Taxi Aéreo
Documento de Arquitetura De Software

Versão 01.00 22 de junho de 2015

Histórico de Revisões

Data	Versão	Descrição	Autor
22/06/2015	0.1	Versão Final do Documento de Arquitetura – Revisado.	Eduardo, Livânio e Uriel
22/06/2006	1.0	Versão Final do Documento de Arquitetura – Atualização.	Eduardo, Livânio e Uriel

1. Introdução

A arquitetura é a base de todo o desenvolvimento e das soluções, envolvidas na implementação, de sistema.

Neste documento iremos detalhar as principais partes da arquitetura proposta para o desenvolvimento do sistema Taxi Aéreo. A arquitetura é utiliza padrões de projeto e padrões Orientados a Objetos com destaque no mercado. Iremos destacar em cada parte da arquitetura o motivo da sua criação e o qual a sua influência para a criação de sistemas de alta coesão, mas com baixo acoplamento.

2. Objetivos

O Documento de Arquitetura do Software provê uma visão geral da arquitetura, usando um conjunto de visões arquiteturais para tratar aspectos diferentes do software.

Este documento serve como um meio de comunicação entre o Arquiteto de Software e outros membros da equipe de projeto sobre as decisões significativas que forem tomadas durante o projeto.

3. Considerações Gerais

As definições arquiteturais de um projeto de desenvolvimento de software em geral seguem as definições necessárias aos vários projetos de uma organização ou instituição e que atenda a todas as necessidades do projeto, desde a segurança, regras de negócio, até a persistência dos dados.

As definições do projeto já documentadas até o presente momento devem guiar as primeiras versões do Documento de Arquitetura do Software, que é desenvolvido durante a fase de Elaboração, uma vez que o propósito dessa fase é estabelecer os fundamentos arquiteturais para o projeto do software.

4. Responsabilidades

O Arquiteto de Software é o responsável por elaborar este documento e por manter a integridade do mesmo durante o processo de desenvolvimento do software. Ele deve:

- Aprovar todas as mudanças arquiteturais significativas e documentá-las.
- Fazer parte do comitê que decide sobre os problemas que tenham algum impacto arquitetural.

5. Arquitetura

O que é? E como é composta?

A arquitetura foi desenvolvida para ser totalmente de alta coesão e baixo acoplamento, onde a estrutura adotada foi a de utilizar o padrão MVC (Model View Controller) e também o padrão DAO (Data Access Object) para fazer a separação do acesso aos objetos e realizando as operações de CRUD fora das camadaz do MVC. Tendo como objetivo ocultar as complexidades existentes nas tecnologias adotadas no desenvolvimento, gerando muito mais conforto e facilidades para os programadores. Com isso, o programador que fosse efetuar uma manutenção no sistema não precisa ter o conhecimento das tecnologias utilizadas, mas sim, ter o conhecimento apenas da última forma que o sistema foi desenvolvido. Conseguindo, assim, criar uma equipe sólida em seus conhecimentos e ao mesmo tempo produtiva.

Além das classes que são peça-chave dentro da arquitetura, foi adicionada também à "última milha", classes de apoio ao desenvolvimento, ou melhor, classes que fazem o papel de ferramentas, onde possuem métodos que serão utilizados da mesma forma em todos os sistemas. Por exemplo, classes responsáveis por fazer a leitura das propriedades do sistema, classes que representam as constantes do sistema, classes que facilitaram a validação e a formatação dos dados de entrada no sistema, entre outros.

6.1. Elementos que compõe a Arquitetura

Quais são os principais elementos da arquitetura?

A arquitetura é composta por alguns elementos, que em conjunto produzem o efeito desejado pela arquitetura como um produto final para o desenvolvimento.

Elementos pertencentes à arquitetura:

- Model
- View
- Controlller
- DAO
- UTILS

Nos próximos capítulos iremos discutir cada componente listado anteriormente e qual o seu papel dentro da arquitetura como um todo, além de discutir de forma sucinta a tecnologia e/ou o padrão adotado para a implementação do mesmo.

6.2. **Model**

Qual a sua principal finalidade?

Quando estamos elaborando uma arquitetura para o desenvolvimento de sistemas, principalmente orientado a objetos, temos que nos preocupar com a separação real das camadas pertencentes à arquitetura. É nesse contexto que começamos a discutir os principais elementos da arquitetura, sendo que agora iremos começar detalhar o Model e seu contexto dentro da arquitetura.

O Model é a camada de negócio, responsável por tratar as regras referentes ao dados da aplicação, regras de negócios, lógica e funções. Dessa forma, no sistema de taxi aéreo cada tabela do nosso banco e seus atributos correspondem a uma classe do Model.

Com a criação do Model, separamos da regra de negócio o controle de acesso a elementos de persistência que são feito em conjunto pelo Controller e pelo DAO.

6.3. **View**

Como é dividida? Quais seus principais elementos?

Agora iremos detalhar a camada de interface com o usuário. Nesse primeiro momento iremos falar sobre as interfaces utilizadas, como ela foi dividida e o que foi envolvido na criação da mesma.

As interfaces foram elaboradas usando-se a API do SWING, que procura renderizar/desenhar por conta própria todos os componentes, ao invés de delegar essa tarefa ao sistema operacional, como a maioria das outras APIs de interface gráfica trabalham.

Dentro do pacote View, as telas foram subdivididas em categorias para cada elemento do Model, e cada uma dessas telas, chama a tela correspondente a operação CRUD que deve vir a ser realizada no sistema. E todas essas telas são "Chamadas" por uma tela principal, que as une que é chamada de TelaPrincipal.java . Tela essa que é instanciada por uma outra tela, chamada de BaseView.java, que funciona como a base para o sistema funcionar, onde nela são também capturadas e externadas as exceções que o sistema possa vir a jogar.

Em JAVA temos um recurso que é muito utilizado e importante no desenvolvimento de sistemas, que é o recurso de Exceptions, ou melhor, exceções. JAVA nos provê vários tipos de exceções já existentes, sendo essas exceções verificadas, exceções de tempo de execução e exceções fatais do sistema. Além das exceções, existe todo um mecanismo para que possamos tratá-las e dar continuidade no fluxo de execução conforme desejamos.

Com isso houve a necessidade de criarmos algumas exceções com mais recursos dentro da nossa arquitetura. Algumas exceções criadas são apenas o encapsulamento de exceções já existentes, só que, com mais recursos que facilitarão o desenvolvimento dos sistemas dentro da arquitetura e outras exceções são novas, e foram criadas para estarem dentro do contexto da arquitetura.

6.4. **Controller**

Onde é implementada a regra de negócio?

Todo sistema é formado por um conjunto de regras de negócio, ou seja, um fluxo lógico que deve ser processado para que tenhamos o resultado desejado. Uma regra de negócio é representada na UML (linguagem de modelagem de sistemas orientada a objetos) através de um caso de uso (modelagem da regra de negócio). Com a análise orientada a objetos a preocupação com a separação real da regra de negócio das demais funcionalidades do sistema é constante e essencial para termos um sistema de baixo acoplamento e de fácil manutenção e extensão.

Por esse motivo foi criado o Controller dentro da arquitetura. O controller é responsável por fazer a mediação da entrada, convertendo-a em comandos para o modelo ou visão. No caso do nosso sistema é ele que é chamado pelas ações da view para realizar as operações CRUD que estão presentes na camada DAO.

Com isso temos para cada caso de uso existente no sistema um controlador responsável por implementá-lo, assim temos um controle transacional muito mais robusto (por caso de uso ou pela interação entre eles), por exemplo, cada método dentro do controlador estará sempre sobre o mesmo contexto transacional.

6.5. **DAO**

O que é essa camada? E pra que serve?

No desenvolvimento de sistemas precisamos em muitas vezes fazer acesso a uma determinada base de dados. Nesse caso as classes presentes na camada DAO servem para dar apoio a esse acesso aos dados.

Nessa camada estão presentes a classe de ConexaoBD.java, que possui os parâmetros de conexão ao banco de dados e que retorna uma conexão ao mesmo. Além de possuir uma classe abstrata GenericDAO.java que implementam controles que são iguais a todos os tipos de acessos a bancos de dados, exigindo que as demais classes específicas herdem dessa classe seus métodos.

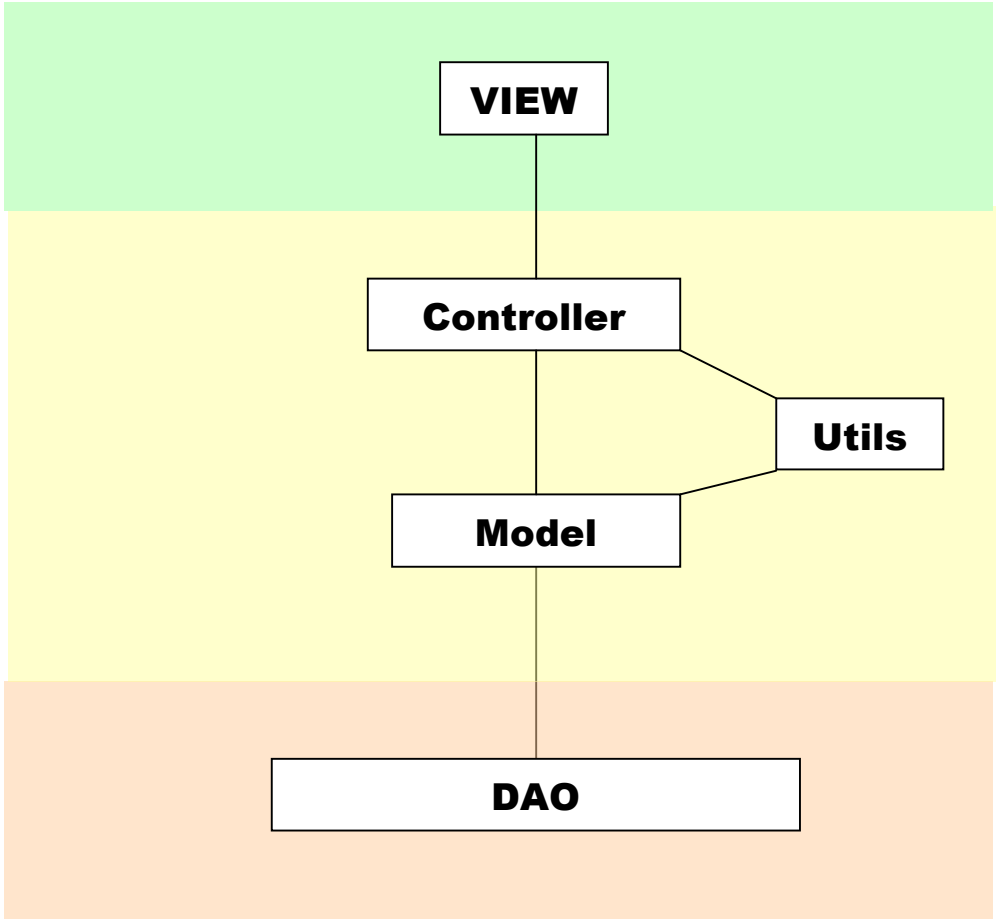
Com esse tipo de solução conseguimos realmente separar das regras de negócio a implementação de funcionalidades básicas de acesso à base de dados, controles transacionais e gerenciadores de conexão.

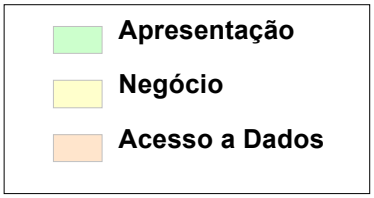
6.6. **UTILS**

O que é essa camada? E pra que serve?

Camada que possui constantes, Strings e funções que são várias vezes utilizadas pelo sistema, dessa forma se evita a reescrita de código e facilita a manutenção do mesmo, pois ao invés de se alterar em N lugares, só se altera em um.

6.7. Desenho GERAL da Arquitetura





6. Objetivos e Restrições Arquiteturais

Esta seção descreve os requisitos e objetivos do software que têm algum impacto na arquitetura, tais como: segurança, proteção de dados, privacidade, portabilidade, distribuição, reuso. Também são descritos nesta seção restrições arquiteturais que se aplicam ao projeto, tais como: estratégias de modelagem e implementação, ferramentas de desenvolvimento, sistemas legados.

8.1. Requisitos básicos

- A arquitetura deve seguir o padrão J2EE/J2SE.
- Windows/Mac OSX como sistema operacional do ambiente de produção.
- Utilização de componentes opensource.
    - Postgres como Banco de Dados da aplicação.