PADRÃO DE ARQUITETURA MVC: MODEL-VIEW-CONTROLLER

(ARCHITECTURAL PATTERN MVC: MODEL-VIEW-CONTROLLER)

JOSUÉ LUCIANO; WALLISON JOEL BARBERÁ ALVES

Centro Universitário UNIFAFIBE – Bebedouro – SP

josueluciano@uol.com.br

Abstract. The average IT investment of a company whose main business is not IT, is between 70% and 80% for software maintenance, and only the remainder to create new software. This means that most of this investment is to maintain the legacy, many are scared because it often means the code hard to maintain (sometimes impossible). It is this challenge that a standard architecture and mounted want to win. The architectural pattern MVC (Model-View-Controller) is designed to be the bridge between the user's mental model and the computational model.

Keywords. Pattern, Architectural, Development, Maintenance, Software, Internet.

Resumo. A média de investimento em TI de uma empresa, cujo negócio principal não é TI, fica entre 70% e 80% para manutenção do software, e somente o restante para criação de software novos. Isso significa que a maior parte deste investimento vai para a manutenção do legado, muitos se assustam, porque muitas vezes significa código difícil de manter (às vezes impossível). É este desafio que um padrão de arquitetura bem montada quer vencer. O padrão de arquitetura MVC (Model-View-Controller) foi criado para ser a ponte entre o modelo mental do usuário e o modelo computacional.

Palavras-chave. Padrão, Arquitetura, Desenvolvimento, Manutenção, Software, Internet.

INTRODUÇÃO

Este trabalho procurou investigar o padrão de arquitetura conhecido como MVC (Model-View-Controller). Surgiu no começo da década de 80, mas se tornou muito popular na criação de aplicações WEB.

Sua dinâmica é simples, todas as requisições da aplicação são direcionadas para a camada Controller, que acessa a camada Model para processar a tal requisição, e por fim exibe o resultado da camada View, o padrão MVC separa as camadas de apresentação, de lógica de negócio e de gerenciamento do fluxo da aplicação, aumentando as capacidades de reutilização e de manutenção do projeto. É muito utilizado para casos em que podem existir múltiplas camadas de apresentação para clientes diversos. Por exemplo, para os usuários diretos de uma aplicação, os dados podem ser apresentados em uma página JSP, mas para os fornecedores pode existir uma interface WML (Wireless Markup Language), enquanto para os usuários remotos pode exisitr uma interface HTML simples. Esse é um dos padrões de desenvolvimento mais populares.

É sugerido a utilização de uma arquitetura fundada em padrões reconhecidos justamente porque sabemos que uma aplicação não vai manter os mesmos requisitos durante toda sua existência. Na verdade, aplicativos do mundo real não mantém os mesmos requisitos nem mesmo durante o desenvolvimento do projeto inicial, quem diria ao longo de toda sua existência. Em inglês o termo camada pode ser traduzido como tier, que tem significado que se aproxima mais de camadas físicas, ou como layer, que significa camadas lógicas.

A primeira vez que se falou em MVC foi em 1979, e quem falou foi Trygve Reenskaug, então funcionário da Xerox que trabalhava no projeto do SmallTalk. Mas foi na WEB que o MVC se popularizou, principalmente na comunidade do software livre.

O MVC foi muito adotado na comunidade Java, mas também vem ganhando grande espaço na plataforma .NET. No final de 2007 a Microsoft anunciou um Framework que dá suporte ao desenvolvimento de aplicações ASP.NET no padrão MVC, e em 9 de abril de 2009 foi lançada a primeira versão do ASP.NET MVC Framework. Atualmente o mercado entende o que é o ASP.Net MVC e para que serve, e está começando a utilizá-lo. Já existe demanda por profissionais que o conheçam no mercado, que está carente de profissionais com este conhecimento.

Padrões de projetos são soluções para problemas que alguém um dia teve e resolveu aplicando um modelo que foi documentado e que pode adaptar integralmente ou de acordo com a necessidade de sua aplicação. O grande desafio das equipes de desenvolvimento de aplicações é cada vez mais produzir aplicativos seguros, eficientes, de fácil manutenção, reutilizáveis e em prazos cada vez menores.

O sucesso para o desenvolvimento de aplicações orientada a objetos está intimamente ligada á arquitetura que será utilizada na construção da aplicação. A tendência indica que esta arquitetura estará baseada na organização da aplicação em camadas e na observação dos padrões utilizados no mercado.

A organização em camadas é a chave para a independência entre os componentes e esta independência é que vai atingir os objetivos de eficiência, escalabilidade, reutilização e facilidade de manutenção. Produzir aplicativos multicamadas num primeiro instante pode parecer mais complexo.

OBJETIVOS

Apresentar o padrão de arquitetura MVC (Model-View-Controller) como uma opção e não questionar se é ou não a melhor opção disponível, pois existem outros padrões de arquiteturas disponíveis no mercado – MVP (Model-View-Presenter) e MVVM (Model-View-ViewModel) - que não fazem parte do escopo apresentado neste artigo.

O grande objetivo do padrão MVC é isolar ao máximo a camada de apresentação.

Durante toda a vida de um aplicativo, o mesmo deverá passar o teste mais difícil de todos: o tempo. A média de investimento em TI de uma empresa, cujo negócio principal não é TI, fica entre 70% e 80% para manutenção de software, e somente o restante para criação de software novos. Isto significa que a maior parte deste investimento vai na manutenção do legado. E só de ouvir a palavra “legado”, muitos já se assustam, porque muitas vezes significa código difícil de manter (às vezes impossível). É esse desafio que uma arquitetura bem montada quer vencer.

As consultorias de arquitetura, sempre recomendam que os padrões sejam utilizados, mesmo para aplicações pequenas. É comum empresas ou usuários afirmarem que um aplicativo é pequeno, não vai ter muitas atribuições, não será alterado no futuro, ou ainda que terá vida curta. É também muito comum que estes aplicativos sobrevivam por muito tempo, não sejam

substituídos, e, pior de tudo, cresçam desordenadamente.

Aplicações monolíticas ou de uma camada: Nos tempos antigos do reinado do grande porte e do computador pessoal, um aplicativo era desenvolvido para ser usado em uma única máquina. Estes aplicativos continham todas as funcionalidades em um único módulo gerado por uma grande quantidade de linhas de códigos e de manutenção nada fácil. A apresentação, a regra de negócio e o acesso à banco de dados estava presente em um mesmo lugar.

Aplicações em duas camadas: A necessidade de compartilhar a lógica de acesso a dados entre vários usuários simultâneos fez surgir as aplicações em duas camadas. Nesta estrutura a base de dados foi colocada em uma máquina específica, separada das máquinas que executavam as aplicações. Nesta abordagem temos aplicativos instalados em estações clientes contendo toda a lógica da aplicação. Um grande problema neste modelo é o gerenciamento de versões, pois para cada alteração os aplicativos precisam ser atualizados em todas as máquinas clientes.

Aplicações em três camadas: Com o advento da internet houve um movimento para separar a lógica de negócio da interface com o usuário. A idéia é que os usuários da WEB possam acessar as mesmas aplicações sem ter que instalar estas aplicações em suas máquinas locais. Neste modelo o aplicativo é movido para o Servidor e um navegador Web (browser) é usado como um cliente. O aplicativo é executado em servidores Web com os quais o navegador Web se comunica e gera o código HTML para ser exibido no cliente. Neste modelo a lógica de apresentação está separada em sua própria camada lógica e física. A separação em camadas lógicas torna os sistemas mais flexíveis permitindo que as partes possam ser alteradas de forma independente. As funcionalidades da camada de negócio podem ser divididas em classes e essas classes podem ser agrupadas em pacotes ou componentes reduzindo as dependências entre as classes e pacotes; podem ser reutilizadas por diferentes partes do aplicativo e até por aplicativos diferentes. O modelo de 3 camadas tornou-se a arquitetura padrão para sistemas corporativos com base na Web.

O MVC foi descrito originalmente em 1979 por Trygve Reenskaug quando trabalhava no projeto Smalltalk-80 ou simplesmente Smalltalk, uma linguagem de programação orientada a objeto, fortemente tipada. Em Smalltalk tudo é objeto: os números, as classes, os métodos, os blocos de código etc. ao contrário de outras linguagens orientadas a objeto; strings, números e caracteres são

implementados como classes em *Smalltalk*, por isso esta linguagem é considerada puramente orientada a objetos. Tecnicamente, todo elemento de *Smalltalk* é um objeto de primeira ordem. O Smalltalk-80 foi lançado para computadores de diversas companhias (Hewlett-Packard, DEC, Apple, IBM, Tektronix) como um teste de portabilidade do ambiente. Ele foi implementado como um compilador de bytecode. O código era de fato compilado, porém não para a linguagem de máquina nativa do computador que executava aplicação e, sim, compilado para uma linguagem de máquina de um computador que não existia: a máquina virtual. A vantagem desse esquema que foi criado com o Smalltalk-80 é que ele tornou-se extremamente portável.

A interface gráfica do usuário (GUI) nasceu na Xerox, e o MVC é o conceito central por trás da GUI em Smalltalk-80.

MÉTODOS

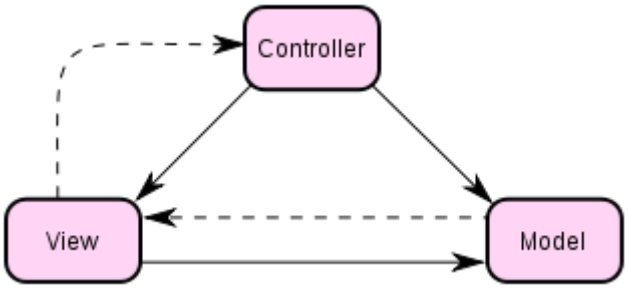
Pesquisando a proposta do autor do MVC na internet e em revistas técnicas especializadas, a arquitetura foi criada para ser a ponte entre o modelo mental do usuário e o modelo computacional.

RESULTADOS

A proposta é útil para possibilitar ao usuário visualizar o mesmo modelo sob diferentes contextos e/ou ponto de vista. No MVC as entradas do usuário, o modelo do mundo real e a interface são explicitamente separados e tratados por três tipos de objetos, onde cada um é especializado em suas próprias tarefas.

CONCLUSÃO

Estes três elementos têm baixo acoplamento e alta coesão, sendo eles: Modelo (Model) – mantém o estado da aplicação, é mais que uma classe para armazenar dados, nele deve estar todas as regras de negócios e também comunicar-se com o banco de dados se necessário; Visão (View) – especifica exatamente como o modelo deve ser apresentado. É a interface do usuário. A visão é dinâmica se adequando a qualquer modificação do modelo; Controlador (Controller) – traduz as interações do usuário com a visão, mapeando-as para tarefas que o modelo irá realizar.



Um diagrama simples exemplificando a relação entre *Model*, *View* e *Controller*. As linhas sólidas indicam associação direta e as tracejadas indicam associação indireta.

REFERÊNCIAS

[1] PEREIRA, R. Guia de java na web. 1.ed. Rio de Janeiro: Ciência Moderna Ltda, 2006. 304p.

[2] REENSKAUG, T.M.H. MVC. 2010. Demonstra o padrão de arquitetura MVC. Disponível em:<http://pt.wikipedia.org/wiki/MVC>. Acesso em 26 jan.2011.

[3] MACORATTI, J.C. Padrões de projetos. 2010. O modelo MVC – Model View Controller. Disponível em<:http://www.macoratti.net/vbn_mvc.htm>. Acesso em 26 jan.2011.

[4] KAY, A.; INGALLS D.; GOLDEBERG A. Smalltalk. 2010. Linguagem orientada a objeto fortemente tipada. Disponível em:<http://pt.wikipedia.org/wiki/Smalltalk>. Acesso em 26 jan.2011.

[5] .NET MAGAZINE. Rio de Janeiro: Grajaú: Devmedia, ano 05, edição nº 56.

[6] .NET MAGAZINE. Rio de Janeiro: Grajaú: Devmedia, ano 05, edição n° 58.

[7] .NET MAGAZINE. Rio de Janeiro: Grajaú: Devmedia, ano 06, edição n° 65.

[8] .NET MAGAZINE. Rio de Janeiro: Grajaú: Devmedia, ano 06, edição n° 68.

[9] .NET MAGAZINE. Rio de Janeiro: Grajaú: Devmedia, ano 06, edição n° 69.