Projeto Arquitetural

Prof Elisabete

Dados baseados no material do Prof. Jacques da UFCG

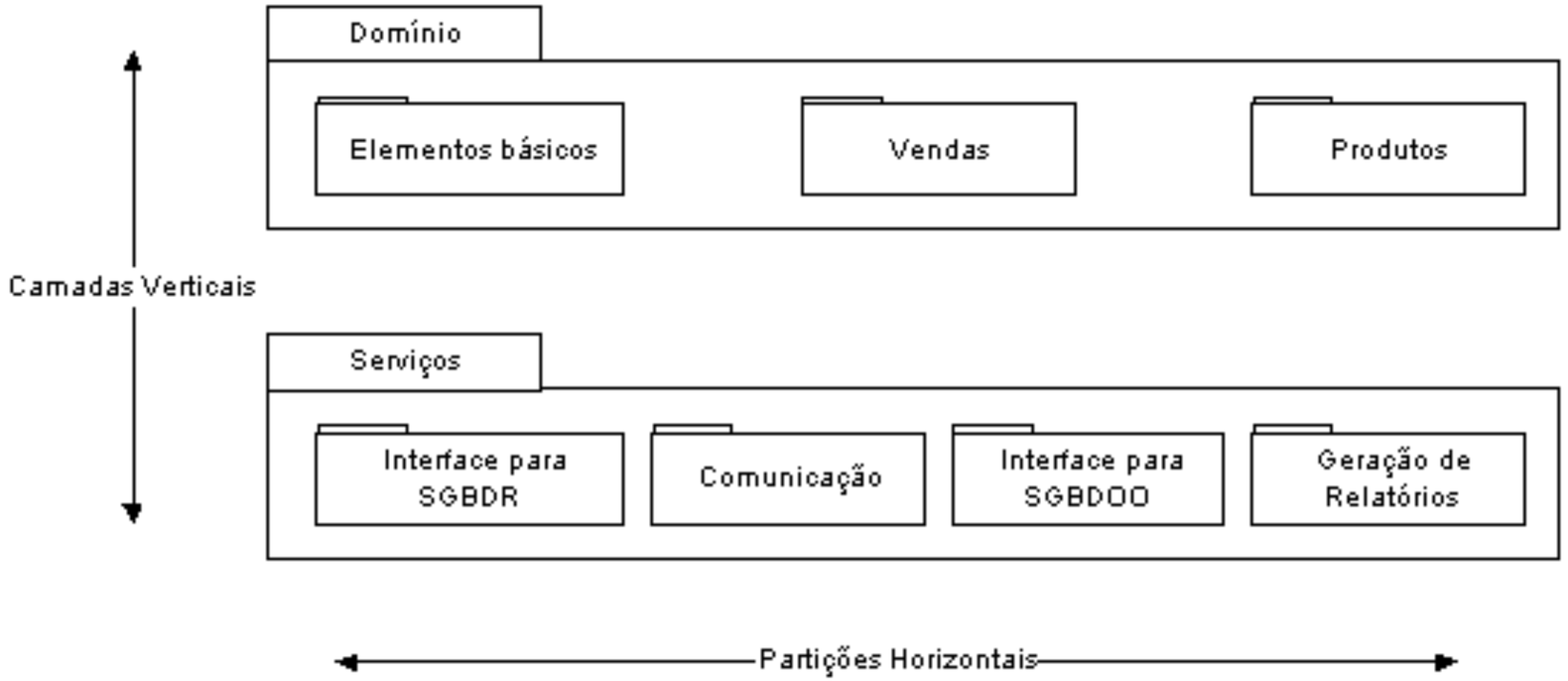
Definição

- São decisões estratégicas que terão consequências profundas ao longo do projeto.

- Exemplos:
  - Modularização do projeto em subsistemas
  - Escolha de uma estrutura de comunicação e controle entre os subsistemas
  - Escolha da divisão de trabalho entre membros de uma equipe ou entre equipes de desenvolvimento
  - Definição das interfaces entre subsistemas para possibilitar o trabalho paralelo de desenvolvimento
  - Escolha de uma estratégia de persistência
  - Escolha do paradigma de SGBD a usar
  - Determinação de oportunidades para o reuso de software
  - Atendimento a requisitos especiais de desempenho
  - Atendimento a outros requisitos (custo, mobilidade, uso de padrões …)
  - Exposição das interfaces para facilitar a futura integração de aplicações (Enterprise Application Integration - EAI)

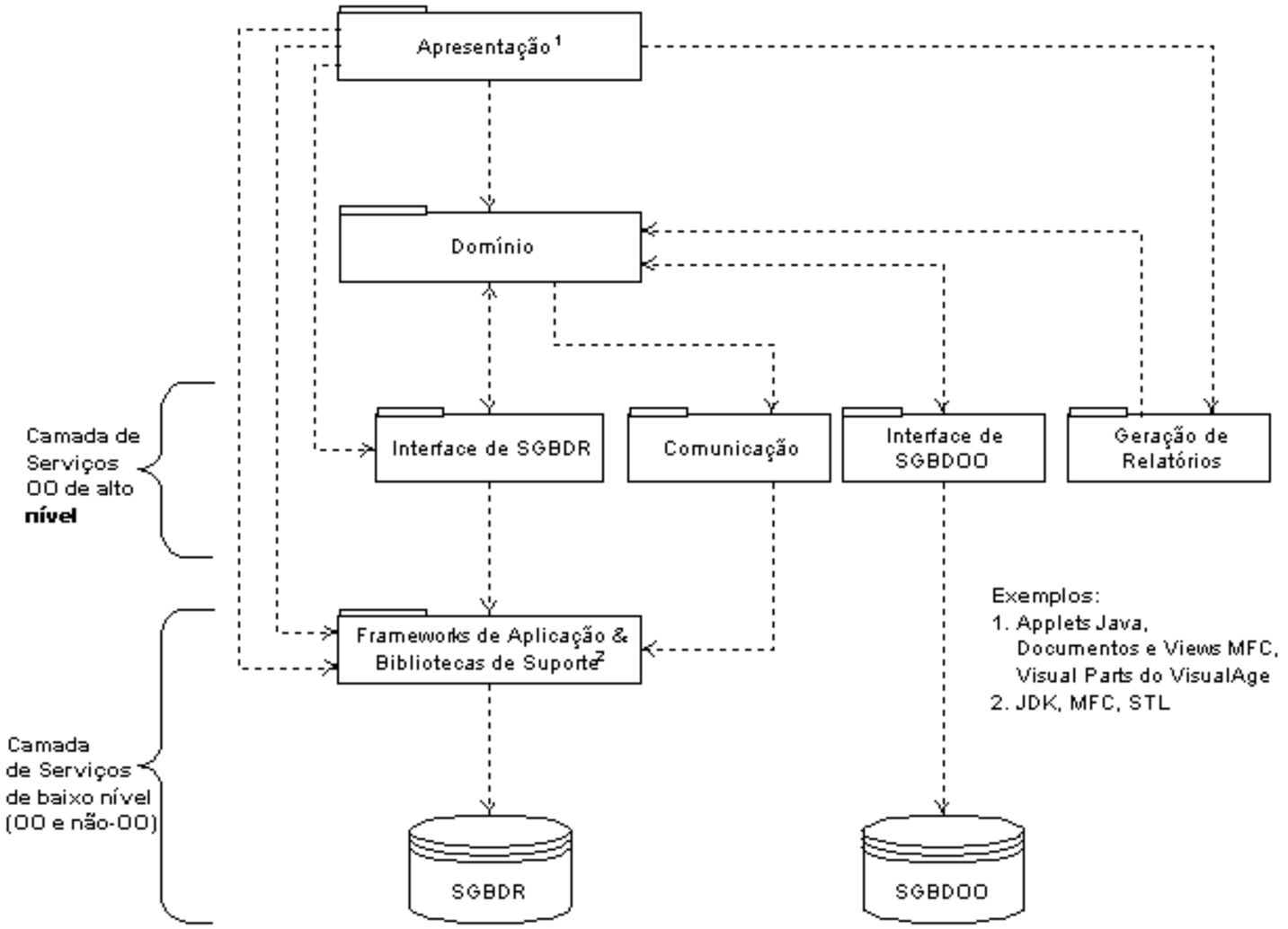
Decomposição do Sistema

- Uma estrutura organizada frequentemente utilizada são as camadas e partições.



Modelagem com UML

- A UML tem o objetivo de evidenciar os subsistemas e suas dependências na construção do software.
- Facilita o entendimento de como a estrutura deverá ser distribuída para a o desenvolvimento.



Paradigmas de Controle Externo

- Controle orientado a procedimento
  - Tem um único thread (tarefa) no programa
  - Métodos executam chamadas para obter dados de entrada e esperam os dados
  - O estado do programa pode ser guardado em variáveis locais aos procedimentos (na pilha de execução)
  - Vantagem: fácil de implementar com linguagens convencionais
  - Desvantagem: qualquer paralelismo inerente aos objetos deve ser mapeado a um fluxo de controle sequencial
  - Frequentemente usado em aplicações que possuem uma interface não gráfica

- Controle orientado a evento
  - Um dispatcher (expedidor) é provido pela linguagem, SGBD, linguagem ou sistema operacional
  - Métodos da aplicação são amarrados a eventos
    - Quando o evento ocorre, o método correspondente é chamado
  - Todos os métodos retornam o controle para o dispatcher em vez de esperar que os dados de entrada cheguem
  - O estado do programa deve ser guardado em variáveis globais já que os métodos retornam o controle (e não ficam com ele)
  - Vantagem: ajuda a separar a lógica da aplicação da interface com o usuário
  - Frequentemente usado com SGBD munido de linguagem de quarta geração
  - Muito usado com bancos de dados
  - Quase sempre usado para controlar uma interface gráfica

Paradigmas de Controle Externo

- Controle concorrente
  - O controle fica com vários objetos autônomos, cada um respondendo a eventos
  - A concorrência é provida pelo sistema operacional
    - Usando paralelismo em hardware ou não
  - Observe que estamos nos referindo a controle concorrente *dentro* da aplicação e não *entre* aplicações
    - Este último é muito comum (sessões paralelas de acesso a BD, por exemplo)
  - Infrequentemente usado, embora seu uso tenda a crescer com o CORBA (framework de objetos distribuídos)

- Controle declarativo
  - O programa usa um fluxo de controle implícito beaseado em statements declarativos
  - Usado em sistemas especialistas baseados em regras

Persistência

- Pontos chaves:
  - Escolha básica da persistência:
    - Dados em memória
    - Dados em arquivos
    - Dados sob controle de um BD
  - Se usar um SGBD, escolha do paradigma de SGBD.
  - Determinação da estratégia de interação entre a aplicação e os dados.

Escolha Básica da Persistência

	Dados na memória	Arquivos	SGBDs
Persistência de dados	Requer hardware especial	Bom suporte	Bom suporte
Custo de aquisição	Custo do hardware especial	Não há	Pode ser caro
Custo total de posse	Variável	Variável	Variável
Quantidade de dados	Limitado pelo hardware	Limitado pelo sostema operacional; A memória limita files em cache	Essencialmente sem limite
Desempenho	Muito rápido	Rápido para acesso sequencial, certos acessos randômicos e para arquivos em cache	Rápido
Estensibilidade	Limitada	Limitada	Excelente
Acesso concorrente	Locking de objetos	Locking de arquivos	Locking de objetos ou de registros; Alguns SGBDs só têm locking de páginas
Recuperação de crash	Shadow memory	Arquivos de backup	Bom suporte
Integridade	Não há	Não há	Projetista pode especificar regras
Suporte a transações	Não há	Não há	Transações curtas
Distribuição	Não há	Não há	Às vezes
Linguagem de consulta	Não há	Parcial	Poderosa
Segurança	Não há	Proteção simples do sistema operacional	Pode ser simples ou sofisticado
Metadados	Não há	Não há	Sim

Benefícios por Tipo

- Dados na memória
  - Dados para dispositivos eletrônicos (celulares, PDAs, ...)
  - Dados para smart cards, cartuchos de jogos
  - Dados para aplicações que não podem tolerar o custo ou falta de confiabilidade de um disco

- Arquivos
  - Dados de grande volume, com pouca densidade de informação (archival files, registros históricos detalhados, logs, ...)
  - Dados crus que devem ser sumarizados num BD (exe.: aquisição de dados, telemetria, ...)
  - Quantidades modestas de dados com estrutura simples (quando não tem SGBD ou este seria complexo demais para a empresa manter)
  - Dados acessados sequencialmente
  - Dados que são lidos por inteiro na memória

- SGBDs
  - Dados que requerem atualização com níveis finos de granularidade por múltiplos usuários
  - Dados que devem ser acessados por várias aplicações
  - Grandes quantidades de dados que devem ser eficientemente manipulados
  - Dados de longa duração e muito valiosos a uma organização
  - Dados que devem ser acessados com sofisticado controle de segurança
  - Dados sujeitos a análise sofisticada para o apoio à decisão

Escolha do Paradigma SGBD

- SGBDRs são maduros mas sofrem de um problema sério: impedance mismatch (imcompatibilidade de tipo) entre objetos e relações.
  - As grandes desvantagens são:
    - Navegação lenta (usando joins de tabelas)
    - Protocolos de travamento inflexíveis (por serem automáticos)
    - Poucos tipos de dados
    - Tudo *tem* que ser uma tabela

- SGBDOOs não são tão maduros mas não sofrem de impedance mismatch e apresentam recursos avançados (evolução de esquema, controle de versões, long transactions, notificações de mudanças, ...)
  - Oferecem navegação rápida mas ainda carecem até de uma teoria completa!

Determinação da Estratégia de Interação entre Aplicação e os dados

- Pré-processador e pós-processador batch
- Arquivos de Script (SQL)
- API de acesso aos dados
- Store Procedures
- Camada Genérica Orientada a Objetos

Perguntas ao Fazer um Projeto Arquitetural

Sobre entidades externas ao sistema

- Quais sistemas externos devem ser acessados?
- Como serão acessados?
- Há integração com o legado a ser feita? Como?

Determinação de oportunidades para o reuso de software

- Há interesse/conveniência/tempo em aproveitar tais oportunidades?
- Como isso pode ser feito?
  - Com componentes?
  - Construindo um framework?

Sobre a organização geral do sistema

- O sistema é centralizado ou distribuído?
- Como modularizar em subsistemas?
- Como minimizar acoplamento entre os pedaços?
- Tais camadas serão lógicas ou terão existência física separada?
- Onde colocar o business logic (como dividir entre as camadas)?
- Qual é o nível de acoplamento, frequência de interações, volumes de dados trocados entre as camadas?
- Qual seria a estrutura de comunicação e controle entre os subsistemas (como ligar as camadas)?
  - Usando o Observer Pattern?
  - Usando o Model-View-Controller Pattern?
  - Usando outro Design Pattern?
- Quais são as interfaces importantes entre os pedaços?
  - Qual é o formato das mensagens (xml, ...)?
  - Como é feita a comunicação remota?
- A programação será feita com qual paradigma? OO?
- Que linguagens e ferramentas serão usadas?

Perguntas ao Fazer um Projeto Arquitetural

Sobre a camada de interface

- O sistema será acessado usando que tipos de clientes?
  - Browser?
  - Uso de applet?
  - Uso de script cliente?
- Como fazer a interface gráfica?
  - Com que ferramentas?
  - Com que componentes visuais?
    - Serão desenvolvidos? comprados?
  - Javascript ou outra linguagem de script do lado cliente será usada?
- Que modelo de componentes visuais será usado?
  - Activex?
  - Javabeans?
- Se a interface usar um browser, como será feita a geração de HTML dinâmico?
  - Servlets?
  - JSP?
  - ASP?
- Que ferramentas usar para a formatação de relatórios?
- Há packages a desenvolver que poderiam ser comuns a várias partes da interface?

Sobre a camada de lógica da aplicação

- Escolha de middleware: qual modelo de componentes de servidor será usado?
  - COM+?
  - EJB?
- Quais são os componentes principais a fazer?
- Serão componentes persistentes?
- Serão componentes com ou sem estado?
- De que forma atender aos requisitos para uso multiusuário?
  - Soluções para resolver a concorrência
- Que APIs serão usadas para acesso a:
  - Dados persistentes?
  - Serviço de mail?
  - Serviço de nomes?
- Há packages a desenvolver que poderiam ser comuns a várias partes da lógica da aplicação?
- Qual é a organização interna dos modulos executáveis?
  - Determinar sub-camadas e partições
- Quais são as interfaces importantes entre os pedaços?
- Onde verificar os business rules?
  - No SGBD?
  - No middleware?
- Como implementar aspectos de segurança?
- Como implementar os requisitos de auditoria?

Perguntas ao Fazer um Projeto Arquitetural

Sobre a camada de dados persistentes

- Quais são as fontes de dados? externas? internas? existentes? novas?
- Como acessa-las?
- Que estratégia de persistência será usada:
  - Na memória (com recursos especiais como pilha)?
  - Em arquivos?
  - Usando um SGBD?
- Qual paradigma de SGBD usar?
  - Relacional?
  - O-R?
  - O-O?
- Onde usar stored procedures para implementar os business rules ou garantir a integridade dos dados?
- Qual será a estratégia de interação entre a aplicação e os dados?
- De que forma atender aos requisitos para uso multiusuário (concorrência)?
- Como resolver o impedance mismatch entre a camada de lógica de aplicação e o esquema de persistência?
- Para grandes escalas, como oferecer connection pooling?
- Como implementar a segurança no SGBD?
- Como será feita a população dos bancos de dados?

Sobre requisitos de desempenho

- Há necessidade de bolar formas especiais de atender aos requisitos de desempenho?
  - Como prover escala?
  - Como prover failover?

O que deve ser produzido?

- Quais são os módulos executáveis a produzir?
  - Executáveis, DLLs, ...
- Quais são os arquivos importantes (de configuração, etc.) e seu formato (xml, ...)?
- Como será resolvida a instalação do produto?
- O que será comprado, feito, modificado?
- O que será gerado automaticamente com uma ferramenta especial?
  - Exemplo: geração de esquema através de ferramentas CASE, ...

Sobre a integração futura

- Que interfaces devem ser expostas para facilitar a futura integração de aplicações (Enterprise Application Integration - EAI)?
  - Pode-se usar uma camada de API para expor a camada de business logic, colocar uma camada de script acima disso e ainda camada(s) de interface para ativar a business logic através de scripts.
  - Vantagens:
    - Fácil criação de macros e outras formas de automação
    - Fácil criação de testes de regressão
    - Clara separação de business logic da interface para o usuário
- Como será feita a exposição?
  - Com XML?
  - Através de uma API?

Perguntas ao Fazer um Projeto Arquitetural

Perguntas adicionais

- Que ferramentas de desenvovimento serão geradas?
- Como será o atendimento a outros requisitos (custo, mobilidade, uso de padrões, etc.)
- Há considerações especiais de administração a levar em conta?
  - Como será a troca de versões?
  - Como será a distribuição do software?
    - Via Internet/Intranet?
    - Via mídia?

Pergunta final

- Você já verificou que a arquitetura bolada pode atender a todos os requisitos levantados?