



Efortis Relationship  
Management Suite Release 2

Arquitetura de Software

Fortis IT Consulting  
Dezembro/2006

# ÍNDICE

---

1.0. INTRODUÇÃO.....	3
2.0. COMPONENTES .....	4
2.1. <i>FRAMEWORK</i> MVC .....	4
2.2. CLASSES DE PERSISTÊNCIA .....	5
2.3. COMPONENTIZAÇÃO JAVASCRIPT .....	6
2.4. AJAX.....	6
3.0. EFORTIS .....	9
3.1. MVC EFORTIS .....	9
3.2. CAMADA <i>CONTROLLER</i> .....	10
3.3. CAMADA VIEW .....	10
3.4. CAMADA MODEL.....	11
3.5. ESTRUTURA DE NAVEGAÇÃO .....	11
3.6. DESENVOLVIMENTO DISTRIBUIDO .....	13
3.7. PERSONALIZAÇÃO DE CLIENTES .....	13

## 1.0. INTRODUÇÃO

---

Efortis Relationship Management Suite Release 2 é voltado para duas importantes tendências tecnológicas de negócio do momento - Computação Orientada ao Serviço e Modularidade:

- **Arquitetura Orientada ao Serviço:** Uma arquitetura de sistema que permite o desenvolvimento de aplicações corporativas de forma modular e voltada para o serviço. Efortis Relationship Management Suite é estruturado para facilitar a customização e o desenvolvimento de novos módulos orientados ao serviço e às necessidades de negócio.
- **Modularidade:** Efortis Relationship Management Suite apresenta uma arquitetura que coordena o uso de módulos distribuídos em diferentes plataformas e ambientes, permitindo agregar soluções com baixo custo e grande escalabilidade.

Para dar suporte a essas necessidades, a solução foi estruturada através das arquiteturas – Modelo em Três Camadas, Ajax e Webservices, abrangendo:

- **Model View Controller:** Permite a separação entre a camada de apresentação da aplicação da camada de negócio, tornando mais fácil o desenvolvimento, customização, compartilhamento de módulos e manutenção do sistema.
- **Pear::DB:** Camada de persistência flexível e compatível com diversos Bancos de Dados.
- **Asynchronous Javascript and XML:** Proporciona uma interação rica, eficiente e rápida, no que se refere ao acesso e à troca de informações.
- **Webservices:** Compatibiliza, integra e disponibiliza informações para outros sistemas e serviços universalizando o compartilhamento de informações.

## 2.0. COMPONENTES

A arquitetura Efortis Relationship Management Suite é composta por quatro componentes: (1) *Framework MVC*, (2) Classe de Persistência, (3) Componentização Javascript e (4) AJAX.

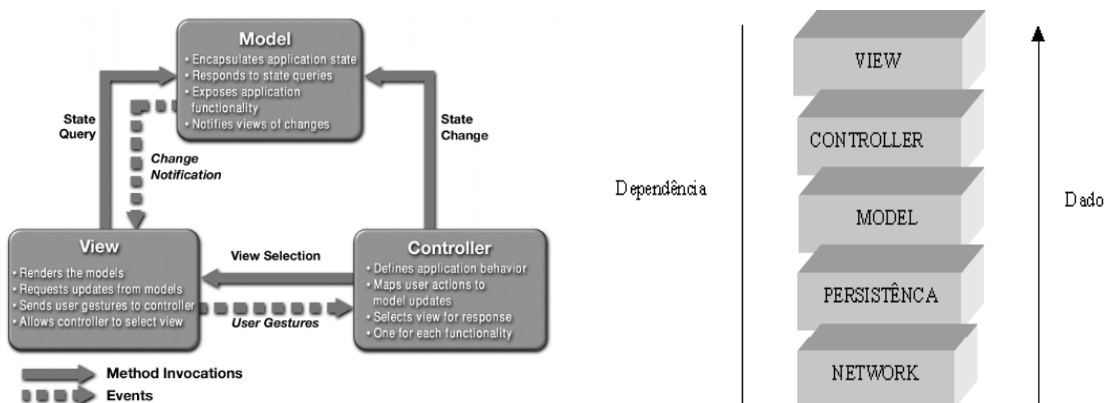
### 2.1. FRAMEWORK MVC

*Framework* voltado para a distribuição, manipulação e visualização de dados similares, por diversas maneiras, através de um *pattern* que facilita o trabalho por conjuntos de habilidades. O MVC eFortis permite a separação da informação, das mais variadas formas, permitindo um rápido e fácil acesso à informação.

O *pattern* é adequado para o desenvolvimento e distribuído em módulos com recursos dinâmicos de personalização, o que promove a portabilidade de interfaces e do *back-end*.

A chave para o MVC é a separação de responsabilidades. As visões podem usar o modelo de dados para exibir resultados, mas elas não são responsáveis por atualizar o Banco de Dados. Os controladores são responsáveis por selecionar uma visão apropriada e por fazer alterações no modelo de dados. O modelo, por sua vez, é responsável por representar os dados da aplicação.

Em certas ocasiões, o modelo de dados inclui a lógica de negócio (as regras para manipular os dados de negócio). Em outras, a lógica existe na camada do controlador.



Estrutura do MVC e esquema de dependência.

Algumas *features* do *framework*:

- **Segurança:** uma única área de leitura por aplicação. Isso torna mais simples a tarefa de proteção do código fonte e dos dados;
- **Instalação Flexível:** o *framework* pode ser instalado fora do diretório raiz do sistema;
- **Projeto Orientado a Objetos:** o *framework* é totalmente implementado com base em orientação a objetos, o que facilita a manutenção;
- **Configuração XML:** toda a parametrização é realizada através de arquivos XML (Extensible Markup Language);
- **Memória Compartilhada:** para a plataforma Unix, há a disponibilidade de utilizar a memória compartilhada para a troca de dados e classes entre diferentes aplicações. Para a plataforma Windows, bibliotecas C++ e Java permitem a comunicação entre o *framework* e a aplicação externa;
- **Jakarta Struts:** o *framework* foi desenhado e baseado no modelo Jakarta Struts. Este modelo tem se mostrado uma solução confiável, extensível e facilmente implementável.

## 2.2. CLASSES DE PERSISTÊNCIA

A camada de persistência é baseada no *framework* Pear::DB, que permite conexões para os Bancos de Dados dBase, FrontBase, InterBase, Informix, Mini SQL, SQL Server, MySQL, Oracle, ODBC, PostgreSQL, SQLite e Sybase.

O uso do *framework* é baseado no modelo de Objetos de Dados ou Containers. Esta metodologia apresenta os seguintes benefícios:

- Método de configuração simples e amigável;
- Armazenamento simples e rápido no Banco de Dados, através do uso apropriado das chaves primárias;
- Uso de *debugger*;

- Validação de dados: textos, datas e números podem ser pré-validados;
- Compilação de junções complexas ou obtenção de dados em consultas secundárias;
- Recurso de *autobuilding*, realizando o *update* automático na base;
- Integração com outros pacotes.

### 2.3. COMPONENTIZAÇÃO JAVASCRIPT

Através da componentização javascript – descrita abaixo –, o processamento das rotinas é transferido para a estação cliente, desonerando o servidor e o consumo de banda e gerando maior flexibilização das interfaces.

- **DBSheet:** Poderoso componente de fácil customização e desenvolvimento. DBSheet apresenta look-and-feel de planilha Excel, tornando-se extremamente simples e de uso.
- **TabControl:** Controlador de páginas que permite um melhor aproveitamento da área e apresenta look-and-feel similar aos componentes do mesmo gênero para aplicações Windows.
- **DataControl:** Componentes de formulário, como caixas de texto e combobox, diretamente ligados ao Banco de Dados.

Todos os *scripts* são comprimidos e maximizados de forma a reduzir o consumo de banda e o processamento, quando em execução.

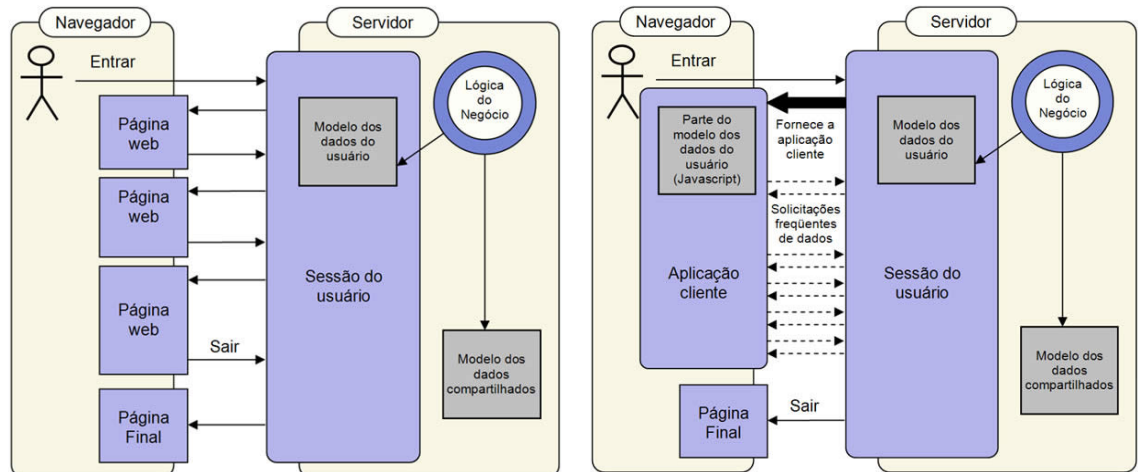
### 2.4. AJAX

Ajax (Asynchronous Javascript And XML) é uma técnica de desenvolvimento para criação de aplicações web interativas, através de solicitações assíncronas de informações. Com isso, não há necessidade de se recarregar toda a página cada vez que o usuário realizar uma mudança.

Segue um exemplo de aplicação web clássica para melhor compreensão das vantagens do uso do modelo Ajax:

### Aplicação Web Clássica

### Aplicação Web Ajax



Esquema de navegação em aplicações web Básica e Ajax.

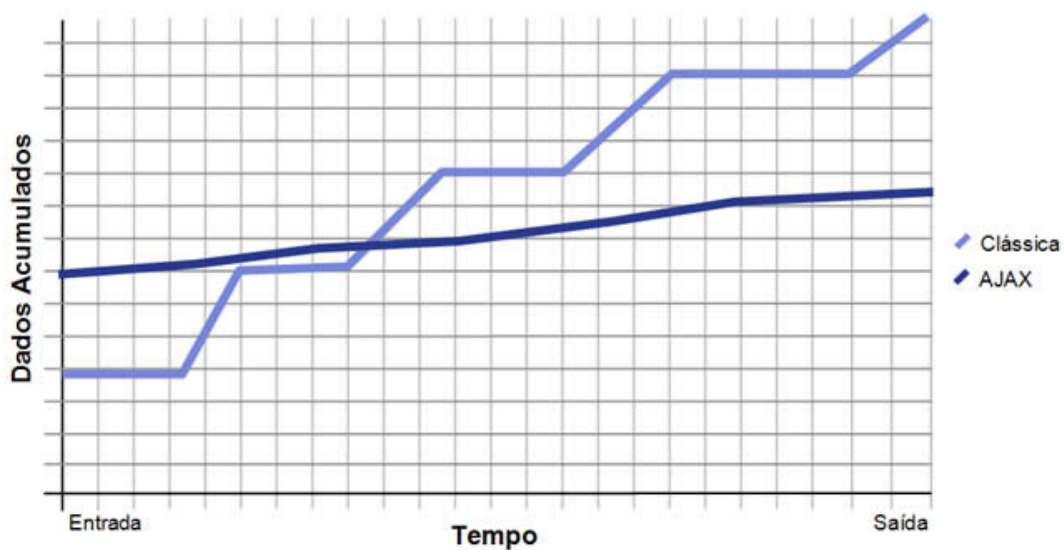
Em uma aplicação web clássica, qualquer interação do usuário é iniciada com uma extensa comunicação entre o cliente e o Servidor, fazendo com que a cada solicitação ao servidor todo o conteúdo antes carregado seja descartado e novamente recarregado.

Desta forma, apesar dos navegadores utilizados no mercado disporem de diversos dispositivos de *cache* e melhora de conexão, o navegador ainda irá sofrer com perdas durante a conexão com o servidor, reindexação dos componentes da página e realocação de memória para componentes dinâmicos.

Para aplicações baseadas em Ajax, grande parte da camada lógica da aplicação é transferida para o navegador. Neste formato, a interação com o usuário torna-se mais rápida já que apenas dados relevantes são trocados com o servidor.

Há um enorme ganho de desempenho, tempo e facilidade na distribuição das camadas, quando se usa esse tipo de interação, pois o usuário só precisa esperar pelo processamento do que foi solicitado, os dados comuns e estáticos como imagem, headers e cabeçalhos são mantidos.

O gráfico abaixo apresenta um comparativo entre a curva de tempo de navegação de uma aplicação web clássica e uma aplicação Ajax.



Tempo de resposta entre aplicações web clássicas e Ajax.

## 3.0. EFORTIS

---

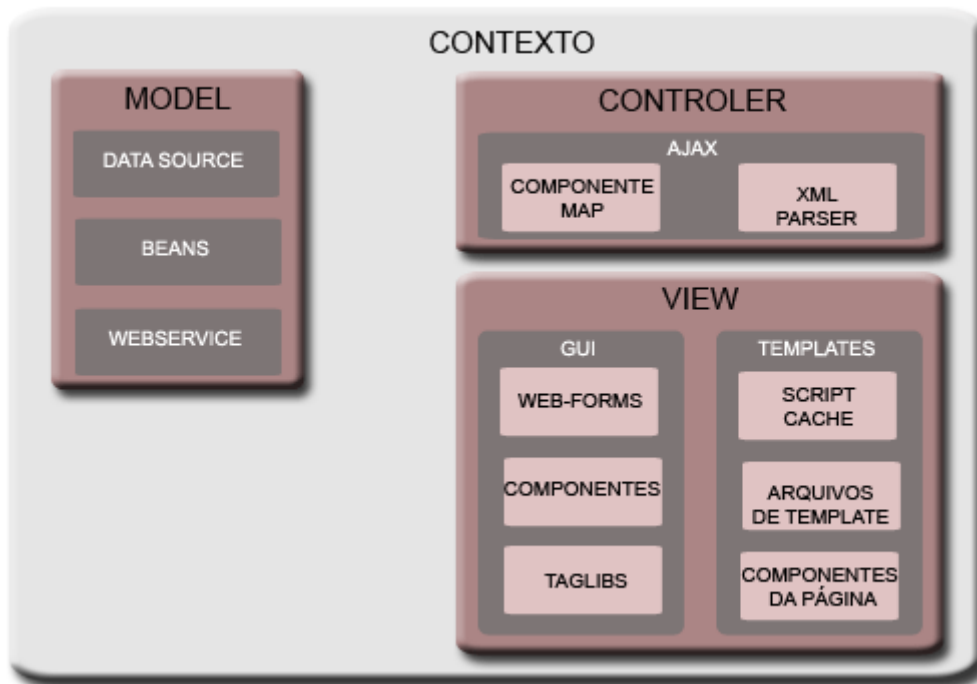
Visando alinhar as tecnologias requeridas e a confiabilidade e segurança da aplicação, foi desenvolvido integralmente o *framework* contendo recursos MVC, Ajax e Componentização.

### 3.1. MVC EFORTIS

Através de inúmeras avaliações dos *frameworks* disponíveis no mercado, concluiu-se que grande parte não está apta para a utilização da tecnologia Ajax, e os que são compatíveis, possuem baixa performance e alta complexidade de uso.

Baseado nestes estudos de mercado, e em conjunto com a vasta experiência e conhecimento da demanda de nossos clientes, o MVC eFortis foi desenvolvido com todos os recursos necessários para obter o melhor resultado em performance, customização e confiabilidade.

O *framework* eFortis segmenta as três camadas como especifica o design *pattern* e encapsula dentro da camada *Controller* uma camada interna para prover as facilidades do Ajax.



Estrutura do MVC eFortis.

### 3.2. CAMADA CONTROLLER

A camada *Controller*, além de prover a convencional navegação e mapeamento dos métodos de negócio bem com as views, dispõe também de uma interação Ajax por meio de requisições XML, integrada a um sistema de *templates* que facilita e torna muito ágil a transferência de informações entre o servidor e o cliente.

Além do mapeamento dos métodos de negócio e interação Ajax, o *Controller* funciona como módulo gerenciador de *cache*. O *framework* dispõe de uma significativa carga de javascript, a fim de que o *Controller* configure automaticamente o navegador do usuário e possa prover um mecanismo de *cache* de duração pré-definida, melhorando significativamente o tempo de carga dos *scripts* no usuário.

### 3.3. CAMADA VIEW

A camada *View*, dispõe de diversos componentes para interfaces modulares, bem como *taglibs* para prover uma interface única de desenvolvimento nos *templates* para toda a equipe de desenvolvimento e outras facilidades descritas abaixo.

O sistema conta com um prático mecanismo para criação de formulários, onde o desenvolvedor, por meio de um simples arquivo de configuração, pode definir a estrutura, bem como a validação, de um formulário. Os Web-Forms são adicionados ao *templates* de maneira transparente para o designer, através de uma simples tag, tornando o desenvolvimento bem flexível e descentralizado, onde o próprio designer pode efetuar seu trabalho sem a necessidade de um desenvolvedor para auxiliá-lo no momento da confecção das Views.

Os componentes são conjuntos de *scripts* pré-formatados e de uso integrado. Através de um arquivo de configuração o desenvolvedor pode parametrizar e utilizar determinado componente em diversas páginas.

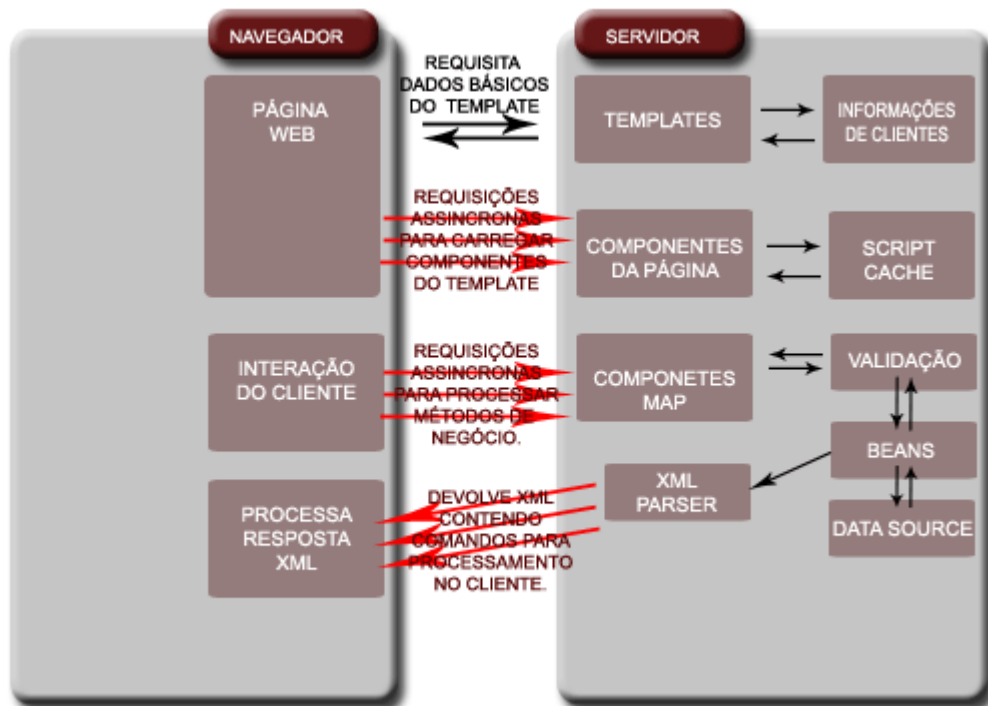
*Taglibs* é um conjunto de comandos que formam uma camada intermediária entre a View e a Model. Com *taglibs* a tarefa de ligar eventos a formulários torna-se bastante simples, bastando somente utilizar comandos que encapsulem as verdadeiras funcionalidades. O resultado é a flexibilização e a maior distribuição do modelo de produção.

### **3.4. CAMADA MODEL**

A camada Model passou por poucas alterações, pois o design *pattern* já define todo o esquema de distribuição e acesso à camada de negócio. No MVC eFortis, apenas foram adicionadas algumas bibliotecas facilitadoras de acesso à Webservices e à Bancos de Dados.

### **3.5. ESTRUTURA DE NAVEGAÇÃO**

A grande preocupação com o desenvolvimento deste *framework* foi a performance de acesso, a facilidade e a rapidez no desenvolvimento de novas *features* para o sistema e o completo desacoplamento das camadas, não apenas macro, mas realmente segmentando nos mínimos detalhes - como por exemplo na camada View, com os *templates* separados dos *scripts*, do CSS, além da individualidade das requisições Ajax e normais no *Controller*.

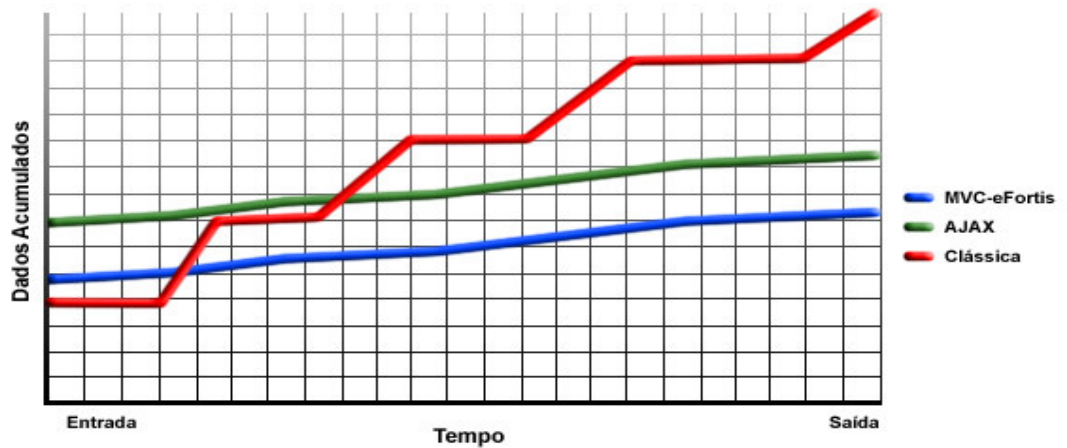


Estrutura de navegação em aplicações MVC eFortis.

Páginas desenvolvidas com Ajax costumam apresentar desvantagens na quantidade de *scripts* necessários para a carga completa da própria página. No MVC eFortis, a implementação de template modificou esse conceito, e fez com que o carregamento da lógica de negócio necessária no lado do cliente fosse maximizado, gerando conteúdo reduzido já que apenas o necessário é construído. Em outras palavras, o sistema só carrega o que o usuário irá utilizar, o que permitiu a criação de páginas mais elaboradas e complexas, e ainda assim, eliminou o problema de tempo de carregamento.

Os recursos de template permitem ainda total flexibilidade na customização do conteúdo, de acordo com as necessidades de cada cliente, permitindo o compartilhamento de códigos com personalização.

Para explorar ao máximo o cacheamento de *scripts* e a codificação HTML, todas as páginas são carregadas em dois tempos. Inicialmente, um script de controle é carregado e este, na última etapa, carrega o restante do conteúdo selecionando *scripts* e *templates*.



Tempo de resposta em aplicações MVC eFortis.

### 3.6. DESENVOLVIMENTO DISTRIBUIDO

O desmembramento em camadas e o uso de *templates* permitem o desenvolvimento e a customização com equipes maiores e com prazos menores, totalizando um esforço inferior a outros modelos empregados.

### 3.7. PERSONALIZAÇÃO DE CLIENTES

O MVC eFortis representou uma grande redução de esforço no desenvolvimento e customização da aplicação. Essa é uma premissa do produto, que oferece customização de interfaces e adequação nas regras de negócio às necessidades do cliente.