

**Flávio Gonçalves Garcia**

***MyFuses: Um Framework Para Desenvolvimento de Aplicações Web PHP.***

Trabalho apresentado ao curso de graduação em Bacharelado em Ciências da Computação da Universidade Católica de Brasília, como requisito para obtenção do Título de Bacharel em Engenharia de Software.

**Orientador:** Prof. Msc. Cândido Salgado

Brasília

2008

## **Termo de Aprovação**

Trabalho de autoria de Flávio Gonçalves Garcia, intitulado “MyFuses: um Framework para Desenvolvimento de aplicações Web”, requisito parcial para obtenção do grau de Bacharel em Ciências da Computação, defendida e aprovada em 21/06/2008, pela banca examinadora constituída por:

---

**Prof. Msc. Cândido Salgado - Orientador**

---

Prof. Msc. Fábio Bianchi - Membro

Brasília

UCB

Dedico este trabalho a meu pai que sempre me ensinou a ser uma pessoa de caráter e nunca desistir de meus objetivos, mas que agora passa pelo momento mais difícil de sua vida, um câncer no cérebro. O esforço despendido neste trabalho foi como se a cura de meu pai dependesse da conclusão desta disciplina.

Agradeço a Deus pelas por todas as conquistas e inspiração concedidas, à minha esposa pelo suporte e compreensão, principalmente durante a conclusão desta disciplina, e a meu irmão e minha mãe, pelo apoio que sempre me deram.

## Resumo

MyFuses é uma reimplementação em PHP do Fusebox, que facilita as tarefas de análise, projeto, codificação e manutenção de sistemas WEB. O MyFuses tem o objetivo de implementar as funcionalidades básicas do Fusebox e sugerir novos mecanismos para melhorar a extensibilidade e interoperabilidade da ferramenta.

**Palavras-chave:** Frameworks, Padrões de Projeto, MVC, SOA, XML, REST, WebServices, PHP, Fusebox.

## **Abstract**

MyFuses is a PHP Fusebox rewrite that makes more easier the analysis, design, programming and maintenance of a WEB system. The MyFuses goal is implement Fusebox basic features and suggest new ways of extensions and interoperability for this tool.

**Key-Words:** Frameworks, Desing Patterns, MVC, SOA, XML, REST, WebServices, PHP, Fusebox.

## Sumário

<b>1 INTRODUÇÃO.....</b>	<b>7</b>
<b>1.1 Motivação.....</b>	<b>7</b>
<b>1.2 Breve histórico.....</b>	<b>8</b>
<b>1.3 Problemas diagnosticados.....</b>	<b>9</b>
<b>1.4 Surgimento da Necessidade.....</b>	<b>10</b>
<b>1.5 Usuários Beneficiados.....</b>	<b>10</b>
<b>1.6 Organizações interessadas.....</b>	<b>10</b>
<b>2 OBJETIVO DA PESQUISA.....</b>	<b>11</b>
<b>2.1 Objetivo Geral.....</b>	<b>11</b>
<b>2.2 Objetivos Específicos.....</b>	<b>11</b>
<b>3 PROPOSTA DA PESQUISA.....</b>	<b>13</b>
<b>3.1 Descrição da Pesquisa.....</b>	<b>13</b>
<b>3.2 Resultados Esperados.....</b>	<b>13</b>
<b>3.3 Restrições da Pesquisa.....</b>	<b>14</b>
<b>3.4 Recursos Necessários.....</b>	<b>14</b>
<b>3.4.1 Descrição de Hardware Ideal.....</b>	<b>14</b>
<b>3.4.2 Descrição de Hardware Mínimo.....</b>	<b>15</b>
<b>3.4.3 Descrição do Software.....</b>	<b>15</b>
<b>3.4.4 Configuração da Rede.....</b>	<b>16</b>
<b>3.4.5 Configuração do Banco de Dados.....</b>	<b>16</b>
<b>3.4.6 Descrição das Instalações.....</b>	<b>16</b>
<b>3.4.7 Pessoal Exigido.....</b>	<b>16</b>
<b>3.5 Relação Custo Benefício.....</b>	<b>16</b>
<b>3.5.1 Robustez x Performance.....</b>	<b>17</b>
<b>3.5.2 Cache x Acesso a disco.....</b>	<b>17</b>
<b>3.5.3 Inovação x Compatibilidade com o Legado.....</b>	<b>17</b>
<b>3.6 Áreas Afetadas.....</b>	<b>17</b>
<b>4 METODOLOGIA DA PESQUISA.....</b>	<b>18</b>
<b>5 DETALHAMENTO DA PESQUISA.....</b>	<b>19</b>
<b>5.1 Vantagens.....</b>	<b>19</b>

<b>5.2 Funcionamento.....</b>	<b>19</b>
<b>5.3 Conceitos Básicos.....</b>	<b>20</b>
<b>5.3.1 Aplicação.....</b>	<b>21</b>
<b>5.3.2 Circuito(Circuits).....</b>	<b>22</b>
<b>5.3.3 Fuseaction.....</b>	<b>23</b>
<b>5.3.3.1 Fuseactions Globais.....</b>	<b>24</b>
<b>5.3.4 Verbo(Verb).....</b>	<b>26</b>
<b>5.3.5 Plugins.....</b>	<b>27</b>
<b>5.4 Classes Mapeadas.....</b>	<b>28</b>
<b>5.5 FLiP.....</b>	<b>29</b>
<b>5.5.1 Wireframing.....</b>	<b>30</b>
<b>5.5.2 Desing de Templates.....</b>	<b>30</b>
<b>5.5.3 Prototipação e DevNotes.....</b>	<b>30</b>
<b>5.5.4 Arquitetura e Fusedocs.....</b>	<b>30</b>
<b>5.5.5 Codificação.....</b>	<b>31</b>
<b>5.5.6 Teste.....</b>	<b>31</b>
<b>6 PROTÓTIPO.....</b>	<b>32</b>
<b>6.1 Contexto.....</b>	<b>32</b>
<b>6.2 Principais Diferenças.....</b>	<b>32</b>
<b>6.3 Componentes Principais do MyFuses.....</b>	<b>33</b>
<b>6.3.1 Núcleo(Core).....</b>	<b>34</b>
<b>6.3.2 Motor(Engine).....</b>	<b>36</b>
<b>6.3.3 Processo(Process).....</b>	<b>37</b>
<b>6.4 Padronização dos Pontos de Extensão.....</b>	<b>38</b>
<b>6.5 REST.....</b>	<b>39</b>
<b>7 CONCLUSÕES.....</b>	<b>40</b>
<b>8 BIBLIOGRAFIA.....</b>	<b>41</b>

# 1 INTRODUÇÃO

## 1.1 Motivação

Em 09 de março de 2004 foi criado um projeto *Open-Source* no repositório *Soruceforge*, o *Iflux*, com o objetivo de facilitar a criação de aplicações web desenvolvidas em PHP, porque naquela época poucos *Frameworks* para *PHP* existiam no mercado, e os disponíveis não eram estáveis o bastante para suprir a demanda do público corporativo. Esta ferramenta visava o desenvolvimento de aplicações utilizando o padrão de projeto *MVC(Model-View-Controler)*. As camadas de Modelo(*Model*) e Visão(*View*) foram solucionadas facilmente pela equipe de desenvolvimento, porém o Controlador(*Controller*), além de ser um componente de desenvolvimento muito complexo, não possuía nenhuma biblioteca que atendesse as demandas exigidas pelo projeto.

Para suprir esta demanda, um outro projeto *Open-Source* foi adotado, escrito originalmente para *ColdFusion*, o *Fusebox*, desenvolvido pela *Fusebox Corporation*, começava a ser portado oficialmente para PHP. Esta ferramenta facilita o processo de levantamento, projeto, desenvolvimento e manutenção de aplicações *web*.

Após um grande tempo de utilização do *Fusebox*, notava-se que a versão para *ColdFusion* era corrigida e evoluída de forma rápida, constante e periódica, e por outro lado, a versão para *PHP* apresentava alguns problemas de implementação e suporte. O ideal era que as duas versões estivessem sincronizadas e possuíssem a mesma política de atualização e correção de falhas ou *bugs*, mas isto não acontecia, limitando o desenvolvimento e evolução do *Iflux*.

O *MyFuses* foi criado para implementar e corrigir as funcionalidades básicas do *Fusebox* para *PHP*, além de adicionar novas funcionalidades que facilitem a interoperabilidade entre aplicações e propor melhorias nas diferentes formas de estender a ferramenta, facilitando o trabalho do desenvolvedor em suas atividades durante o seu dia a dia.

## ***1.2 Breve histórico***

Em 14 de setembro de 2004, o *Fusebox 4.0 PHP* foi escolhido para compor a camada de controle do *Iflux*, com o objetivo de atender projetos de grande porte, nos quesitos organização e controle de uma aplicação, possibilitando que vários programadores desenvolvessem suas tarefas de forma paralela, proporcionando a equipe de desenvolvimento maior produtividade e qualidade nos produtos desenvolvidos.

A primeira versão utilizada era a 4.0, que tinha problemas grotescos de implementação, corrigidos pela 4.1, lançada pela *Fusebox Corporation* em 14 de agosto de 2005, corrigindo alguns erros da versão anterior, além de trazer a maioria das funcionalidades básicas que a versão similar desenvolvida para *ColdFusion* propunha.

O *Fusebox* para *ColdFusion* apresentava atualizações e correções constantes, mas a versão para PHP estava estagnado. A necessidade de corrigir os problemas apresentados pelo *Fusebox 4.1* ficavam evidentes, principalmente quando era necessário utilizar recursos avançados da ferramenta, que possibilitavam melhor reutilização do código. Outra preocupação estava quando as aplicações desenvolvidas com a *Fusebox* cresciam em número de funcionalidades, notava-se uma queda considerável na performance da aplicação.

Alguns pedidos para correção da ferramenta foram submetidos a *Fusebox Corporation*, e como não foram atendidos, fato que obrigou uma manutenção forçada no código fonte da biblioteca original.

Este grande ciclo, forçado, de manutenção do código fonte do *Fusebox* para *PHP*, propiciou um grande conhecimento do funcionamento e estruturas de dados utilizados no *framework*, abrindo portas para o desenvolvimento de uma ferramenta atualizada, que corrigisse os erros de implementação, esta ferramenta foi batizada de *MyFuses*.

Logo nas primeiras versões do *MyFuses*, a *Fusebox Corporation* lançou o *Fusebox 5.0* para *PHP*, que apesar de conter várias inovações nos padrões de desenvolvimento, a taxa de erros nos testes inviabilizaram sua utilização.

As primeiras versões de teste do *MyFuses* definiram um protótipo bem primário do modelo de estrutura de dados, assim como o funcionamento básico do controlador, já era possível testar o funcionamento de uma aplicação, porém essas funcionalidades precisavam evoluir.

Durante o período deste trabalho de conclusão de curso, houve um amadurecimento do modelo de estrutura de dados e funcionamento do controlador, adicionando a possibilidade de trabalhar com várias aplicações. As estratégias de carregamento e montagem das estruturas de dados foram reescritas duas vezes, com o objetivo de melhorar a performance da ferramenta.

Também neste período, todo o estudo de interoperabilidade da ferramenta foi realizado, adicionando as funcionalidades para utilização de *webservices* e REST, além de melhorar o tratamento de exceções dando maior qualidade e estabilidade a ferramenta.

### ***1.3 Problemas diagnosticados***

A versão de *Fusebox* que foi utilizada como base para desenvolvimento do *MyFuses* foi a 4.1, por ser a mais estável. Alguns problemas diagnosticados nesta ferramenta foram:

- O paradigma de desenvolvimento da ferramenta é o estruturado, utilizando funções, porém em muitas partes do código é possível observar que não há muita reutilização de código;
- As estruturas de dados da ferramenta são baseadas em *arrays*, que se encontram no contexto global da aplicação, possibilitando que um programador facilmente sobrescreva a esta estrutura de dados;
- Algumas funcionalidades, destinadas a melhorar reutilização de código, faziam com que a aplicação entrasse em *loop* infinito;
- O cache da aplicação é descentralizado, cada aplicação possui uma pasta de cache, dificultando a manutenção e implantação de aplicações;
- A própria *Fusebox Corporation* questionava algumas estruturas de dados propostas na versão 4.1, já colocando em listas de discussão algumas alternativas, e depois de algum tempo, disponibilizou a versão 5.0 com várias soluções, mas só estavam disponíveis na implementação para *ColdFusion*;
- Efeito degradante de performance em aplicações maiores, que apresentavam um crescimento no número de funcionalidades.

### ***1.4 Surgimento da Necessidade***

A falta de extensibilidade, atualização e correção do *Fusebox* começou a limitar o crescimento do *Iflux*, fazendo com que em março de 2006 começassem algumas experiências para implementação de algumas estruturas do *Fusebox*, além da resolução de requisições feita ao controlador. Durante essas experiências, constatou-se, que ao invés do *Iflux* utilizar o *Fusebox*, seria mais interessante estender o controlador.

Como a *Fusebox Corporation* não sinalizava nenhuma iniciativa em resolver a situação do *Fusebox* 4.1, em 09 de agosto de 2006, iniciou o projeto *MyFuses*, que além de implementaria os padrões da versão 5, suportaria o legado das aplicações desenvolvidas com a versão 4.1, e adicionaria funcionalidades de interoperabilidade entre aplicações situadas em uma mesma máquina ou em servidores diferentes.

### ***1.5 Usuários Beneficiados***

Os usuários beneficiados com a implementação do *MyFuses* são todos os analistas, engenheiros de software e programadores que utilizam ou se interessem em utilizar o *Fusebox* para desenvolvimento aplicações *web* para *PHP*.

### ***1.6 Organizações interessadas***

O Ministério do Meio Ambiente já utiliza o *Fusebox* em larga escala, tendo mais de 15 aplicações em produção, fazendo que esta organização seja a mais interessada no desenvolvimento da solução.

A Poliedro Informática e Consultoria também expressou interesse em adotar o *MyFuses* como plataforma padrão de desenvolvimento de aplicações *PHP*.

## 2 OBJETIVO DA PESQUISA

### 2.1 *Objetivo Geral*

A pesquisa tem o objetivo de apresentar uma solução que substitua a implementação do *Fusebox* para *PHP*, melhorando a sua manutenibilidade, extensibilidade, interoperabilidade e robustez, fazendo com que haja uma outra alternativa a ferramenta da *Fusebox Corporation*, que seja compatível com o legado, e que possibilite utilizar o mesmo processo de desenvolvimento adotado pela ferramenta de origem.

### 2.2 *Objetivos Específicos*

A pesquisa apresenta vários objetivos específicos, que abrangem tanto o aprendizado da solução proposta pela *Fusebox Corporation*, a formulação de uma proposta que atenda os requisitos que atendam ao legado já desenvolvido, além de propor novas soluções de extensibilidade e interoperabilidade para a ferramenta.

Dentre os objetivos específicos que podemos descrever para a pesquisa estão:

- Mapeamento das estruturas básicas para implementação de uma solução baseada em *Fusebox*;
- Mapeamento das estratégias de carregamento e montagem das estruturas de dados necessárias para representar uma aplicação *Fusebox*;
- Mapeamento do fluxo de resolução de uma requisição do usuário, desde seu início até a entrega de um conteúdo completo;
- Proposta de uma estrutura, orientada a objeto, que atenda aos requisitos básicos suportados pela ferramenta original, que apresente uma melhora de qualidade nos quesitos manutenibilidade e extensibilidade;
- Implementação das soluções propostas pelo padrão proposto pela versão do *Fusebox* 5.0;
- Proposta de uma estrutura que possibilite a utilização de mais de uma aplicação por um controlador;
- Proposta de estratégias que facilitem o consumo de *WebServices* por uma aplicação

desenvolvida em *MyFuses*;

## 3 PROPOSTA DA PESQUISA

### 3.1 Descrição da Pesquisa

A pesquisa consiste em mapear as estruturas de dados contidas no Fusebox, modelando uma estrutura nova, orientada a objetos, e mais extensível, formando assim, o núcleo da biblioteca *MyFuses*.

A estratégia de carregamento dos arquivos de configuração e montagem dos elementos do núcleo do *MyFuses*, é fundamental, e será o ponto mais crítico, pois dela depende toda a performance e funcionalidades do sistema, nesse aspecto, é possível definir o custo benefício entre a velocidade e a robustez da solução.

Também será alvo da pesquisa, buscar meios para melhorar a interoperabilidade entre as aplicações desenvolvidas utilizando *MyFuses*, facilitando a reutilização de recursos e funcionalidades por meio das estruturas de dados do núcleo do *Framework*, além de possibilitar meios que facilitem a utilização de *Webservices* ou *REST* para que aplicações *MyFuses* consigam se comunicar facilmente com outras que utilizem outras bibliotecas ou tecnologias em seu desenvolvimento.

### 3.2 Resultados Esperados

Os resultados esperados ao final da pesquisa são:

- Ter um produto semelhante ao *Fusebox* e que apresente melhor qualidade;
- Desenvolver uma estrutura de dados orientada a objeto, garantindo melhor encapsulamento e integridade;
- O controlador *MyFuses* deverá suportar mais de uma aplicação por instância, e gerenciar a interoperabilidade entre elas;
- Suportar facilmente o consumo de *Webservices*;
- Facilitar a conversão de estruturas de dados em *XML* e *Json*;
- Desempenho satisfatório em relação ao *Fusebox*;
- Ter uma ferramenta mais estável, com relação a degradação de performance, de acordo com o tamanho da aplicação;

- Padronização das pontos de extensão da *Framework*.

### **3.3 Restrições da Pesquisa**

A pesquisa não abordará os seguintes aspectos:

- Reutilização de estruturas de dados de aplicações *MyFuses* que estejam em servidores diferentes;
- Criação automática de *webservices*;
- Geração automática de código.

### **3.4 Recursos Necessários**

#### **3.4.1 Descrição de Hardware Ideal**

Considerando que o *MyFuses*, tem como finalidade desenvolver aplicações corporativas, com uma quantidade grande de acessos e processamento de dados a cada requisição, necessitamos de máquinas que consigam suportar essa carga.

No ciclo de desenvolvimento teremos pessoas desenvolvendo aplicações em computadores *desktop* e servidores hospedando as aplicações geradas para ambientes de internet, intranet e extranet.

A máquina utilizada para o desenvolvimento da aplicação necessita de uma boa capacidade para suportar a execução simultânea de um servidor *HTTP* e uma *IDE PHP*, além de suportar a execução de um browser, e dependendo do projeto, uma máquina para simular senários com alta carga de requisições. O hardware ideal desta máquina é um computador com mais de 2.0 Ghz de velocidade de processador, 2GB de *ram*.

A máquina utilizada para hospedar as aplicações desenvolvidas com o *MyFuses* necessita de uma boa capacidade para processamento, pois como as aplicações tendem a ser interoperáveis, ele deve estar preparado para suprir este aumento na quantidade de requisições ao servidor. O hardware ideal é uma máquina com quatro ou mais processadores e 4 ou mais Gb de *ram*, considerando que esta máquina várias aplicações hospedadas e será

utilizada somente como servidor de aplicações *PHP*.

Com relação a disco rígido é necessário um estudo de caso considerando o tamanho do sistema operacional utilizado, softwares necessários para desenvolvimento e hospedagem das aplicações, e a quantidade de aplicações hospedadas na máquina, com objetivo de dimensionar o tamanho desta unidade de disco rígido.

### **3.4.2 Descrição de Hardware Mínimo**

Como hardware mínimo, para o computador de desenvolvimento será necessário uma máquina com 1.0 Ghz de velocidade de processamento, 512Mb de *ram*, já o servidor já é um computador com um processador de 1.6 Ghz e 1.0 Gb de *ram*.

Vale lembrar que o servidor é destinado somente a gerenciar aplicações *PHP*, e que nos dois casos, servidor e estação de desenvolvimento, o desempenho será bastante limitado.

### **3.4.3 Descrição do Software**

O software necessário para utilização do *MyFuses* pode ser descrito pelo ponto de vista do servidor *HTTP*, contando com a configuração da extensão *PHP* que será utilizada neste servidor, do sistema operacional, e a *IDE* para desenvolvimento de aplicações.

O servidor *HTTP* testado que tem a utilização garantida é o *Apache 2.0* ou superior, da *Apache Foundation*. Já a extensão do *PHP* utilizada na configuração deste apache, é a versão 5.2 ou superior, com as extensões de *json*, *libxml*, *Reflection*, *SimpleXML*, *SPL* e *xml*. Se a aplicação necessita de acesso a *SGBS*, *LDAP*, ou qualquer outro recurso, habilite as extensões necessárias para o funcionamento destes recursos.

Quanto ao sistema operacional, todos os sistemas operacionais que suportem o servidor *HTTP Apache*, são compatíveis com o *MyFuses*, por exemplo: Windows, Linux, Unix entre outros, descritos no site do fornecedor.

Sobre a *IDE* para desenvolvimento, é interessante, do ponto de vista de produtividade e qualidade de desenvolvimento, adotar ferramentas que tenham o recurso de completar códigos e mapear bibliotecas, e tenham uma perspectiva de projeto. No mercado existem várias *IDEs* que possuem estes recursos para linguagem *PHP*, podemos citar alguns como, o *Zend Studio for Eclipse* da *Zend Technologies*, Eclipse com o *plugin PDT(PHP Development*

*Tools*) da *Eclipse Foundation*, entre entre outras.

#### **3.4.4 Configuração da Rede**

As configurações de rede necessárias para desenvolver aplicações *MyFuses* são as mesmas que as adotadas para uma aplicação *web* em *PHP* comum. A maneira de como a rede será configurada depende do planejamento do projeto feito por uma equipe infraestrutura, considerando aspectos importantes como a demanda esperada de usuários do sistema, performance, entre outras coisas, o *MyFuses* não demanda nenhuma configuração especial da rede.

#### **3.4.5 Configuração do Banco de Dados**

O *MyFuses* não controla a camada de acesso a banco de dados de uma aplicação. Por esse motivo é irrelevante discutir a configuração de Banco de Dados, este fator vai depender dos critérios adotados para adoção de um tipo de *SGBD*. Todos os tipos de banco de dados e forma de acesso a eles compatíveis com *PHP* são suportados pelo *MyFuses*, em sua essência, as aplicações desenvolvidas com a ferramenta são aplicações normais em *PHP*, porém estruturadas de uma forma mais racional e organizada.

#### **3.4.6 Descrição das Instalações**

Não se aplica.

#### **3.4.7 Pessoal Exigido**

A equipe alocada em projeto que adotará o *MyFuses* como plataforma de desenvolvimento terá os mesmos integrantes de um projeto normal, e estará sujeita a uma variação de pessoas e papéis de acordo com o planejamento, orçamento, tamanho e prazo do projeto.

### **3.5 Relação Custo Benefício**

Para que a pesquisa cumpra seus objetivos, é necessário observar algumas relações de custo benefício, que são elas:

### ***3.5.1 Robustez x Performance***

Como um dos objetivos da pesquisa é gerar um novo modelo de estruturas de dados orientado a objetos, como o Fusebox representava seus dados em forma de array, é necessário ter muito cuidado para que a solução não se torne muito lenta em relação a ferramenta de origem.

### ***3.5.2 Cache x Acesso a disco***

O PHP tem uma peculiaridade, que é não controlar o estado de aplicação, dando a possibilidade de gerenciar o estado de sessão, que é armazenado, por padrão, em arquivos no disco. A mesma estratégia será utilizada no gerenciamento que o *MyFuses* fará das variáveis de aplicação, porém é necessário racionalizar o acesso a disco desnecessário, para evitar a troca de contexto ao trabalhar com o cache das estruturas geradas pela ferramenta.

### ***3.5.3 Inovação x Compatibilidade com o Legado***

O *MyFuses* tem como alvo inovar algumas funcionalidades do Fusebox, porém é importante que a maioria do código legado funcione na nova ferramenta, para facilitar o esforço de migração.

## ***3.6 Áreas Afetadas***

As áreas afetadas pelo *MyFuses* são as responsáveis pelo desenvolvimento e manutenção da infra-estrutura de uma instituição. É necessário que a área de desenvolvimento aprenda a utilizar a ferramenta, e recomendado que o pessoal de infra-estrutura tenha noção da arquitetura e configuração da biblioteca e aplicações nos servidores.

#### **4 METODOLOGIA DA PESQUISA**

Do ponto de vista da natureza a pesquisa se classifica como pesquisa aplicada.

Do ponto de vista da abordagem do problema, podemos dizer que a pesquisa é quantitativa, explicativa e intervencionista.

## 5 DETALHAMENTO DA PESQUISA

Será abordado o *Fusebox* para PHP versão 4.1, por ser a única versão completamente funcional, porém alguns conceitos adicionados no *Fusebox* 5.0 e 5.5 para *ColdFusion* serão considerados, para abranger a evolução do *framework*. Isto não afetará o resultado do trabalho, pois não houve alterações estruturais consideráveis entre estas versões.

### 5.1 Vantagens

Além de ser uma ferramenta para estruturar aplicações *web*, o *Fusebox* serve para orientar a análise e implementação, fazendo que a equipe de desenvolvimento tenha como base as estruturas lógicas do *framework* para montagem do software.

A *Fusebox Corporation* também desenvolveu um processo que controla o ciclo de vida de um projeto (*FLIP*), e boas práticas de desenvolvimento que são divulgados pelos desenvolvedores do *Fusebox* e membros da comunidade de usuários. A ferramenta agrega a equipe de desenvolvimento uma cultura de boas práticas e um processo bem definido.

Entre as vantagens ao utilizar o *Fusebox* em um projeto, estão:

- a análise do projeto é orientada pelas estruturas do *Fusebox*;
- melhora as métricas, e por consequência os prazos;
- contempla um ciclo de desenvolvimento o *FLiP*;
- padroniza a comunicação da equipe de desenvolvimento;
- documentação mínima e obrigatória de arquitetura (arquivos descritores *xml*);
- arquitetura com baixo acoplamento e alta coesão;
- produtividade durante a manutenção.

### 5.2 Funcionamento

O *Fusebox* é um controlador que é mapeado por arquivos descritores *xml*, que definem toda a estrutura e fluxo de execução de uma aplicação. Porém o custo de processamento desses arquivos é muito alto, para resolver este problema, é feito um cache das estruturas geradas para agilizar a resolução de uma requisição.

Existem dois modos em que o *Fusebox* opera, desenvolvimento e produção. No modo de desenvolvimento, os arquivos descritores são processados em toda requisição, considerando que eles são alterados constantemente. Já no modo de produção, este processamento só ocorrerá uma vez, e as próximas requisições utilizarão o cache, neste caso é considerado que como a aplicação foi finalizada, os arquivos de configuração não sofrem alteração. Para aplicações com muitas estruturas definidas nos arquivos de configuração, esta economia de processamento é visível, ao ponto de diminuir o tempo de execução de uma requisição de segundos para décimos ou centésimos de segundo.

Existe uma variável chamada *fuseaction variable*, que é o ponto de chaveamento das funcionalidades que o controlador precisa resolver e mandar a resposta para o usuário. Esta variável recebe o nome de um circuito e um nome de um *fuseaction*, estas estruturas serão detalhadas na explicação dos conceitos básicos, a partir deste momento, ele executa o processamento da requisição de acordo com a figura 5.1.

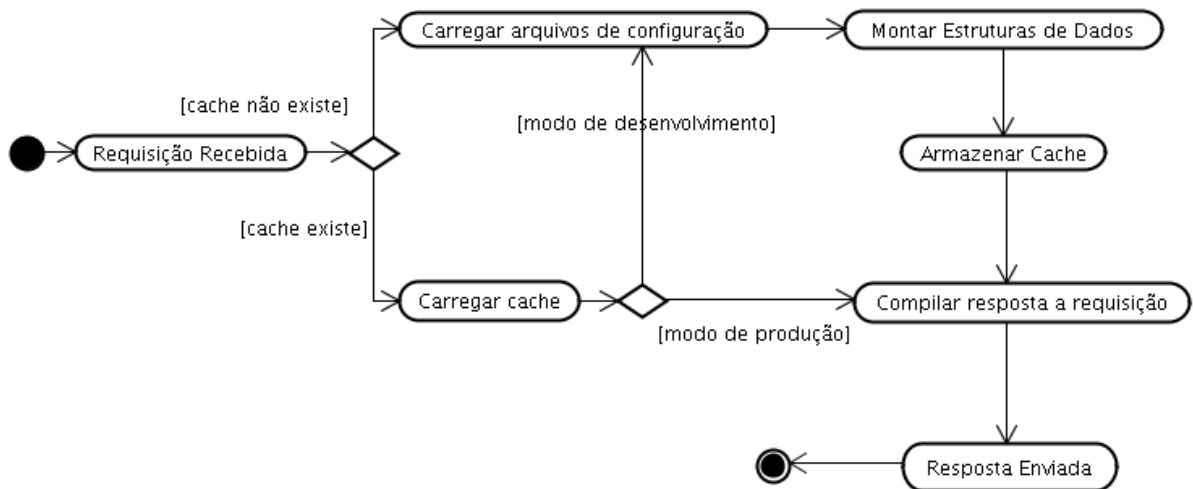


Figura5.1: Ciclo de básico de uma requisição

### 5.3 Conceitos Básicos

Para compreender o funcionamento e estrutura do *Fusebox*, é preciso entender alguns conceitos básicos a fim de visualizar a estrutura de uma aplicação, como ela é configurada e como se divide.

### 5.3.1 Aplicação

Uma aplicação *Fusebox* é uma estrutura criada para resolver um domínio particular de um problema, considerando seu escopo e limitações . Exemplos de aplicações que podem ser desenvolvidas em *Fusebox* são:

- cadastros em geral;
- portais ou gerenciadores de conteúdos;
- ferramentas de pesquisa de conteúdo;
- em fim, aplicações utilizadas em ambiente *web*.

Basicamente, uma aplicação *Fusebox* é composta de circuitos(*circuits*), que correspondem a diretórios onde se encontram a implementação dos requisitos funcionais do software, que são os *fuseactions*, utilizados tanto como uma resposta final para o usuário(*html, xml, pdf, json*), quanto como uma funcionalidade interna da aplicação, que pode ser reutilizada em vários pontos do sistema (QUARTO-VONTIVADAR, 2003, at al.). Um *fuseaction* possui ou mais fuses, arquivos PHP com uma finalidade bem clara e específica, por exemplo, conectar no banco ou exibir a tabela de usuários, aumentando a clareza e o entendimento das funcionalidades da aplicação. Há uma abordagem, mais orientada a objetos, onde os fuses são representados por métodos de classes ou objetos mapeados na aplicação.

Segundo (PEETERS, 2005, at al.), existem dois tipos de arquivos que compõem uma aplicação *Fusebox*, o arquivo que define a aplicação, o "*fusebox.xml*", e o arquivo que define os *circuits* "*circuit.xml*". O arquivo da aplicação, que entre outras coisas, define o endereçamento e nome dos circuitos, modelando a estrutura da aplicação. Para cada circuito definido, será criado um arquivo "*circuit.xml*" dentro de seu diretório que define seus *fuseactions*.

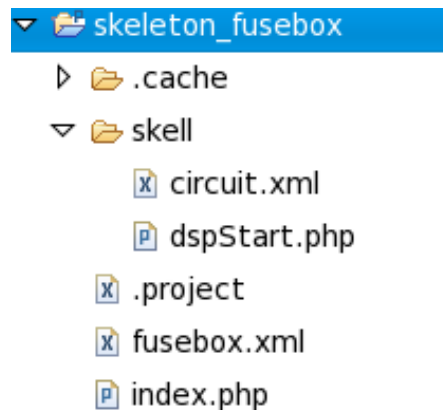


Figura5.2: Estrutura básica de uma aplicação *Fusebox*.

Na figura a cima é possível visualizar a estrutura de uma aplicação desenvolvida com o *Fusebox*. Toda requisição é recebida pelo arquivo “index.php”, ele inclui a biblioteca e executa o processamento da requisição. No arquivo “*fusebox.xml*” são colocadas as definições de um circuito conforme a figura abaixo:

```

1<?xml version="1.0" encoding="UTF-8"?>
2
3<fusebox>
4  <circuits>
5    <circuit alias="skell" path="skell/" parent="" />
6  </circuits>
7
8
```

Figura5.3: Definição de um circuito do arquivo “*fusebox.xml*”.

Dentro do diretório do circuito encontra-se um arquivo “*circuit.xml*” que define um *fuseaction*:

```

1<?xml version="1.0" encoding="UTF-8"?>
2<circuit access="public">
3
4  <fuseaction name="start" >
5    <include file="dspStart.php" />
6  </fuseaction>
7
8</circuit>
```

Figura5.4: Estrutura básica de uma aplicação Fusebox

Este *fuseaction* adiciona o fuse “*dspStart.php*” no processamento, ou seja o controlador *Fusebox*, ao receber uma requisição ao *fuseaction start* do circuito *skell*, exibirá o conteúdo da página “*dspStart.php*”.

### 5.3.2 Circuito(Circuits)

Circuitos são unidades lógicas que compõem uma aplicação *Fusebox*. É necessário no mínimo um *circuit* para adicionar uma *fuseaction* em uma aplicação. Circuitos são agrupadores das funcionalidades de um sistema (FUSEBOX CORPORATION, 2008).

Os nomes utilizados para definir os circuitos, geralmente serão os mesmos termos utilizados por um usuário para descrever um sistema. Por exemplo, durante a entrevista de levantamento de requisitos, o usuário demanda um formulário para adicionar e editar uma pessoa, além de uma tela que traga a listagem das pessoas cadastradas em um sistema, podemos concluir que a aplicação terá um circuito chamado pessoa.

Gerentes de projeto e analistas de negócio costumam ver o circuito como um módulo ou um pacote da aplicação, o que não deixa de ser uma boa formas de definir esta estrutura.

Os circuitos são mapeados dentro do arquivo de configuração da aplicação, “*fusebox.xml*”, definindo o seu nome e endereço físico. A partir do momento da definição de circuito, é necessário criar um diretório com o mesmo nome utilizado no mapeamento do arquivo “*fusebox.xml*”, e dentro deste diretório, criar o arquivo de configuração deste circuito “*circuit.xml*”, onde será mapeado seus *fuseactions*, ou seja, suas funcionalidades.

Exemplos:

```

15 <circuits>
16   <circuit alias="main" path="main/" parent="" />
17   <circuit alias="basic" path="basic/" parent="main" />
18 </circuits>

```

Figura5.5: Mapeamento dos *circuits* dentro do arquivo “*fusebox.xml*”

```

1<?xml version="1.0" encoding="UTF-8"?>
2<circuit access="public">
3
4</circuit>

```

Figura5.6: Mapeamento dos *circuits* dentro do arquivo “*fusebox.xml*”

### 5.3.3 Fuseaction

*Fuseactions* são as funcionalidades que um circuito disponibiliza para uma aplicação. Para ajudar com a compreensão desta estrutura, de uma forma bem simples, todas as páginas

de um sistema são *fuseactions*(PEETERS, 2007, at al.). Por exemplo, a página que exibe a lista de pessoas, seria o *fuseaction* listar do circuito pessoa.

O conceito de *fuseaction* não está ligado obrigatoriamente a uma página da aplicação, mas sim a uma funcionalidade, reuso de código. É possível *fuseactions* que ao invés de imprimir uma tabela com a lista de pessoas na tela do navegador, simplesmente crie um *array* de objetos do tipo Pessoa, possibilitando que o sistema utilize esta funcionalidade toda vez que necessitar desse tipo de dado.

O *fuseaction* é utilizado para executar todo tipo de operação, conexão com banco de dados, impressão de um *html* ou *pdf*, ou qualquer outro tipo de processamento. O valor mais importante que essa estrutura agrega é que conseguimos carregá-las em qualquer parte do sistema.

Exemplo:

```

1<?xml version="1.0" encoding="UTF-8"?>
2<circuit access="public">
3
4  <fuseaction name="list">
5    </fuseaction>
6
7</circuit>

```

Figura5.7: Definição de um *fuseaction* dentro de um arquivo “*circuit.xml*”.

### 5.3.3.1 Fuseactions Globais

*Fuseaction* globais são estruturas utilizadas para melhorar a reutilização de código durante a montagem de uma aplicação.

Toda requisição ao controlador recebe o nome de processamento, o Fusebox estabeleceu algumas fases para resolução desta requisição, que são, pré-processamento, pós-processamento, *pré-fuseaction* e *pós-fuseaction*.

É possível estabelecer *fuseactions* globais de pré-processamento e pós-processamento tanto no escopo de aplicação como no de circuito, facilitando a montagem de cabeçalhos, rodapés, conexão de banco de dados, ou qualquer outra função que necessite ser executada em um contexto global.

Uma boa prática, no que diz respeito a *fuseactions* globais é focar funcionalidades

particulares da aplicação, para funcionalidades mais gerais, que podem ser utilizadas em várias aplicações, como rotinas de segurança ou *log*, é melhor utilizar outro recurso do *Fusebox*, *plugins*, que serão explicados mais adiante.

Os *fuseactions* globais são definidos nos arquivos de configuração, assim como representado nas figuras 5.8 e 5.9.

```

49 <globalfuseactions>
50 <preprocess>
51 <do action="principal.cabecalho" />
52 </preprocess>
53 <postprocess>
54 <do action="principal.rodape" />
55 </postprocess>
56 </globalfuseactions>

```

Figura5.8: Definição dentro de um arquivo “*fusebox.xml*”.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <circuit access="public">
3
4 <prefuseaction>
5 <do action="rodape" />
6 </prefuseaction>
7
8 <postfuseaction>
9 <do action="cabecalho" />
10 </postfuseaction>

```

Figura5.9: Definição dentro de um arquivo “*circuit.xml*”.

O processamento de uma requisição, considerando os *fuseactions* globais seguem o fluxo da figura 5.10.

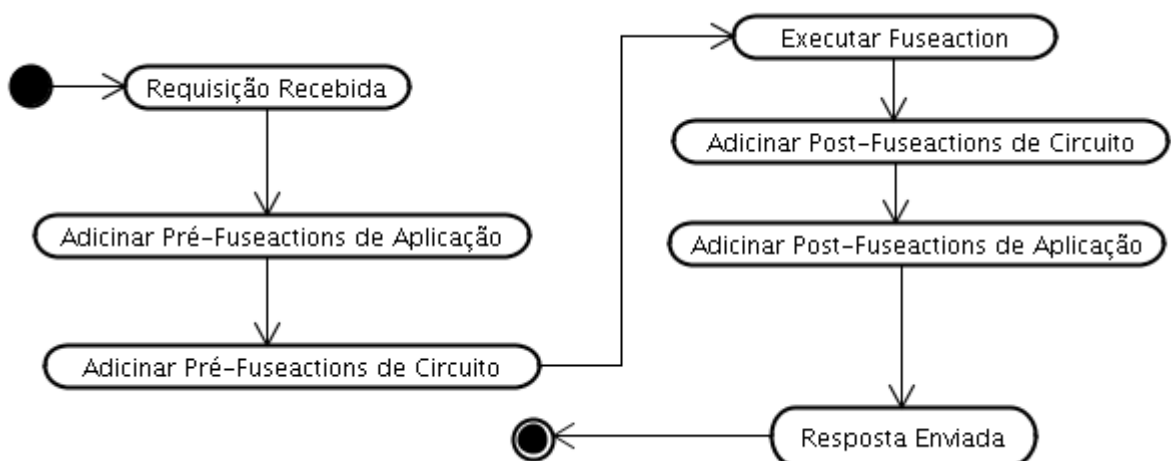


Figura5.10: Ciclo de vida de uma requisição considerando os *fuseactions* globais.

Um dos defeitos encontrados no *Fusebox* para PHP, na versão 4.x, é a implementação dos *fuseactions* globais de circuito, eles fazem a aplicação entrar em *loop* infinito, já a versão 5.0 conseguiu retroceder neste aspecto, além de não corrigir o erro da versão anterior, ao utilizar os *fuseactions* globais de aplicação, a aplicação também entra em *loop* infinito.

### 5.3.4 Verbo(Verb)

Os verbos(*verbs*) são pequenos comandos que são executados durante o processamento de um *fuseaction*. Os *verbs* possuem uma finalidade específica, com um tipo de processamento próprio, e a sua principal objetivo é automatizar funções, como inclusão de páginas, execução de outros *fuseactions*, redirecionamento para outras páginas, entre outras coisas (PEETERS, 2007, at. al.). Há também verbos que são utilizados para interferir no fluxo de execução de um *fuseacion* como estruturas de repetição e condicionais.

Tomando como exemplo um circuito chamado pessoa e o *fuseaction* *exibir*, é preciso que executar algumas pequenas ações, verbos, para chegar ao objetivo final que é *exibir* uma pessoa, este exemplo está representado na figura 5.11.

```

2<ircuit>
3  <fuseacition name="exibir">
4    <do action="banco.conectar" />
5    <do action="pessoa.listar" />
6    <if condition="count($pessoas)">
7      <true>
8        <include file="dspListaPessoas.php" />
9      </true>
10     <false>
11       <include file="dspListaVazia.php" />
12     </false>
13   </if>
14 </fuseacition>
15</ircuit>

```

Figura5.11: *Fuseaction* pessoa.exibir

Neste exemplo é possível identificar cinco verbos dentro do *fuseaction* *exibir* do circuito *pessoa*, que são:

- *do*, executa um outro *fuseaction*;
- *if*, divide o processamento em verdadeiro e falso;
- *true*, parte verdadeira do processamento do *if*;

- *false*, parte falsa do processamento do if;
- *include*, inclui um arquivo no processamento;

Um recurso interessante inserido no *Fusebox* 4.1 são os *lexicons*, ou verbos customizados (*custom verbs*), que possibilita estender o *framework* sem alterar o código fonte da biblioteca. Os *lexicons* eram configurados no arquivo “*fusebox.xml*” e tinham a sua implementação bastante confusa, por este motivo, da versão 5.0 em diante, os *lexicons* foram retirados do arquivo “*fusebox.xml*”, e seu método de implementação ficou mais simplificado e claro.

### 5.3.5 Plugins

*Plugins* são utilizados para adicionar uma lógica mais elaborada a estrutura do *framework* de forma simples e organizada, sem alterar o *Fusebox*. A utilização do *plugin* pode ser confundida com a de um *fuseaction* global, a diferença básica é que eles tem um foco mais estrutural, são utilizados para adicionar recursos de segurança, montagem de *links* da aplicação, logs, processamento de erros, raramente são utilizados para atuar no negócio da aplicação.

Plugins são definidos no arquivo “*fusebox.xml*”, figura 5.12, e eles podem atuar, nas fases de pré-processamento, pós-processamento, pré-*fuseaction*, pós-*fuseaction*, erro no processamento e erro no *fuseaction*. As fases de erro de processamento e *fuseacion*, não são implementadas pelo *Fusebox* para PHP, por isto não serão considerado nesta pesquisa.

```

56     <plugins>
57         <phase name="preProcess">
58             <plugin name="BreadCrumb" path="plugins/BreadCrumb/" />
59         </phase>
60         <phase name="preFuseaction">
61         </phase>
62         <phase name="postFuseaction">
63         </phase>
64         <phase name="fuseactionException">

```

Figura5.12: Definição de um plugin de pré-processamento no arquivo “*fusebox.xml*”.

O processamento do *plugin* é mandatório em relação ao processamento de um *fuseaction* global durante uma fase da resolução de uma requisição, primeiro o *plugin* será executado, logo em seguida o *fuseaction*.

O processamento de uma requisição, considerando os *plugins* seguem o fluxo da figura 5.13.

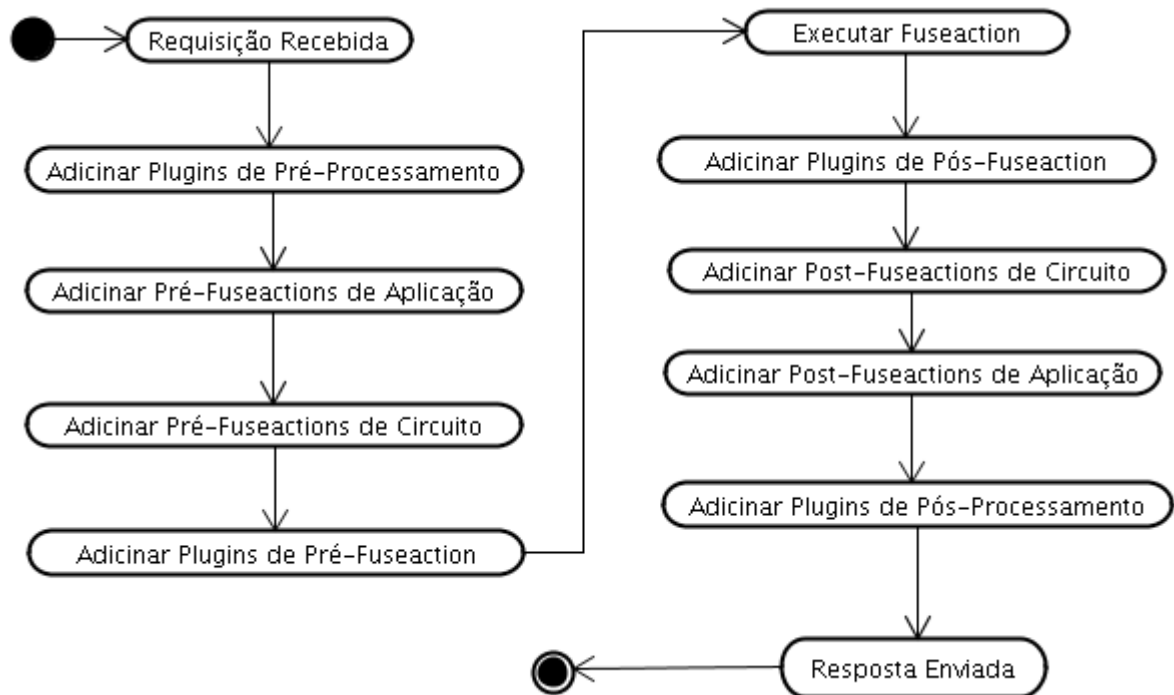


Figura5.13: Ciclo de vida de uma requisição considerando os plugins.

O grande problema dos *plugins* no *Fusebox*, é que eles são arquivos PHP com uma programação que atua no contexto global, é preciso criar um arquivo para registrar o *plugin* no *array* global *myFusebox*, e depois criar o arquivo que contém a execução da funcionalidade. É possível alterar o *array myFusebox* em qualquer hora do processamento e em qualquer lugar da aplicação, tornando esta estrutura muito insegura. O *plugin* será um dos pontos de extensão que sofrerão mais mudanças no *MyFuses*.

#### 5.4 Classes Mapeadas

As classes mapeadas servem para adicionar a implementação de uma aplicação, uma abordagem mais orientada a objetos. É possível considerar que cada método público de uma classe mapeada seja um fuse, o que dá mais flexibilidade ao desenvolvimento, pelo fato de objetos apresentarem encapsulamento, herança, polimorfismo, além de vários outros recursos e técnicas de padrão de projetos disponíveis para utilização.

Uma classe mapeada é definida no arquivo “*fusebox.xml*” de acordo com a figura

5.14.

```

21 <classes>
22   <class alias="Account" classpath="classes/Account.class.php" />
23 </classes>

```

Figura5.14: Definição de uma classe no arquivo “fusebox.xml”.

Após mapeada, uma classe pode ser utilizada em um *fuseaction* utilizando os verbos *instantiate* e *invoke*, o primeiro, serve para instanciar um objeto, o segundo serve para executar um método deste objeto.

```

18 <fuseaction name="instantiateInvoke">
19   <include file="dspInstantiate.php" />
20   <instantiate class="Account" object="account" >
21     <argument value="tgreets" />
22   </instantiate>
23   <invoke object="account" method="getName" returnvariable="login" />
24   <include file="dspInvoke.php" />
25   <invoke object="account" method="setName" >
26     <argument value="rmark" />
27   </invoke>
28   <invoke object="account" method="getName" returnvariable="login" />
29   <include file="dspInvoke.php" />
30 </fuseaction>

```

Figura5.15: Utilização de classes mapeadas em um *fuseaction*.

O exemplo da figura 5.15, mostra a utilização do *invoke* e do *instantiate* utilizando o padrão do *Fusebox* 5.0, na versão anterior, em vez de utilizar o parâmetro *method*, era utilizado *methodcall*, onde era informado a execução do método com seus argumentos, o que deixava a leitura do *xml* um pouco confusa, com a abordagem nova, fica mais fácil de trabalhar com estes verbos.

## 5.5 FLiP

O *Fusebox Lifecycle Process (FLiP)* é uma metodologia de desenvolvimento de projetos criada pela *Fusebox Corporation* que tem o objetivo aumentar a chance de sucesso durante a produção de um software (FUSEBOX CORPORATION, 2008).

A premissa segundo (CORFIELD, 2008) do FLiP é que nenhum desenvolvimento da parte de negócio deve ser iniciado até a interface com usuário (*front end*) estiver completa, a idéia é eliminar tudo o que o usuário possa interferir antes de do desenvolvimento da lógica do software. Este raciocínio esta no fato que os usuários não sabem muito o que eles querem

antes de ver algo concreto, como uma tela ou um relatório. O foco desta metodologia é o aceite do usuário.

Um projeto de software envolve o usuário em dois pontos do processo, no começo, onde é muito cedo para os usuários darem um feedback relevante, e no fim, quando é muito tarde. As maiores reclamações sobre projetos de software são: o sistema não faz aquilo que o usuário deseja.

As fazem que fazem parte do FLip são:

#### **5.5.1 Wireframing**

É um esqueleto simples da aplicação, sem nenhuma parte gráfica. Tem o objetivo de definir o fluxo da aplicação e a lógica de negócio que o cliente necessita em sua aplicação.

#### **5.5.2 Desing de Templates**

É a criação das telas de interface com o usuário.

#### **5.5.3 Prototipação e DevNotes**

É junção da primeira e segunda fase. Utilizando o wireframe como mapa da aplicação, e os *templates* como a base da interface com o usuário, o protótipo é criado.

O protótipo consiste de HTML estático, nenhum código de negócio é criado, o objetivo é montar uma estrutura navegável com as telas da aplicação, para que o usuário tenha noção do produto que será gerado. Esta método de trabalho facilita o serviço em alterar o protótipo.

O cliente verifica o protótipo e adiciona comentários e perguntas no *DevNotes*, um fórum de discussão bem simples, que está anexado a cada tela da aplicação.

#### **5.5.4 Arquitetura e Fusedocs**

A partir do momento que o protótipo é finalizado, o arquiteto começa a quebrar a aplicação em circuitos, fuseactions e fuses. Para cada fuse é adicionado um comentário com a descrição do objetivo necessário a alcançar no trecho de código.

Nesta fase é feita a conexão das percepções do usuário a respeito do sistema, com os

objetivos a serem cumpridos pelos desenvolvedores.

#### ***5.5.5 Codificação***

Codificação dos fuses em conformidade com os fusedocs.

#### ***5.5.6 Teste***

Nesta fase é executado um teste para cada fuse, verificando se a funcionalidade está funcionando corretamente e se estão retornando os valores corretos com relação a suas entradas.

## 6 PROTÓTIPO

Nesta parte serão abordados os aspectos envolvidos na construção do protótipo da pesquisa, assim como alguns pontos principais, como estratégias de desenvolvimento, estrutura, principais diferenças e recursos novos adicionados.

### 6.1 Contexto

Vários problemas foram encontrados na implementação do *Fusebox*, porém o mais grave está em sua concepção, ou estratégia de desenvolvimento, de forma estruturada, com várias funções que atuam e criam variáveis e estruturas em contexto global. A falta de encapsulamento é um ponto de vulnerabilidade que direcionou o desenvolvimento do *MyFuses* orientado a objetos.

Outro ponto que foi abordado foi a melhora dos pontos de extensão da ferramenta, para isso foi pesquisado uma série de técnicas e padrões de projeto com o objetivo de construir uma plataforma fácil e intuitiva de desenvolver e anexar novas estruturas e funcionalidades sem alterar o núcleo da ferramenta, dois exemplos de componentes que tiveram uma atenção especial foram os *plugins* e os verbos customizados.

No desenvolvimento *MyFuses* um dos principais objetivos foi adicionar a ferramenta a capacidade de trabalhar com mais de uma aplicação em um único controlador, fazendo com que as funcionalidades destas aplicações estivessem disponíveis entre elas, além de criar novas funcionalidades para receber e produzir *webservices*.

### 6.2 Principais Diferenças

Segue uma descrição das principais diferenças, do ponto de vista do usuário final, ou seja os desenvolvedores de aplicações, entre o *MyFuses* e o *Fusebox*.

Uma aplicação desenvolvida em *Fusebox* funcionará perfeitamente se for portada para *MyFuses*, mas será necessário reescrever os *plugins* e *custom verbs*, se a aplicação utilizar estes recursos.

Por padrão, o arquivo de definição da estrutura da aplicação terá o nome de “myfuses.xml” ao invés de “fusebox.xml”, mas o *MyFuses* aceita as duas nomenclaturas. Esta funcionalidade tem o objetivo de facilitar a migração das aplicações legadas para a biblioteca

nova.

É possível mapear no arquivo “*index.php*” várias aplicações, fazendo com que uma instancia do controlador consiga compartilhar os seus recursos entre elas.

Com relação aos *plugins* e verbos customizados, sua implementação segue um padrão definido por algumas interfaces e algumas classes abstratas que auxiliam o desenvolvimento de novos tipos destas estruturas.

O gerenciamento do cache do *MyFuses* é centralizado, ou seja, é gerado em uma pasta localizada na raiz da biblioteca. No *Fusebox* o cache era armazenado na raiz da aplicação, complicando a publicação e manutenção da aplicação em ambiente de versionamento(*cvs*, *subversion*, etc...), geralmente o desenvolvedor publicava a pasta de cache de seu ambiente de desenvolvimento no servidor de produção, trazendo alguns transtornos. O cache centralizado diminuí este risco, e ao invés da equipe de publicação se preocupar com a limpeza do cache em vários pontos, um único ponto dever ter uma atenção especial, que é a pasta “*parsed*” localizada na raiz da biblioteca.

As mensagens de erros geradas pela aplicação mostram descrições detalhadas sobre qual estrutura que está ocasionando o erro, com uma sugestão do tipo de ação que deve ser tomada para resolver o problema. Esse tipo de melhoria facilita o trabalho de depuração, principalmente quando a aplicação atinge um certo número de circuito e fuseactions, já que sem esse recurso, a manutenção se torna um pouco cansativa.

Como a estrutura do *MyFuses* é toda orientada a objeto, se alguma estrutura do *Fusebox* foi utilizada em alguma parte da aplicação, será necessário reescrever este trecho. Apesar deste custo de migração, as estruturas do *MyFuses* são muito mais seguras do que as do *Fusebox*, pois seus dados são encapsulados e protegidos contra uma alteração acidental, por parte de um desenvolvedor, fazendo com que a aplicação tenha seu funcionamento comprometido.

### **6.3 Componentes Principais do MyFuses**

O framework divide-se basicamente em *core*(núcleo), pacote onde as estruturas de dados do *MyFuses* foram desenvolvidas, *engine*(motor) pacote onde os componentes responsáveis pelo carregamento e montagem das estruturas se encontram, e *process*(processo)

pacote dos componentes responsáveis pelo ciclo de vida da aplicação, e requisição do usuário.

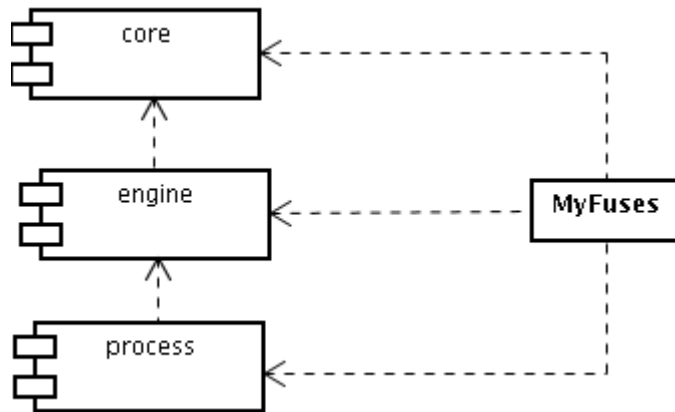


Figura6.1: Componentes principais do *MyFuses*.

### 6.3.1 Núcleo(Core)

A estrutura utilizada pelo *MyFuses* é basicamente uma árvore de objetos, onde o relacionamento hierárquico entre a raiz e suas folhas podem ter vários significados, mas eles estão dispostos estrategicamente com o objetivo de facilitar geração de código tanto para cache, como para execução das instruções de uma requisição.

Basicamente o controlador possuía somente uma aplicação, porém em um segundo momento foi adicionado a habilidade de controlar mais de uma aplicação. Cada aplicação possui um ou mais circuitos, cada circuito um ou mais *fuseactions*, e cada *fuseaction* vários verbos. A aplicação pode armazenar vários *plugins*, mapeamento de classes e parâmetros de inicialização. Estes são os componentes que forma estrutura do núcleo(*core*) do *MyFuses*.

Uma premissa foi adotada na implementação das entidades, para cada entidade desenvolvida, criar uma interface para definição do comportamento delas durante o processamento de uma requisição, para depois implementar classes abstratas ou concretas. Esta abordagem favorece a extensibilidade das estruturas da ferramenta.

Duas interfaces foram criadas no núcleo para facilitar a geração de cache e código a partir de um elemento raiz da árvore, *ICacheable* que define o comportamento de um elemento que tenha que gerar cache, e o *IParseable* que define o comportamento de um elemento que tenha que gerar código de execução.

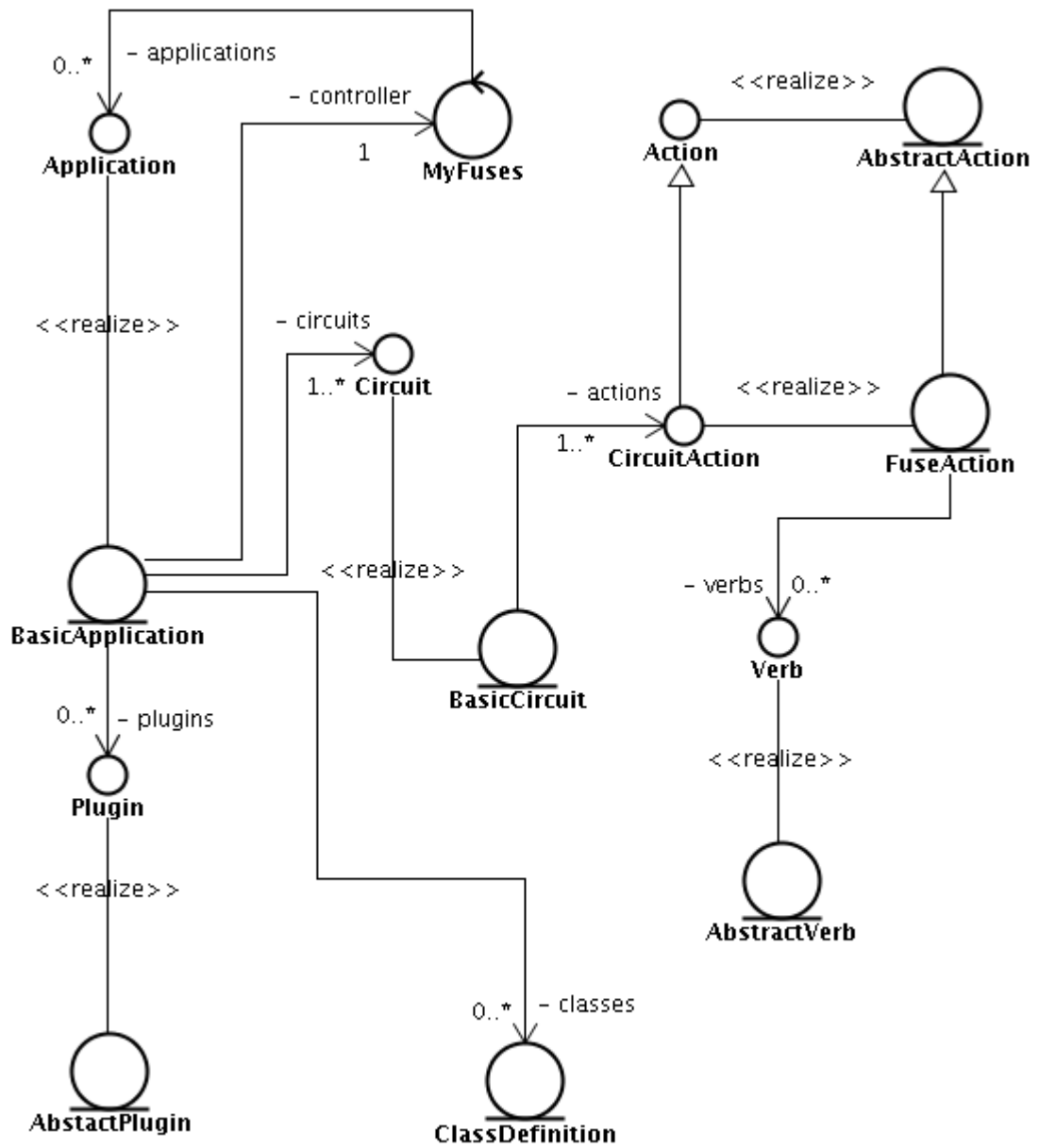


Figura6.2: Estrutura básica dos elementos do núcle(*core*).

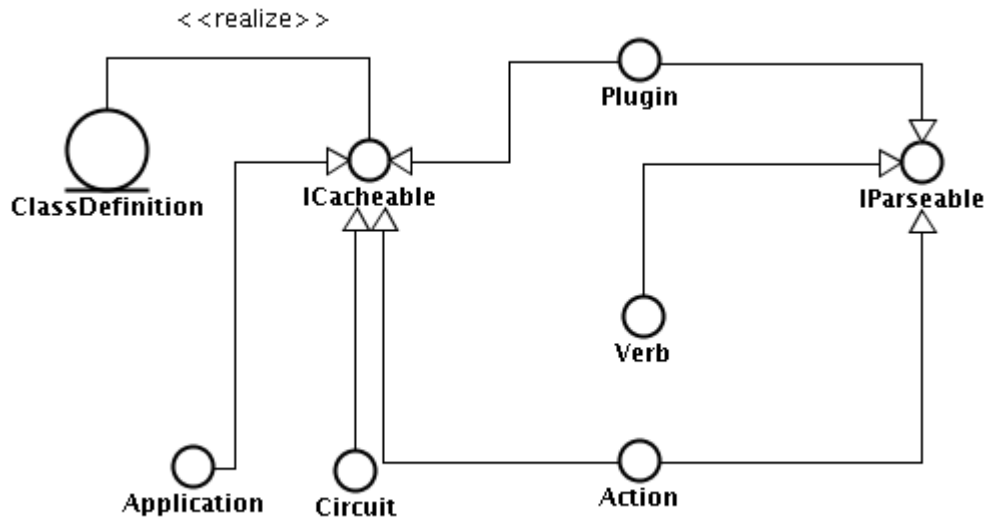


Figura6.3: Interfaces responsáveis por cache e geração de código.

### 6.3.2 Motor(Engine)

O Motor(Engine) é a parte do *framework* que tem o objetivo de fazer três duas funções, carregar os arquivos descritores *xml* e montar as estruturas de dados em memória.

Esta parte do *MyFuses* é a mais sensível pois o maior gasto de processamento no que diz respeito às estruturas de dados estão aqui, por este motivo várias abordagens de desenvolvimento foram utilizadas a fim de otimizar seu funcionamento e aproximar o processamento do *MyFuses* com o *Fusebox*.

A primeira abordagem adotada no desenvolvimento do motor foi carregar todas as estruturas para memória, montar o cache e depois montar a resposta para o usuário, isto dava muita flexibilidade ao usuário da ferramenta, pois ele conseguiria utilizar todas as estruturas definidas sem restrições, mas deixava o processamento muito lento em ambiente de produção, ficando em média 78% mais lento em relação ao *Fusebox*, o que era bastante insatisfatório.

Neste momento o motor só tinha um componente, o carregador(*loader*), que fazia tanto o carregamento, como a montagem das estruturas de dados do núcleo, esta abordagem era um pouco experimental, e era claro que este componente estava sobrecarregado, em relação a seus papéis.

Uma nova abordagem foi utilizada, já que o carregamento de todas as estruturas, causa uma perda de performance. Neste novo modo de executar o carregamento e montagem

das estruturas, um novo componente foi adicionado ao processo, o montador(*Builder*).

Outra novidade foi adicionada ao processamento utilizado pelo motor, em vez produzir e receber o cache das estruturas de dados, seria utilizado um meta-dados com a representação serializada dos objetos a serem utilizados, com isso o carregador, transformava os arquivos descritores em um meta-dados, o montador transformava o meta-dado nas estruturas de dados que seriam utilizadas naquele processamento, e a árvore de meta-dados era passada enviada para o cache.

Com esta abordagem, o *MyFuses* estava 40% mais rápido em relação ao *Fusebox* em desenvolvimento e 69% mais lento em modo de produção. A falha estava no fato de que sempre o montador deveria converter os meta-dados em estrutura de dados válidas, era necessário uma abordagem que misturassem as duas primeiras abordagens, com objetivo de deixar o processamento em modo de produção aceitável.

Com a última abordagem utilizada, estrutura de dados que sempre eram utilizadas, como circuitos, classes, parâmetros de inicialização, *fuseactions* globais e *plugins* sempre são carregados, porém *fuseactions* e verbos são carregados por demanda, somente se necessário. Esta nova forma de executar o processamento foi concebido devido ao fato que a tendência é existir mais *fuseactions* e verbos em uma aplicação do que a soma de todas as outras estruturas existentes, mas durante a execução de um único processamento de uma requisição, o número de *fuseaction* e verbos utilizados é pequeno.

O resultado foi que o *MyFuses* em desenvolvimento ficou 43% mais rápido em relação ao *Fusebox*, e em produção, 31% mais lento. Este desempenho é bastante satisfatório pois as estruturas de dados utilizadas pelo *MyFuses* são muito mais caras, no que diz respeito a memória, do que as utilizadas pelo *Fusebox*, porém a nova ferramenta se tornou muito mais estável e segura.

Outra curiosidade é que com classes mapeadas o *MyFuses* mantém a mesma performance, e em alguns casos, supera o *Fusebox*, principalmente quando o número de classes mapeadas é grande.

### **6.3.3 Processo(Process)**

O processo é a parte do *MyFuses* que controla o ciclo de vida do processamento de

uma requisição. A criação da requisição, o carregamento e montagem das aplicações, compilação e envio da resposta para o usuário, além do armazenamento do cache das estruturas de dados são feitas pelos componentes do processo.

As estruturas componentes do processo tem o objetivo mais de organizar a implementação(código fonte) do que outro objetivo estrutural mandatório, mas que tem fundamental importância para manter a manutenibilidade e clareza do *MyFuses* como um todo.

Os componentes do processo são o ciclo de vida (*MyFusesLifeCycle*), a requisição (*FuseRequest*), o debugador (*MyFusesDebugger*) e a fila de processamento(*FuseQueue*).

#### ***6.4 Padronização dos Pontos de Extensão***

Um dos principais objetivos do *MyFuses* está a padronização dos pontos de extensão da ferramenta, principalmente pelo fato das estruturas de dados foram convertidas para uma estrutura orientada a objetos. O próprio *framework* pode ser estendido assim como os circuitos e fuseactions, fazendo com que seja possível utilizar o *MyFuses* para desenvolver outros *frameworks*.

Mas por padrão, dois pontos de extensão sofreram alterações especiais, para eles, foram definidas interfaces especiais e classes abstratas para facilitar a implementação dessas estruturas, que são os *plugins* e os verbos customizados.

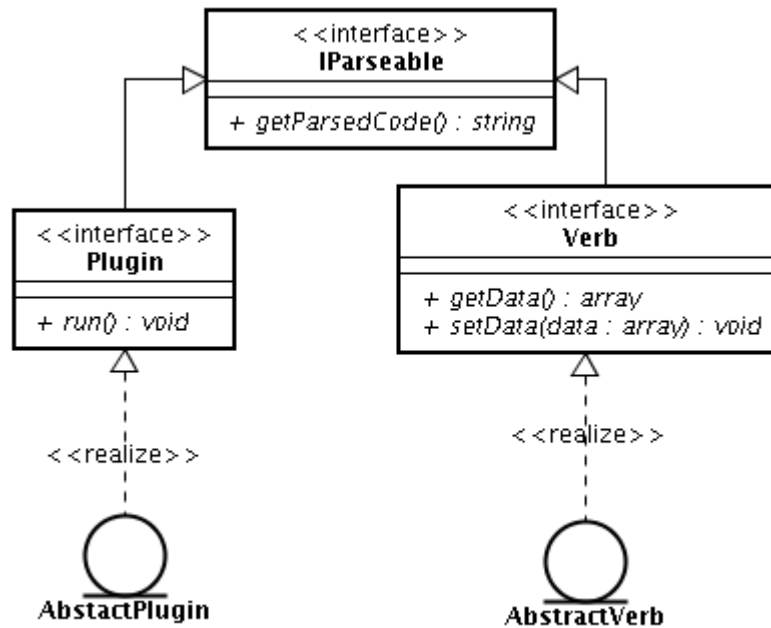


Figura6.4: Interfaces responsáveis por cache e geração de código.

Com essa estrutura, os pontos de extensão padrões, ficam mais fáceis de serem desenvolvidos, sendo necessário só estender de suas classes abstratas, e implementar os métodos necessários impostos pelas interfaces.

### 6.5 REST

A última implementação do *MyFuses* foi o desenvolvimento de verbos que implementassem *REST*, com isso podemos produzir e consumir esse tipo de abordagem de serviço web.

As implementações de *REST* disponíveis no *MyFuses* são *xml* e *json*. Para esses dois tipos de dados foram implementados tanto uma possibilidade de gerar e consumir o serviço.

Para essa implementação foi criado um *namespace* de verbo customizado chamado “data”. Os verbos customizados que fazem parte do “data” são *toJson*, que transforma uma estrutura de dados PHP em *json*, e *fromJson* que transforma um dado do tipo *json* em estruturas PHP, para trabalhar com *xml*, os verbos criados foram *toXml* e *fromXml* que tem o funcionamento dos verbos utilizados para trabalhar em *json*.

## 7 CONCLUSÕES

O *MyFuses* é uma boa alternativa para o desenvolvimento de aplicações PHP utilizando o padrão *Fusebox*, dando mais flexibilidade e extensibilidade a ferramenta e proporcionando maior facilidade de manutenção e correção de erros.

O controlador evoluiu sem comprometer a performance da ferramenta, possibilitando o consumo e produção de serviços, além de suportar mais de uma aplicação, de forma que hoje em dia podemos ver o *MyFuses* como um orquestrador de serviços.

É possível evoluir ainda mais a ferramenta cada vez mais, mesmo por que o padrão da ferramenta de origem não é estático, e uma das metas do *MyFuses* é estar sincronizada com a versão corrente do *Fusebox* para *ColdFusion*.

Uma das evoluções que são possíveis é criar uma forma de um circuito representar um *webservices*, e um *fuseaction*, um método deste serviço. O trabalho de tratamento de exceções pode ser mais elaborado, fazendo com que, ao apresentar algum erro ao desenvolvedor, algumas dicas de resolução sejam apresentados. Outras áreas que podem ser pesquisados, são alternativas que possibilite o balanceamento de carga e clusterização das aplicações.

A grande dificuldade do projeto esteve no fato da linguagem PHP não controlar o contexto de aplicação, fazendo com que fosse necessário simular este contexto. O maior trabalho da pesquisa foi analisar e implementar o carregamento, cache e utilização do contexto de aplicação criado, isto inclui todas as estruturas de dados utilizadas pelo *MyFuses*.

Como todo *framework* para desenvolvimento de aplicações *web*, é necessário desenvolver um ambiente, de forma intuitiva e fácil, que possibilite a criação das estruturas da aplicação com auxílio passo-a-passos e modelos, aumentando assim a produtividade.

A ferramenta desenvolvida será disponibilizada para a comunidade, sob uma licença open-source, possibilitando que outras pessoas contribuam para a evolução da ferramenta. A licença de distribuição do *MyFuses* será a *Mozilla Public Licence 1.1* e o site do projeto será <http://www.sourceforge.net/projects/myfuses>.

## 8 BIBLIOGRAFIA

- QUARTO-VONTIVADAR**, Jhon. Discovering Fusebox – Second Edition. EUA: Techspedition Press, 2003. 469 p.
- PEETERS**, Jeff. The Fusebox Guide Bookiscovering Fusebox. EUA: Proton Arts, 2005. 76 p.
- PEETERS**, Jeff. Fusebox 5 & FLiP: Master Class ColdFusion Apps. EUA: Proton Arts, 2007. 471 p.
- CORFIELD**, Sean: Fusebox Lifecycle Process, disponível em <http://corfield.org/FLiP/index.cfm?&fuseaction=methodology.steps>, Acessado em 05/04/2008.
- Fusebox Corporation**: disponível em <http://www.fusebox.org>, Acessado em 05/04/2008.
- Mozilla Public Licence 1.1**: <http://www.mozilla.org/MPL/MPL-1.1.html>, acessado em 05/05/2008.