Ruby on Rails

Framework pode ser considerado como "Uma arquitetura desenvolvida com o objetivo de se obter a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização" Mattsson (1996). E ele é composto por hot-spots e frozen-spots. Os hot-spots são os pontos flexíveis dentro do framework, são pontos genéricos da aplicação que devem ser adaptados para o contexto da aplicação. Já os frozen-spots são responsáveis por definir a arquitetura geral do sistema e os relacionamentos entre as diversas classes.

O framework de caso de estudo é o Ruby on Rails. RoR é um framework de código aberto escrito na linguagem Ruby, na qual é especializado para desenvolvimento web, e suas aplicações se utilizam do padrão arquitetural MVC (Model-View-Controller). E o mesmo possui 3 filosofias básicas: **DRY: Dont Repeat Yoursef**, **Convention over Configuration**, **Less Software**.

Como dito anteriormente, Ruby on Rails usa o padrão arquitetural MVC, com a intenção de manter a manutenibilidade da aplicação. A Model centraliza a lógica de negócio, a camada view é responsável por exibir essa lógica, e a Controller lida com o fluxo da aplicação. A imagem abaixo explicita seus componentes e comunicação entre os mesmos em uma aplicação.

Por ser um framework, RoR possui diversos pontos de extensões. Esses pontos são basicamentes módulos Ruby, e os mais notáveis são: **ActionController**, **ActiveRecord** e **ActionMailer**.

Para o autor Felix Bachmann, modelo de componente "representa um elemento da arquitetura do sistema na qual são definidos os padrões e convenções impostas aos componentes do sistema, de como a descrever a função de cada um e as interações entre si". Abaixo está descrito a função de cada um e como eles interagem.

- Active Support: é o componente responsável por fornecer extensões de linguagem Ruby, utilidades. Muitas coisas que são utilizadas no desenvolvimento Rails vem desse componente.

- Active Resource: Fornece uma interface de comunicação para as aplicações desenvolvidas em Ruby on Rails, a classe ActiveResource::Base é a principal desse componente e é responsável por mapear recursos RESTful como modelos em um aplicativo rails. Funciona de forma parecida com o ActiveRecord

- Active Record: Active Record representa a letra M da arquitetura MVC, no desenvolvimento de aplicações rails, a camada modelo é responsável por representar as regras de negócio e lógica do projeto. Esse componente facilita a criação e uso dos objetos de negócio que estão armazenados na base de dados da aplicação. É uma implementação do padrão Active Record, que por essência é um sistema de mapeamento de relacionamento de objetos. Ele atua basicamente como um framework de ORM.

- Active Model: é uma biblioteca contendo vários módulos de desenvolvimento de classes que utilizam de features do Active Record.

- Action Mailer: componente que permite envio de emails utilizando-se de classes mailer e views. As Mailers funcionam como controllers, e as classes normalmente herdam de ActionMailer::Base. Cada classe criada tem um mailer e uma view para gerenciamento dos eventos. Componente Action:

- Action Controller: representa o 'C' do MVC. Após a requisição ser determinada para qual controller seguir, essa controller processa a requisição e gera o resultado esperado. Esse componente faz boa parte do trabalho braçal de se tratar uma requisição de forma a facilitar pro desenvolvedor.

- Action Dispatch: módulo responsável pelo redirecionamento das requisições para as controllers

- Action View: módulo responsável por exibir o resultado gerado pelas controllers, representa o 'V' do MVC. Essas views são escritas utilizando o formato html.erb e já são fornecidos vários métodos por parte desse módulo para criação das mesmas.

Em Rails, um componente de software são os módulos que são passíveis de serem estendidos. Um exemplo claro, é o ActiveRecord mencionado anteriormente. A estrutura do módulo do ActiveRecord é basicamente um conjunto de diversas composições com vários outros módulos, ou seja, o módulo é composto por uma classe chamada Base que contém vários includes e extends de diversos módulos.

A implementação é ocultada através da herança nos hotspots fornecidos pelo framework, na qual é possível alterar certos comportamentos com o uso de sobrescritas porém a implementação original do componente continua inacessível ao desenvolvedor. Assim sendo, para adaptarmos os componentes, se utiliza de Herança, na qual a maioria dos componentes gerados são instâncias de componentes fornecidos pelo Rails, e utiliza-se do polimorfismo para adaptá-los e criar componentes próprios.

Existe 3 modos para composição componentes / frameworks, são eles:

- **Componente - Componente**: é realizado através de composição, que define interação entre os componentes e funcionalidades da aplicação, a especificação dessas interações classifica-se como contrato em nível de aplicação.

- **Componente - Framework**: composição que possibilita interação entre os frameworks de componentes e seus componentes, tais interações permite que o framework gerencie os componentes, e essas interações classificam-se como contrato em nível de sistema.

- **Framework - Framework**: composição que possibilita interações entre diferentes frameworks e permite composição de componentes definidos em diferentes frameworks, essas interações classificam-se como contrato de interoperação.

Agora a comunicação entre Componentes/Frameworks é dada através das interfaces que compõem o framework que possuem associações com outras interfaces, permitindo a comunicação das mesmas. Por exemplo no pacote de Action o ActionDispatch utiliza da ActionController para instanciar uma view, e nessa Controller é definida a lógica de qual view

deve ser renderizada para o usuário. Também sobre o Action Pack, todos módulos presentes dentro desse pacote estendem o Active Support e utilizam de alguns métodos fornecidos por tal componente.

O diagrama abaixo explica como cada componente se comunica em uma aplicação Ruby on Rails.

Referências

- SPAGNOLI, Luciana; BECKER, Karin. Um estudo sobre o desenvolvimento baseado em componentes.
- Ruby on Rails Architectural Design, disponível em (https://adrianmejia.com/blog/2011/08/11/ruby-on-rails-architectural-design/) . Acesso em 26/06/2018.