MVC & FRAMEWORK

Marcos Vilela Fonseca, André Luiz Alves

Coordenação de Pós-graduação Latu Sensu – PROPE / PUC – GOIAS

Avenida Universitária, 1440 Setor Universitário CEP: 74605-010

Goiânia – GO – Brasil

marcos@canion.com.br, andre.luiz@ucg.br

André Luiz Alves – Especialista

Abstract. *After 31 years of its inception the MVC remains an unknown in the IT community, especially of software developers in general. An innovative idea so simple, obvious, practical and fundamental spent so much time almost unnoticed. Your application alone could revolutionize the way you're programming today, even after so many advances in programming languages and techniques because it is a concept, an organizational model that regardless of programming language that is intended to bring excellent results. This article aims at re-presenting the MVC and highlight the advantages of its use in manufacturing programs, especially to improve the organization of source code thus facilitating their understanding and maintainability.*

Resumo. Após 31 anos de sua concepção o MVC continua um desconhecido da comunidade de TI, principalmente dos desenvolvedores de programas em geral. Uma idéia inovadora tão simples, obvia, prática e fundamental passou tanto tempo praticamente despercebida. Sua aplicação poderia sozinha revolucionar o modo com se faz programas ainda hoje, mesmo após tantos avanços em linguagens e técnicas de programação pois se trata de um conceito, um modelo organizacional que independente da linguagem de programação que se pretende adotar produz excelentes resultados. Este artigo tem como objetivo re-apresentar o MVC e destacar as vantagens de sua utilização na fabricação de Programas, especialmente para melhorar a organização do código fonte facilitando seu entendimento e manutenibilidade.

Palavras-Chave: Modelo-Visão-Controle. Normas e Padrões para Programação.

Nota do autor: No inicio do meu curso de Pós-graduação na PUC-GOIAS em 2009, quando fui formalmente apresentado ao MVC tive a nítida sensação que seria esse meu projeto final, mas, havia ainda muito curso pela frente e fui esperando para ver outras novidades interessantes. No decorrer do curso o MVC sempre retornava como um assunto que merecia ser explorado, me incomodava a baixa utilização do MVC no dia a dia do programador comum como metodologia organizacional de código fonte, acima de tudo frustrava-me a sua baixa recomendação e o desmerecido esquecimento pela comunidade de TI fora do ambiente acadêmico.

MVC – Modelo, Visão e Controle

1) Introdução

Existe muita dificuldade em programar, gerenciar, documentar e manter o código fonte de uma aplicação quando este se apresenta como uma mistura de código de acesso a dados, código de lógica ("negócio") e código de apresentação (monolítico). Tais aplicações são difíceis de manter, porque a interdependência entre todos os componentes pode causar fortes efeitos em cascata, sempre que uma alteração é feita em qualquer lugar. A princípio a facilidade de se ter tudo no mesmo lugar torna difícil ou impossível a reutilização porque existe uma alta dependência de tantas outras classes. Adicionar novas telas de consulta, manutenção, exclusão e relatório muitas vezes requer reimplementações ou cortar e colar o código da lógica de negócios, o que requer uma manutenção em vários lugares, o código de acesso de dados sofre do mesmo problema, sendo cortado e colado entre os métodos de lógica de negócios. A proposta do MVC é separar este código em três partes o Modelo (dados), a Visão (apresentação) e o Controle (fluxo e regra de negócio).

Não se trata de programação em três camadas, devido à forte divulgação e utilização de programação em três camadas nos anos 90 e o reaparecimento do MVC na mesma época as duas técnicas podem ser confundidas. Mas o MVC não e programação em três camadas, mas uma camada de programação dividida em três partes em mesmo nível e organizada segundo a sua função. A principio pode parecer trabalhoso dividir um código fonte em três partes se é perfeitamente possível fazer a mesma coisa em código único, mas, o simples fato de separar por funções ou responsabilidade permite uma manutenabilidade mais ágil menos sujeita a erros e mais clara, tudo que um programador precisa quando pretende dar manutenção em um código principalmente se foi originalmente implementado por outro programador.

Sendo assim essa idéia deveria estar sendo implementada e ser conhecida a muito tempo pela maioria dos programadores mas, mas pelo contrário, muitos estão descobrindo por conta própria a duras penas e necessidade de se separar o código para se organizar. Com a introdução das chamadas novas linguagens e seus ambientes RAD (Rapid Aplication Development) Desenvolvimento rápido de aplicação, parte dessa organização passou a ser efetuada pelo ambiente de desenvolvimento sem a percepção e participação do programador. Mesmo nesses ambientes a aplicação do MVC propositadamente implementada pelo programador poderá resultar em altos ganhos de produtividade, principalmente quando se leva em conta o produto pronto (projeto, desenvolvimento, testes, correções e entrega).

O MVC facilita a produção via aplicação de Design Patterns (Padrões de projeto) e facilita a criação e aplicação de FrameWorks (plataforma de trabalho padronizado) porque com sua melhor organização e separação por funções (responsabilidades) fica mais fácil projetar sistemas que produzem sistemas.

Para a *Gamma et al.* o *MVC* é:

MVC é composto por três tipos de objetos. O modelo é o objeto de aplicação, a visão é a apresentação na tela e o controlador define a maneira como a interface do usuário reage às entradas do mesmo. Antes do MVC, os projetos de interface para o usuário tendiam em agrupar esses objetos. MVC para aumentar a flexibilidade e a reutilização. (GAMMA et al. 2000, p. 20).

2) Origem

A documentação original da criação do MVC é disponibilizada pelo seu autor Trygve Reenskaug em seu site: http://folk.uio.no/trygver/ escrita originalmente entre os anos de 1978/1979 esse método organizacional se revela incrivelmente atual e moderno. Visitando os documentos originais da criação do MVC pode-se comprovar que o modelo permanece válido até hoje 31 anos depois e sem a necessidade de alterações para se aplicar no mundo atual da programação.

Segundo Trygve, a primeira implementação do MVC e a documentação escrita foi realizada quando ele fazia uma visita cientifica ao Xerox Palo Alto Research Laboratory (PARC) nos anos de 1978/1979. Basicamente o autor criou uma solução obvia para um problema geral: proporcionar aos usuários o controle de grandes e complexas massas de dados. É fascinante obter a informação da origem do MVC de sua fonte original e notar que mesmo após tanto tempo ela continua válida e util.

Os primeiras implementações do MVC foram feitas na linguagem Smalltalk e Trygve não participou delas diretamente, demonstrando que o modelo já nasceu autônomo e completo. Trygve publicou varias atualizações do seu projeto MVC, especialmente chamam a atenção uma revisão de 20 de agosto de 2003 em que ele resume e condensa o que ele chama de *MVC pattern language* (Padrão de linguagem MVC). Este documento mais claro e completo que o original facilita a absorção das idéias por traz da arquitetura MVC.

http://folk.uio.no/trygver/2003/javazone-jaoo/MVC_pattern.pdf

3) MVC

3.1) Modelo

Segundo Trygve (1979) O modelo e uma representação ativa de uma abstração em forma de dados em um sistema de computação.

Podemos entender o modelo como a camada responsável pela persistência dos dados na aplicação, com todas as funções e responsabilidades que essa área exige. Assim pode-se inferir que a "camada de dados" conterá todas as funcionalidades como incluir, obter dados, gravar, e excluir, alem de outras responsabilidades relacionadas com o modelo de dados como: controle de chaves, ordenação, etc. A figura 1 abaixo mostra a forma de integração do modelo com as partes de Visão e o Controle. Para Trygve (1979) essa camada devia representar uma coisa "qualquer coisa" de forma abstrata uma representação do objeto real seja um carro, uma casa, uma pessoa.

Em uma análise básica devemos imaginar a figura do modelo em mundo real, e essa abstração que vai possibilitar um perfeito entendimento do modelo que se deseja. Quando um modelo real (por exemplo, um carro) consegue ser bem descrito em forma de modelo de dados a abstração foi eficiente. Neste ponto se percebe que a representação em modelo de dados não conseguiria ser tão eficaz se tivesse que se misturar a outras funcionalidades que não fosse somente à descrição do objeto. Nesse mesmo sentido vale a decomposição das partes do objeto, no mesmo exemplo acima poderíamos ter que "armazenar" o objeto motor que é parte integrante do carro, e mais uma vez percebe-se que essa atividade fica mais fácil e clara quando se preocupa somente em descrever fielmente o objeto e seus componentes.

O Modelo pode (deve?) ter métodos encapsulados, situações de estados, exporem funcionalidades, pesquisas e relacionamentos com outros objetos relacionados a dados. Assim, esta parte do código deverá conter toda a

estrutura de dados e seus métodos de inclusão, alteração (consulta) e exclusão, com os adicionais de cardinalidade, integridade, controle de transações. Evidentemente algumas dessas funcionalidades poderão ser transferidas ao SGBD se o mesmo possuir capacidade para suportá-las, no entanto deve-se considerar que a portabilidade entre SGBD é inversamente proporcional a utilização de suas potencialidades exclusivas, existindo quem defenda a utilização apenas do SQL padrão (ANSI) como forma de obter compatibilidade ou ainda da utilização do SGBD apenas como repositório de dados buscando-se neste caso a portabilidade máxima.

O Modelo não pode extrapolar sua função de armazenar fielmente o objeto, como exemplo, manipular informações para facilitar a apresentação (mascaras) ou ainda realizar checagens, remontar dados em outras formas, transformações para padronizações de tamanhos etc. Deve-se sempre ter em mente que o modelo poderá estar aberto a novos clientes de visualização a qualquer hora e qualquer manipulação poderá privilegiar um ou outro visualizador. Porém isso não pode significar falta de eficiência em guardar e recuperar rapidamente e de forma inteligente as informações.

O modelo não precisa saber onde será apresentado seus dados, basta disponibilizá-los corretamente e garantir a persistência.

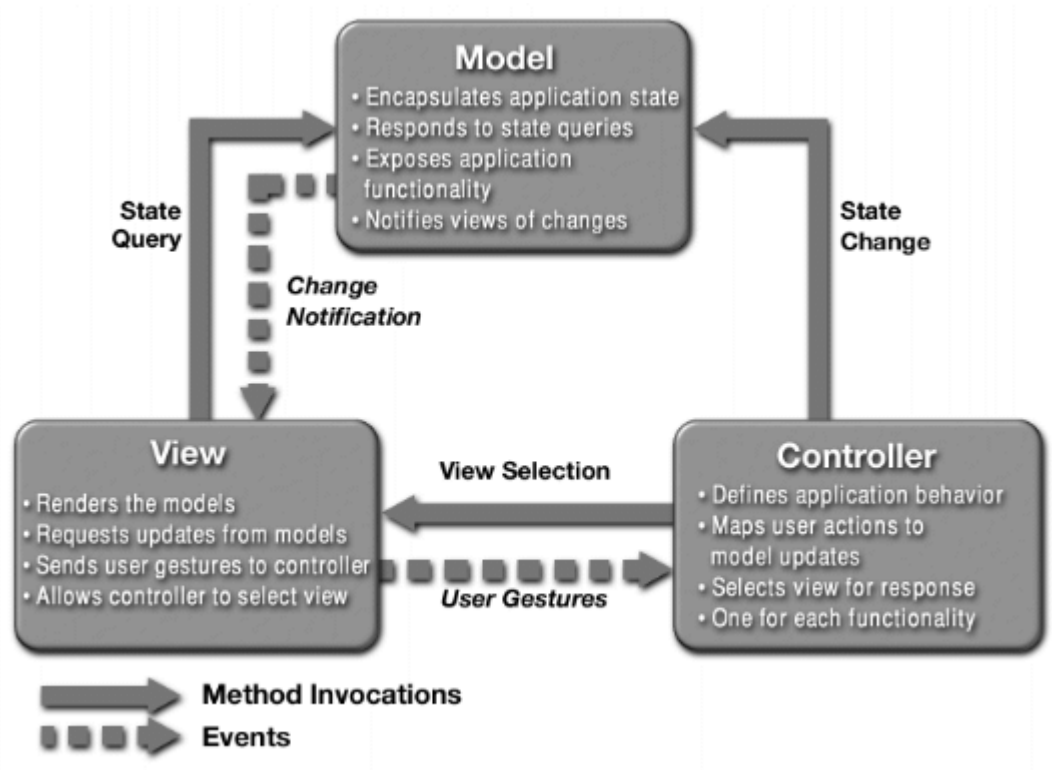


Figura 1. © 2000-2002 Sun Microsystems

3.2) Visão

Representa a parte visual de apresentação do modelo para o usuário, é a interface de interação com o mundo exterior da aplicação, seja uma tela, um consulta ou relatório. É construída com base no modelo, mas não controla o modelo e nem e controlada por ele. Pode ser notificada e se atualizar baseada em mudanças de estado do modelo. Deve refletir o mais fielmente o modelo em mundo real, não se trata mais de um modelo de dados, mas da cópia do objeto real de forma a que o usuário possa visualizar todas as características do objeto real e seus componentes. É a Visão que deve transformar os dados em informações, manipular, mascarar, ajustar, alinhar, concatenar, separar, substituir códigos por descrições que melhor representem o objeto real.

Uma característica especial dessa aplicação em MVC é a possibilidade de se "trocar" de interface facilmente desde que se acionem corretamente os métodos conhecidos, isso tornaria possível a adição de uma camada visual nova "Skin" em toda aplicação de forma rápida e sem muitos efeitos colaterais se a parte modelo estiver bem estruturada e for eficiente, pode-se trocar a pele/casca mantendo-se a persistência encapsulada (isolada).

Pode ser difícil, mas não impossível se pensar em trabalhar inclusive com linguagens diferentes para Visão, podendo-se ousar no designer cores e formas apostando em uma linguagem mais arrojada para esse fim, basta para isso que exista alta acoplagem com o Modelo/Controle, essa característica pode também ser explorada para se desenvolver uma visão-web baseada em linguagens nativas da web (java, .net, flash, etc) deixando o Modelo e Controle em linguagens mais robustas e confiáveis em gerenciamento de dados e regras de negócios (código legado).

A Visão deve ser entendida como uma ambiente de passagem, de armazenamento temporário (memória), portanto deve ser ágil, simples, clara e por tudo isso desprovida de inteligência (excetuando-se métodos básicos de validação de dados).

A Visão não precisa saber onde e como seus dados (temporários) serão armazenados de forma persistente, isso é função do modelo.

3.3) Controle

Com o modelo (persistência) e a visão (apresentação) desconectados (assíncrono), é de se esperar que tudo se torne uma bagunça onde nada se liga a nada e ninguém se entende. O Controle vem para gerenciar esta comunicação e controlar o fluxo de dados, regras de negócios, ações de usuários (como cliques, menus teclas de atalhos e outras formas de interação).

O controle interpreta as ações, solicitações e interações do usuário e comunica-se com o modelo que por sua vez executa a solicitação e atualiza a visão com o resultado da solicitação. Assim o comportamento da aplicação passa sempre pelo controle que tem ainda as regras de negócios da aplicação. Atuando assim como um elo entre a visão e o modelo.

Em um sistema complexo que não seguisse o MVC o código de regra de negocio ficaria em meio a um emaranhado juntamente com a construção de tela e persistência de dados e numa necessidade de manutenção no código fariam qualquer um perder o controle se não se mantivesse altamente concentrado, ou não tivesse acesso a excelente documentação.

Framework:

O uso do MVC facilita o desenvolvimento e a aplicação de um avanço no processo de codificação, programas que criam programas (framework). A automação de parte do processo de codificação e vista por muitas pessoas de TI com desdém, afirma-se que seria essa linha de montagem produto da preguiça, além de prejudicar a personalização e o processo criativo.

Porém é possível mapear muitas operações semelhantes, onde um framework poderia sim ser aplicado perfeitamente. É possível estabelecer um padrão de aplicativo que faz busca, apresentação e persistência de dados, e esse padrão ocorre com muita freqüência em um programa complexo, nas atividades especializadas como as regras de negocio o melhor é codificar de maneira tradicional para se atingir a melhor solução.

Apesar da grande aceleração produzida por um ambiente RAD, a aplicação de um FrameWork provoca uma produtividade insuperável e pode ser desenvolvido praticamente em qualquer linguagem mesmo as mais antigas, para as quais não existem RAD.

Na empresa Cânion Software foi desenvolvido um FrameWork para geração automática de manutenção de tabelas armazenadas em SGBD. Desenvolvido em Linguagem Delphi gera código fonte Delphi em padrão MVC. O programa gerador conecta-se ao banco de dados e o usuário escolhe a tabela alvo do FrameWork, em seguida cria três pastas com denominadas M,V e C, dentro delas serão gerados os códigos de pesquisa, inclusão, manutenção e exclusão, já com todo o tratamento de menu e controle de usuários.

Esse processo tem a vantagem de aliar velocidade, confiabilidade e padronização, muitos erros básicos são evitados ao se optar pelo processo automático. Evidentemente após a geração alguma personalização pode ser adicionada para se fazer um ajuste fino de alguma necessidade especifica.

A adição de um novo campo na tabela causa uma intervenção manual ainda, mas a empresa trabalha numa melhoria do processo que poderá alterar automaticamente o código conforme as necessidades de mudanças no BD.

O processo automático fere alguns princípios da Engenharia de Software já que "desvia" a seqüência do processo para o Banco de Dados antes da codificação, mas sem duvida os ganhos de produtividade podem justificar um pouco essa "flexibilidade".

Conclusão:

Com o entendimento mais profundo do padrão de arquitetura MVC, nota-se que a sua aplicação produz ganhos de qualidade em qualquer linguagem, mesmo aquelas procedurais e não gráficas podem se beneficiar dessa metodologia padrão de processo produtivo de software.

O MVC assim como outros padrões de arquitetura ajuda a produzir programas mais complexos e com menos erros. Após 3 décadas continua revolucionando o pensamento produtivo e a realidade da comunidade de TI agora fora do ambiente acadêmico. O tempo gasto entre a sua criação e a sua efetivação no mercado empresarial reacende uma velha questão. É a sociedade que não busca as novidades e inovações de pesquisa ou a universidade não se abre suficientemente divulgando mais eficientemente os seus avanços.

Certamente a produtividade pode andar junto com a qualidade desde que boas técnicas sejam adotadas no processo. Entre tantas técnicas vale pesquisar e aplicar o MVC talvez a mais simples, obvia e eficaz de todas.

Referencias

http://folk.uio.no/trygver/

http://folk.uio.no/trygver/2003/javazone-jaoo/MVC_pattern.pdf

METSKER, Steven John. ***Padrões de projeto em Java***. Porto Alegre: Bookman, 2004.

GAMMA, Erich et al. ***Padrões de Projeto:*** *Soluções reutilizáveis de software Orientado a Objetos.* Porto Alegre: Bookman, 2000.

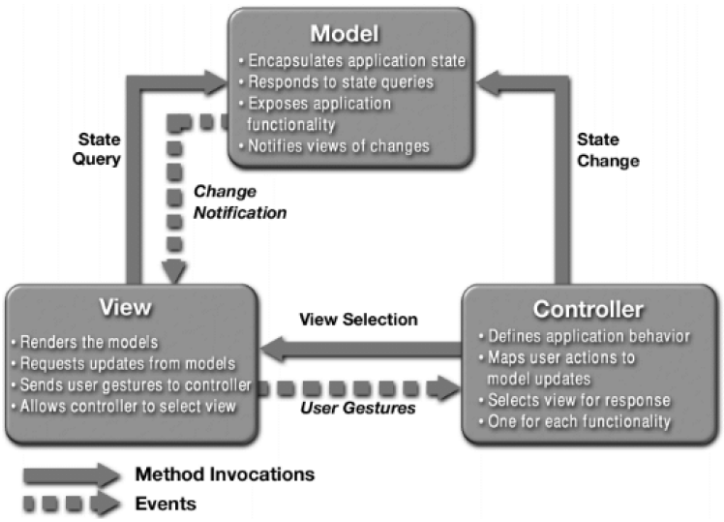
SOMMERVILLE, Ian. ***Engenharia de Software.*** São Paulo: Addison Wesley, 2003.

FREEMAN & FREEMAN, Eric & Elizabeth***. Padrões de Projetos:*** Seu cérebro em padrões de projetos. Rio de Janeiro: ALTABOOKS, 2007.

PRESSMAN, Roger S. ***Engenharia de Software.*** São Paulo: Pearson Education do Brasil, 1995.

MVC e Frameworks – Exercícios

1. O que significa MVC e o que é? Dê exemplos.
2. Defina cada um dos elementos que formam o MVC, exemplificando-os.
3. Por que em um sistema seguindo os conceitos MVC, a visão não modifica diretamente os modelos?
4. Qual conceito de orientação a objetos os modelos devem seguir? Por quê? Dê exemplos.
5. Explique em poucas palavras o que acontece na figura a seguir.



6. Seguindo os conceitos do MVC, o que deve ser feito para que se possa mudar a visão sem modificar os demais elementos?
7. O que é um framework? Dê exemplos de alguns que você conhece.
8. Qual a relação entre frameworks e MVC?