname) {
        this.surname = surname;
    }

    //FullName
    public String getFullName() {
        String fullName = this.forename;
        if (this.enterSurname) fullName += " " + this.surname;
        return fullName;
    }
}
```

RIFE

web/Index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>RIFE</title>
    </head>

    <body>
        <r:v name="content"/>

        <r:bv name="content">
            <r:v name="content_form"></r:v>
        </r:bv>

        <r:b name="A">

            <form action="${v SUBMISSION:FORM:A/}" method="post">
                <r:v name="SUBMISSION:PARAMS:A"/>
                <table>
                    <tr>
                        <td>Forename:</td>
                        <td><r:v
name="FORM:INPUT:forename">id="forename"</r:v></td>
                    </tr>
                    <tr>
                        <td>Enter a Surname?</td>
                        <td><r:v
name="FORM:CHECKBOX:enterSurname">id="enterSurname"</r:v></td>
                    </tr>
                </table>
                <input type="submit" value="Next" />
            </form>
        </r:b>

        <r:b name="B">
            <form action="${v SUBMISSION:FORM:B/}" method="post">
                <r:v name="SUBMISSION:PARAMS:B"/>
                Surname: <r:v name="FORM:INPUT:surname">id="surname"</r:v>
                <input type="submit" name="Previous" value="Previous" />
                <input type="submit" value="Next" />
            </form>
        </r:b>

        <r:b name="C">
            <form action="${v SUBMISSION:FORM:C/}" method="post">
                <r:v name="SUBMISSION:PARAMS:C"/>
                <p>Hello <r:v name="fullName" /></p>
                <input type="submit" value="Previous" />
            </form>
        </r:b>
    </body>
</html>
```

web/participants.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE rep SYSTEM "/dtd/rep.dtd">
<rep>
    <participant param="rife.xml">ParticipantSite</participant>
</rep>
```

web/rife.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE site SYSTEM "/dtd/site.dtd">
<site>
    <arrival destid="Index"/>
    <element implementation="app.Index"/>
</site>
```

src/app/Index.java

```
package app;

import com.uwyn.rife.engine.Element;
import com.uwyn.rife.engine.annotations.Elem;
import com.uwyn.rife.engine.annotations.Param;
import com.uwyn.rife.engine.annotations.Submission;
import com.uwyn.rife.engine.annotations.SubmissionBean;
import com.uwyn.rife.template.Template;

@Elem(
    submissions = {
        @Submission(
            name = "A",
            beans = {@SubmissionBean(beanclass = Name.class, group =
NameMetaData.A)}
        ),
        @Submission(
            name = "B",
            beans = {@SubmissionBean(beanclass = Name.class, group =
NameMetaData.B)},
            params = {@Param(name = Index.PARAM_PREVIOUS)}
        ),
        @Submission(
            name = "C"
        )
    }
)
public class Index extends Element {
        public final static String PARAM_PREVIOUS = "Previous";

        private Name name = new Name();

        @Override
        public void processElement() {
            Template template = getHtmlTemplate("index");

            do {
                generateForm(template, this.name);
                template.setBlock("content_form", "A");
                print(template);
                pause();

                template.clear();
                fillSubmissionBean(this.name);
            } while (duringStepBack());
            if (this.name.getEnterSurname()) do {
                generateForm(template, this.name);
                template.setBlock("content_form", "B");
                print(template);
                pause();

                template.clear();
                fillSubmissionBean(this.name);
                if (hasParameterValue(PARAM_PREVIOUS)) stepBack();
            } while (duringStepBack());

            generateForm(template, this.name);
            template.setBlock("content_form", "C");

            //generate the full name
            String fullName = this.name.getForename();
            if (this.name.getEnterSurname()) fullName += " " +
this.name.getSurname();
            template.setValue("fullName", template.getEncoder().encode(fullName));

            print(template);
            pause();
            stepBack(); //the only button here is Previous
        }

        @Override
        public Object clone() throws CloneNotSupportedException {
                Index cloned = (Index)super.clone();
                cloned.name = this.name;
                return cloned;
        }
}
```

src/app/Name.java

```
package app;

public class Name {

    private String forename, surname;
    private boolean enterSurname;

    //Forename
    public String getForename() {
        return this.forename;
    }
    public void setForename(String forename) {
        this.forename = forename;
    }

    //EnterSurname
    public Boolean getEnterSurname() {
        return this.enterSurname;
    }
    public void setEnterSurname(Boolean enterSurname) {
        this.enterSurname = enterSurname;
    }

    //Surname
    public String getSurname() {
        return this.surname;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }
}
```

src/app/NameMetaData.java

```
package app;

import com.uwyn.rife.site.ConstrainedBean;
import com.uwyn.rife.site.ConstrainedProperty;
import com.uwyn.rife.site.MetaData;

public class NameMetaData extends MetaData<ConstrainedBean, ConstrainedProperty> {
    public final static String A = "A";
    public final static String B = "B";

    public void activateMetaData() {
        addGroup(A)
            .addConstraint(new ConstrainedProperty("forename"))
            .addConstraint(new ConstrainedProperty("enterSurname"));

        addGroup(B)
            .addConstraint(new ConstrainedProperty("surname"));
    }
}
```

Wicket

app/Name.java

```
package app;

import org.apache.wicket.IClusterable;

public class Name implements IClusterable {

    private String forename, surname;
    private boolean enterSurname;

    //Forename
    public String getForename() {
        return this.forename;
    }
    public void setForename(String forename) {
        this.forename = forename;
    }

    //EnterSurname
    public Boolean getEnterSurname() {
        return this.enterSurname;
    }
    public void setEnterSurname(Boolean enterSurname) {
        this.enterSurname = enterSurname;
    }

    //Surname
    public String getSurname() {
        return this.surname;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }

    public String getFullName() {
        String fullName = this.getForename();
        if (this.getEnterSurname()) fullName += " " + this.getSurname();
        return fullName;
    }
}
```

app/App.java

```
package app;

import org.apache.wicket.protocol.http.WebApplication;

public class App extends WebApplication
{
        public Class getHomePage()
        {
                return Index.class;
        }
}
```

app/Index.java

```
package app;
```

```java
import org.apache.wicket.Component;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.form.*;
import org.apache.wicket.model.*;
import org.apache.wicket.extensions.wizard.*;
import org.apache.wicket.extensions.wizard.WizardModel.ICondition;
import org.apache.wicket.markup.html.WebComponent;

public class Index extends WebPage {
    public Index() {
        NameWizard wizard = new NameWizard("wizard");
        add(wizard);
    }

    private class NameWizard extends Wizard {
        private Name name;

        //Name
        public Name getName() {
            return this.name;
        }
        public void setName(Name name) {
            this.name = name;
        }

        private NameWizard(String id) {
            super(id);
            this.name = new Name();

            setModel(new CompoundPropertyModel(this));
            WizardModel model = new WizardModel();
            model.add(new A(this));
            model.add(new B(this));
            model.add(new C(this));
            init(model);

        }

        //get rid of default button bar
        @Override
        protected Component newButtonBar(String id) {
            return new WebComponent(id);
        }

        //steps A-C
        private class A extends WizardStep {
            public A(Wizard wizard) {
                this.add(new TextField("name.forename"));
                this.add(new CheckBox("name.enterSurname"));
                add(new NextButton("next", wizard));
            }
        }

        private class B extends WizardStep implements ICondition {
            public B(Wizard wizard) {
                this.add(new TextField("name.surname"));
                add(new PreviousButton("previous", wizard));
                add(new NextButton("next", wizard));
            }

            public boolean evaluate() {
                return getName().getEnterSurname();
            }
        }

        private class C extends WizardStep {
            public C(Wizard wizard) {
                this.add(new Label("name.fullName"));
                add(new PreviousButton("previous", wizard));
            }
        }
    }
}
```

app/Index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>Wicket</title>
</head>
<body>
    <span wicket:id="wizard" />
</body>
</html>
```

app/Index$NameWizard.html

```
<wicket:panel>
    <div class="wicketExtensionsWizard">
        <form wicket:id="form" class="wicketExtensionsWizardForm">
            <div wicket:id="view" class="wicketExtensionsWizardViewInner" />
            <span wicket:id="header"/>
            <span wicket:id="feedback"/>
            <span wicket:id="buttons"/>
        </form>
    </div>
</wicket:panel>
```

app/Index$NameWizard$A.html

```
<wicket:panel>
  <table>
    <tr>
      <td>Forename:</td>
      <td><input type="text" wicket:id="name.forename" /></td>
    </tr>
    <tr>
      <td>Enter a Surname?</td>
      <td><input type="checkbox" wicket:id="name.enterSurname" /></td>
    </tr>
  </table>
  <input wicket:id="next" type="submit" value="next" />
</wicket:panel>
```

app/Index$NameWizard$B.html

```
<wicket:panel>
    Surname: <input type="text" wicket:id="name.surname" /><br />
    <input wicket:id="previous" type="submit" value="previous" />
    <input wicket:id="next" type="submit" value="next" />
</wicket:panel>
```

app/Index$NameWizard$C.html

```
<wicket:panel>
    Hello <span wicket:id="name.fullName" />
    <input wicket:id="previous" type="submit" value="previous" />
</wicket:panel>
```

ASP.NET

Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" Inherits="Default"
CodeFile="Default.aspx.cs" ValidateRequest="false" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
  <head id="Head1" runat="server">
    <title>A</title>
  </head>
  <body>
    <form id="form" runat="server">
      <asp:Wizard ID="Wizard" runat="server" DisplaySideBar="False"
OnNextButtonClick="Next">
        <WizardSteps>
          <asp:WizardStep runat="server" ID="A">
            <table>
              <tr>
                <td>Forename:</td>
                <td><asp:TextBox ID="Forename" runat="server" /></td>
              </tr>
              <tr>
                <td>Enter a Surname?</td>
                <td><asp:CheckBox ID="EnterSurname" runat="server" /></td>
              </tr>
            </table>
          </asp:WizardStep>
          <asp:WizardStep runat="server" ID="B">
            Surname: <asp:TextBox ID="Surname" runat="server" />
          </asp:WizardStep>
          <asp:WizardStep runat="server" ID="C">
            Hello <asp:Literal ID="Name" runat="server" />
          </asp:WizardStep>
        </WizardSteps>
        <FinishNavigationTemplate>
          <asp:Button ID="FinishPreviousButton" runat="server"
CausesValidation="False" CommandName="MovePrevious" Text="Previous" />
        </FinishNavigationTemplate>
      </asp:Wizard>
    </form>
  </body>
</html>
```

Default.aspx.cs

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Default: Page {
    protected void Page_PreRender(object sender, EventArgs e) {
        if (this.Wizard.ActiveStep == this.C) {
            string name = this.Forename.Text;
            if (this.EnterSurname.Checked) name += " " + this.Surname.Text;
            this.Name.Text = Server.HtmlEncode(name);
        }
    }

    protected void Next(object sender, WizardNavigationEventArgs e) {
        if (this.Wizard.ActiveStep == this.A && !this.EnterSurname.Checked)
            this.Wizard.MoveTo(this.C);
    }
}
```

ASP.NET MVC

Controllers/AppController.cs

```
using System;
using System.Web;
using System.Web.Mvc;
using App.Models;

namespace App.Controllers {
    public class AppController : Controller {

        private Name _name;
        private Name name {
            get {
                return this._name ??
                    (this._name = (this.Session["Name"] as Name ??
                        (Name)(this.Session["Name"] = new Name())));
            }
        }

        public ActionResult A() {
            return View("A", this.name);
        }

        public RedirectToRouteResult fromA(string forename, bool enterSurname) {
            this.name.forename = forename;
            this.name.enterSurname = enterSurname;
            return RedirectToAction(this.name.enterSurname ? "B" : "C");
        }

        public ActionResult B() {
            return View("B", this.name);
        }

        public RedirectToRouteResult fromB(string surname) {
            this.name.surname = surname;
            return RedirectToAction("C");
        }

        public ActionResult C() {
            return View("C", this.name);
        }
    }
}
```

Models/Name.cs

```
namespace App.Models {
    public class Name {
        public string forename;
        public bool enterSurname;
        public string surname;

        public string FullName {
            get {
                string name = this.forename;
                if (this.enterSurname) name += " " + this.surname;
                return name;
            }
        }
    }
}
```

Views/A.aspx

```
<%@ Page Language="C#" Inherits="ViewPage<App.Models.Name>" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>A</title>
</head>
<body>
    <% using (Html.BeginForm("fromA", "App")) { %>
        <table>
            <tr>
                <td>Forename:</td>
                <td><%= Html.TextBox("forename", Model.forename) %></td>
            </tr>
            <tr>
                <td>Enter a Surname?</td>
                <td><%= Html.CheckBox("enterSurname", Model.enterSurname) %></td>
            </tr>
        </table>
        <button type="submit">Next</button>
    <% } %>
</body>
</html>
```

Views/B.aspx

```
<%@ Page Language="C#" Inherits="ViewPage<App.Models.Name>" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>B</title>
</head>
<body>
    <% using (Html.BeginForm("fromB", "App")) { %>
        Surname: <%= Html.TextBox("surname", Model.surname) %>
        <button type="submit">Next</button>
    <%= Html.ActionLink("Previous", "A") %>
    <% } %>
</body>
</html>
```

Views/C.aspx

```
<%@ Page Language="C#" Inherits="ViewPage<App.Models.Name>" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>C</title>
</head>
<body>
    <p>Hello <%= Model.FullName %></p>
    <%= Html.ActionLink("Previous", Model.enterSurname ? "B" : "A") %>
</body>
</html>
```

PHP

.htaccess

```
php_flag magic_quotes_gpc off
php_flag session.auto_start on
```

A.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>A</title>
  </head>
  <body>
    <form method="post" action="fromA.php">
      <table>
        <tr>
          <td>Forename:</td>
          <td><input type="text" name="Forename" value="<?=
htmlspecialchars($_SESSION['Forename']) ?>"/></td>
        </tr>
        <tr>
          <td>Enter a Surname?</td>
          <td><input type="checkbox" name="EnterSurname" value="Yes" <?php if
($_SESSION['EnterSurname']) echo 'checked="true"' ?> /></td>
        </tr>
      </table>
      <input type="submit" value="Next" />
    </form>
  </body>
</html>
```

fromA.php

```
<?php
$_SESSION['Forename'] = $_POST['Forename'];
$_SESSION['EnterSurname'] = ($_POST['EnterSurname'] == "Yes");
include($_SESSION['EnterSurname'] ? 'B.php' : 'C.php');
?>
```

B.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>B</title>
</head>
<body>
  <form method="post" action="C.php">
    Surname:
    <input type="text" name="Surname" value="<?=
htmlspecialchars($_SESSION['Surname']) ?>" />
    <input type="submit" value="Next" />
  </form>
  <a href="A.php">Previous</a>
</body>
</html>
```

C.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>C</title>
  </head>
  <body>
    <p>Hello <?php
      $name = $_SESSION['Forename'];
      if (isset($_POST['Surname'])) {
        $_SESSION['Surname'] = $_POST['Surname'];
        $name .= ' ' . $_POST['Surname'];
      }
      echo htmlspecialchars($name);
    ?></p>
    <?= $_SESSION['EnterSurname'] ? '<a href="B.php">' : '<a href="A.php">'
?>Previous</a>
  </body>
</html>
```

Ruby on Rails

app/controllers/App_controller.rb

```
class AppController < ApplicationController
  before_filter :initialize_session

  def A
  end

  def fromA
    redirect_to :action => @name.enterSurname ? "B" : "C"
  end

  def B
  end

  def C
     @previous = @name.enterSurname ? "B" : "A"
  end

  private
    def initialize_session
      @name = session[:name] ||= Name.new
      @name.attributes = params[:name]
   end
end
```

app/models/Name.rb

```
class Name < ActiveRecord::Base
  attr_accessor :forename, :surname
  attr_reader :enterSurname

  #don't try to connect to the database!
  def initialize
  end

  def enterSurname=(enter)
    @enterSurname = (enter == "1") #HTTP checkbox value
  end

  def full_name
    name = @forename
    name += " " + @surname if @enterSurname
    return name
  end
end
```

app/views/app/A.rhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>A</title>
</head>
<body>
    <form method="post" action="fromA">
        <table>
            <tr>
```

```
                <td>Forename:</td>
                <td><%= text_field "name", "forename" %></td>
            </tr>
            <tr>
                <td>Enter a Surname?</td>
                <td><%= check_box "name", "enterSurname" %></td>
            </tr>
        </table>
        <input type="submit" value="Next" />
    </form>
</body>
</html>
```

app/views/app/B.rhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>B</title>
</head>
<body>
    <form method="post" action="C">
        Surname: <%= text_field "name", "surname" %>
        <input type="submit" value="Next" />
    </form>
    <a href="A">Previous</a>
</body>
</html>
```

app/views/app/C.rhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>C</title>
</head>
<body>
    <p>Hello <%= h(@name.full_name) %></p>
    <a href="<%= @previous %>">Previous</a>
</body>
</html>
```

Seaside

App.st

A

```
WAComponent subclass: #A
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'App'!

!A methodsFor: 'as yet unclassified'!
renderContentOn: html
	html table: [html
		tableRow: [html
			tableData: [html text: 'Forename:'];
			tableData: [html textInput on: #forename of: self session
person]];
		tableRow: [html
			tableData: [html text: 'Enter a Surname?'];
			tableData: [html checkbox on: #enterSurname of: self
session person]]].! !
```

App

```
WAComponent subclass: #App
	instanceVariableNames: 'screens current'
	classVariableNames: ''
	poolDictionaries: ''
	category: 'App'!

!App methodsFor: 'as yet unclassified'!
back
	current := current - 1.
	(current = 2 and: [self session person enterSurname not ]) ifTrue:
[current := 1]! !

!App methodsFor: 'as yet unclassified'!
children
	^screens! !

!App methodsFor: 'as yet unclassified'!
initialize
	super initialize.
	screens := {A new . B new . C new}.
	current := 1.! !

!App methodsFor: 'as yet unclassified'!
next
	current := current + 1.
	(current = 2 and: [self session person enterSurname not]) ifTrue:
[current := 3]! !

!App methodsFor: 'as yet unclassified'!
renderContentOn: html
	html form:
		[html render: (screens at: current).
		current > 1 ifTrue:
			[html submitButton callback: [self back]; value: 'Back'].
```

```
                 current < screens size ifTrue:
                         [html submitButton callback: [self next]; value:
'Next']]! !

!App methodsFor: 'as yet unclassified'!
states
        ^ Array with: self with: self session person! !


"-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- "!

App class
        instanceVariableNames: ''!

!App class methodsFor: 'as yet unclassified'!
canBeRoot
        ^true! !
```

AppSession

```
WASession subclass: #AppSession
        instanceVariableNames: 'person'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'App'!

!AppSession methodsFor: 'accessing'!
person
        ^ person! !

!AppSession methodsFor: 'accessing'!
person: anObject
        person := anObject! !


!AppSession methodsFor: 'as yet unclassified'!
initialize
        super initialize.
        person := Name new.! !
```

B

```
WAComponent subclass: #B
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'App'!

!B methodsFor: 'as yet unclassified'!
renderContentOn: html
        html text: 'Surname:'.
        html textInput on: #surname of: self session person.! !
```

```
WAComponent subclass: #C
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'App'!

!C methodsFor: 'as yet unclassified'!
renderContentOn: html
        html text: 'Hello ', self session person fullName.! !
```

Name

```
Object subclass: #Name
        instanceVariableNames: 'forename surname enterSurname'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'App'!

!Name methodsFor: 'as yet unclassified'!
fullName
        | name |

        name := forename.
        enterSurname ifTrue: [name := name, ' ', surname].
        ^name.! !


!Name methodsFor: 'accessing'!
enterSurname
        ^ enterSurname! !

!Name methodsFor: 'accessing'!
enterSurname: anObject
        enterSurname := anObject! !

!Name methodsFor: 'accessing'!
forename
        ^ forename! !

!Name methodsFor: 'accessing'!
forename: anObject
        forename := anObject! !

!Name methodsFor: 'accessing'!
surname
        ^ surname! !

!Name methodsFor: 'accessing'!
surname: anObject
        surname := anObject! !
```

WASH

Main.hs

```
import WASH.CGI.CGI hiding (standardQuery)

--replace WASH's standardQuery with one that uses htmlHeader instead of
standardPage
standardQuery ttl elems = ask $ htmlHeader ttl (makeForm elems)

data Name = Name {forename::String, enterSurname::Bool, surname::String}

main = run $ screenA (Name "" False "")

screenA name = standardQuery "A" $
                do params <- table $ do forenameF <- tr $ do td (text "Forename:")
                                                             td (textInputField $
attr "value" $ forename name)
                                        enterSurnameF <- tr $ do td (text "Enter a
Surname?")
                                                                 td
(checkboxInputField $ if enterSurname name then attr "checked" "" else empty)
                                        return (F2 forenameF enterSurnameF)
                   submit params (fromA name) $ attr "value" "Next"

fromA name (F2 forenameF enterSurnameF) = let name' = name {forename = value
forenameF, enterSurname = value enterSurnameF} in
                                          if value enterSurnameF then screenB
name' else screenC name'

screenB name = standardQuery "B" $
                do text "Surname: "
                   surnameF <- textInputField $ attr "value" $ surname name
                   submit surnameF (fromB name True) $ attr "value" "Next"
                   submit surnameF (fromB name False) $ attr "value" "Previous"

fromB name next surnameF = let name' = name {surname = value surnameF} in
                           if next then screenC name' else screenA name'

screenC name = standardQuery "C" $
                do text $ "Hello " ++ forename name ++ if enterSurname name then "
" ++ surname name else ""
                   br empty
                   submit0 (if enterSurname name then screenB name else screenA
name) $ attr "value" "Previous"
```

Arc

App.arc

```
;page

(mac page (title . body)
    `(tag html
        (tag head
            (tag title (pr ,title)))
        (tag (body) ,@body)))

;checkbox http://arclanguage.org/item?id=7334

(def opmin (key val)
    `(if ,val (pr " " ',key)))

(attribute input checked opmin)

(def checkbox (name (o checked))
    (gentag input type 'checkbox name name checked checked))

;the app

(defop A req (A (table)))

(def A (name)
    (page "A"
        (aform [fromA name _]
            (tab
                (row (pr "Forename:") (input "forename" (name 'forename)))
                (row (pr "Enter a Surname?") (checkbox "enterSurname" (name
'enterSurname))))
            (submit "Next"))))

(def fromA (name req)
    (= (name 'forename) (arg req "forename"))
    (= (name 'enterSurname) (arg req "enterSurname"))
    (if (name 'enterSurname) (B name) (C name)))

(def B (name)
    (page "B"
        (aform [fromB name _]
            (pr "Surname: ")
            (input "surname" (name 'surname))
            (buts "step" "Next" "Previous"))))

(def fromB (name req)
    (= (name 'surname) (arg req "surname"))
    (if (is (arg req "step") "Next") (C name) (A name)))


(def C (name)
    (page "C"
        (pr "Hello " (eschtml (name 'forename)))
        (if (name 'enterSurname) (pr " " (eschtml (name 'surname))))
        (br)
        (onlink "Previous" (if (name 'enterSurname) (B name) (A name)))))
```

```
(serve 80)
```