VRaptor - Simple And Quick Web Framework

Alexandre Freire , Paulo Silveira

[1]Department of Computer Science
Institute of Mathematics and Statistics
University of São Paulo

{alex,paulo}@arca.ime.usp.br
vraptor.org

Abstract. *This paper describes the development of VRaptor, a framework that allows developers to create dynamic web applications quick and easily. Developed by Arca, VRaptor is free software, available under the Apache 2.0 license. We will present motivations behind Arca's undertaking to create yet another web development framework for the J2EE specification. We will then describe the framework's architecture and it's usage patterns. We will justify our architectural decisions based on our main objectives: keeping the framework simple, clean, and easy to learn and use. We will briefly mention other free software tools and frameworks that interact with VRaptor in some way .*

1. Introduction

Web Application development has been the focus of many businesses and institutions since only after a couple of year's of the advent of the Internet. Today, many web application frameworks [7] are available to developers. Choices exist in various languages and "flavors". There is an overwhelming number of free software projects providing us with tools for web development. When faced with the task of adopting an alternative to develop a system, an architect may have to consider solutions ranging from containers and application servers to programming languages and frameworks. Some of the most known open source and free software projects available today are PHP, Struts, WebWorks, Zope, Tomcat, Jetty, and JBoss. Also, an even number of commercial offerings for frameworks and application servers exists as well. This miscellanea of options makes it hard to decide which project to use when it comes the time to develop a system.

"Arca" [1], a group of free software developers fostered by the Department of Computer Science at IME/USP, has focused on web development since it's foundation. Having worked on many end user applications like Panda, Texugo, and Mico, Arca's developers have experimented with most of the choices in tools available today . During the creation of these applications we strived to learn more about all of these technologies. Together, the members of our group have developed for all of the most known frameworks nowadays.

These frameworks all have very nice features and patterns, however, many of them lean more to a declarative approach on programing instead of an imperative one. This choice means using lots of configuration files (mostly XML metadata descriptors), which in our opinion ends up cluttering the developers perception of what is really happening in his application, increasing the learning curve for new comers to the framework, and making applications harder to debug.

Throughout our existence, our group has studied and discussed most of these frameworks. We have had many ideas on how to make things quicker and easier for the developer. “eXtreme Programming’ [4]’, our favorite methodology, influenced us when idealizing what a framework had to do, since much of our development focus on XP practices. Faced with many choices we decided to create our own framework, one that had to be agile, allowing it’s developers to test their code easily and refactor it mercilessly, and at the same time had to be simple to understand and easy to learn.

Earlier this year we had the opportunity to develop a portal for SINAPE [2], the National Symposium on Probability and Statistics organized by the Brazilian Association for Statistics (ABE). We then decided to use the funds from this contract to sponsor the development of this new framework. We also decided to use some XP practices when developing the framework, adding features as the need for them arose in the portal.

And so VRaptor was born. Our main objectives with this framework are to keep the code simple and clean, so as to encourage outside contribution, but most importantly, to keep it simple to learn and use, so that new developers can choose to use it over all the other available choices. Particularly, we want to provide good documentation in portuguese, to help the brazilian free software community.

2. Development

VRaptor was created to solve our development needs for the SINAPE portal. We developed the portal using the eXtreme programming methodology, therefore VRaptor had to facilitate the practices we were following. We needed a framework that would grow organically from our “real-world” needs. We knew that our end-user portal would have to accommodate changes easily, we knew that we would do the simplest thing we possibly could to solve a particular problem and we knew that our code had to be kept consistent, simple and clear.

We took all these premises in consideration when choosing how the developer would use VRaptor, favoring a more imperative approach over the declarative one that most frameworks today seem to prefer. We decided to keep configuration down to one XML file, where the user coupled the components. Components themselves would be completely decoupled, so as to facilitate testing.

New features were added added to the framework when the need for them arose, keeping the design clean and easy to understand and use. We made a conscious effort to postpone development of features we did not really need, therefore VRaptor does not suffer from a flaw common to many frameworks, known as “feature overflow”. We believe this makes the learning process much easier for a new developer.

Refactoring code created for VRaptor should also be easy, so we chose to use symbols from objects created by the programmer, like method and class names, in the configuration XML file. These symbols can easily be altered by an automated tool. Avoiding much use of meta-data we empower the developer to welcome change, for he does not have to modify lot’s of configuration files by hand when deciding to add new parameters to a method, or to change a method name.

3. Objectives

We strived to achieve many objectives in VRaptor, we shall briefly discuss these goals and why we chose them.

3.1. Simplicity and ease of use

VRaptor does not offer many choises to the developer, we made this decision to avoid confusing new users of the framework with configuration options. The XML configuration file is clean and consistent, there are not many ways to do the same thing, only one "right" way. This reduces a common confusion, of how things work "under the covers", that is present in the minds of many developers that are trying to learn how to use a unfamiliar framework.

VRaptor has few main abstractions, Interceptors, Domains, Chains, and Views. With a couple of these objects a developer can deliver complex logic in dynamic web applications. By use of introspection in the framework's engine we can allow these components to be loosely coupled, this permits easy component reuse and "on-the-fly" configuration.

Because components are decoupled from web-specific objects as well, tests are a whole lot easier to write.

3.2. Avoiding "too much magic"

One of our main objectives is to avoid the known anti-pattern, present in many frameworks available today, called "too much magic" [5]. This anti-pattern states that when a developer cannot understand how the framework interacts with his code he is awed by what "seems to be the magic" that makes his application work.

This makes it really hard for the developer when a new bug is found in his application, most times he feels that things were working before and he doesn't know what changed that made the application behave differently.

In VRaptor it is clear when and where the framework will instantiate objects declared in the application, the developer knows how the framework will call the methods defined in in his objects and how their lifecycle is controlled, he also knows how to get instances of objects he needs.

3.3. A national alternative

Another of our objectives was to create a national alternative to all the international web application frameworks available. We are currently working on documentation in portuguese and have recently made our first public release. The framework has been used to develop a couple of "real-world" applications, including GUJ's [3] new website.

4. Patterns and Paradigms

Applications developed with VRaptor follow a couple of architectural patterns and paradigms, making the developer's life easier [9].

4.1. Model View Controller

Model View Controller [6] breaks the application into three: the user input and output, the modeling of our business logic and the application controller. These are well defined parts in a VRaptor application.

VRaptor has built-in support for well know, free software, web view layers: Velocity[**?**], Freemarker [**?**] and JSP. the model classes are written by the developer, and the controller is VRaptor itself.

4.2. Interceptors and Aspects

All application have some aspects that do not really belong to the core business logic. Commonly known aspects are logging, security and persistency. Programming code related to these aspects should not be mixed with code that deals with business logic. In VRaptor, the developer can choose to put this type of code in interceptors. Interceptors are also responsible for the components dependecy configuration, as we will see in the next section.

VRaptor provides some default interceptors, but the developer will also need to write his own interceptors to make his application more robust.

4.3. Inversion Of Control

Inversion of Control addresses a component's configuration and dependency resolution: there is no need for a component to know exactly with which other components it will be dealing with[8] . This configuration will be handled by the framework, VRaptor acts as a component container, managing their lifecycle.

Newer web and Inversion of Control frameworks (like Apache Avalon, PicoContainer, or SpringFramework) support inversion of control through dependency injection, writting some descriptors (mostly hard to read and long XML files) to tell the container how it shall create and manage components. This is the declarative approach, as mentioned before.

VRaptor uses an imperative approach: dependency configuration must me done in plain java code. This code goes inside the interceptors, which are responsible for preparing components that will be used by the business logic code. Note that the configuration is still decoupled from the components themselves, if we need to change a component that is being used by our business logic, we simply change the interceptor. There is no need to touch the business logic at all.

5. Conclusion

We believe VRaptor is a real contender when considering a good framework for web application development. We are committed to providing a real alternative for the Brazilian market and welcome any contributions from the community.

References

[1] Arca project - http://www.arca.ime.usp.br

[2] http://www.abe.org/sinape

[3] Grupo de Usuarios Java - http://www.guj.com.br

[4] [Beck 1999] - *Extreme Programming Explained.*

[5] Too-much magic anti-pattern - http://lsd.student.utwente.nl/jicarilla/TooMuchMagic

[6] [Model 1988] - *"The Model-View- Controller (MVC) ParadigmUser Interfaces"* OOPSLA'88 Tutorial, ACM.

[7] Ralph E. Johnson. Components, Frameworks, Patterns. In ACM SIGSOFT Symposium on Software Reusability, 1997.

[8] C. Szyperski, Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 1998.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.