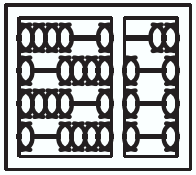
Arquitetura de Software

Cecília Mary Fischer Rubira
Patrick Henrique da Silva Brito



Instituto de Computação (IC)
Universidade Estadual de Campinas
(Unicamp)
INF064'2008

Roteiro

(1) Introdução e Definição dos Conceitos;

(2) Visões Arquiteturais;

(3) Estilos e Padrões Arquiteturais.

Parte 1: Introdução e Definição dos Conceitos

Problema (I)

- Um problema crucial em compor um sistema de software a partir de componentes (módulos ou subsistemas) é garantir que a composição dos módulos satisfaz um dado conjunto de:

  (1) Restrições globais de integridade, e.g., restrições de comunicação e coordenação entre componentes e invariantes estruturais, e

  (2) Atributos de qualidade, e.g., evolvabilidade, adaptabilidade, disponibilidade, segurança, persistência.

Problema (II)

- Essas restrições e atributos são sistêmicos, isto é, elas não podem ser providas pelas partes individualmente mas sim pela composição/conjunto.
- Exemplo:
  - Suponha a existência de 2 componentes tolerantes a falhas que são compostos para formar um terceiro componente tolerante a falhas.
  - Não há garantia que o terceiro componente seja tolerante a falhas pelo fato das suas partes serem tolerantes a falhas.

Problema (III)

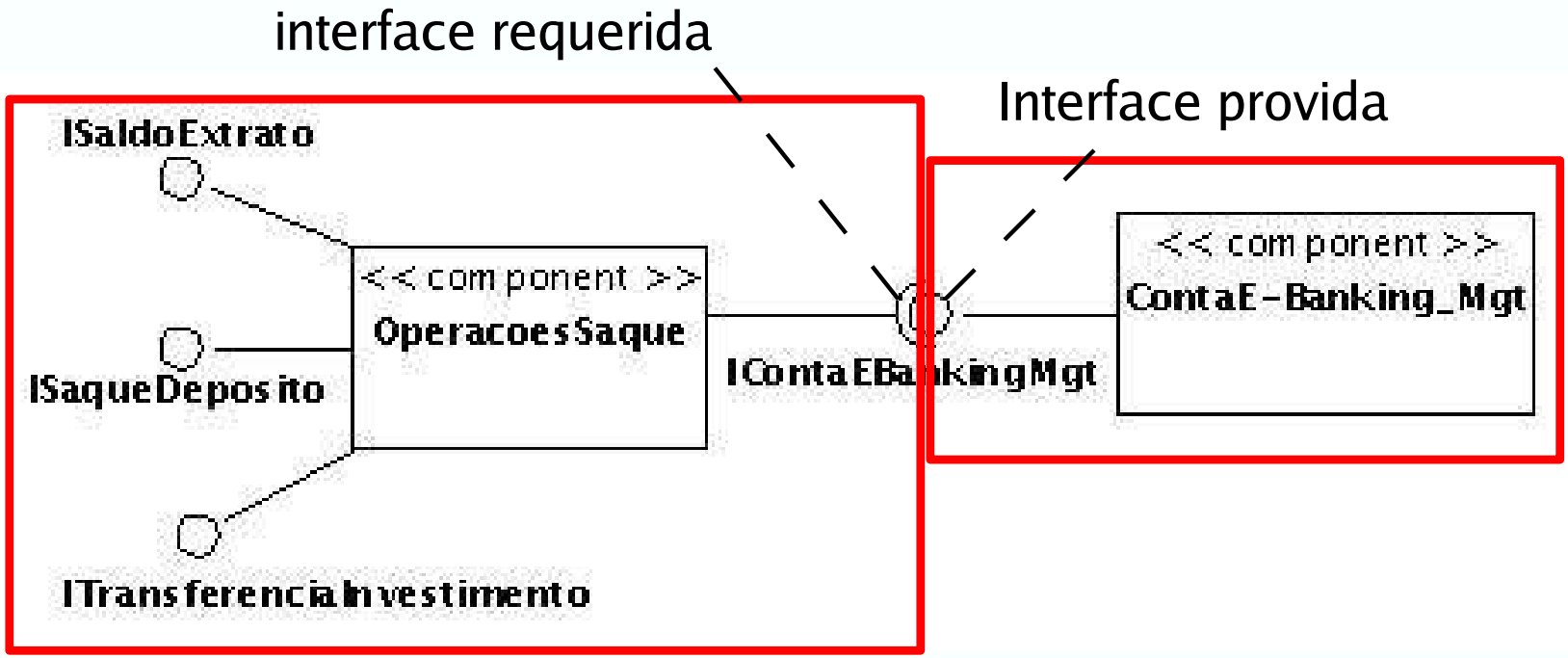
- Portanto, é necessário um modelo arquitetural para descrever a arquitetura do sistema:
  - Com um grau de abstração mais alto do que implementação e
  - Independente de plataforma de implementação e linguagem de programação.

Definição

- A arquitetura de software de um sistema descreve um modelo de nível alto de abstração (i.e. "sobe a montanha e vê o panorama") em termos de:
  - **Componentes Arquiteturais** (módulos ou componentes) que executam a computação (requisitos funcionais do sistema), e
  - **Conectores Arquiteturais** explícitos que unem os componentes e coordenam as interações entre os componentes para que a composição satisfaça restrições e atributos de qualidade globais do sistema específico a ser construído.
  - **Configuração Arquitetural** representa um conjunto de componentes e conectores interligados.

Componentes Arquiteturais (I)

- Representam os módulos funcionais do sistema;
- São responsáveis pela execução dos serviços em si;



Componentes Arquiteturais (II)

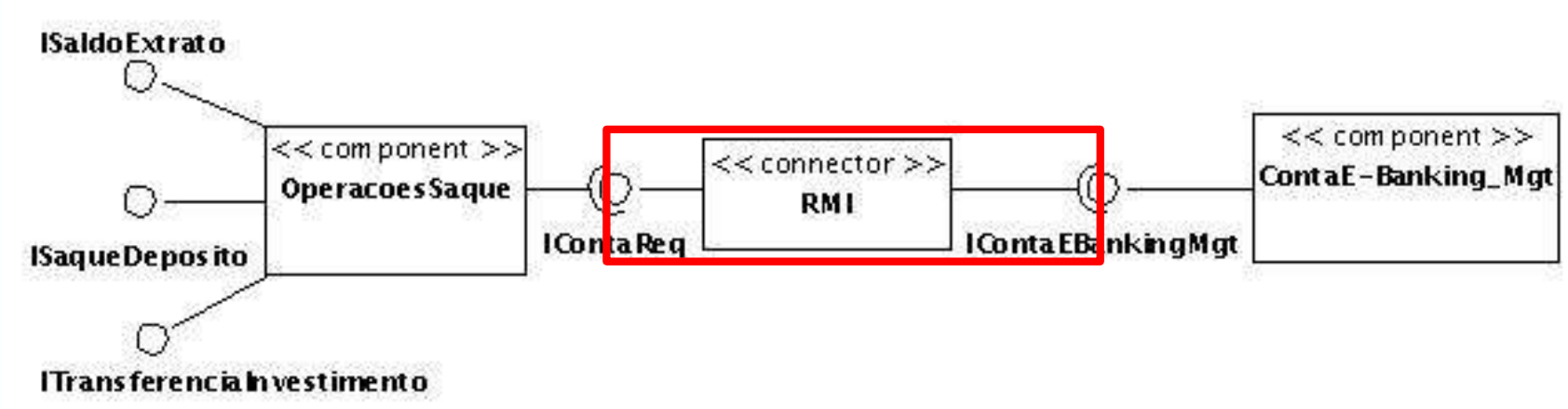
- *"Components are specified by interfaces; they correspond roughly to compilation units of conventional programming languages and other user-level objects (e.g., files."* [Shaw1996]
- *"(A component is) a primary computational element or data store of a system, such as a client, server, filter, object, blackboard, or database."* [Garlan2000]

Exemplos de Componentes

- Web browser;
- Servidor WEB;
- Servidor DNS (*Domain Name System*);
- Gerenciador de Bancos de Dados;
- Componente de Controle de Estoque.

Conectores Arquiteturais (I)

- São tipos de componentes mais simples;
- Materializam explicitamente a interação entre os componentes do sistema.



Conectores Arquiteturais (II)

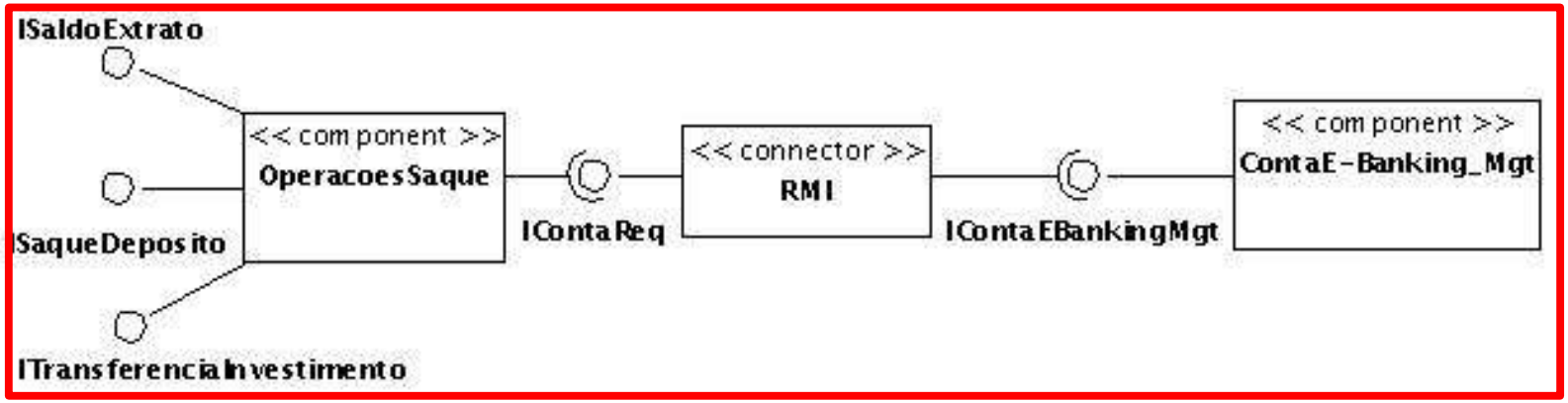
- *"Connectors are specified by protocols; they mediate interactions among components. That is, they define the rules governing component interaction and specify any auxiliary implementation mechanism required."* [Shaw1996]
- *"(A connector is) A mediator of the communication and coordination activities among components. Examples include pipes, procedure call, event broadcast, a client-server protocol or a SQL link between a database and an application."* [Garlan2000]

Exemplos de Conectores

- Call/return;
- Java RMI (*Remote Name System*);
- ODBC (*Open Database Connectivity*);
- HTTP (*Hyper-Text Transfer Protocol*).

Configurações Arquiteturais (I)

- Representam um conjunto de componentes e conectores interligados.



DBC x Arquitetura de Software

- Enquanto a arquitetura lida com elementos conceituais abstratos, os componentes DBC são concretos, representando necessariamente um único módulo funcional do sistema;
- A arquitetura é utilizada normalmente para definir a relação entre os "grandes módulos" do sistema, que podem ser refinados até atingir uma granularidade de DBC;
- Enquanto a arquitetura é o principal artefato da fase de integração do sistema, a seleção dos componentes de DBC utilizados é o objetivo da fase de engenharia de aplicação.

O que é um Componente?

- Ainda não se tem uma definição de consenso:
  - *"Um componente como um elemento arquitetural. Este componente deve prover e estar em conformidade com um conjunto de interfaces"*;
  - *"Um componente como um elemento implementacional acessados através de interfaces bem documentadas que podem ser descobertas em tempo de execução"*;
  - *"Um componente como um elemento implementacional mas que faz parte de um contexto arquitetural específico"*.

Motivação (I)

- A motivação principal para a definição de uma arquitetura de software é:
  - Proporcionar um plano concreto com o objetivo de predizer o comportamento do sistema **antes** dele ser construído e **guiar** o seu desenvolvimento, sua manutenção e sua evolução através do ciclo de vida do sistema.
- A importância do papel da arquitetura começou a ser reconhecido por desenvolvedores e pesquisadores em meados da década de 90.

Motivação (II)

- O projeto de uma boa arquitetura de software ajuda efetivamente na construção de um produto de software que atende os requisitos dos interessados do sistema (*stakeholders*).
- *Stakeholders* são pessoas ou organizações interessadas na construção do sistema;
- Uma arquitetura inapropriada certamente trará conseqüências desastrosas para um projeto.

Motivação (III)

- Todo sistema já implementado tem uma arquitetura de software embutida nele, que pode ser decomposta e reconstruída em termos de componentes e conectores (*architecture reconstruction* e *architecture recovery*);
- Padrões de interações entre os componentes dão apoio para raciocinar sobre os atributos de qualidade do sistema como um todo;
- O modelo arquitetural representa uma abstração de alto nível para que os stakeholders do sistema pensem a respeito dos seus atributos (requisitos funcionais e não-funcionais) e para que façam decisões de extensões, customizações e evoluções a respeito do sistema.

Projeto Arquitetural

- Deve ser uma das primeiras atividades de um processo de desenvolvimento de software;
- Representa a ligação entre os métodos de análise e projeto;
- Normalmente o projeto arquitetural é conduzido em paralelo com outras atividades de especificação;
- Envolve a identificação dos principais componentes do sistema e das ligações e regras de interação entre eles;
- A saída final do projeto arquitetural é a arquitetura de software propriamente dita.

Ciclo de Negócio da Arquitetura

(The Architecture Business Cycle – ABC)

- Por décadas, os engenheiros de software aprenderam a construir sistemas baseados exclusivamente nos requisitos técnicos do sistema;
- A arquitetura de software engloba a estrutura ou estruturas de um sistema de software complexo e grande;
- A visão arquitetural é **abstrata**, ignorando detalhes de implementação, algoritmos e representação de dados;
- Ela se concentra no comportamento e interação entre elementos caixa-preta.

Eficácia na Comunicação

- A arquitetura serve como uma ferramenta importante para comunicação, para tomada de decisões (*reasoning*), para análise e para o crescimento de sistemas;
- A arquitetura de software é o resultado de um conjunto de decisões de negócio e técnicas;
- Portanto, ela é um resultado de influências (explícitas e implícitas): técnicas, de negócios e sociais;
- Os requisitos são explícitos mas existem influências que são implícitas.

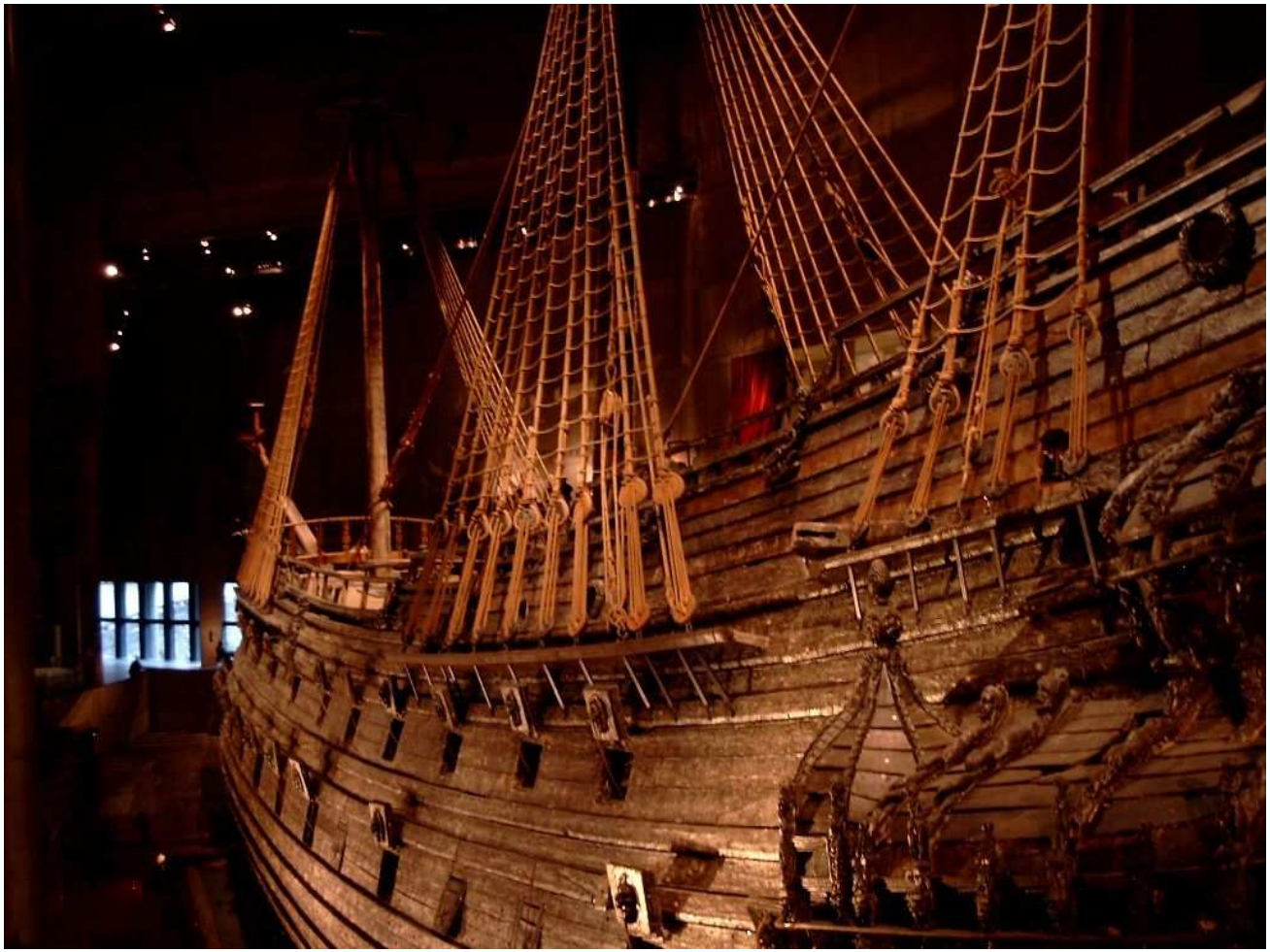
Exemplos

1. Um mesmo conjunto de requisitos passados para uma organização A e B provavelmente produzem arquiteturas diferentes;
2. Um mesmo arquiteto dentro de uma organização com um conjunto de requisitos pode tomar decisões diferentes, dependendo se o projeto tem pouco prazo ou muito prazo, produzindo assim arquiteturas diferentes.

Perguntas

1. Que fatores influenciam a arquitetura de um sistema?
2. Como a arquitetura afeta a organização onde ela está inserida?

Um caso de fracasso:
O Navio Sueco VASA (I)



Um caso de fracasso: O Navio Sueco VASA (II)

- 1620, Suécia em guerra com a Polônia;
- O rei sueco, Gustave Adolphus, manda construir um navio novo;
- Requisitos do navio: 70 metros de comprimento para comportar 300 soldados, com 64 canhões pesados dispostos em dois *decks*;
- O arquiteto do Vasa, Henrik Hybertsson, um construtor de navio holandês foi contratado;
- Ele não tinha construído navios com *deck* duplo de canhões.

Um caso de fracasso: O Navio Sueco VASA (III)

- Ninguém tinha construído um navio com o tamanho e armamento do Vasa;
- O arquiteto teve que balancear vários interesses críticos (muitas vezes conflitantes):
  1. rapidez para a entrega e instalação,
  2. desempenho,
  3. funcionalidade,
  4. confiabilidade,
  5. custo.

Um caso de fracasso: O Navio Sueco VASA (IV)

- O arquiteto era responsável por vários stakeholders:
  1. o rei que estava pagando o projeto (cliente);
  2. tripulação do navio (usuários finais).
- De acordo com a sua experiência, ele projetou o Vasa como se fosse um navio com apenas um *deck* de canhões e depois extrapolou o projeto para comportar um *deck* duplo de canhões;
- Ele morreu um ano antes do navio ficar pronto;
- O navio demorou 8 anos para ser construído.

Um caso de fracasso: O Navio Sueco VASA (V)

- Em 1628, o navio afundou na baía de Estocolmo na sua viagem inaugural;
- Água entrou pelas portas abertas dos canhões que foram disparados para a inauguração;
- 12 homens morreram da tripulação de 150;
- O navio foi bem construído mas sua arquitetura foi mal projetada;
- O arquiteto fez um trabalho pobre em: gerenciamento de riscos e gerenciamento dos requisitos do cliente.

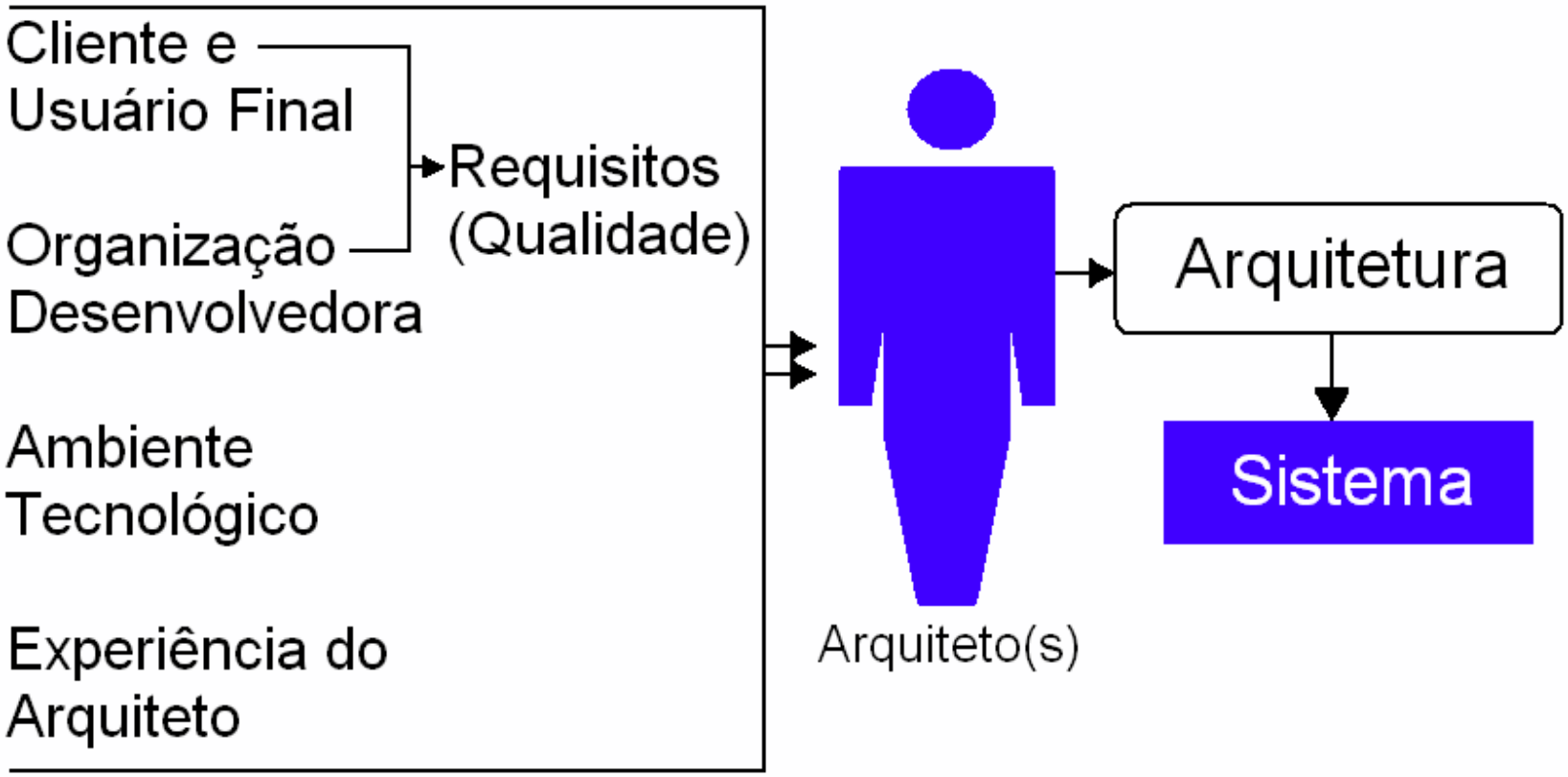
Um caso de fracasso: O Navio Sueco VASA (VI)

- O que o arquiteto poderia ter feito antes para evitar que apenas no dia da entrega do navio fosse descobrir os problemas?
  1. estudos de casos para verificar se a arquitetura satisfazia os requisitos exigidos,
  2. métodos para avaliar a arquitetura antes que um sistema fosse construído a partir dela, para que a chance dos riscos acontecerem fosse diminuída,
  3. técnicas para desenvolvimento incremental baseado na arquitetura para encontrar falhas de projeto antes que fosse tarde demais.

Influências na Arquitetura (I)

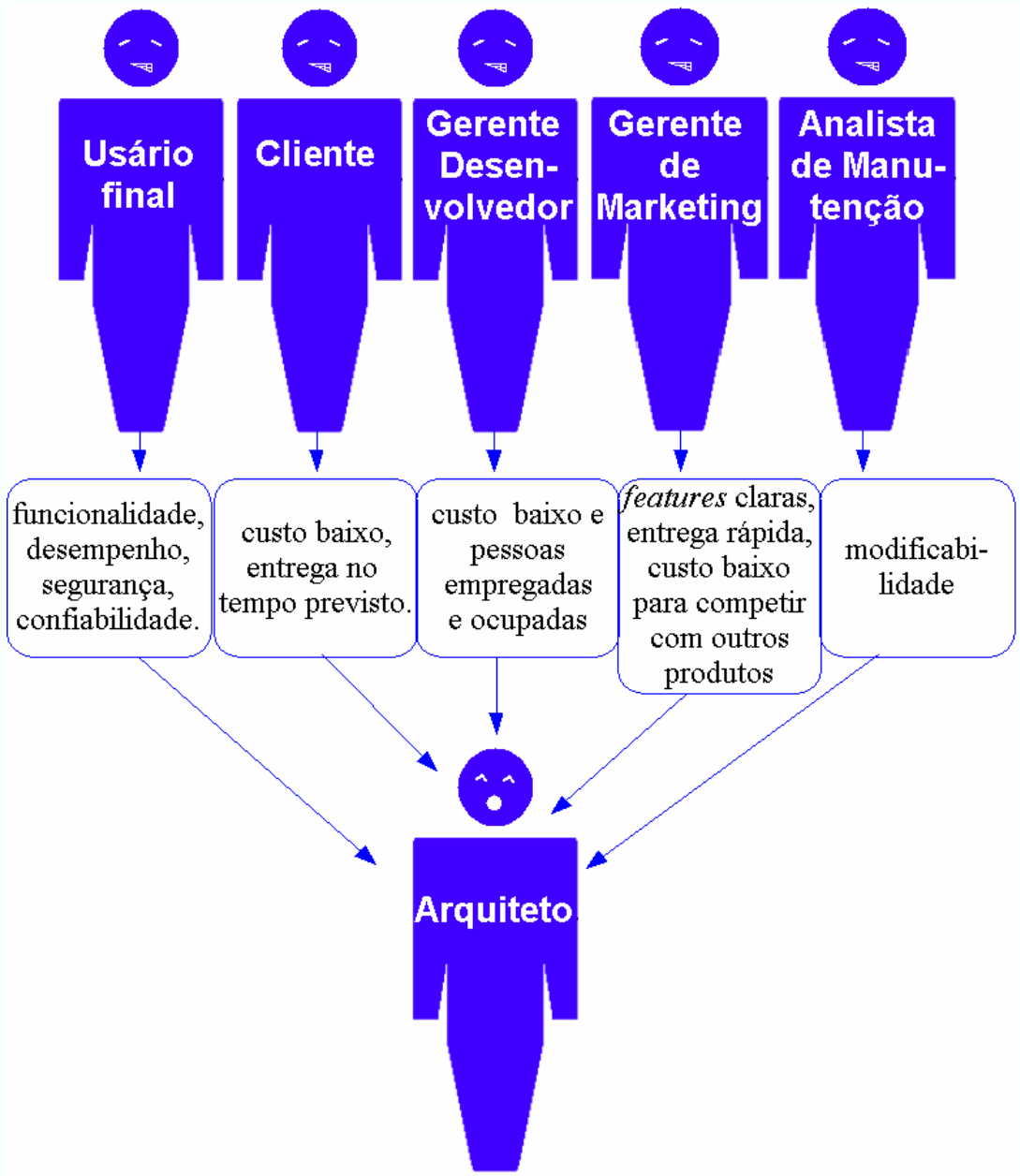
- Arquitetos são influenciados:
  - pelos requisitos da aplicação (funcionais e não-funcionais),
  - pela estrutura e objetivos da organização desenvolvedora,
  - pelo ambiente técnico disponível, e
  - pela sua própria experiência.

Influências na Arquitetura (II)



Arquitetos são Influenciados pelos Stakeholders

- Clientes: paga a conta;
- Usuário Final: usa o sistema;
- Arquitetos;
- Gerentes de Projeto;
- Analistas e Projetistas de Desenvolvimento e Manutenção;
- Programadores;
- Testadores.
- Portanto, diferentes *stakeholders* diferentes têm interesses/preocupações particulares.


Usário final
Cliente
Gerente Desen-volvedor
Gerente de Marketing
Analista de Manu-tenção
funcionalidade, desempenho, segurança, confiabilidade.
custo baixo, entrega no tempo previsto.
custo baixo e pessoas empregadas e ocupadas
features claras, entrega rápida, custo baixo para competir com outros produtos
modificabi-lidade
Arquiteto

Atributos do Sistema (I)

- Observáveis em tempo de execução:
  - Desempenho: relação entre o tempo esperado e o tempo gasto para a execução das funcionalidades;
  - Disponibilidade: minimização dos riscos dos serviços ficarem indisponíveis;
  - Confiabilidade: minimização dos riscos dos serviços funcionarem incorretamente;
  - Usabilidade: facilidade de operação do sistema, de acordo com o seu público alvo.

Atributos do Sistema (II)

- Não observáveis em tempo de execução:
  - Adaptabilidade: facilidade de se alterar o sistema;
  - Portabilidade: facilidade de adaptar o sistema em plataformas distintas;
  - Reusabilidade: facilidade de se reutilizar os componentes do sistema em projetos posteriores;
  - Testabilidade: facilidade que o sistema oferece para auxiliar a execução dos testes.

Atributos do Negócio

- Prazo de mercado (*time to market*): o tempo máximo desejado entre a idealização do sistema e a sua implantação em um cliente;
- Custo: o esforço necessário para desenvolver o sistema;
- Vida útil: qual o tempo esperado que o sistema será utilizado;
- Mercado alvo: quem são os principais candidatos a usuários do sistema;
- Personalização: facilidade de se ajustar o sistema, de acordo com as necessidades de cada cliente;
- Integrabilidade: facilidade de integração com sistemas existentes.

Atributos da Arquitetura

- Integridade conceitual e consistência;
- Precisão e completude;
- Facilidade de implementação.

Compromissos Existentes (I)

- A descrição da arquitetura é o artefato mais inicial que possibilita as prioridades entre interesses competidores (não-ortogonais) serem analisados, e é também o artefato que manifesta as qualidades do sistema;
- A concepção do sistema para alcançar essas qualidades é o resultado de compromissos feitos que provavelmente não estão documentados em lugar nenhum.

Compromissos Existentes (II)

- $\uparrow$ segurança, $\downarrow$ menor desempenho $\Longrightarrow$ *necessidade de processamento adicional*;
- $\uparrow$ confiabilidade (*reliability*), $\downarrow$ manutenibilidade $\Longrightarrow$ *aumento da complexidade de implementação*;
- $\downarrow$ custo de manutenção, $\uparrow$ custo de desenvolvimento $\Longrightarrow$ *especificação e atualização da documentação*.

Arquitetos são Influenciados pela Organização Desenvolvedora

- A organização pode querer investir em negócios num curto prazo, reutilizando artefatos já existentes;
- A organização pode querer investir a longo prazo numa nova infra-estrutura e pode ver o sistema como uma forma de financiar e estender essa infra-estrutura;
- A estrutura da organização pode influenciar a arquitetura: divisão de um componente para ser construído por especialistas externos.

Arquitetos são Influenciados pelo Ambiente Técnico Disponível

- O ambiente técnico que é corrente quando a arquitetura é projetada irá influenciar a arquitetura.
  - Exemplo: EJB .NET

Arquitetos são Influenciados pela Experiência do Arquiteto (I)

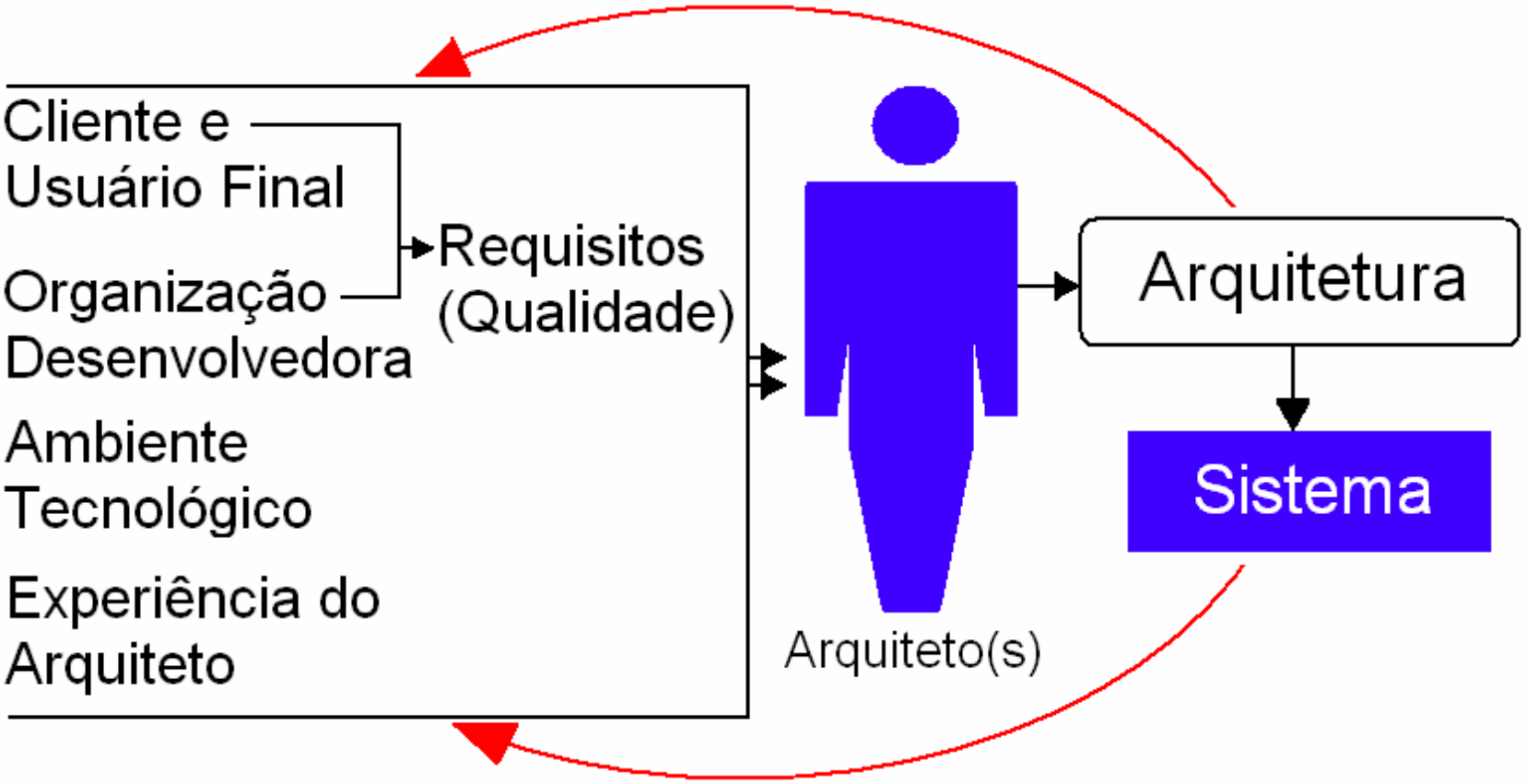
- Se um arquiteto tem um bom resultado com uma abordagem arquitetural particular é provável que ele a aplique em outros projetos;
- O contrário também é verdade;
- Exemplo: objetos distribuídos

Arquitetos são Influenciados pela Experiência do Arquiteto (II)

Importante:

- Arquitetos devem identificar e engajar ativamente os stakeholders para solicitar as suas necessidades e expectativas;
- Sem esse engajamento ativo, os stakeholders, em algum momento, podem argumentar com os arquitetos que a arquitetura proposta não é aceitável, portanto, atrasando o projeto.

O Ciclo Arquitetônico
Architecture Business Cycle



A Arquitetura Pode Afetar ... (I)

- A estrutura da organização desenvolvedora:
  - A arquitetura prescreve uma estrutura para o sistema;
  - As unidades identificadas são a base para a estrutura da equipe de desenvolvimento, para orçamento dos custos, para o estabelecimento de prazos, para controle de acompanhamento.
- Os objetivos da empresa desenvolvedora:
  - Uma arquitetura bem sucedida pode habilitar a empresa a ganhar uma determinada fatia do mercado.

A Arquitetura Pode Afetar ... (II)

- Os requisitos do cliente:
  - O cliente pode pedir que o próximo sistema seja também baseado na mesma arquitetura dado que o sistema anterior foi bem sucedido.
- A experiência do arquiteto é influenciada pela construção do sistema:
  - O arquiteto vai adicionar mais conhecimento à sua base de experiências.
- Poucos sistemas de fato irão influenciar o ambiente técnico:
  - CORBA, Java, Web são exemplos de tecnologias que apareceram nos anos 90.

Etapas do Desenvolvimento Centrado na Arquitetura

1. Criação de um plano de negócios;
2. Compreensão dos requisitos do sistema;
3. **Criação ou seleção da arquitetura do sistema;**
4. **Representação e divulgação da arquitetura;**
5. **Avaliação da arquitetura;**
6. **Implementação do sistema baseado na arquitetura;**
7. **Evolução do sistema.**

Etapa 1 – Criação de um Plano de Negócio

- Preço;
- Público alvo;
- Prazo (*time to market*);
- Integração com outros sistemas;
- Restrições do sistema.

Etapa 2 – Compreensão dos Requisitos

- Casos de Uso;
- Autômatos Finitos;
- Linguagens de Especificação Formais (sistemas críticos);
- Análise de Domínio (modelo de domínio);
- Protótipos.

Como lidar com requisitos conflitantes?

- Requisitos Funcionais;
- Requisitos de Qualidade;
- Escala de Prioridades.

Etapa 3 – Criação ou Seleção da Arquitetura do Sistema (I)

- Estilos Arquiteturais;
- Padrões Arquiteturais;
- Criatividade.
- Consistência;
- Simplicidade.

Etapa 3 – Criação ou Seleção da Arquitetura do Sistema (II)

- Frederick P. Brooks Jr., livro "The Mythical Man-Month: Essays on Software Engineering – Anniversary Edition", Addison-Wesley, 1995, 1975, pai do IBM 360.
- Brooks argumenta que a "integridade conceitual" é a chave para uma arquitetura sólida de sucesso;
- Integridade conceitual reflete a harmonia do todo, isto é, reflete **um** (e somente um) conjunto de idéias de projeto que funcionam harmoniosamente, ao invés de conter muitas idéias boas mas que são independentes e desacordadas e no conjunto não se casam.

Etapa 3 – Criação ou Seleção da Arquitetura do Sistema (III)

- Ele afirma que a integridade conceitual só pode ser alcançada por um número bem pequeno de arquitetos que se unem para projetar a arquitetura do sistema;
- Esse argumento anterior implica que temos uma "elite" de arquitetos e uma classe de trabalhadores (implementadores)?

Etapa 4 – Representação e Divulgação da Arquitetura

- Diagramas UML;
- Linguagens formais (*Architecture Description Language - ADL*)
- Completa;
- Precisa;
- Compreensível;
- Fácil de Manipular.

Etapa 5 – Avaliação da Arquitetura

- Análise da Arquitetura;
- Simulação de Funcionamento;
- Avaliação Baseada em Cenários.
- Propriedades de Execução;
  - consistência com padrões comuns de comportamento.
- Propriedades de Manutenção;
  - Facilidade em apoiar mudanças.
- Propriedades de Evolução;
  - Portabilidade, reusabilidade, facilidade de adaptação.

Etapa 6 – Implementação do Sistema Baseado na Arquitetura

- Geração Automática de Código;
- Inspeção e Análise de Código.
- Conformidade com a Arquitetura
  - O primeiro passo em direção à garantia de conformidade da arquitetura é ter uma representação explícita e bem definida da comunicação entre os elementos;
  - É recomendado utilizar ferramentas de apoio à criação e manutenção da arquitetura (não apenas ao código fonte).

Etapa 7 – Evolução do Sistema

- Manter a conformidade com a arquitetura
  - Garantir que a implementação continua em conformidade com a arquitetura.
- Evoluir a arquitetura

Sistema de e-banking (I)

- O objetivo do sistema de *e-banking* é oferecer um canal para um Banco disponibilizar via Internet a seus clientes, seus principais produtos e serviços. Nesse cenário, o sistema deve ter como requisitos para seu desenvolvimento a flexibilização para acréscimo de novos serviços e funcionalidades. O sistema *e-banking* deverá reutilizar também um sistema simplificado de controle de conta-corrente.

Sistema de e-banking (II)

Exemplos de serviços e funcionalidades que devem ser atendidos pelo sistema *e-banking*:

1. **Administração do Sistema**:
   (a) Configuração de perfis de usuários;
   (b) Bloqueio/Desbloqueio de usuários;
   (c) *Logs* de auditoria.
2. **Cadastros de Clientes**:
   (a) Cadastro e Alteração de dados cadastrais básicos (endereço, ramo de atividade, etc.);
   (b) Cadastro de contas para transferências;
   (c) Alteração de senhas de acesso e assinatura eletrônica.

Sistema de e-banking (III)

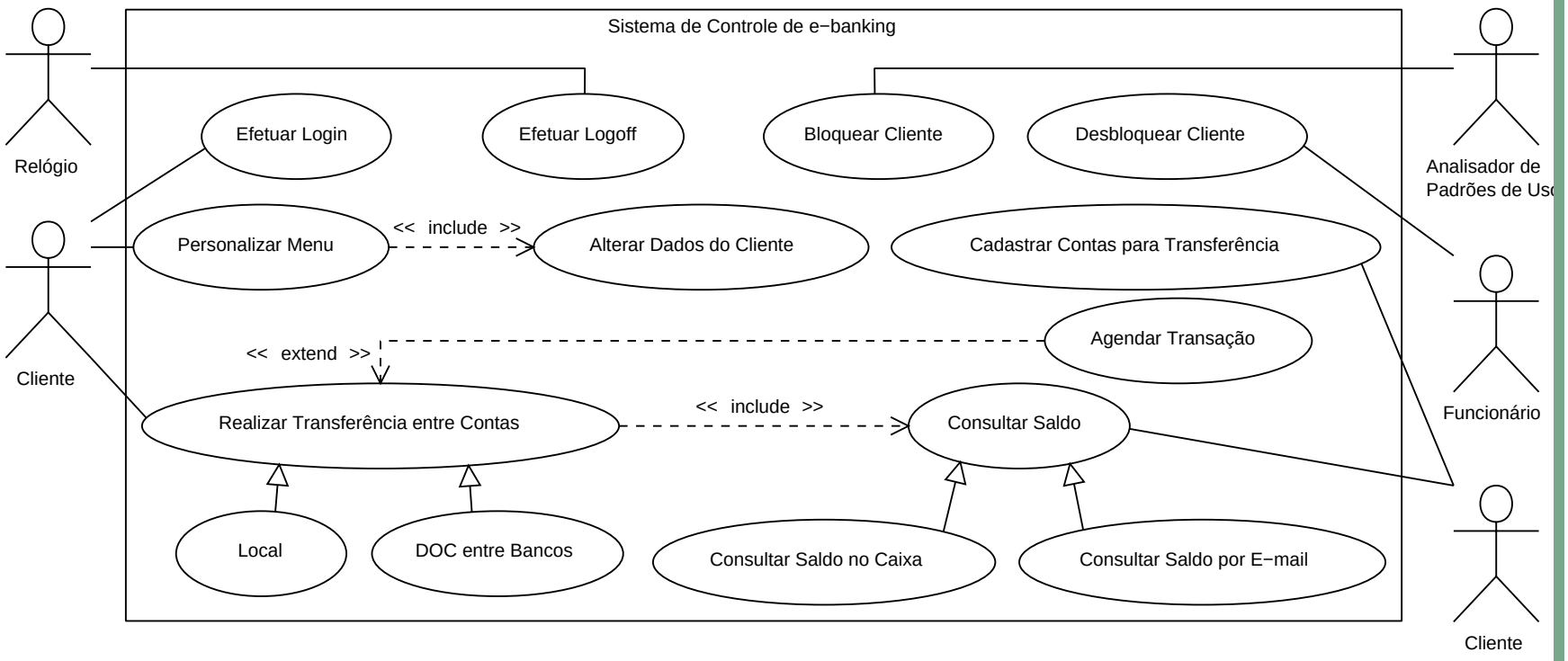
3. **Contas Correntes**:
   (a) Consultas de saldos e extratos (por período);
   (b) Extrato por *e-mail*;
   (c) Transferências entre contas;
   (d) Envio de Doc's C e D;
   (e) Agendamentos de Operações;
   (f) Histórico de Operações.
4. **Chat**:
   (a) Atendente on-line;
   (b) *E-mail* para gerente.

Sistema de e-banking (IV)

5. **Personalização** (***My Banking***): Menu permanente com as opções selecionadas pelo cliente como mais utilizadas.

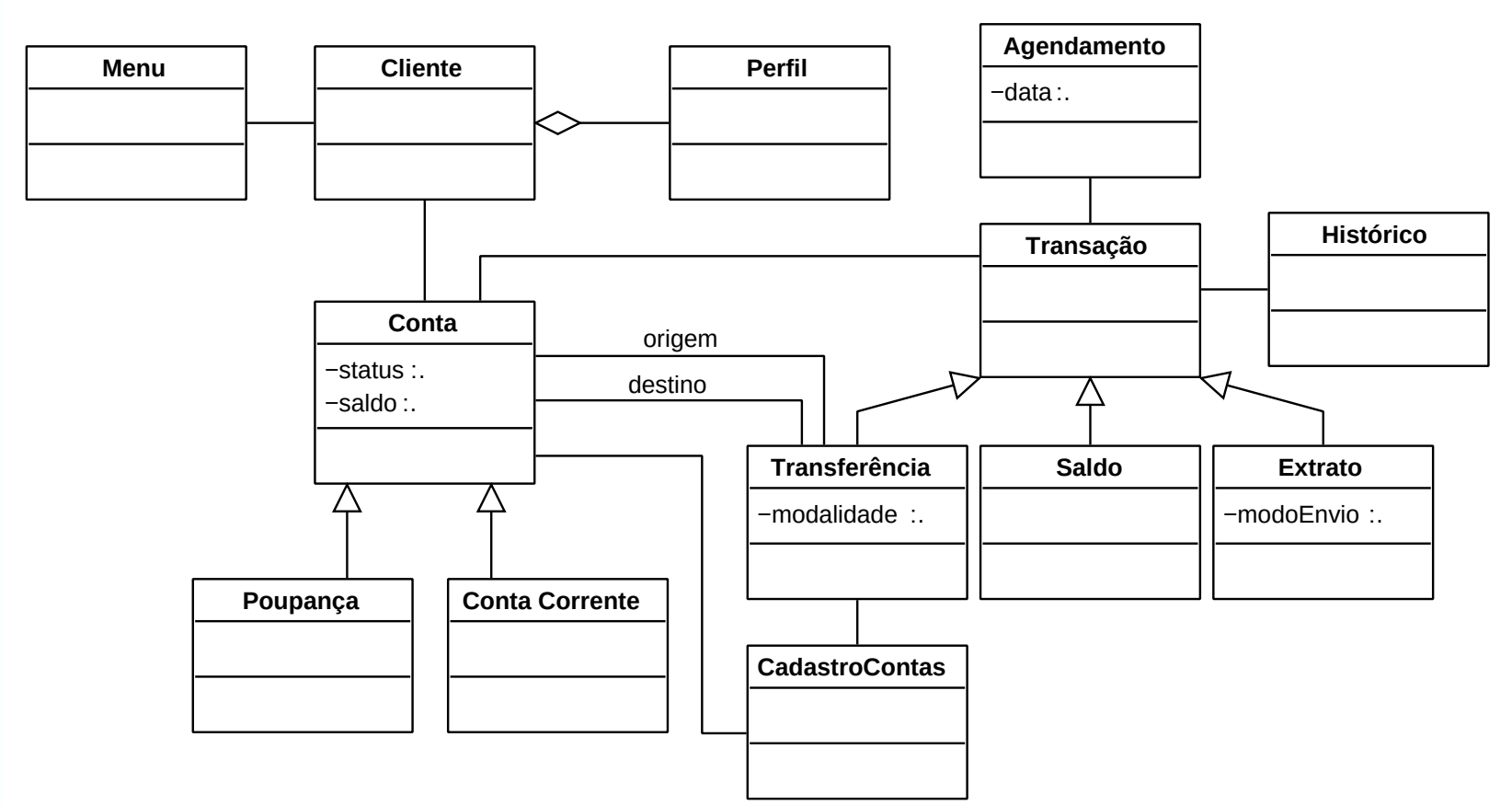
6. **Segurança**:
   (a) Acesso ao *login* e demais dados com criptografia;
   (b) Controle de Acesso;
   (c) Arquitetura deve prever que o sistema interno de conta-corrente não possa ser acessado diretamente pelas páginas do *e-banking*. Além disso, devem estar situados em máquinas diferentes.

Diagrama de Casos de Uso



Modelo Conceitual

- Representação das principais entidades envolvidas no problema:



Exercício

- Identifique os principais interessados e os requisitos de qualidade críticos para o sucesso do projeto do *e-banking*;
- Caracterize os fatores tecnológicos e sociais que afetam o desenvolvimento do *e-banking*;
- Descreva o ciclo arquitetônico do *e-banking*.

Solução Proposta

Interessados

1. Cliente;
2. Usuário;
3. Banco;
4. Arquiteto;
5. Projetista;
6. Desenvolvedores externos.

Requisitos Não-Funcionais (I)

1. Sistema WEB: distribuição (**Alta**);
2. Separação explícita entre dados e aplicação: segurança dos dados (**Alta**);
3. O diálogo do sistema deve ser personalizável de acordo com o cliente: adaptabilidade (**Média**);
4. O sistema deve ser fácil de evoluir: evolvabilidade (**Alta**);
5. O sistema deve estar preparado para mudanças das regras de negócio do domínio, por exemplo, mudanças na economia impostas pelo banco central: evolvabilidade (**Alta**);
6. A comunicação entre o cliente e o servidor utiliza o protocolo HTTP: distribuição - via HTTP (**Alta**);

Requisitos Não-Funcionais (II)

7. Os dados são uma peça chave para a segurança do banco e devem estar seguros e cifrados (com criptografia): segurança dos dados (**Alta**);
8. O desempenho é um requisito esperado desse sistema, apesar de não ser considerado crítico: eficiênia (**Média**);
9. A escalabilidade é um requisito obrigatório nessa aplicação WEB, pois o cliente espera uma média de 10.000 de acessos simultâneos: escalabilidade (**Alta**);
10. Os sistemas internos não podem ser acessados diretamente pelas páginas do *e-banking*: encapsulamento (**Alta**). Além disso, devem estar situados em máquinas diferentes: distribuição (**Alta**);

Requisitos Não-Funcionais (III)

11. O cliente deseja reutilizar ao máximo as bibliotecas existentes na empresa. Essas bibliotecas implementam pequenas funcionalidades, úteis para a implementação de várias camadas: reutilização (**Baixa**);
12. Uso de assinatura eletrônica: segurança de acesso (**Média**);
13. *Log* de Auditoria: rastreabilidade (**Alta**).

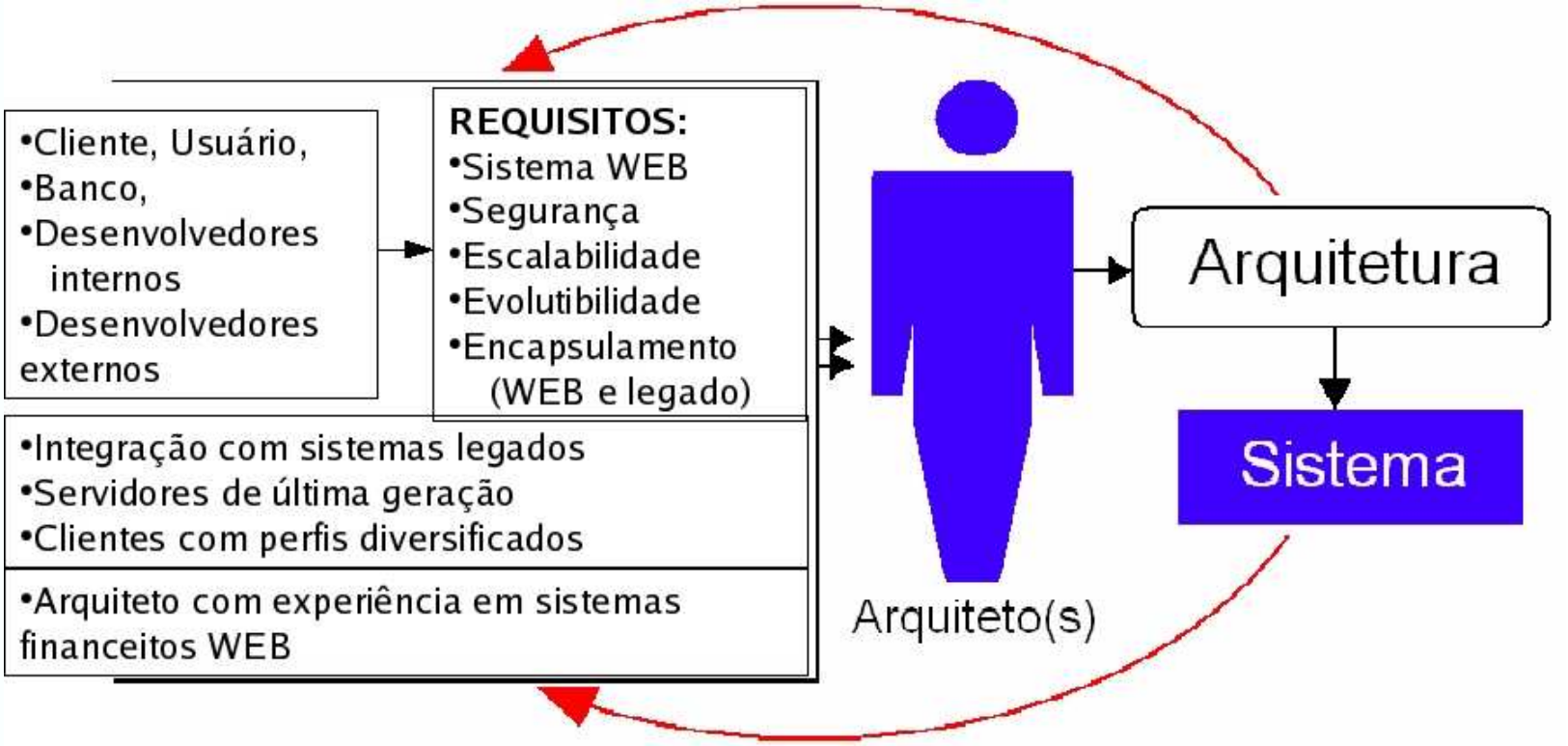
Fatores Tecnológicos e Sociais

- O sistema deve ser integrado ao sistema legado atual;
- O banco possui servidores de última geração, e *links* de fibra ótica com a Internet;
- Linguagem acessível a todas as classes sociais e idades;
- Acessibilidade para diferentes tipos de deficiências.

Ciclo Arquitetônico (I)

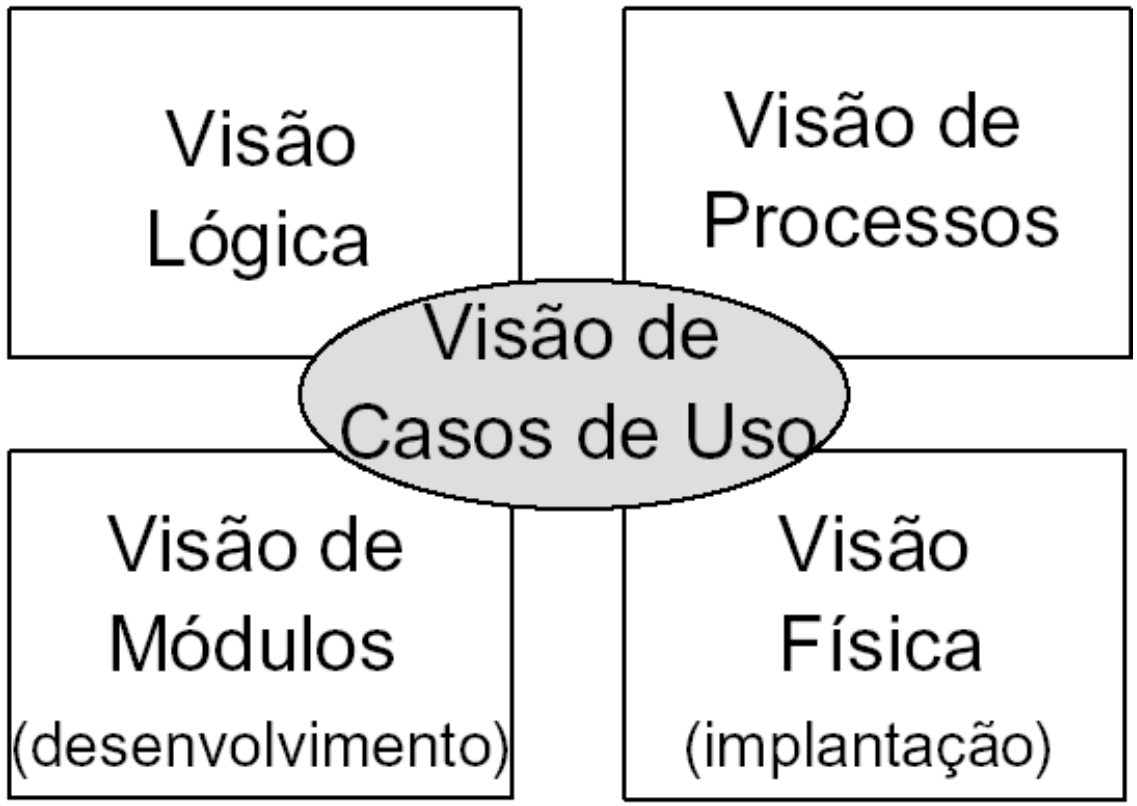
1. Clientes exigentes em relação aos requisitos não-funcionais;
2. O cliente e o usuário têm disponibilidade de interação com o arquiteto, caso necessário;
3. Apesar da maturidade da organização, ela não tem corpo técnico para a implementação na tecnologia desejada:
   - O desenvolvimento provavelmente será feito por terceiros (uma solução é componentizar os módulos).
4. As exigências do cliente e do usuário são factíveis, isto é, computacionalmente implementáveis;
5. O arquiteto responsável é uma pessoa experiente, capaz de expor várias decisões de projeto para as necessidades do cliente.

Ciclo Arquitetônico (II)



Visões Arquiteturais

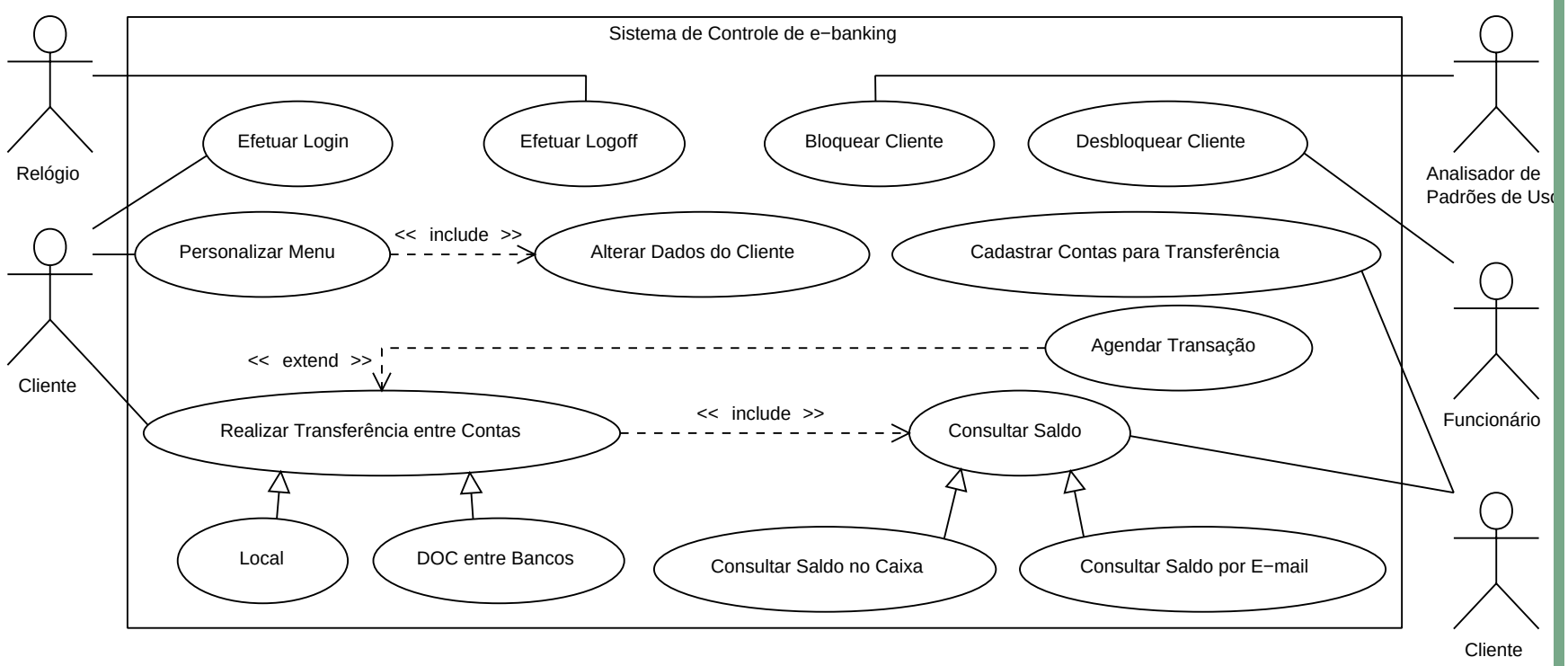
Visões Arquiteturais



Casos de Uso (I)

- Representam os requisitos do sistema (funcionais e não-funcionais);
- É útil para todas as outras visões;
- Casos de uso do sistema;
- Diagrama e Cenários.

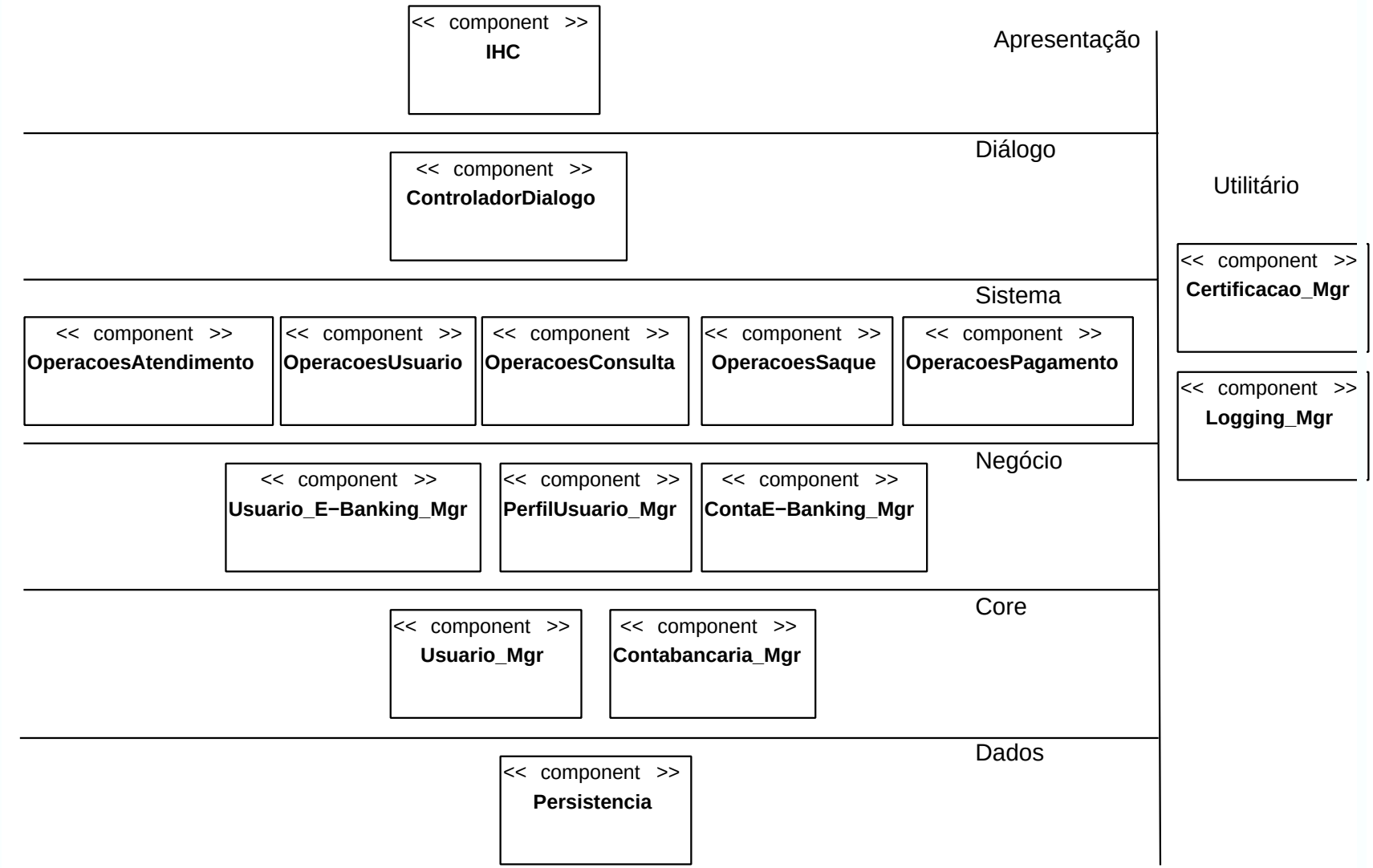
Casos de Uso (II)



Visão Lógica (I)

- Visão estrutural do sistema;
- Destaque aos módulos funcionais (componentes arquiteturais) e as dependências entre eles (conectores arquiteturais);

Visão Lógica (II)



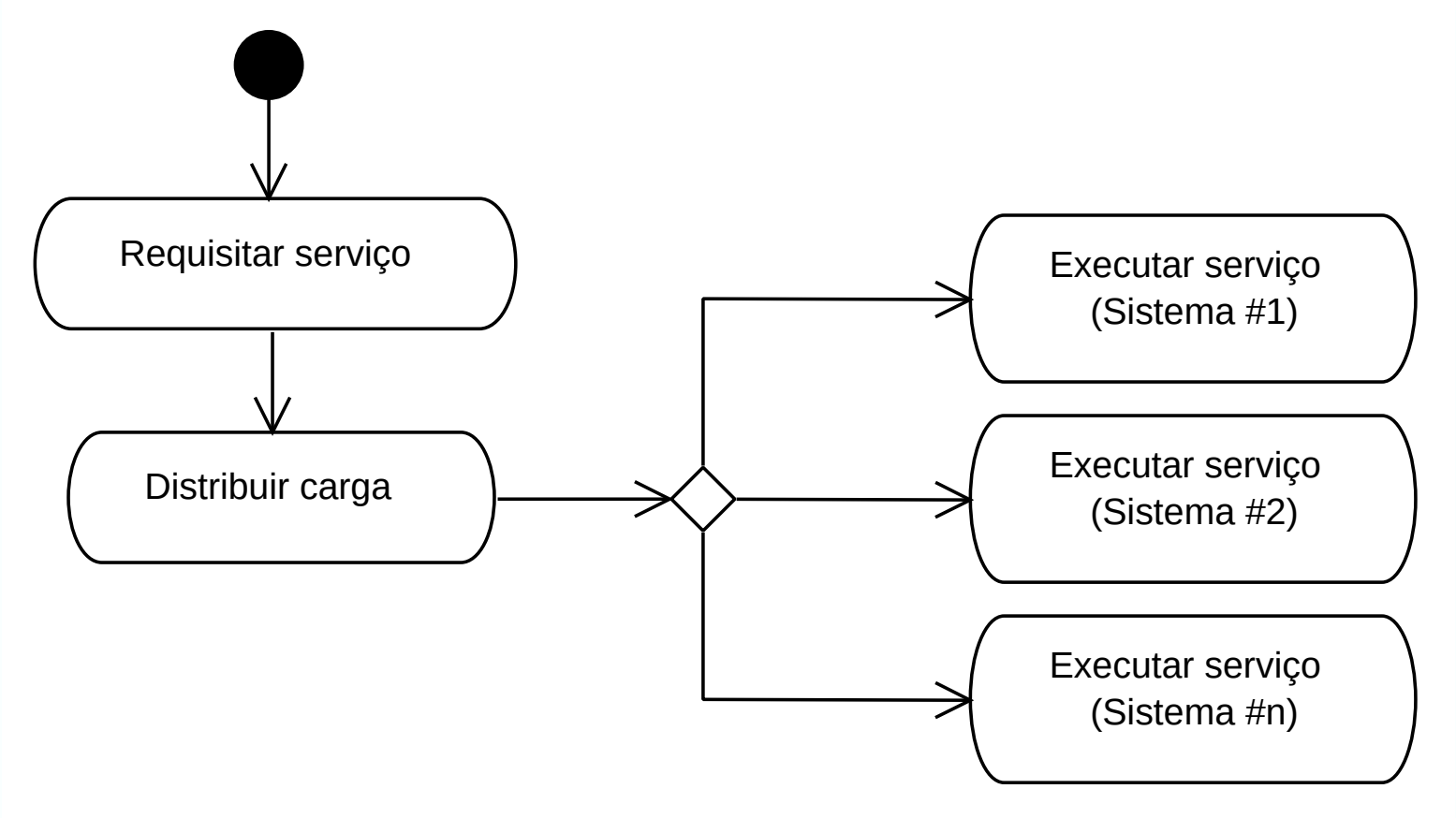
Visão Lógica (III)

- **IHC.** Devido à necessidade de um terminal para interação com o usuário;
- **ControladorDialogo.** Para flexibilizar a exibição dos menus (processamento entre a interface gráfica e o sistema);
- **OperacoesAtendimento.** Derivado dos casos de uso básicos: efetuar login, efetuar logoff, bloquear cliente, desbloquear cliente;
- **OperacoesUsuario.** Derivado dos casos de uso de usuário: personalizar menu, alterar dados do cliente;
- **OperacoesConsulta.** Derivado dos casos de uso de consulta: consultar saldo, consultar extrato;
- **OperacoesSaque.** Derivado dos casos de uso que envolvam redução do saldo: efetuar transferência, cadastrar conta para transferência;

- **OperacoesPagamento.** Derivado dos casos de uso de pagamento: pagar contas ;
- **Usuario_E-Banking_Mgr.** Derivado da entidade Cliente do modelo conceitual (descoplamento do e-banking);
- **PerfilUsuario_Mgr.** Derivado da entidade Perfil do modelo conceitual;
- **Conta_E-Banking_Mgr.** Derivado da entidade Conta do modelo conceitual (descoplamento do e-banking);
- **Usuario_Mgr.** Derivado da entidade Cliente do modelo conceitual;
- **Contabancaria_Mgr.** Derivado da entidade Conta do modelo conceitual;
- **Persistência.** Derivado da necessidade de se armazenar os dados de forma permanente;

Visão de Processos

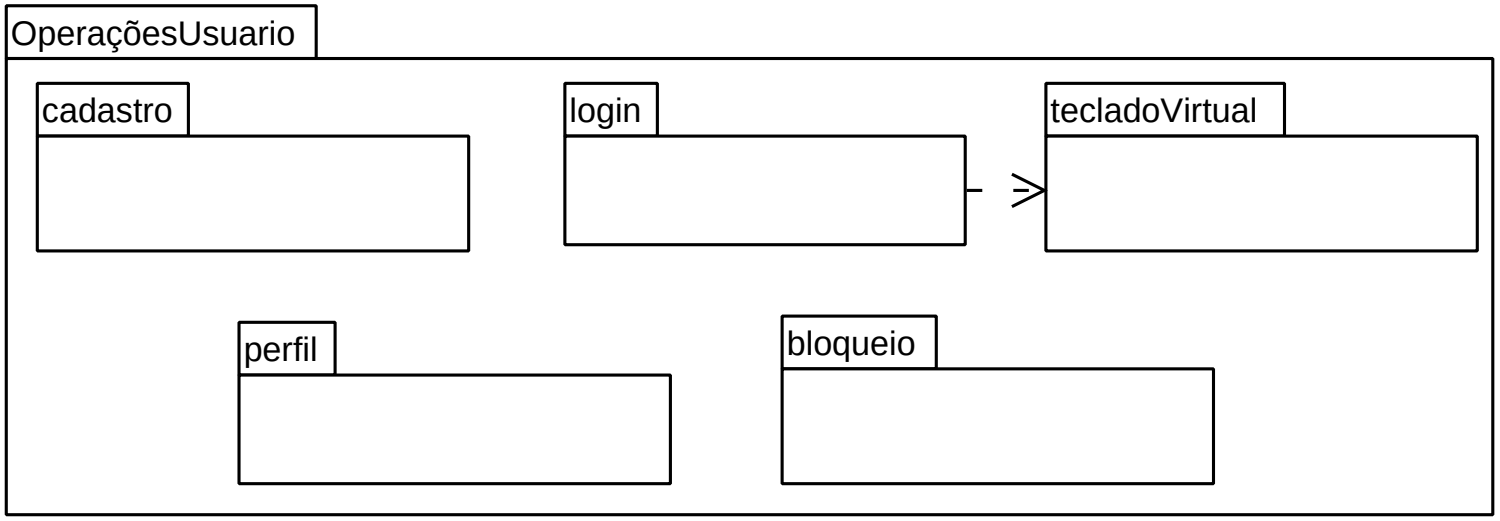
- Representação de aspectos dinâmicos do sistema, tais como eficiência (execução concorrente), escalabilidade (distribuição de carga), confiabilidade (julgamento de vários resultados), etc.



Visão de Desenvolvimento

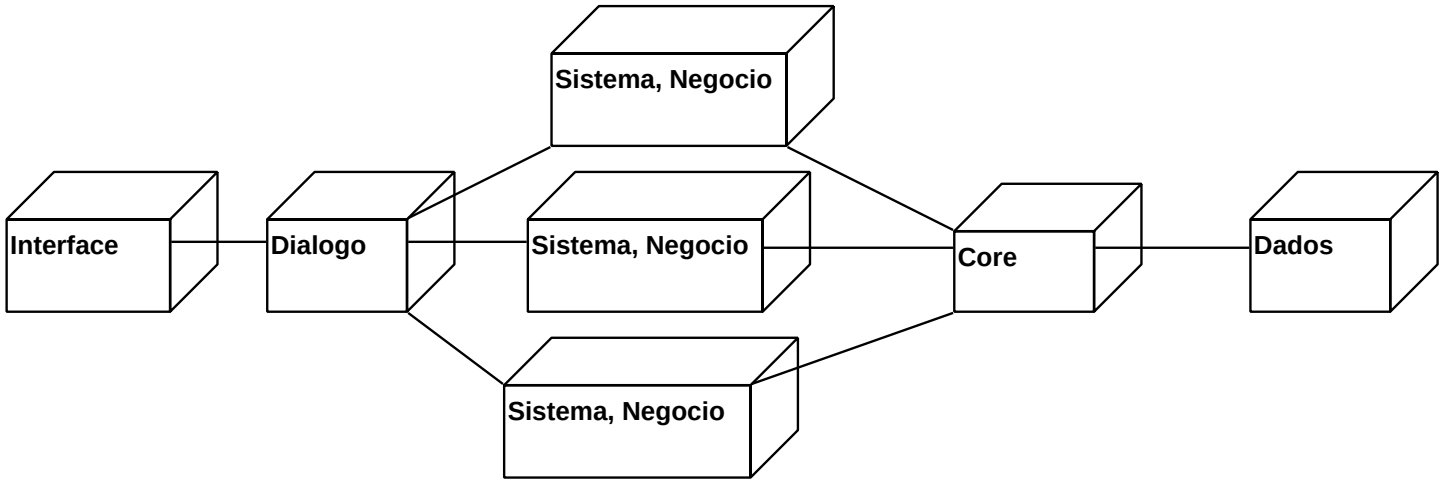
(componente `OperaçõesUsuario`)

- Componentes são vistos como unidades de código (granularidade menor que os componentes da visão lógica).

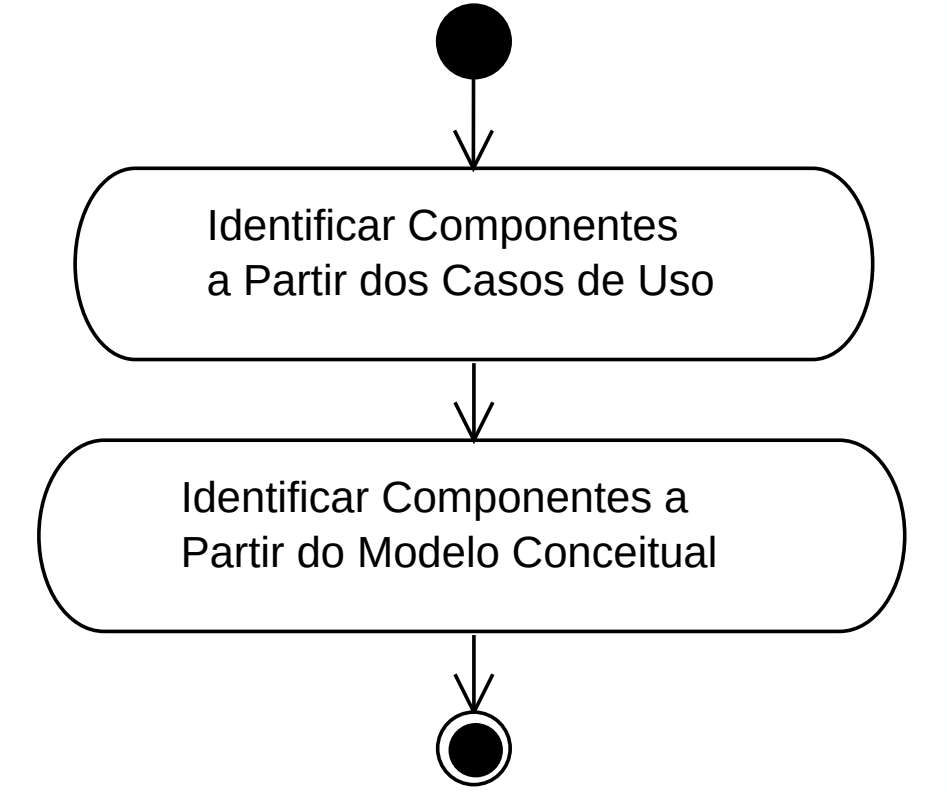


Visão Física

- Representação dos nós de hardware e distribuição física.



Identificação dos componentes da Visão Lógica



Identificar Componentes a Partir dos Casos de Uso

- Para cada uma das funcionalidades do sistema, deve existir um componente que a implemente;
- Um componente pode implementar mais de uma funcionalidade;
- As funcionalidades implementadas por um mesmo componente devem ser coesas entre si.

Identificar Componentes a Partir do Modelo Conceitual

- Primeiramente, deve-se destacar as entidades do modelo conceitual consideradas principais;
- A identificação das entidades principais pode se basear na percepção das entidades que guardam informações importantes para as funcionalidades do sistema (necessárias ou atualizadas pelos casos de uso);
- Para cada entidade principal, deve ser definido um componente para gerenciar a manipulação de informações relativas à entidade.

Exercício

1. Dado o seguinte enunciado de um sistema de Controle de Videolocadora:

   Um sistema de controle para uma videolocadora tem por objetivo automatizar o processo de locação e devolução de fitas de vídeo. Deve-se manter um controle dos cadastros de clientes e seus respectivos dependentes e também um controle sobre o acervo de fitas e sua movimentação.

   Os clientes podem executar operações que envolvem locação, devolução e compra de fitas. Um cliente *VIP* pode alugar um número ilimitado de fitas; caso contrário, o número máximo de fitas é limitado a três. O pagamento pode ser efetuado no ato da locação ou da devolução e pode ser feito em dinheiro, com cartão de crédito, ou através de "cheque-vídeo", que é comprado

antecipadamente. Se pagar com "cheque-vídeo", o cliente recebe um desconto especial. Em relação à devolução, caso a fita não seja devolvida no prazo previsto, uma multa será cobrada. Caso o cliente perca ou danifique uma fita alugada, ele deve ou pagar uma multa equivalente ao preço de uma fita nova, ou comprar uma nova fita para substituir a que foi danificada.

(a) Especifique o diagrama de casos de uso do sistema, identificando os relacionamentos entre os diversos casos de uso do sistema usando $<< include >>$ e $<< extend >>$.

(b) Especifique o modelo conceitual do sistema da vídeolocadora;

(c) Especifique o ciclo arquitetônico do sistema da vídeolocadora.

Estilos e Padrões Arquiteturais

Detalhamento da Etapa 3: Criação ou Seleção da Arquitetura do Sistema

Um estilo arquitetural...

- Caracteriza uma família de sistemas que são relacionadas pelo compartilhamento de propriedades estruturais e comportamentais (semântica);
- Define um vocabulário para os elementos da arquitetura: componentes e conectores;
- Além disso, apresenta ainda regras e restrições sobre a combinação dos componentes arquiteturais.

Motivos para Utilizar

- O estilo arquitetural de um sistema é útil para indicar a disposição geral dos seus elementos;
- O conhecimento dos estilos arquiteturais existentes pode simplificar a fase de projeto arquitetural de uma aplicação;
- Porém, a maioria dos sistemas grandes não seguem um único estilo arquitetural (são heterogêneos).

Exemplos de Estilos Arquiteturais

Arquitetura Centrada em Dados (I)

- Componentes:

  Unidades de processamento que materializa os requisitos funcionais do sistema.

- Conectores:

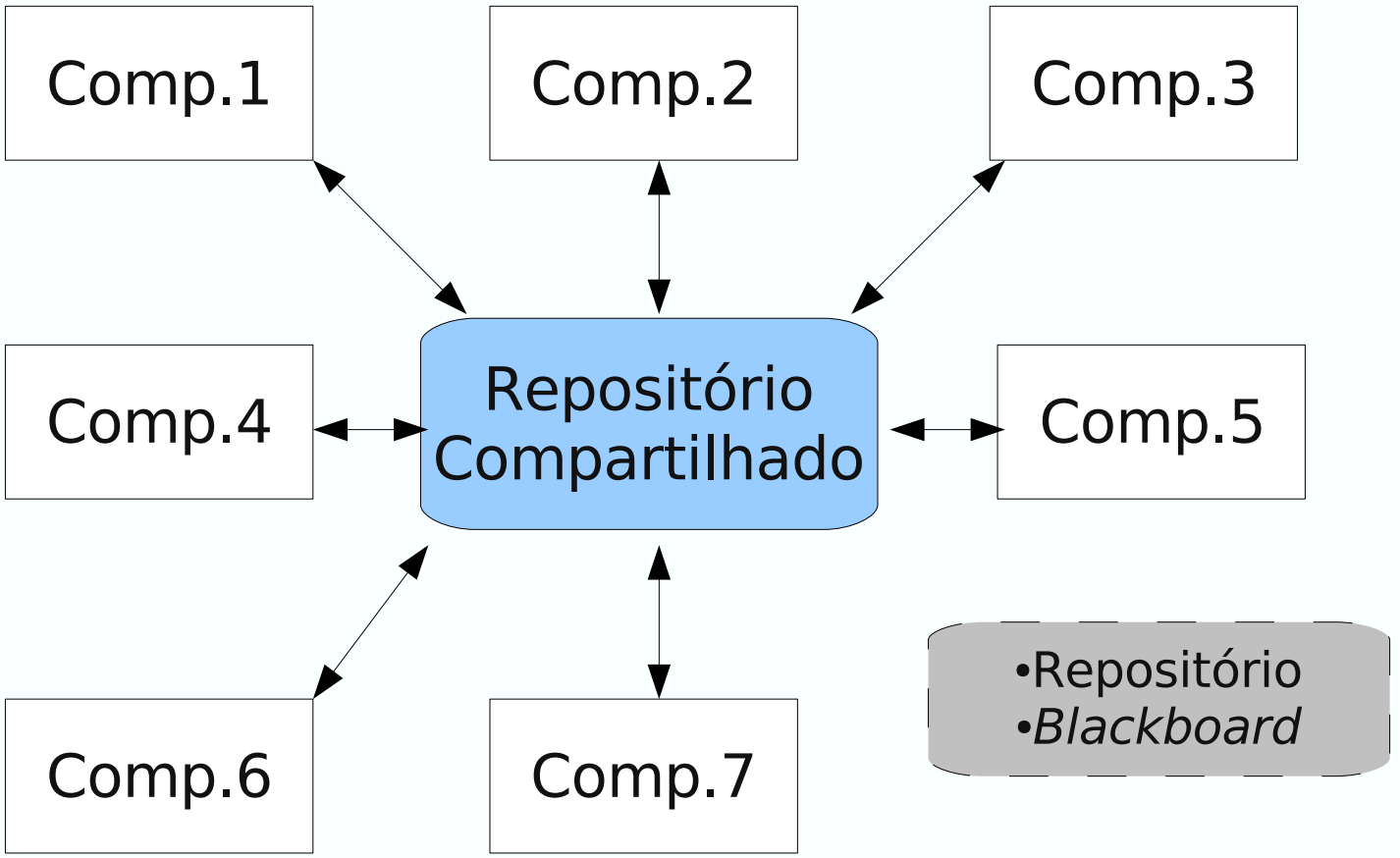
  Repositórios de dados, que é compartilhado por todos os componentes.

- Restrições
  - Os componentes só se comunicam através do compartilhamento de dados.

Arquitetura Centrada em Dados (II)

- Sub-sistemas necessitam trocar dados entre si. Essa troca pode acontecer de duas maneiras:
  - Compartilhamento através de um banco de dados ou repositório comum a todos os componentes;
  - Cada componente mantém o seu próprio banco de dados e transfere os dados explicitamente uns aos outros.
- Quando é necessário compartilhar um grande volume de dados, o modelo de repositório centralizado é mais utilizado. Existem duas variações mais conhecidas desse estilo arquitetural:
  - Repositório $\rightarrow$ depósito de dados "passivo";
  - *Blackboard* $\rightarrow$ depósito de dados "ativo";

Arquitetura Centrada em Dados (III)



Arquitetura Centrada em Dados (IV)

- Vantagens:
  - Proporciona um modo eficiente de compartilhamento de um grande volume de dados;
  - Os componentes não necessitam conhecer a maneira como os dados são produzidos ou gerenciado (*backup*, segurança, etc. são transparentes).
  - O modelo de compartilhamento pode ser definido unicamente pelo repositório.

Arquitetura Centrada em Dados (V)

- Desvantagens:
  - Sub-sistemas necessitam lidar com o mesmo modelo de dados (o do repositório). É um compromisso inevitável;
  - A evolução dos dados é uma atividade difícil e custosa;
  - Políticas de gerenciamento específicas são mais difíceis de serem implementadas;
  - É difícil de implementar distribuição de forma eficiente.

Arquitetura de Componentes Independentes

- Cada componente é autônomo, isto é, um não exerce controle sobre o outro;
- Redução do acoplamento entre os componentes;
- Existem dois tipos principais:
  1. Arquitetura de Processos Comunicantes: componentes independentes para atingir escalabilidade ("cliente-servidor");
  2. Arquitetura Baseada em Eventos: cada componente especifica claramante os eventos que podem lançar e tratar.

Arquitetura Cliente-Servidor (I)

- Componentes:

  São agrupados de acordo com a sua localização física: cliente ou servidor.

- Conectores:

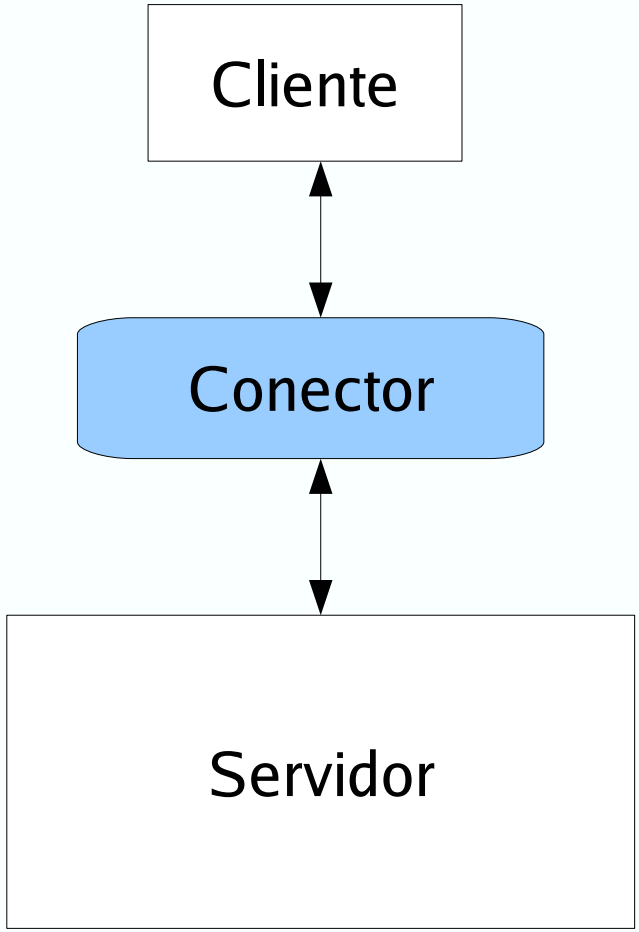
  Pontes de comunicação entre os componentes clientes e os componentes servidores. Normalmente implementam algum serviço de comunicação remota.

- Restrições
  - Os clientes não são unidades autônomas, dependendo explicitamente dos servidores.

Arquitetura Cliente-Servidor (II)

- É um estilo voltado para sistemas distribuídos, que mostra como os dados e o processamento é dividido entre um conjunto de componentes;
- Um conjunto de servidores oferecem serviços específicos, tais como impressão, gerenciamento de dados, etc;
- Um conjunto de clientes podem executar esses serviços;
- A comunicação entre eles se dá normalmente através de uma rede remota.

Arquitetura Cliente-Servidor (III)



Arquitetura Cliente-Servidor (IV)

- O conector pode ser representado de forma explícita ou implícita;
- Quando tornar o conector explícito?

  - protocolo, requisitos de qualidade relativos à comunicação.

- Vantagens:
  - A distribuição dos dados é feita de uma forma direta e natural;
  - Torna o uso de redes de computadores uma atividade efetiva. Sua popularização reduziu o valor dos hardwares de comunicação;
  - É escalável, isto é, fácil de adicionar novos servidores ou atualizar os já existentes.

Arquitetura Cliente-Servidor (V)

- Desvantagens:
  - Não apresenta um modelo de compartilhamento de dados, então cada componentes normalmente organiza os dados da sua maneira. O compartilhamento de dados pode se tornar ineficiente;
  - Cada servidor implementa seu próprio controle, podendo haver redundância de esforço;
  - Não há um registro central de nomes e serviços. Por isso a localização de um servidor ou serviço específico pode ser uma tarefa difícil.

Arquitetura Baseada em Eventos (I)

- Componentes:

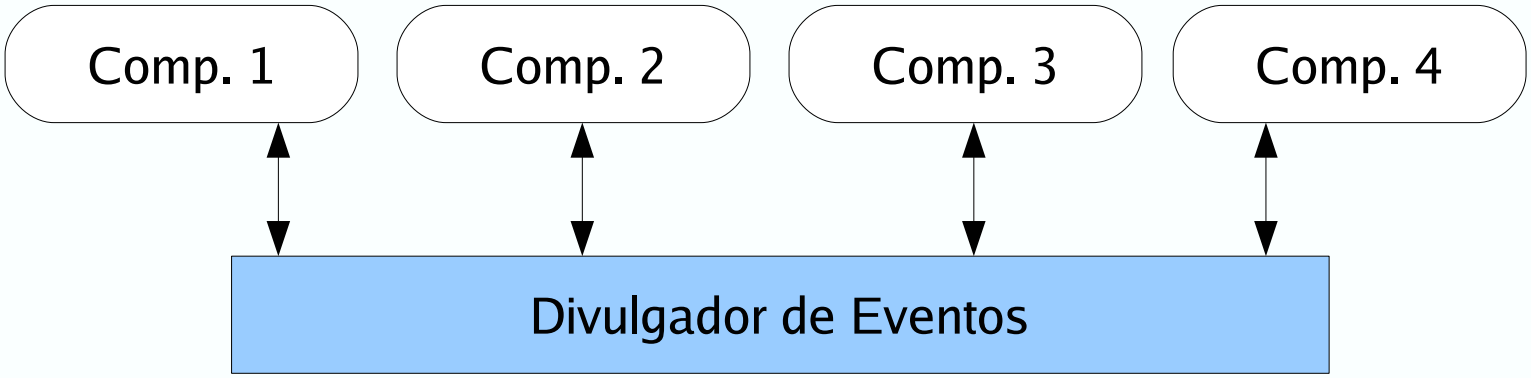
  Unidades de processamento de dados que produzem e tratam eventos que acontecem no sistema.

- Conectores:

  Elementos que disseminam os eventos entre os componentes do sistema.

- Restrições
  - A comunicação entre os componentes deve acontecer exclusivamente através dos eventos do sistema;
  - Os conectores apenas comunicam eventos, não executam transformação de dados.

Arquitetura Baseada em Eventos (II)



Arquitetura Baseada em Eventos (III)

- Tipos de conectores:
  - **Modelo implícito (*broadcast*)**. Um evento é enviado em *broadcast* para todos os componentes. Qualquer componente que pode tratar o evento pode fazê-lo. Um exemplo desse tipo de arquitetura é o padrão *observer*.
  - **Modelo explícito (dirigido por interrupção)**. Utilizado em sistemas de tempo real, onde interrupções são detectadas por um tratador de interrupções e repassadas para outros componentes que as processam.

Modelo Implícito

- A política de controle não é embutida no evento ou no componente cliente da requisição. Os componentes decidem quais os eventos que eles se interessam;
- Os componentes se registram como interessados em determinados eventos. Quando ocorre algum desses eventos, o controle é transferido a todos os componentes que o tratam;
- Porém, os componentes que lançam um evento não conhecem os possíveis tratadores;
- É efetivo para integrar componentes em diferentes computadores de uma mesma rede.

Modelo Explícito

- Existem tipos conhecidos de interrupções. Para cada um deles são atribuídos os possíveis tratadores;
- Cada tipo é associado a uma localização de memória e a seleção dos tratadores é normalmente implementada no hardware (desempenho crítico);
- Possibilita respostas rápidas, porém são mais complexos e difíceis de validar;
- São utilizados para desenvolver sistemas de tempo real, que necessitam de respostas rápidas a eventos específicos.

Arquitetura Baseada em Eventos (IV)

- Vantagens:
  - Indicada para a modelagem de sistemas concorrentes, uma vez que cada uma das unidades são autônomas e os eventos produzidos podem ser tratados concorrentemente;
  - A autonomia das unidades e a baixa dependência entre elas (dependência indireta através dos eventos), a torna escalável e fácil de evoluir (adicionar tratadores para novos eventos).
- Desvantagens:
  - É difícil ter um controle efetivo dos eventos que circulam nela;
  - É difícil alterar o conjunto de eventos que fluem no sistema, tendo em vista o número de ajustes necessários em todos os componentes que os tratam.

Arquitetura Call-Return (I)

- Componentes:

  Unidades de processamento que materializam os requisitos funcionais do sistema.

- Conectores:

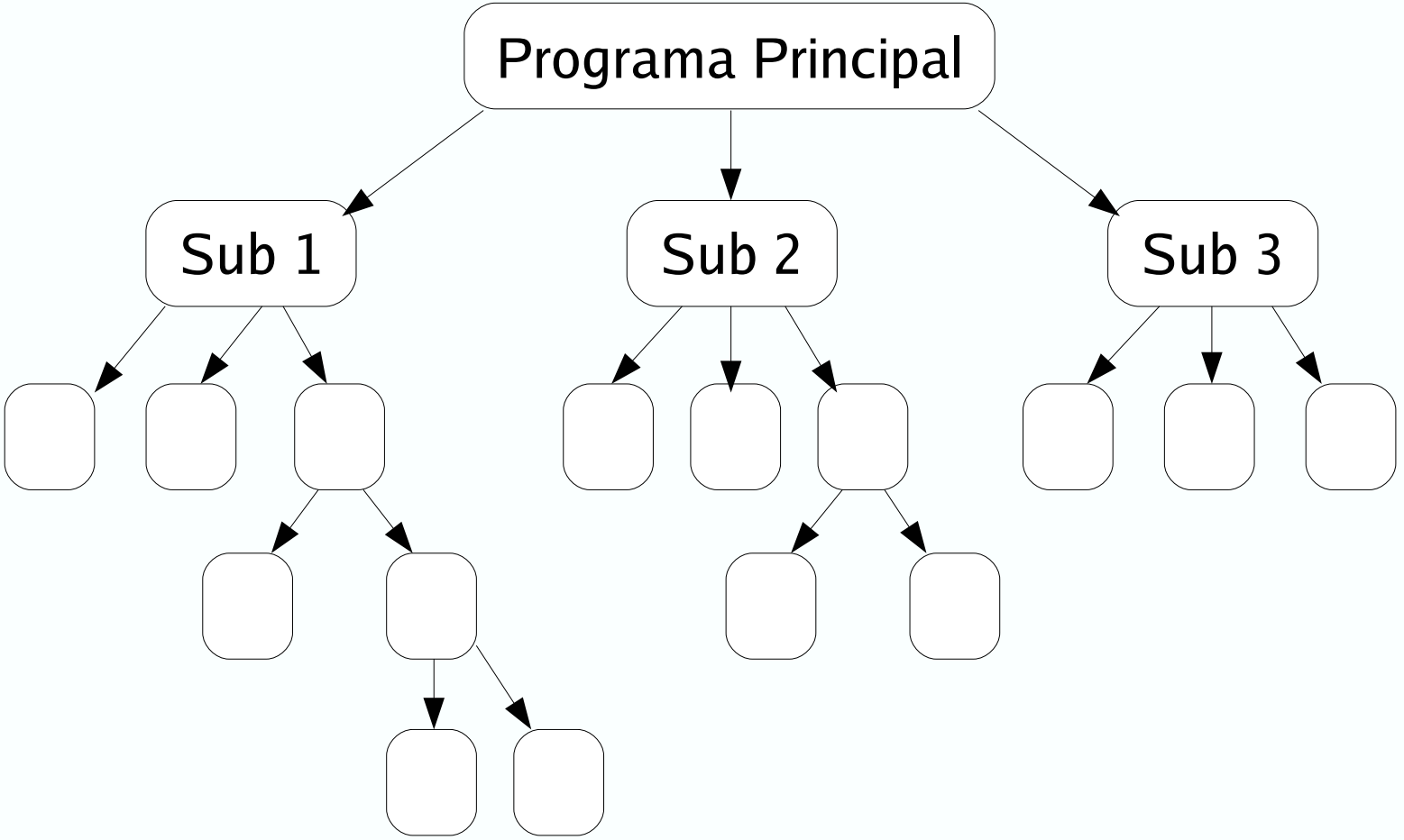
  Componentes se comunicam diretamente entre si através de chamadas.

- Restrições
  - Nenhuma.

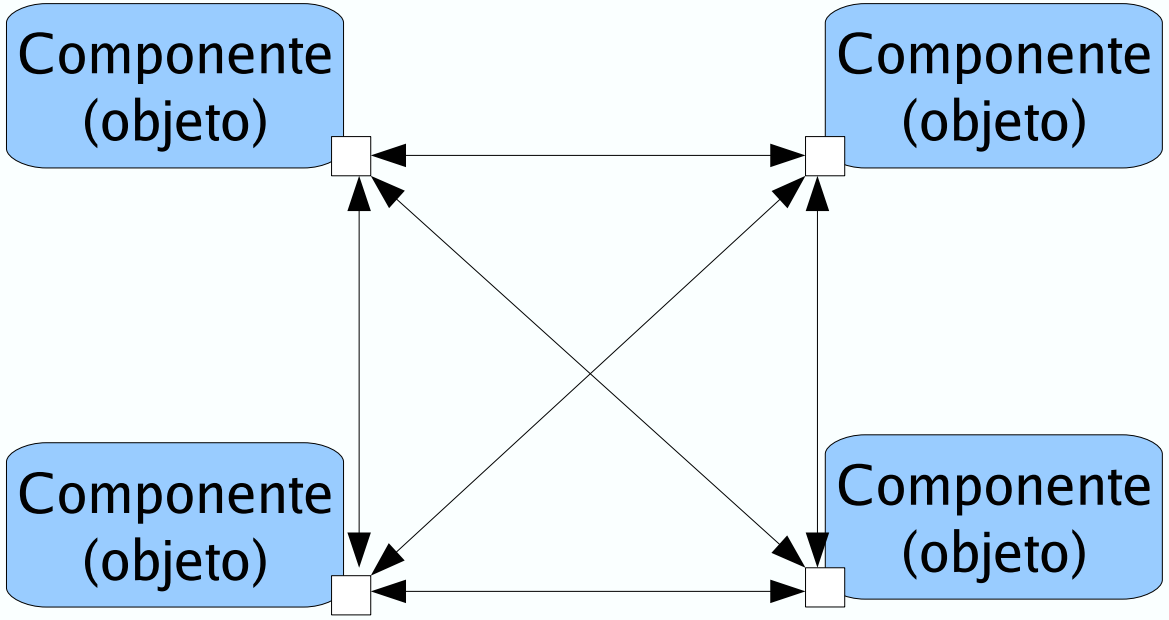
Arquitetura Call-Return (II)

- Estrutura o sistema em um conjunto de componentes fracamente acoplados entre si, com interfaces bem definidas;
- Um componente de controle possui a responsabilidade de gerenciar a execução de outros componentes;
  - Modelo de subrotinas *top-down*, onde o controle é exercido pelo topo da hierarquia de sub-rotinas;
  - É aplicável a sistemas seqüenciais.
- Quando implementado, um modelo de controle é utilizado para coordenar a seqüência de execução das operações;
- Um tipo conhecido de arquitetura *call-return* é a orientada a objetos, onde cada objeto é uma unidade autônoma e a comunicação é feita por requisição e resposta, através da troca de mensagens.

Programa Principal-Subrotinas



Arquitetura Orientada a Objetos (I)



Arquitetura Orientada a Objetos (II)

- Vantagens:
  - Componentes são fracamente acoplados, logo a sua implementação pode ser modificada sem afetar outros objetos;
  - Melhor desempenho, uma vez que a ausência de restrições de comunicação facilita a estruturação dos componentes do sistema e reduz a necessidade de indireção entre as chamadas;
  - Os componentes podem refletir entidades do mundo real;
  - As linguagens de programação OO são amplamente utilizadas;

Arquitetura Orientada a Objetos (III)

- Desvantagens:
  - Alterações nas interfaces podem causar problemas e, por isso, entidades complexas dificilmente são representadas como objetos;
  - A ausência de restrições de comunicação dificulta a evolução do sistema, como por exemplo, a substituição de componentes.

Arquitetura em Camadas (I)

- É um tipo de call-return;
- Componentes:

  São camadas com um alto nível de abstração.

- Conectores:

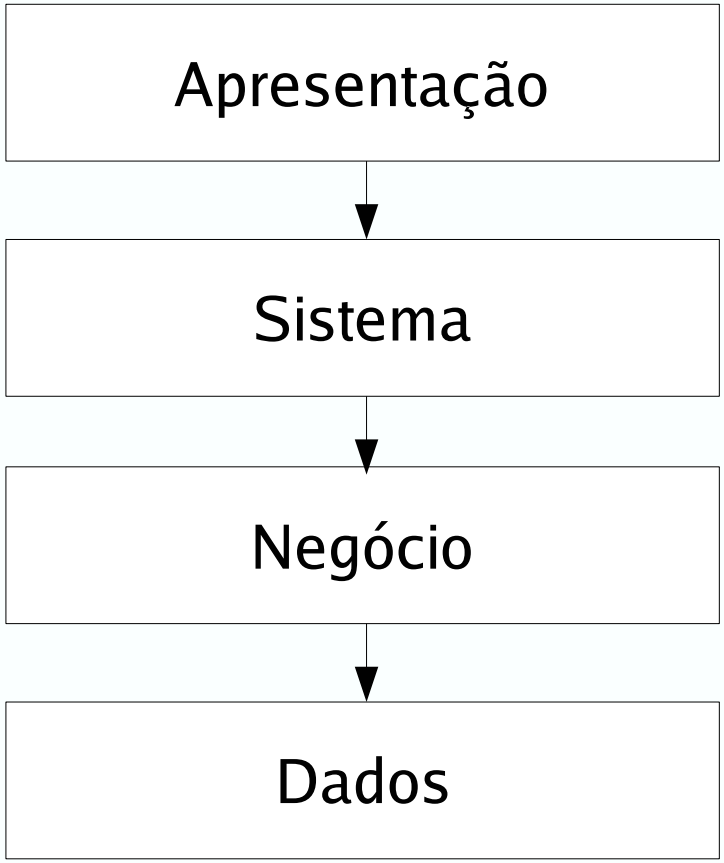
  Fluxos de comunicação entre camadas.

- Restrições
  - Uma camada somente **recebe requisição** de uma camada imediatamente superior;
  - Uma camada somente **envia requisição** para uma camada imediatamente inferior.

Arquitetura em Camadas (II)

- Utilizada para modelar a interação entre componentes;
- Organiza o sistema em um conjunto de camadas, cada camada oferece um conjunto próprio de serviços e só depende da camada imediatamente inferior;
- Dá suporte ao desenvolvimento incremental de componentes em várias camadas. Quando a interface de uma camada sofre alguma alteração, apenas a camada adjacente é afetada.

Arquitetura em Camadas (III)



Arquitetura em Camadas (IV)

- Vantagens:
  - Melhor controle da complexidade do sistema, pelo fato de uma camada abstrair todos os seus detalhes internos;
  - Facilidade de evolução do sistema. O fato de uma camada só conhecer a camada imediatamente inferior a ela, reduz a dependência entre os componentes do sistema;
- Desvantagens:
  - A comunicação restrita pode comprometer o desempenho do sistema, aumentando o número de indireções.

Arquiteturas de Fluxos de Dados

- O sistema é visto como uma série de transformações complementares;
- Possui o objetivo de maximizar a reutilização de funcionalidades pequenas e muito bem definidas;
- Existem dois tipos principais:
  1. Arquitetura *Pipes and Filters*: transformação incremental da informação (saída de um $\rightarrow$ entrada do próximo);
  2. Arquitetura em *Batch*: cada componente é independente e a única restrição entre eles é que a execução deve seguir uma seqüência pré-definida.

Arquitetura Pipes and Filters (I)

- Componentes:

  Unidades de processamento de dados ("filtros de dados").

- Conectores:

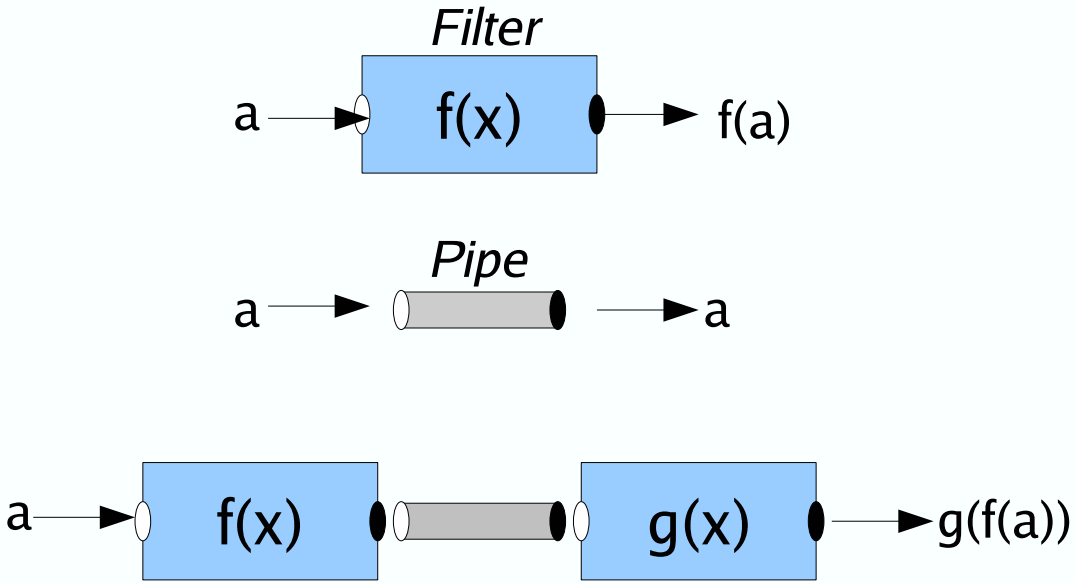
  Dutos (*pipes*) de comunicação entre os filtros.

- Restrições
  - Apenas os filtros podem transformar os dados;
  - Os dutos apenas efetuam transporte entre filtros.

Arquitetura Pipes and Filters (II)

- As transformações funcionais processam os dados fornecidos como entrada e geram dados de saída;
- Um exemplo clássico de uma aplicação bem sucedida de uma arquitetura *pipes and filters* é o *shell* do UNIX: `ls | grep ''objetos''`
- Variantes dessa abordagem são muito comuns. A ocorrência de transformações seqüenciais caracteriza um tipo particular desse estilo: a seqüência *batch*, que é utilizada em sistemas de processamento de dados;
- Não é indicado para sistemas que necessitam trocar informações (ida e volta).

Arquitetura Pipes and Filters (III)



Arquitetura Pipes and Filters (IV)

- Vantagens:
  - Dá suporte à reutilização das transformações;
  - Oferece uma organização intuitiva, útil na comunicação com os interessados (*stakeholders*);
  - É fácil adicionar novas transformações à arquitetura;
  - É relativamente simples de implementar, tanto no caso de sistemas seqüenciais, quanto concorrentes.
- Desvantagens:
  - Necessita de um formato comum para transferência de dados nos *pipelines*;
  - A implementação de sistemas que interagem através de eventos é muito dificultada.

Arquitetura de Máquina Virtual (I)

- Componentes:

  Unidades de processamento que materializam os requisitos funcionais do sistema.

- Conectores:

  Elementos responsáveis pela comunicação entre os componentes.

- Restrições
  - Existência de dois papéis bem definidos: tradução e execução.

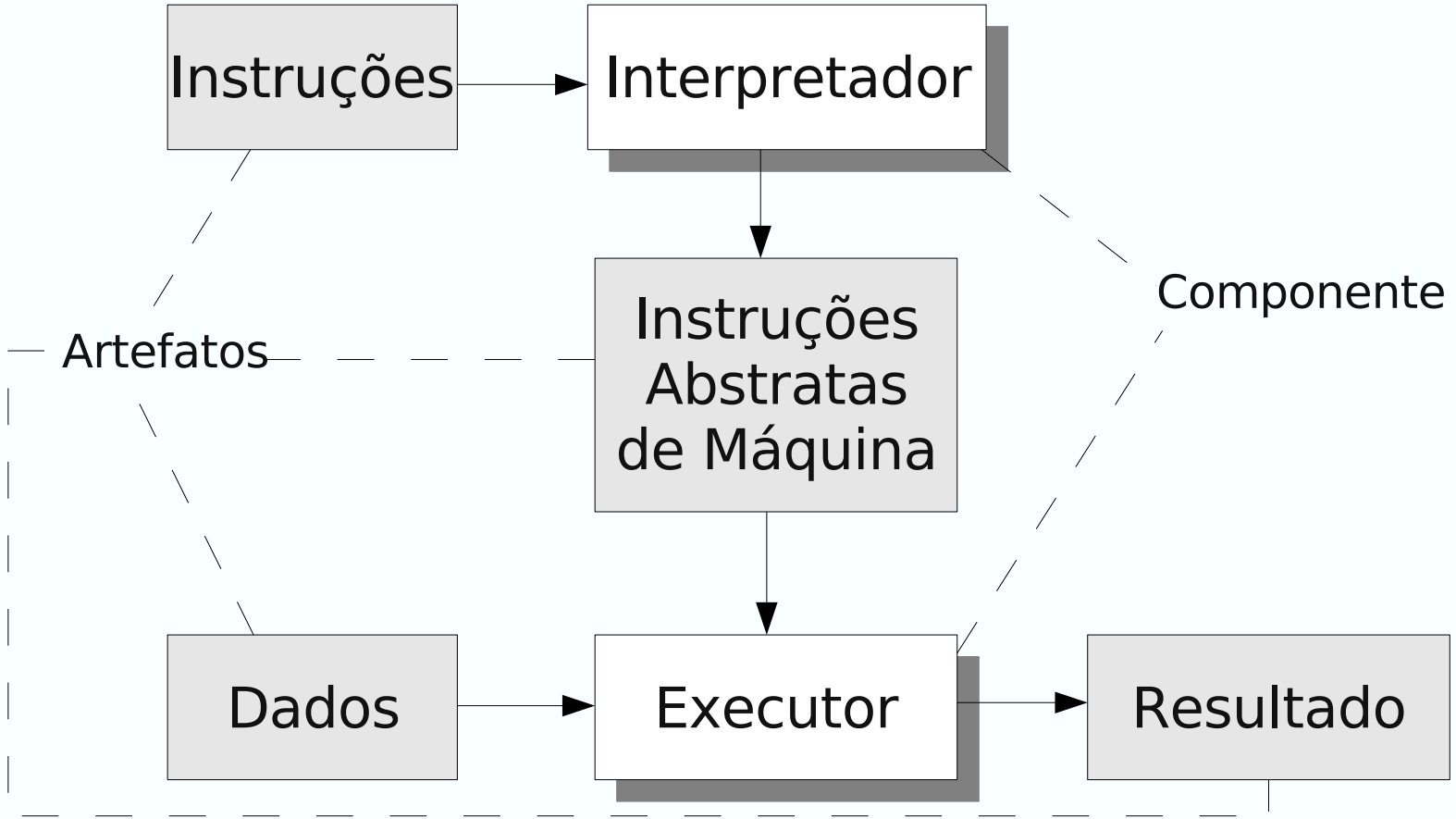
Arquitetura de Máquina Virtual (II)

- É indicado quando é necessário executar atividades intermediárias, antes da execução da funcionalidade propriamente dita;
- Proporciona uma maior independência dos componentes que implementam os requisitos funcionais do sistema (desacoplamento).
- Esse estilo também é conhecido como "máquina abstrata";

Arquitetura de Máquina Virtual (III)

- Possui duas variações principais:
  - Interpretador;
  - Baseado em regras.
- Exemplos de aplicações reais:
  - Máquina virtual Java (interpretador);
  - *Framework* de execução .NET (interpretador);
  - Ferramenta de execução de regras DROOLS (baseado em regras);
  - Sistemas especialistas (baseado em regras).

Arquitetura de Máquina Virtual (IV)



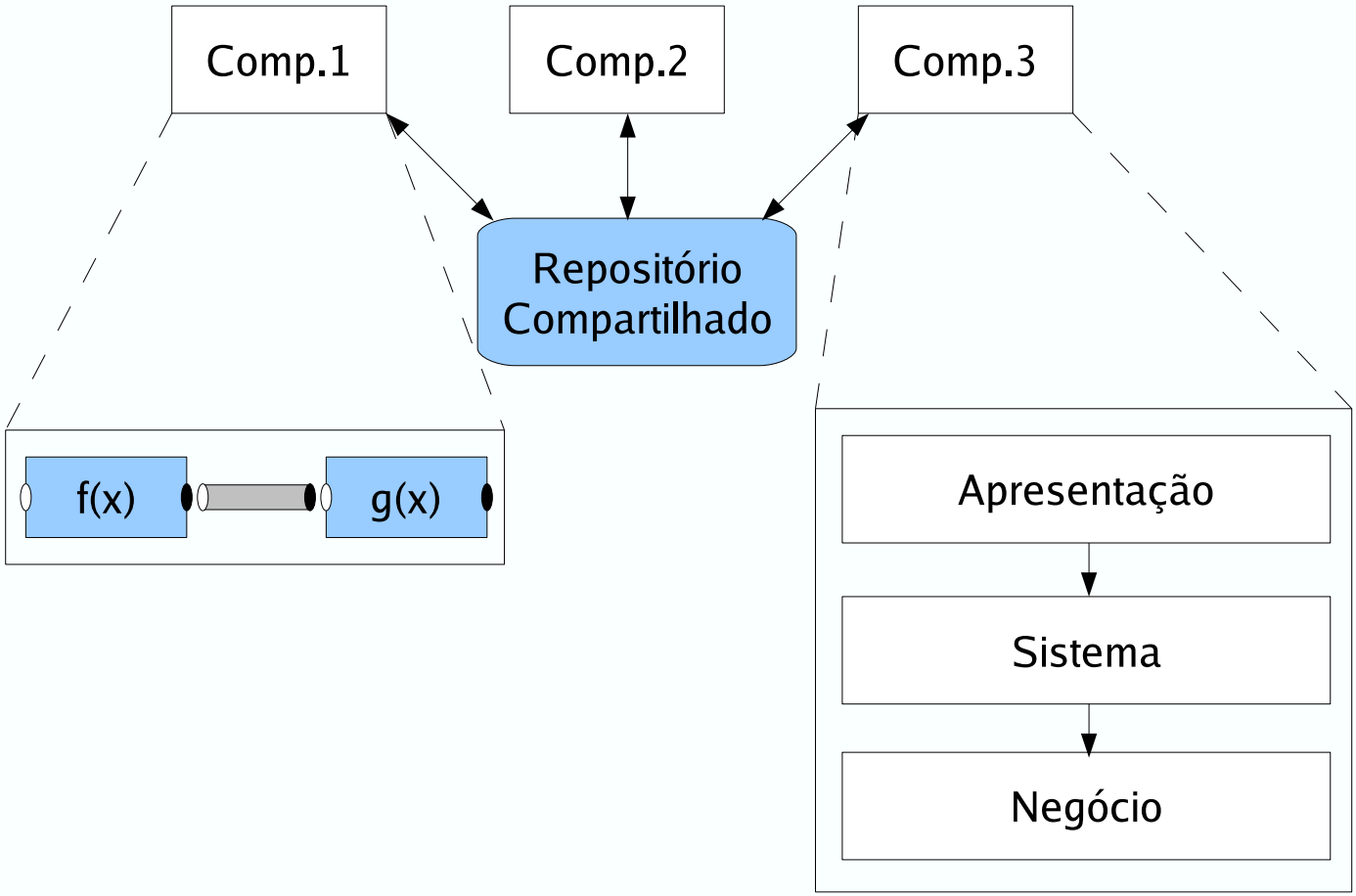
Arquitetura de Máquina Virtual (V)

- Vantagens:
  - Desacoplamento entre o núcleo de execução e o que será executado (entrada);
  - Facilidade de evolução do sistema;
  - Facilidade de adaptação do sistema para várias plataformas;
- Desvantagens:
  - O desempenho pode ser comprometido (passos intermediários de execução);
  - A complexidade do sistema aumenta, tendo em vista a adição de novos passos de processamento (tradução).

Arquiteturas Heterogêneas (I)

- Combina vários estilos:
  - Cada componente da arquitetura pode seguir um estilo específico.
- Na prática, é a mais utilizada, pois possibilita a reutilização de qualquer componente, independentemente da sua arquitetura interna;
- É a melhor maneira combinar as vantagens e contornar as desvantagens de um estilo arquitetural específico. Por exemplo:
  - Uma arquitetura abstrata estruturada em camadas;
  - Módulos com desempenho crítico estruturados com *call-return*;
  - Módulos reutilizados seguindo o estilo *pipes and filters*, etc.

Arquiteturas Heterogêneas (II)



Padrões Arquiteturais

Padrões Arquiteturais (I)

*"Um padrão arquitetural (*architectural pattern*) descreve um esquema básico de organização para estruturar sistemas de software, definindo um conjunto de componentes, suas responsabilidades, e estabelecendo regras e diretrizes para organizar o relacionamento entre esses componentes."*
[Buschman, 1996]

Padrões Arquiteturais x Estilos Arquiteturais

"By architectural style we mean a set of design rules that identify the kinds of components and connectors that may be used to compose a system or subsystem, together with local or global constraints on the way the composition is done. ... the patterns community concentrates on documenting proven solutions, including architectural styles, in the context of the specific kinds of problem for which each solution is useful"
[Mary Shaw, 1997]

Padrões Arquiteturais (II)

- Arquiteturas (**inclusive estilos arquiteturais**) voltadas para solucionar problemas específicos;
- Facilitam a reutilização de soluções arquiteturais bem sucedidas;
- Auxiliam na decomposição do sistema de forma ordenada, evitando a proliferação de componentes;
- Facilitam a integração de diferentes estilos arquiteturais.

Padrões Arquiteturais (III)

- Segundo Christopher Alexander:

  "cada padrão descreve um problema que ocorre recorrentemente em nosso ambiente e então descreve uma solução para o problema, de tal maneira que você pode usar essa solução um milhão de vezes, sem nunca utilizá-la duas vezes da mesma maneira".

Elementos de um Padrão

- Um padrão (arquitetural, de análise, de projeto, etc.) é descrito através das seguintes informações:
  1. Nome;
  2. Problema;
  3. Solução;
  4. Conseqüências;
  5. Exemplos (opcional).

A Escolha de Um Padrão Arquitetural

- Um padrão arquitetural relaciona um estilo arquitetural a um grupo de sistemas (ou requisitos não-funcionais) que ele é indicado;
- A especificação da arquitetura pode consistir na escolha de um padrão (ou estilo) arquitetural adequado ao sistema;
- Seguir um determinado padrão arquitetural significa adotar estrutura conhecida e adequada para o sistema específico;
- Sistemas de informação (sistemas comerciais em geral) possuem normalmente uma arquitetura em camadas, onde as camadas correspondem aos seus componentes.

Os 8 Padrões Arquiteturais de Buschman

- Camadas (*Layers*);
- *Pipes and Filters*;
- Quadro Negro (*Blackboard*);
- Broker (ou *broadcast*);
- MVC (*Model-View-Controller*);
- PAC (*Presentation-Abstraction-Control*);
- *Microkernel*;
- Reflexão (*Reflection*);

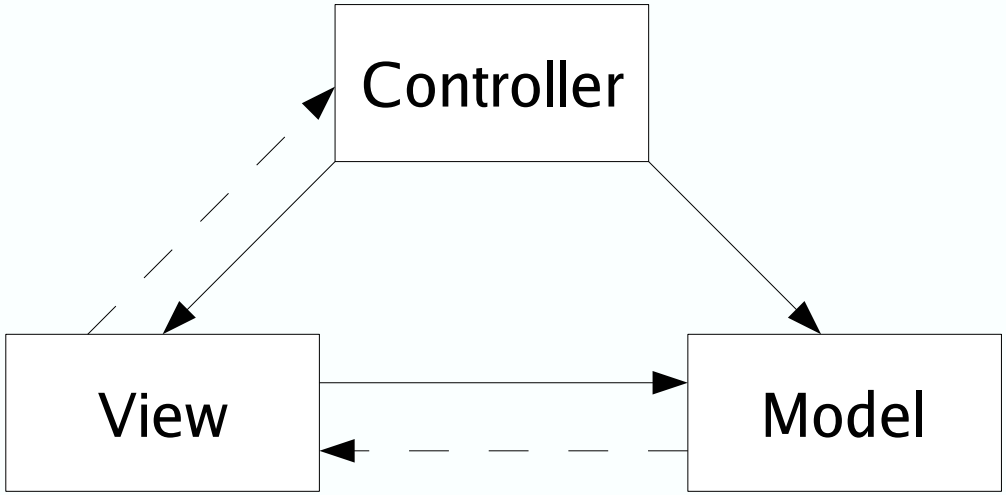
O Padrão Arquitetural MVC (I)

- **Nome:** *Model-View-Controller* (MVC).
- **Problema:** A interface com o usuário (I.U.) pode mudar com freqüência. O banco de dados (B.D.) é compartilhado por diversas aplicações. Não é desejável rever toda a aplicação quando a I.U. mudar, ou ter que alterar o B.D. quando uma regra de negócio for alterada.

O Padrão Arquitetural MVC (II)

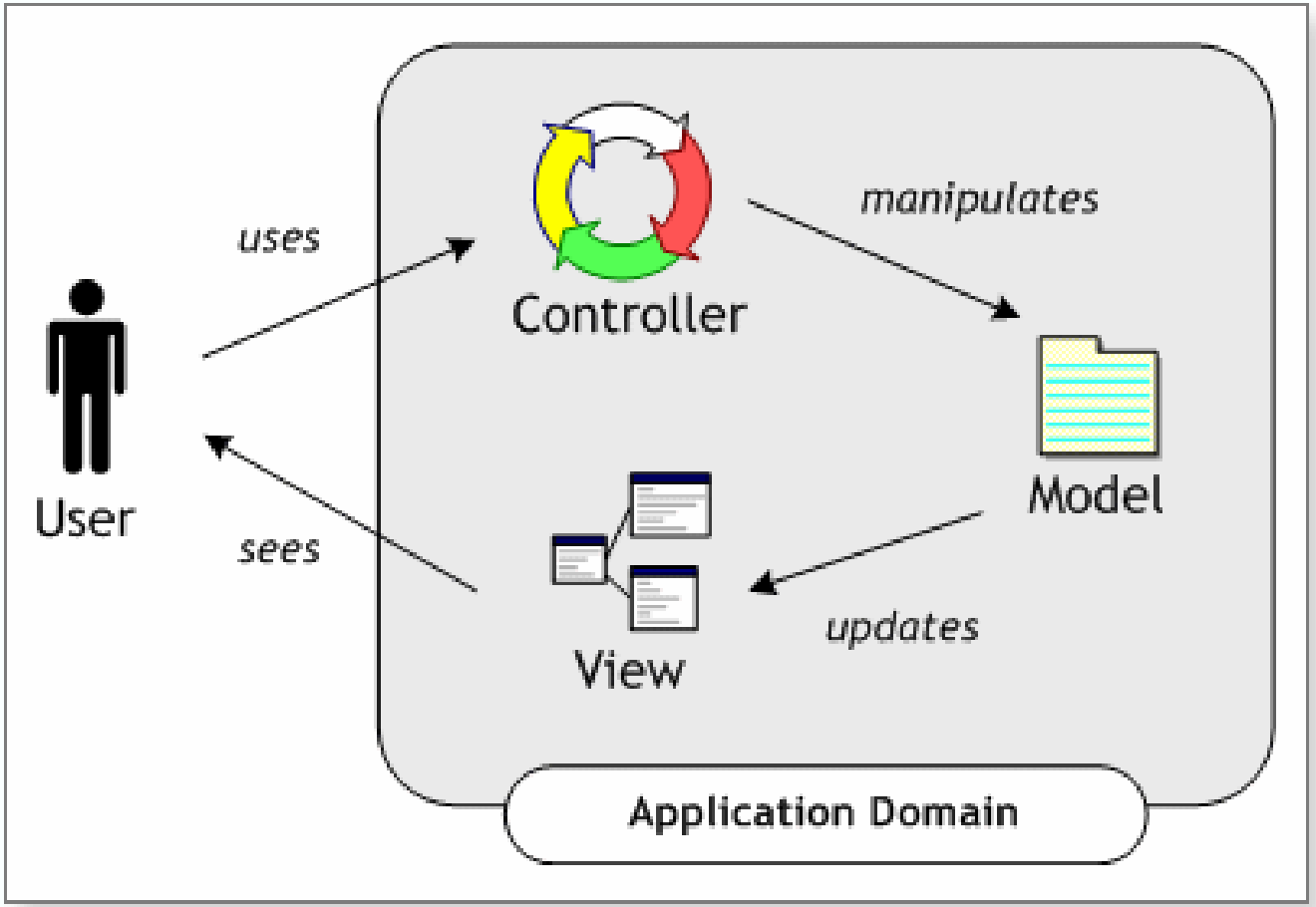
- **Solução:** Dividir as aplicações em três módulos com responsabilidades distintas:
  - *Model*: armazena os dados;
  - *View*: recebe, recupera e exibe os dados;
  - *Controller*: gerencia os eventos; coordena *model* e *view*.
- **Conseqüências:** Desacoplamento entre os módulos do sistema, através da separação clara da responsabilidade entre modelo, visão e controle da lógica do negócio.

Estrutura do Padrão Arquitetural MVC

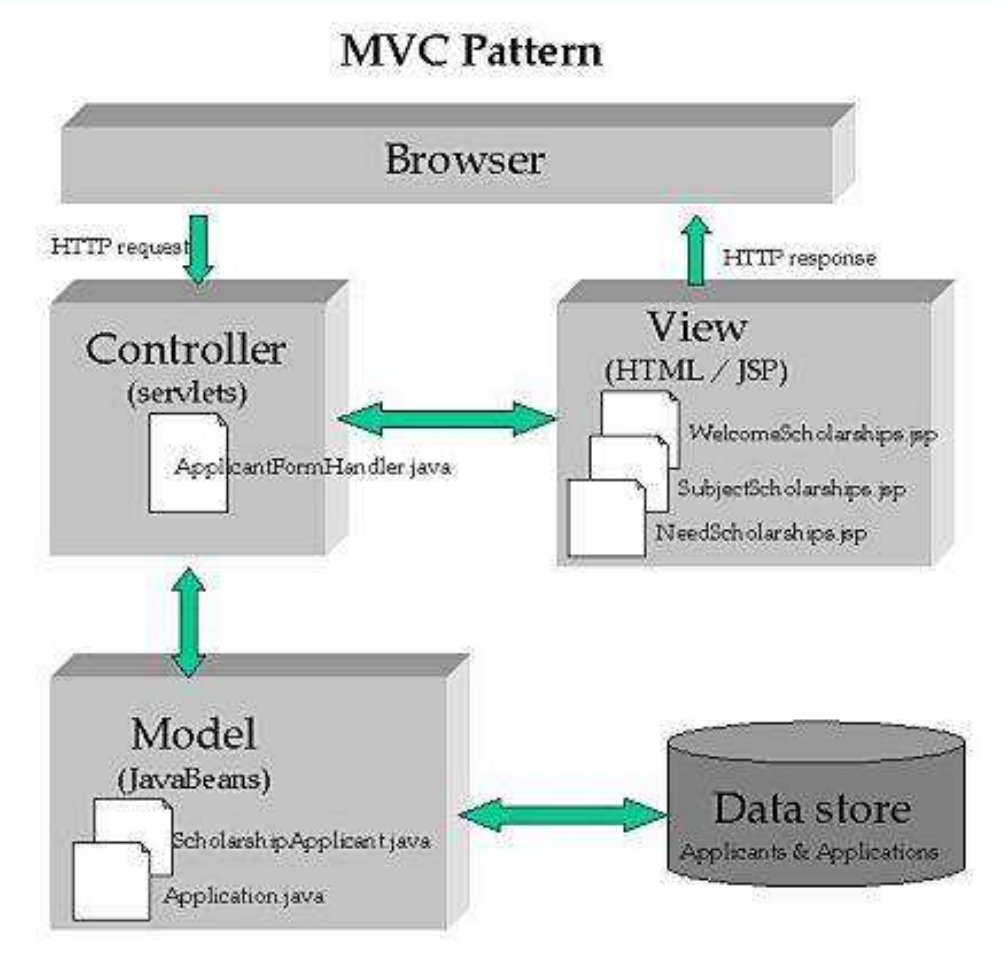


Funcionamento do Padrão Arquitetural MVC

- Cenário onde o estado do `model` é alterado:



Exemplo Prático do Padrão Arquitetural MVC



Estrutura da Solução do Padrão Arquitetural MVC

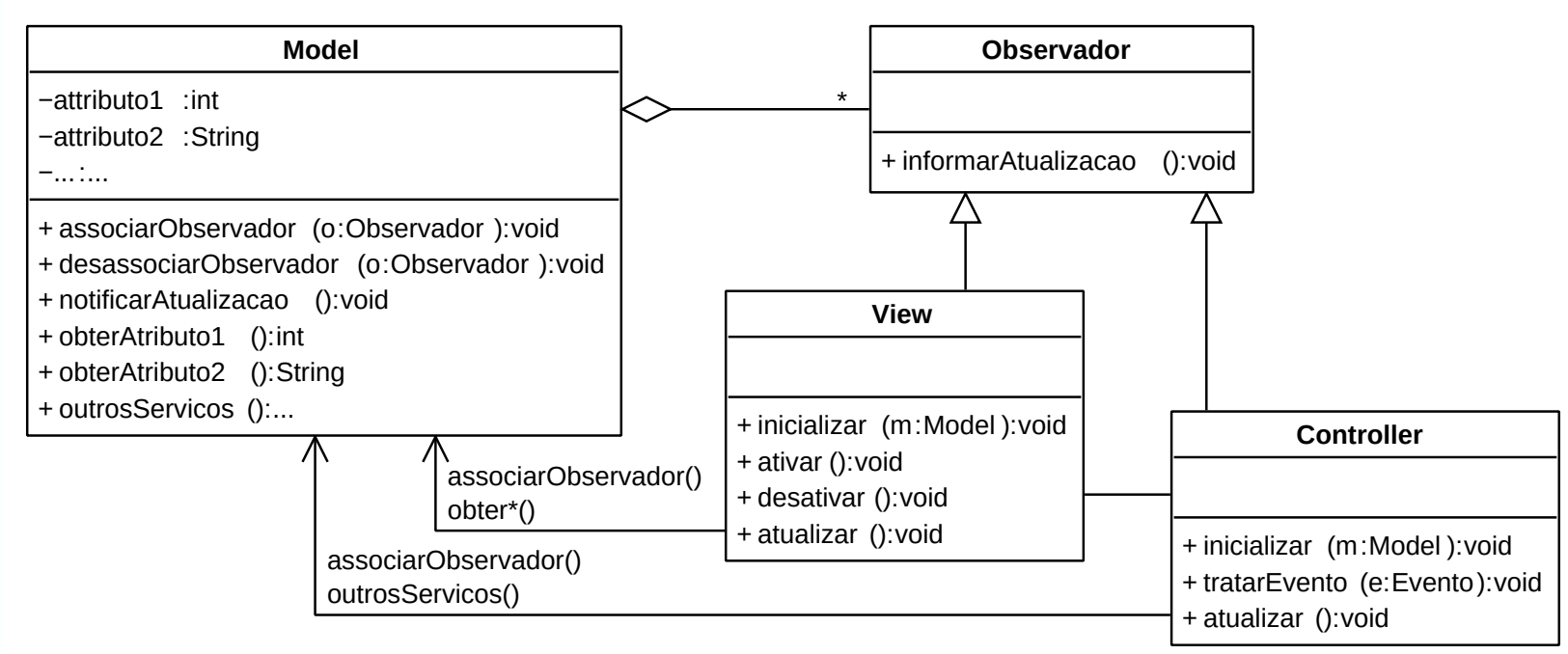
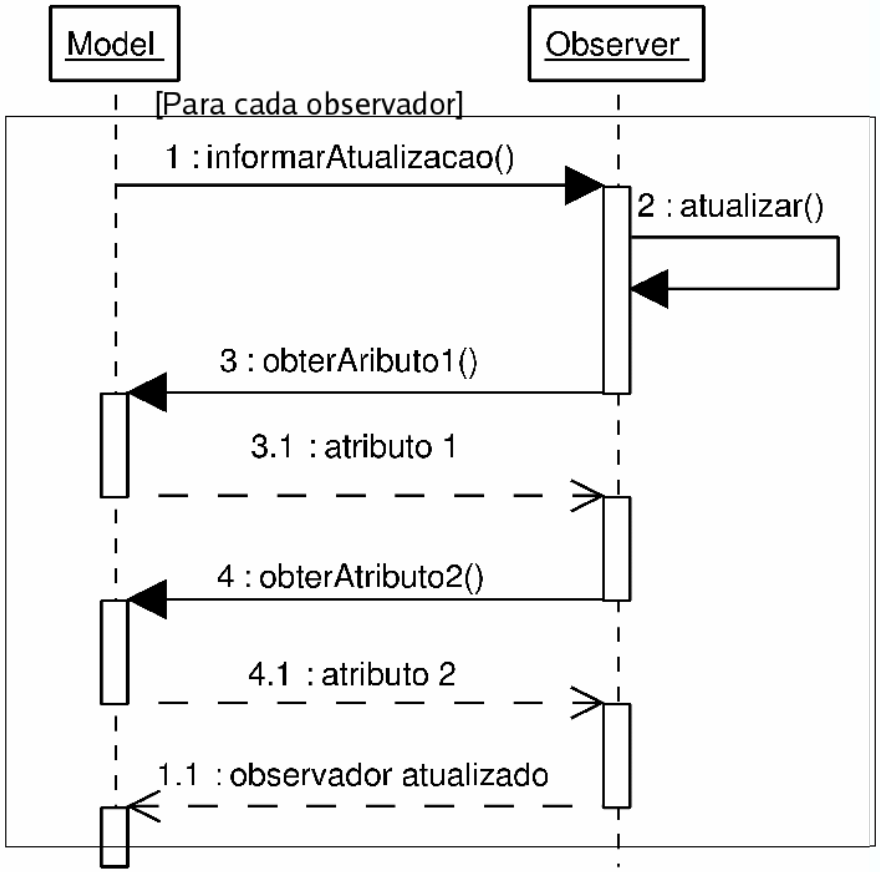


Diagrama de Seqüência do Padrão Arquitetural MVC

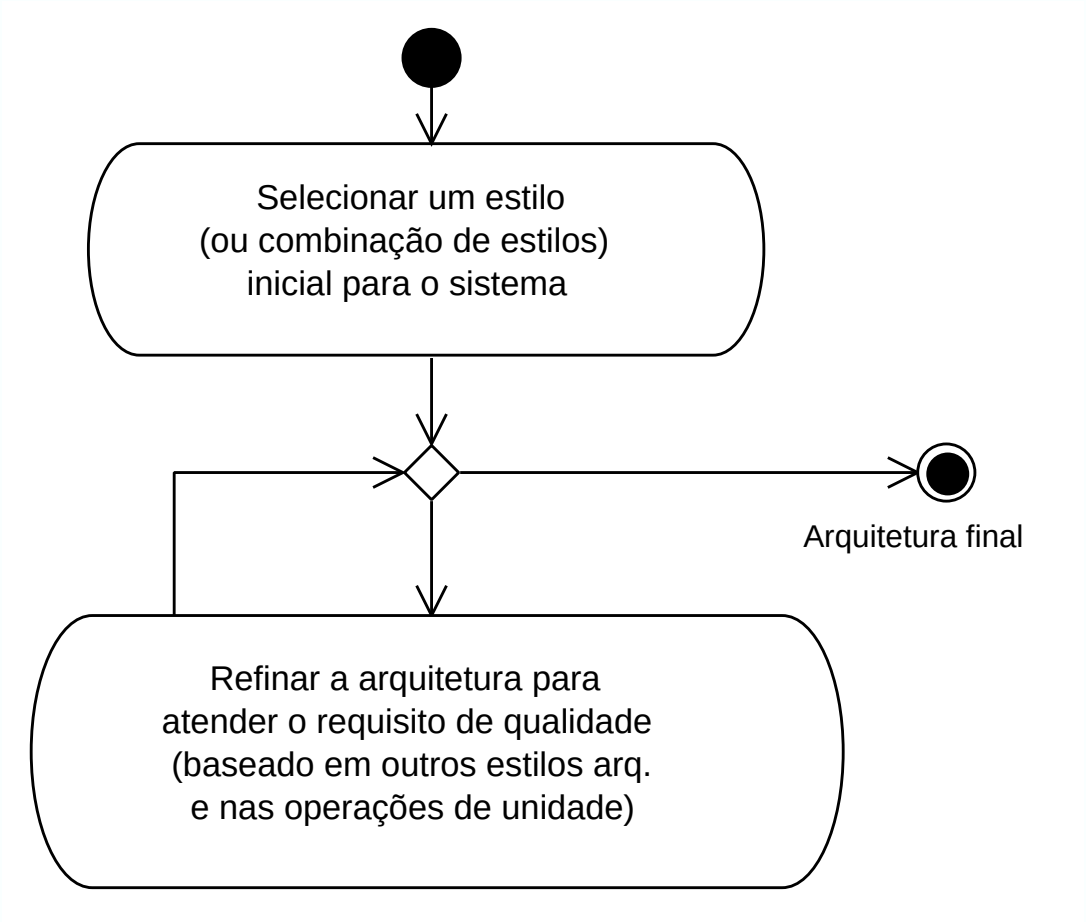


Usos de Estilos e Padrões Arquiteturais

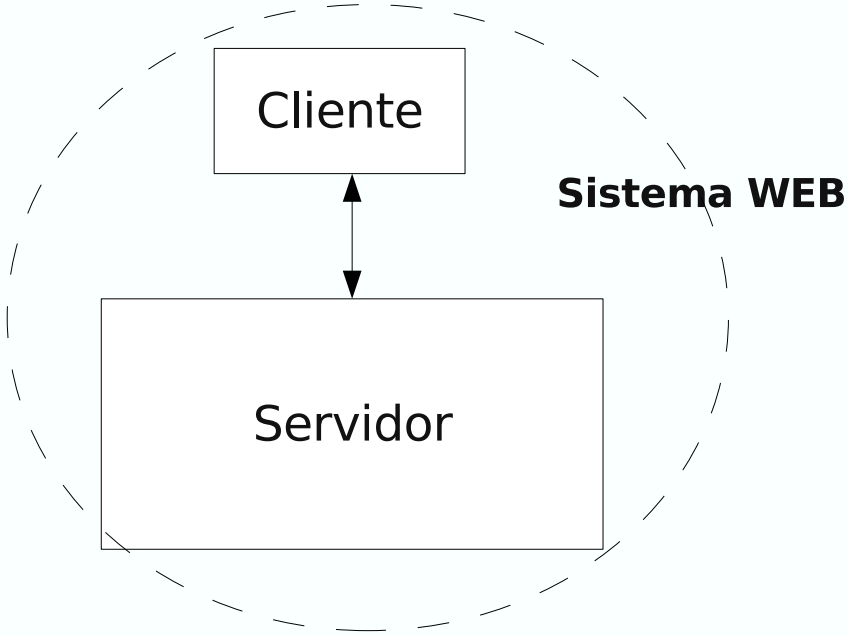
- Como um ponto de partida para a fase de projeto arquitetural;
- Como uma *checklist* da fase de projeto;
- Como uma maneira de organizar o trabalho da equipe de desenvolvimento;
- Como uma maneira de identificar componentes candidatos à reutilização;
- Como um vocabulário a respeito dos tipos da aplicação.

Exemplo Passo a Passo:
Arquitetura do Sistema de E-banking

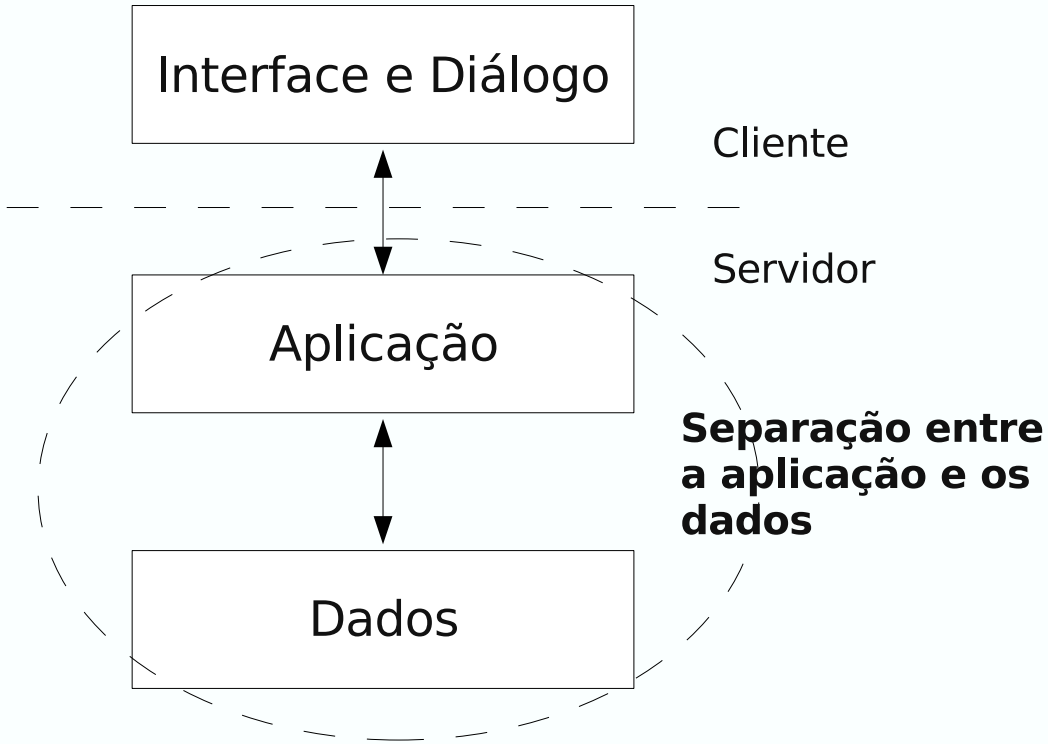
Atividades Executadas



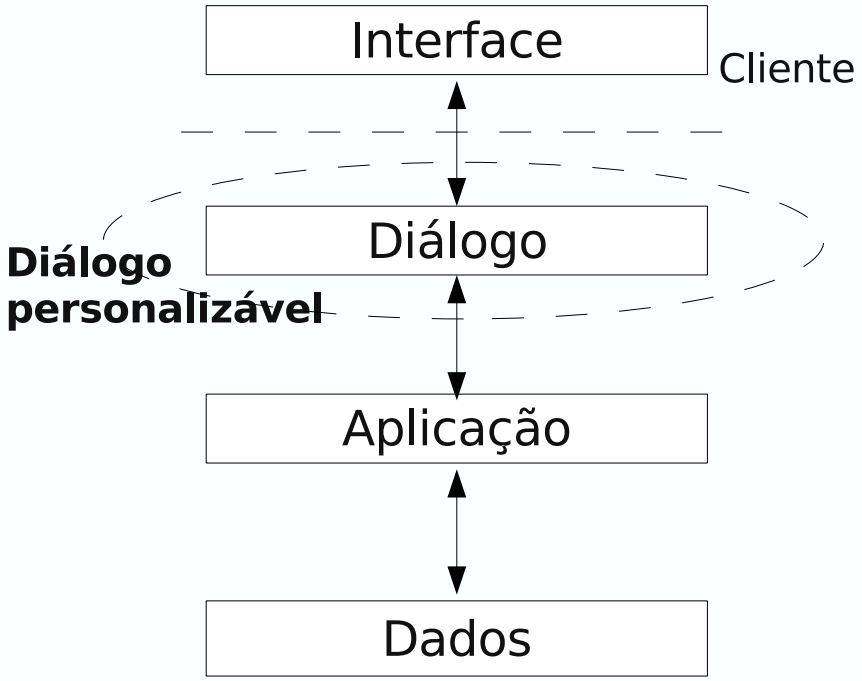
Arquitetura Inicial



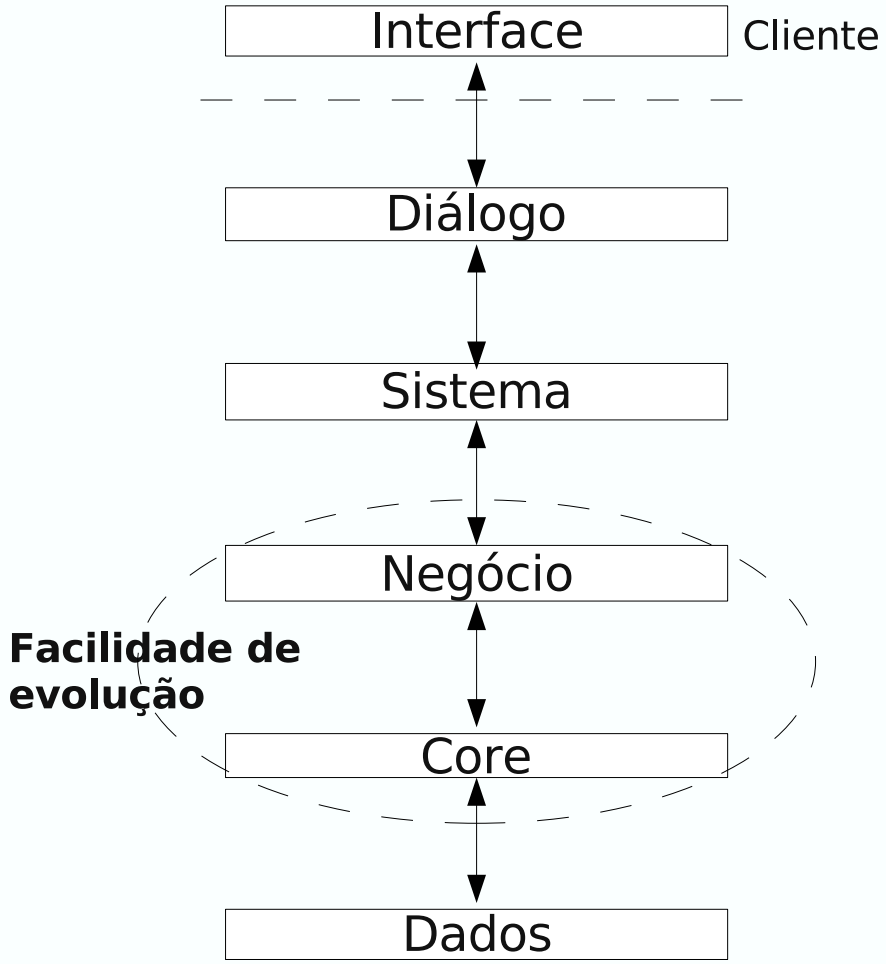
Versão 2


Interface e Diálogo
Cliente
Servidor
Aplicação
Separação entre a aplicação e os dados
Dados

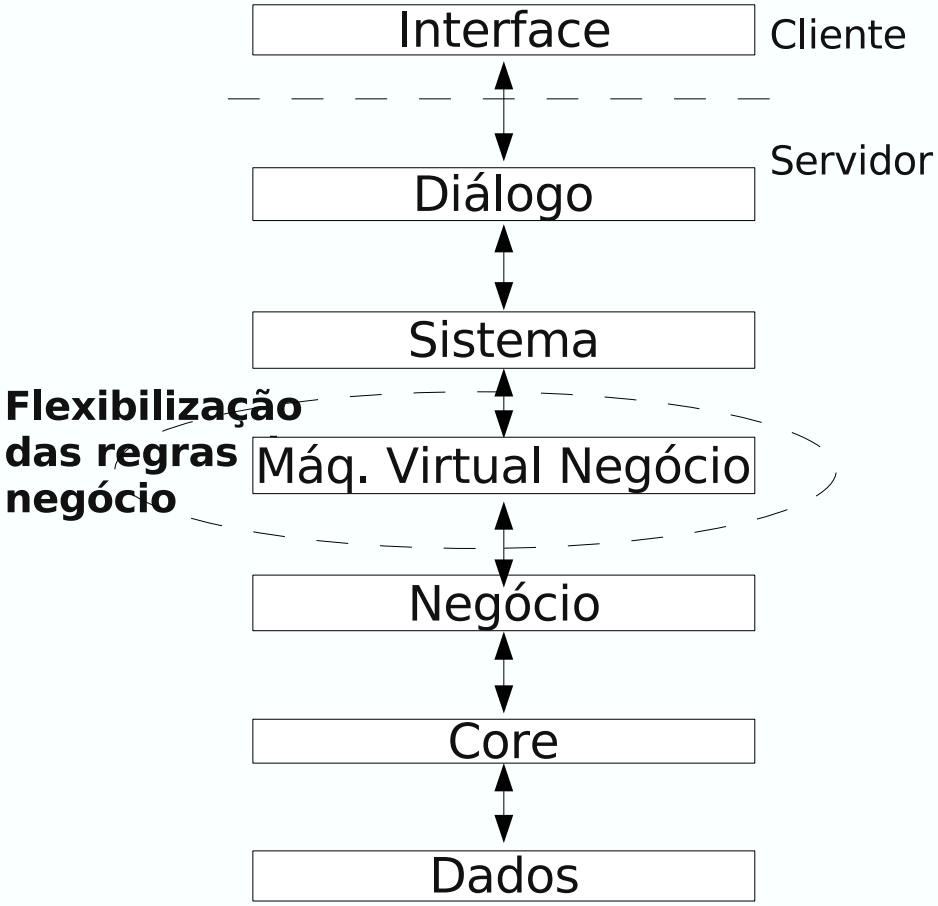
Versão 3


Interface
Cliente
Diálogo
Diálogo
personalizável
Aplicação
Dados

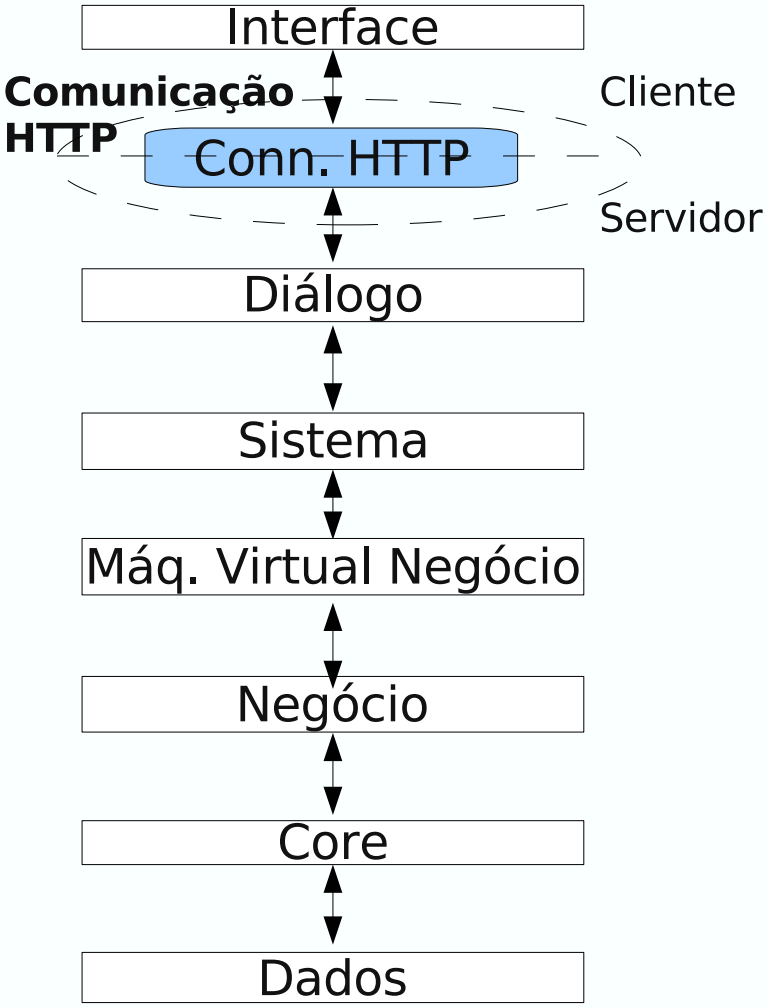
Versão 4


Interface
Cliente
Diálogo
Sistema
Negócio
Facilidade de evolução
Core
Dados

Versão 5


Interface
Cliente
Servidor
Diálogo
Sistema
Flexibilização
das regras
negócio
Máq. Virtual Negócio
Negócio
Core
Dados

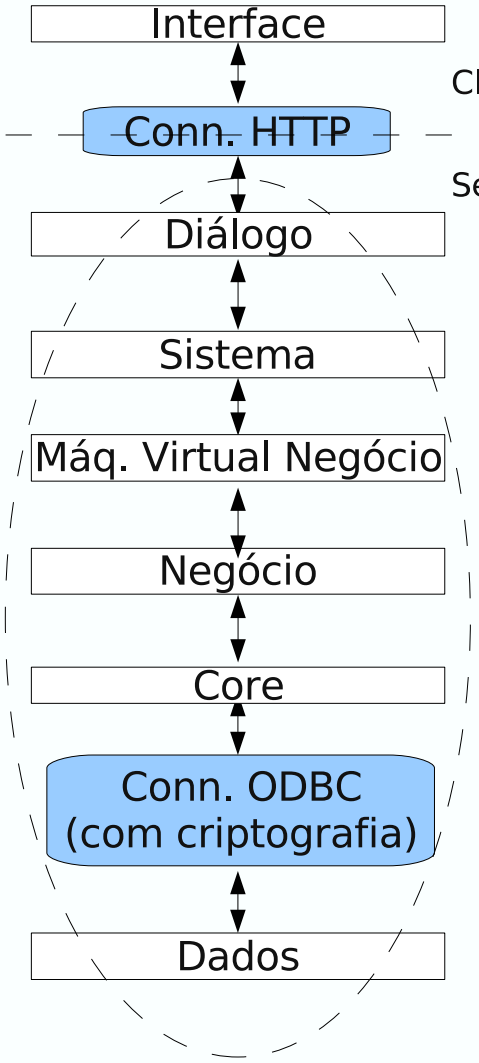
Versão 6


Interface
Comunicação HTTP
Cliente
Conn. HTTP
Servidor
Diálogo
Sistema
Máq. Virtual Negócio
Negócio
Core
Dados

Versão 7


Interface
Cliente
Conn. HTTP
Servidor
Diálogo
Sistema
Máq. Virtual Negócio
Negócio
Core
Dados remotos
e cifrados
Conn. ODBC
(com criptografia)
Dados

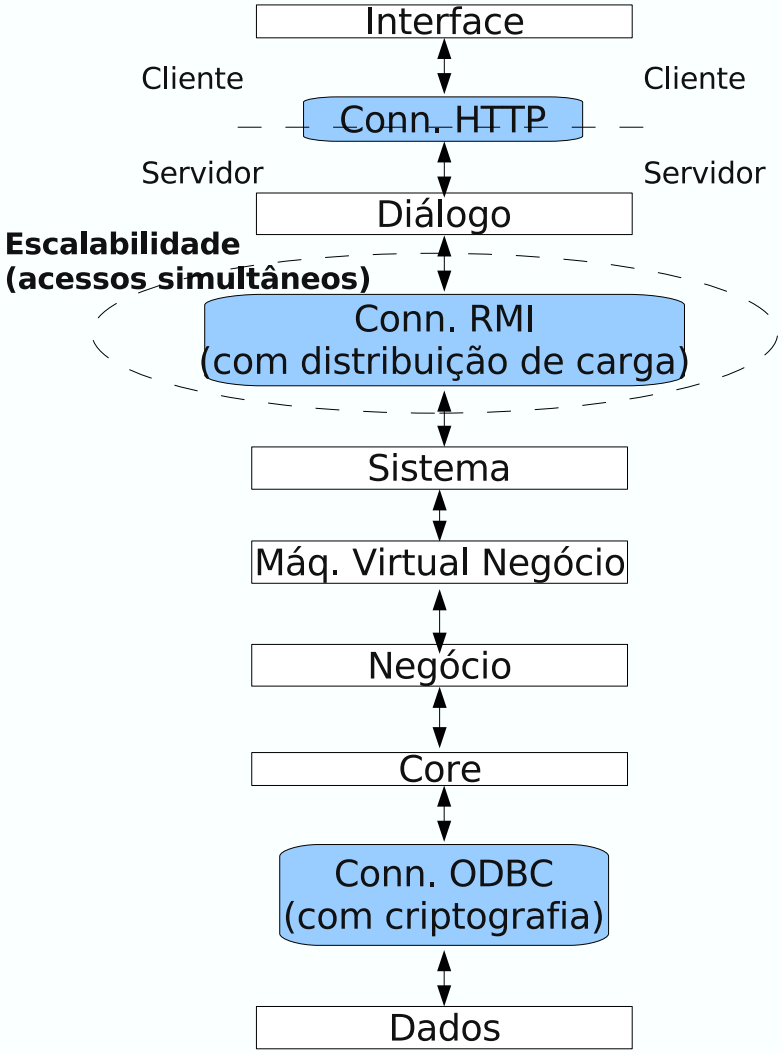
Versão 8



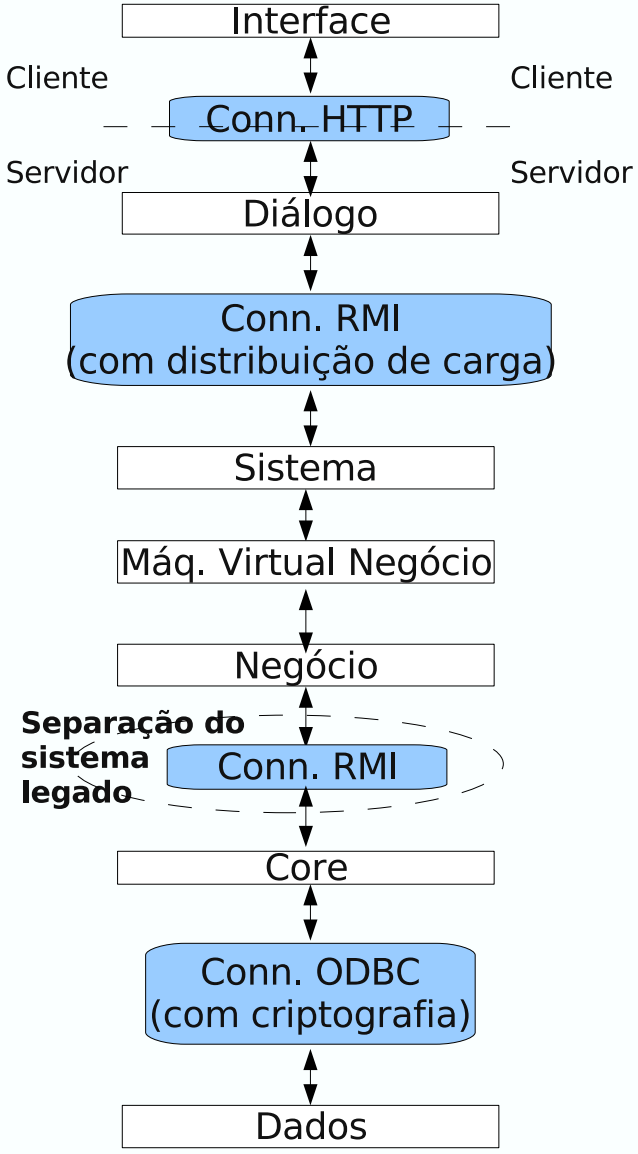
Desempenho: *tradeoff*

- O número de camadas não foi reduzido;
- Os outros requisitos possuem maior prioridade:
  - diálogo flexível;
  - fácil evolução;
  - segurança dos dados.

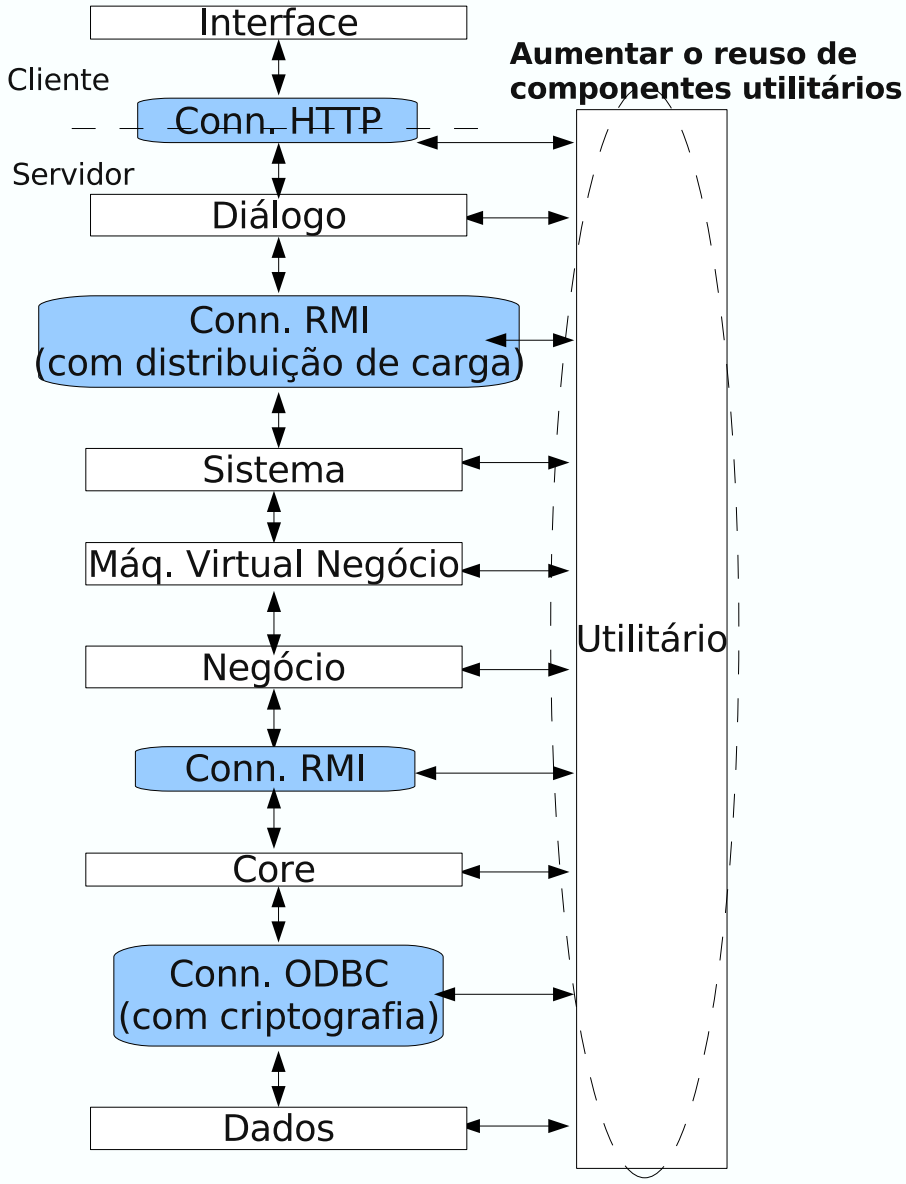
Versão 9


Interface
Cliente
Cliente
Conn. HTTP
Servidor
Servidor
Diálogo
Escalabilidade
(acessos simultâneos)
Conn. RMI
(com distribuição de carga)
Sistema
Máq. Virtual Negócio
Negócio
Core
Conn. ODBC
(com criptografia)
Dados

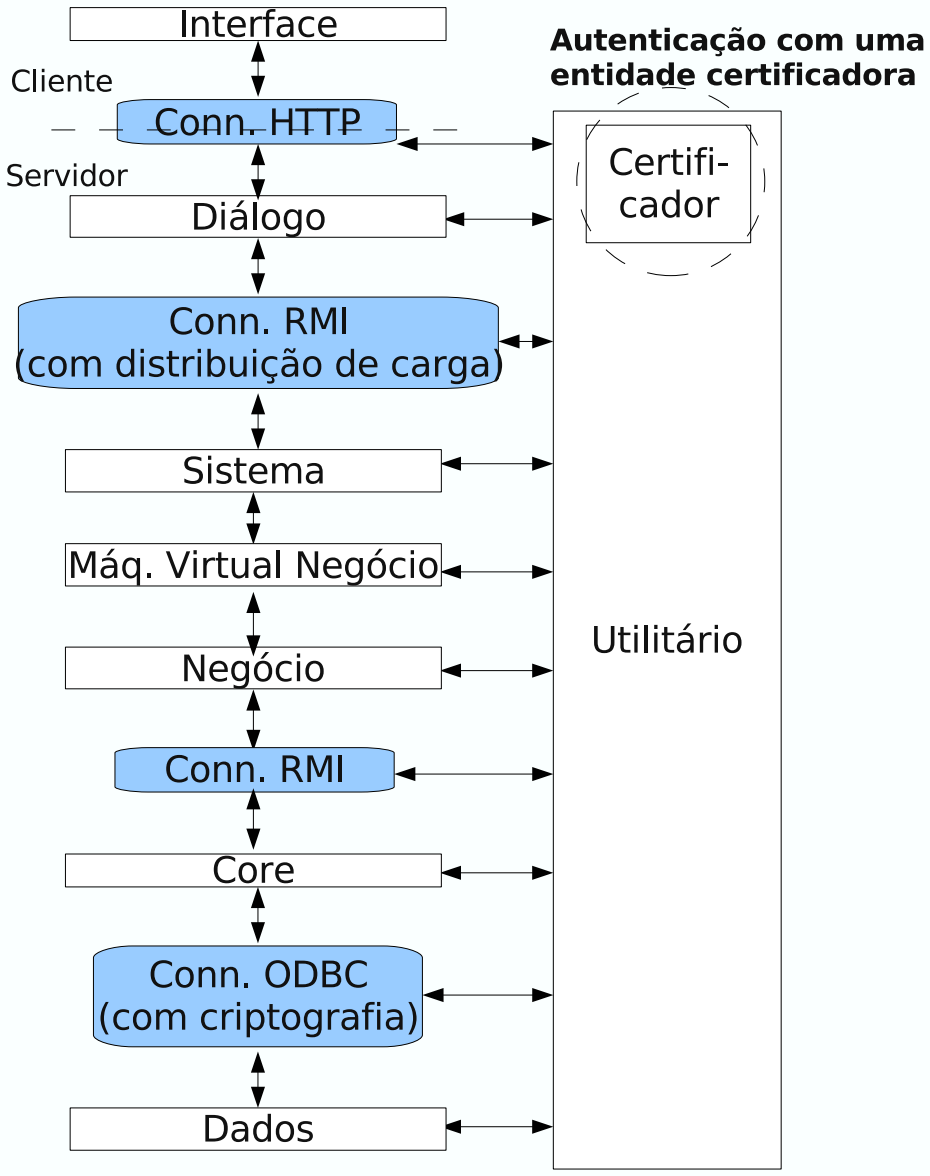
Versão 10


Interface
Cliente
Cliente
Conn. HTTP
Servidor
Servidor
Diálogo
Conn. RMI
(com distribuição de carga)
Sistema
Máq. Virtual Negócio
Negócio
Separação do
sistema
legado
Conn. RMI
Core
Conn. ODBC
(com criptografia)
Dados

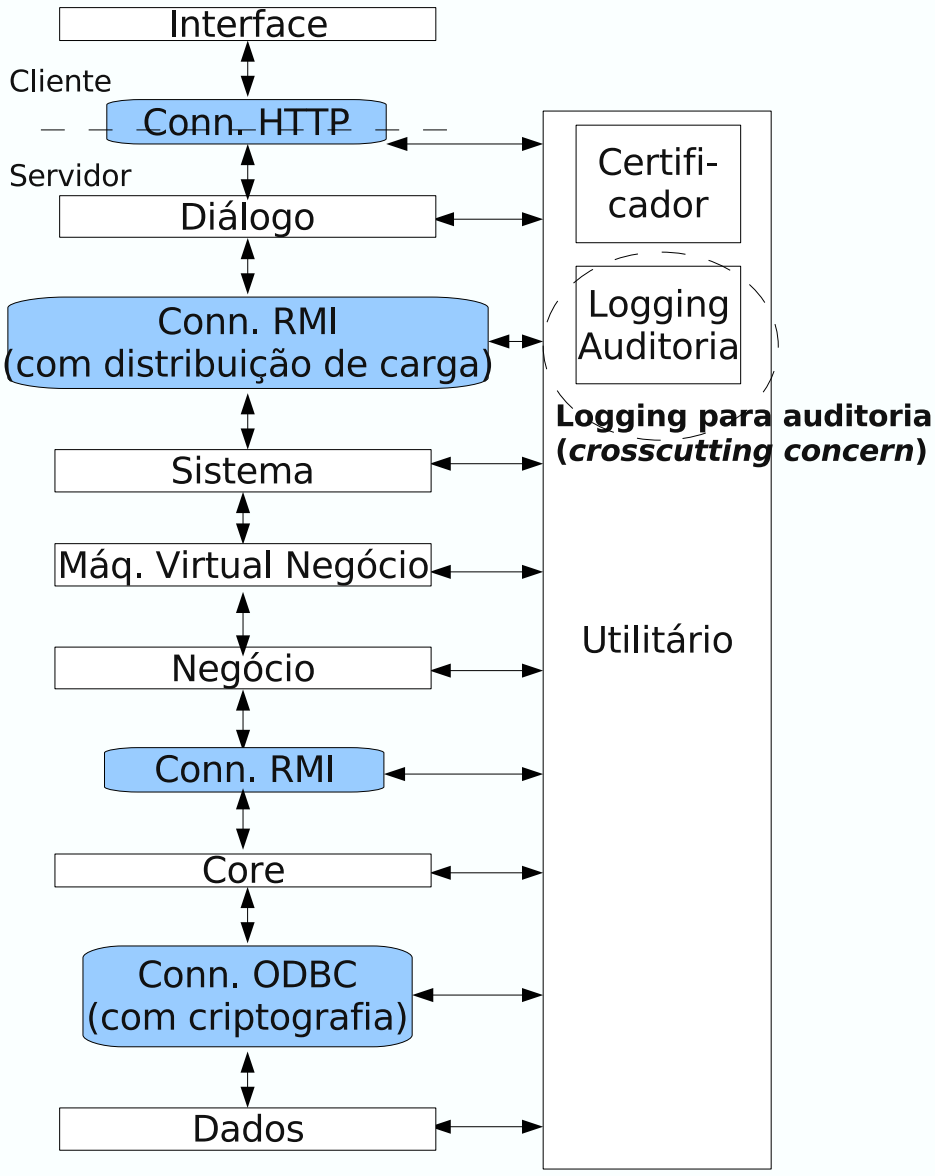
Versão 11


Interface
Cliente
Conn. HTTP
Servidor
Diálogo
Conn. RMI
(com distribuição de carga)
Sistema
Máq. Virtual Negócio
Negócio
Conn. RMI
Core
Conn. ODBC
(com criptografia)
Dados
Aumentar o reuso de
componentes utilitários
Utilitário

Versão 12


Interface
Cliente
Conn. HTTP
Servidor
Diálogo
Conn. RMI
(com distribuição de carga)
Sistema
Máq. Virtual Negócio
Negócio
Conn. RMI
Core
Conn. ODBC
(com criptografia)
Dados
Autenticação com uma
entidade certificadora
Certifi-
cador
Utilitário

Versão 13


Interface
Cliente
Conn. HTTP
Servidor
Diálogo
Conn. RMI
(com distribuição de carga)
Sistema
Máq. Virtual Negócio
Negócio
Conn. RMI
Core
Conn. ODBC
(com criptografia)
Dados
Certifi-
cador
Logging
Auditoria
Logging para auditoria
(crosscutting concern)
Utilitário

Exercício

1. As questões seguintes se referem ao sistema de Controle de Videolocadora apresentado anteriormente:
   (a) Qual o estilo arquitetural que você consideraria inicialmente?
   (b) Especifique a arquitetura do sistema, a partir do estilo arquitetural selecionado no item 1a.

Arquiteturas de Referência

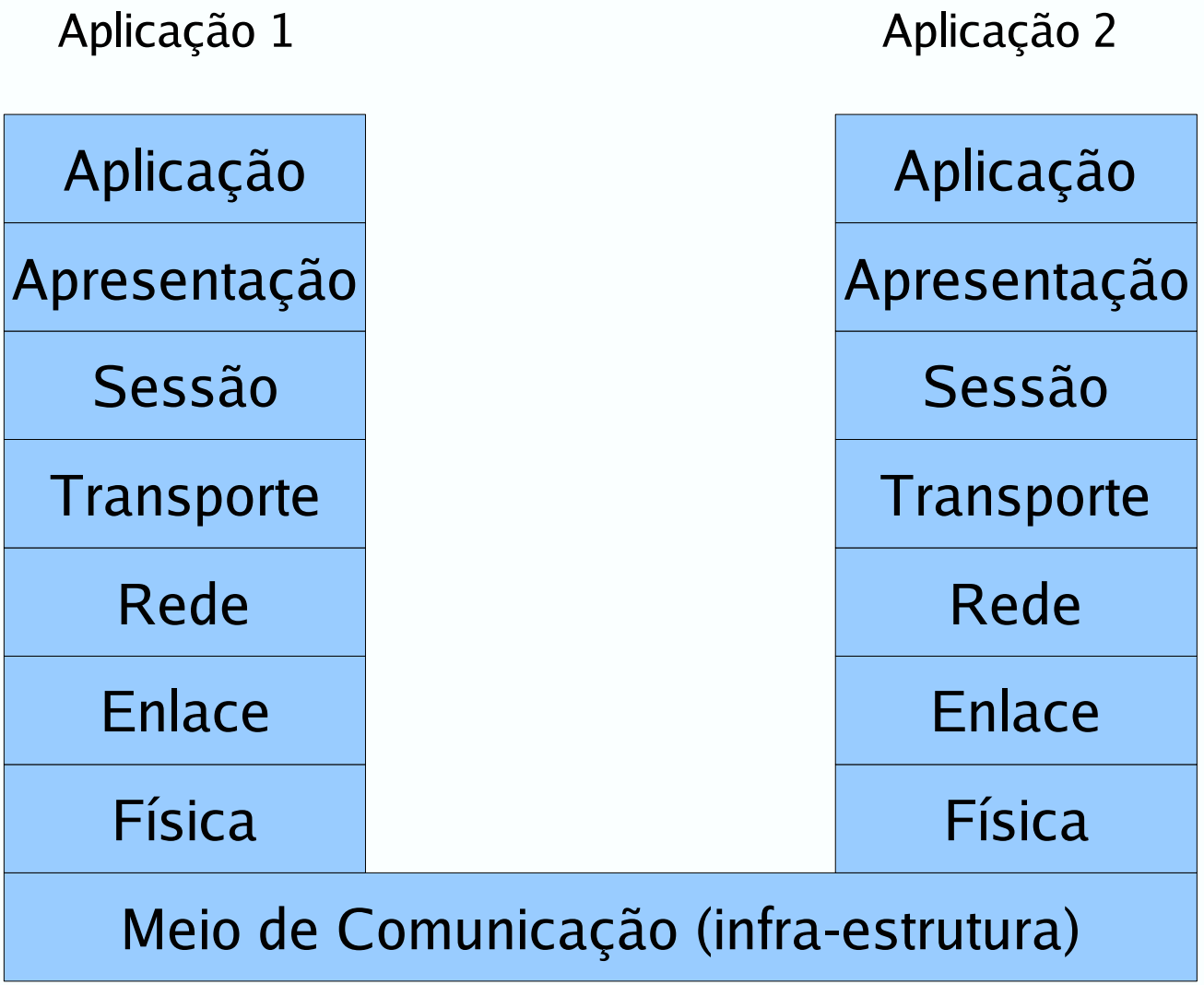
Arquiteturas de Referência (I)

- Estilos e padrões arquiteturais podem ser utilizados como referência inicial para o projeto da arquitetura do sistema;
- Os estilos arquiteturais podem ser adequados para domínios específicos de aplicações;
- Nesses casos, dizemos que um estilo arquitetural (ou uma combinação de estilos) pode ser considerado uma **arquitetura de referência** para tipos de aplicações específicos.

Arquiteturas de Referência (II)

- Modelos de referência são derivados de um estudo cuidadoso do domínio da aplicação, ao invés de analisar aplicações existentes;
- Podem ser utilizados como a base para a implementação do sistema ou para comparar diferentes sistemas. Esses modelos atuam como um padrão sobre os quais os sistemas podem ser avaliados;
- O modelo OSI é um exemplo de modelo em camadas para sistemas de comunicação.

A Arquitetura de Referência OSI



Arquiteturas de Referência (III)

- Os sistemas e aplicações são projetados para atender as necessidades de uma organização;
- Nos negócios que possuem muitas características em comum, as suas aplicações tendem a ter uma arquitetura comum que reflete os requisitos da aplicação;
- Uma arquitetura genérica é configurada e adaptada para criar um sistema que satisfaz requisitos específicos.

Uso de Arquiteturas de Referência

- Como um ponto de partida para a fase de projeto arquitetural;
- Como uma *checklist* da fase de projeto;
- Como uma maneira de organizar o trabalho da equipe de desenvolvimento;
- Como uma maneira de identificar componentes candidatos à reutilização;
- Como um vocabulário a respeito dos tipos da aplicação.

Exemplos de Arquiteturas de Referência

Tipos de Aplicações Mais Comuns (I)

- Aplicações de processamento de dados:
  - Aplicações centradas em dados que processam dados em grupo (*batch*), sem explicitar intervenções do usuário durante o processamento.
- Aplicações de processamento de transações:
  - Aplicações centradas em dados que processam as requisições dos usuários e atualizam as informações do banco de dados;
  - Os **sistemas de informação** são um tipo de aplicação baseada em processamento de transações.

Tipos de Aplicações Mais Comuns (II)

- Sistemas de processamento de eventos:
  - Aplicações onde as ações do sistema dependem da interpretação de eventos do ambiente da aplicação.
- Sistemas de processamento de linguagem:
  - Aplicações onde as intenções dos usuários são especificadas em uma linguagem formal, que por sua vez, é processada e interpretada pelo sistema.

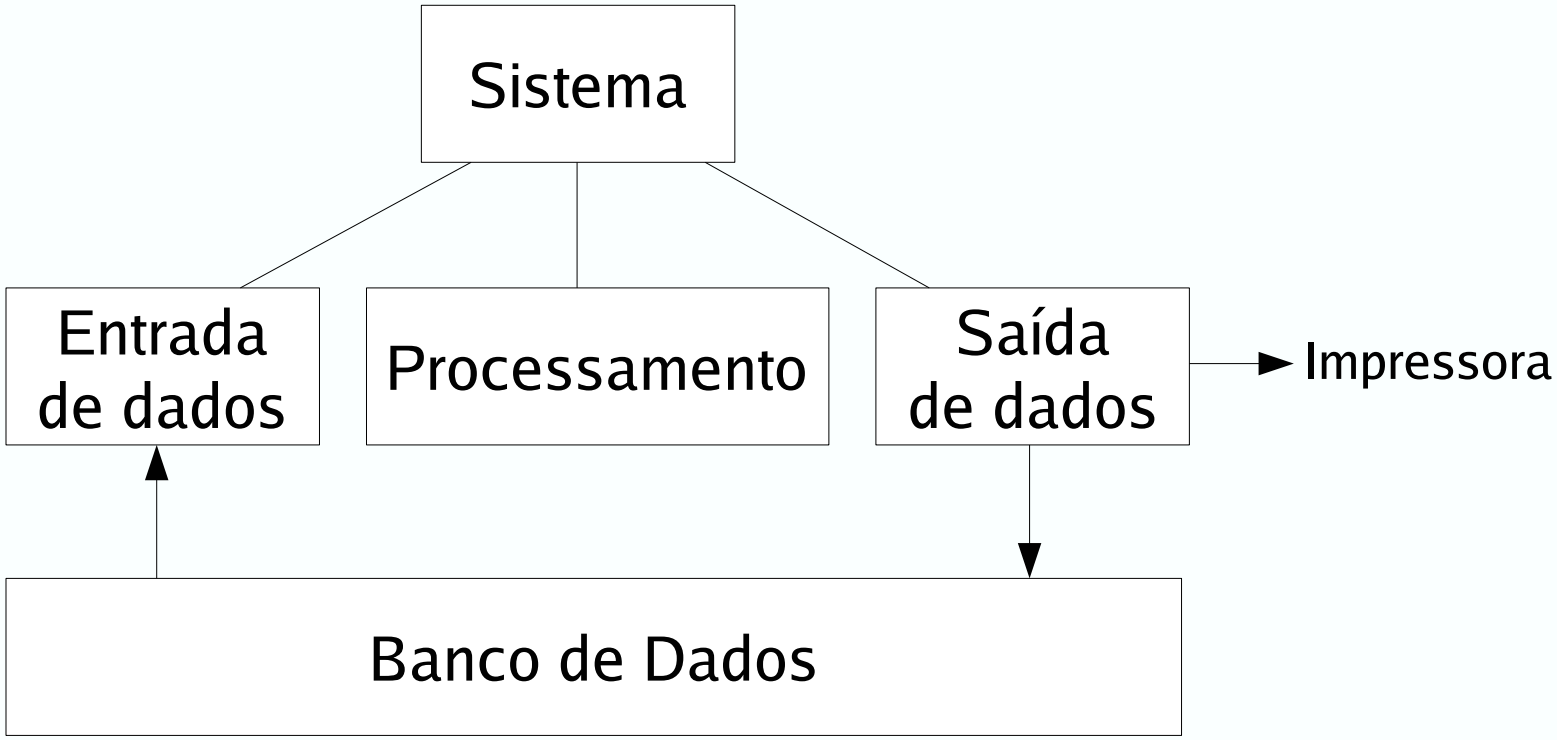
Exemplos de Aplicações

- Aplicações de processamento de dados:
  - Sistema de faturamento;
  - Sistema de folha de pagamento;
- Aplicações de processamento de transações:
  - Sistemas de *e-commerce*;
  - Sistemas de reserva.
- Sistemas de processamento de eventos:
  - Processadores de texto;
  - Sistemas de tempo real.
- Sistemas de processamento de linguagem:
  - Compiladores e interpretadores;
  - Interpretadores de comandos.

Sistema de Processamento de Dados (I)

- Sistemas que são centrados em dados, onde as bases de dados utilizadas são normalmente muito maiores que o software em si;
- Os dados são as entradas e saídas dos processamentos em *batch*:
  - **Entrada**: Um conjunto de números e leitores associados de um medidor de consumo elétrico;
  - **Saída**: Um conjunto de contas correspondentes ao conjunto de dados fornecido como entrada, uma conta para cada usuário.
- Sistemas de processamento de dados normalmente têm uma estrutura de: $entrada \rightarrow processamento \rightarrow saida$.

Sistema de Processamento de Dados (II)



Sistema de Informação (I)

- Sistemas de informação têm uma arquitetura genérica que pode ser organizada utilizando o estilo arquitetural em camadas, com pelo menos cinco camadas:
  - Interface com o usuário;
  - Gerenciador de diálogo com o usuário;
  - Processador de dados;
  - Gerenciador de informações, que controla o armazenamento e recuperação de dados;
  - Sistema de armazenamento de dados (banco de dados em si).

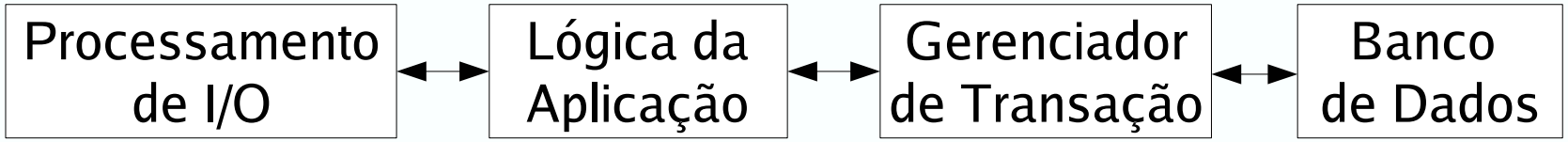
Sistema de Informação (II)

Interface com o Usuário
Gerenciador de Diálogo com o Usuário
Processador de dados
Gerenciador de informações (controle de recuperação/armazenamento)
Banco de Dados Relacional

Sistema de Processamento de Transações (I)

- Processa as requisições dos usuários, podendo utilizar ou atualizar informações da base de dados.
- De uma perspectiva do usuário, uma transação é:
  - Qualquer seqüência de operações que satisfazem um objetivo;
  - Por exemplo: encontrar os horários disponíveis para os vôos de Londres a Paris em um determinado dia.
- Usuários fazem requisições assíncronas para serviços que são processados pelo gerenciador de transação.

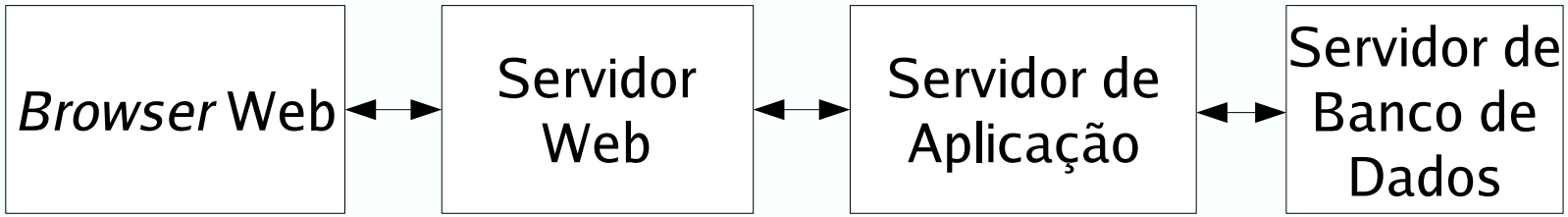
Sistema de Processamento de Transações (II)



Sistema de E-commerce (I)

- Sistemas de comércio eletrônico (*e-commerce*) são sistemas de gerenciamento de recursos que operam através da Internet;
- Eles são normalmente organizados utilizando uma arquitetura em várias camadas, com as camadas da aplicação associadas umas com as outras;
- É semelhante à arquitetura dos sistemas de processamento de transações, mas com outros papéis definidos para as camadas.

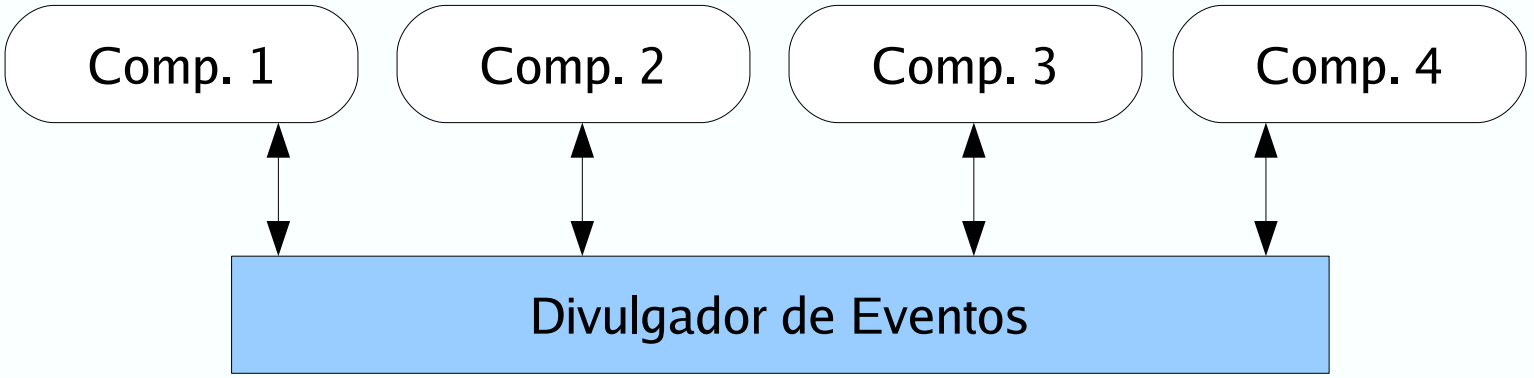
Sistema de E-commerce (II)



Sistema de Processamento de Eventos (I)

- Esses sistemas respondem a eventos que ocorrem no ambiente do sistema;
- Sua característica chave é que os eventos podem ocorrer a qualquer instante. Dessa forma, a arquitetura deve estar organizada para tratá-los a qualquer momento;
- Ex: processadores de texto, jogos, etc.

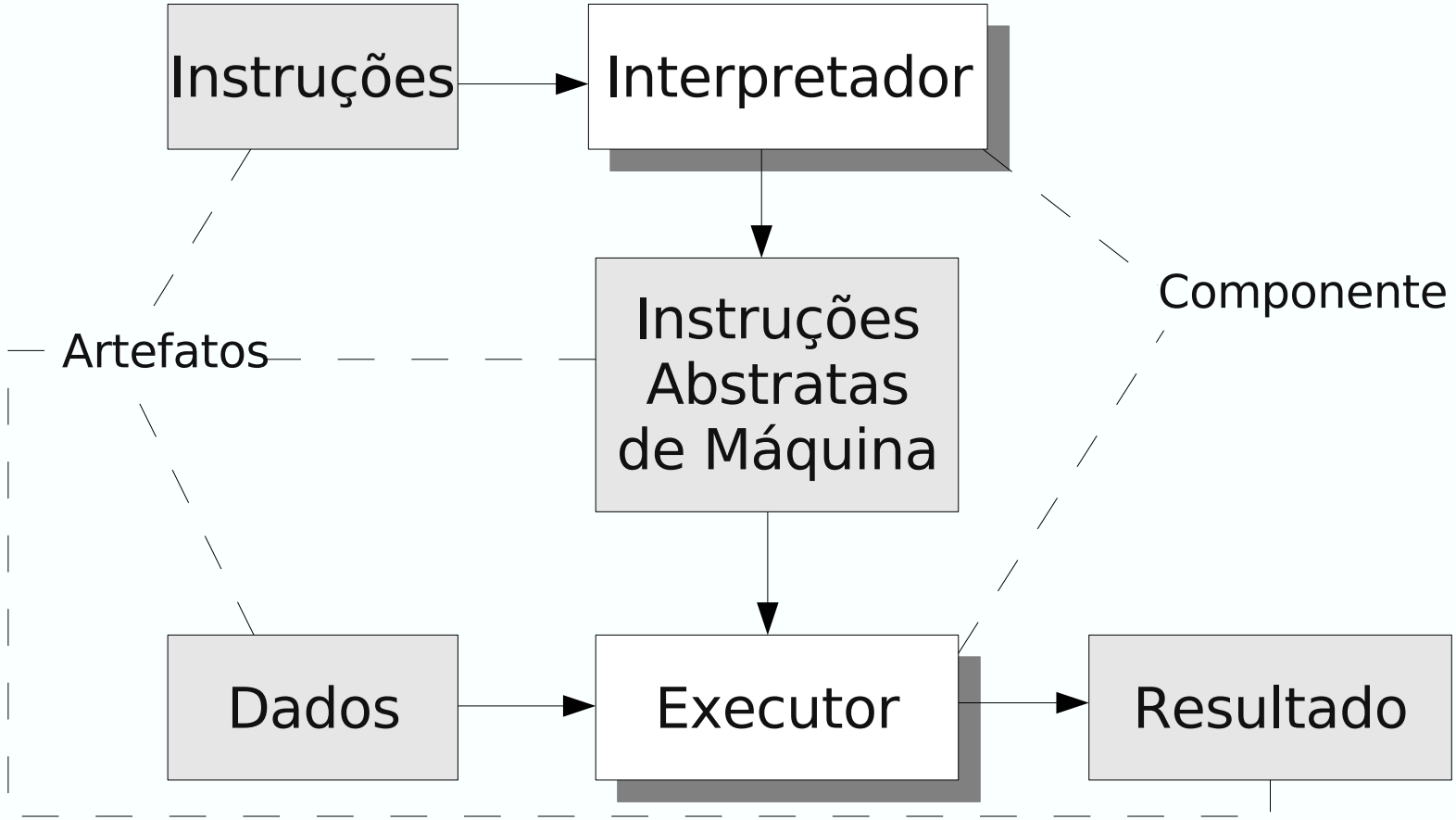
Sistema de Processamento de Eventos (II)



Sistema de Processamento de Linguagem (I)

- Recebe uma linguagem (natural ou artificial) como entrada e gera alguma outra representação da linguagem como saída;
- Pode incluir um interpretador para atuar de acordo com as instruções presentes na linguagem processada;
- Usado em situações onde a maneira mais fácil de resolver um problema é descrever um algoritmo ou descrever os dados do sistema:
  - Ferramentas de descrição baseadas em meta-modelos, módulos baseados em regras variáveis (interpretadas), ferramentas de geração de código, etc.

Sistema de Processamento de Linguagem (II)



Componentes de um Compilador

- Analisador léxico;
- Tabela de símbolos;
- Analisador sintático;
- Árvore sintática;
- Analisador semântico;
- Gerador de código.

Desenvolvimento Baseado em Arquitetura

Desenvolvimento Baseado em Arquitetura

- Montagem da equipe;
- Criação do esqueleto do sistema;
- Uso de padrões;
- Garantir a conformidade.

Montagem da Equipe

- Estrutura Conceitual (Visão Lógica);
- Divisão em subdomínios;
- Equipes especializadas.

Criação do Esqueleto do Sistema

- Interações entre componentes ("glues");
- Componentes "stub";
- Áreas de risco;
- Funcionalidades Específicas.

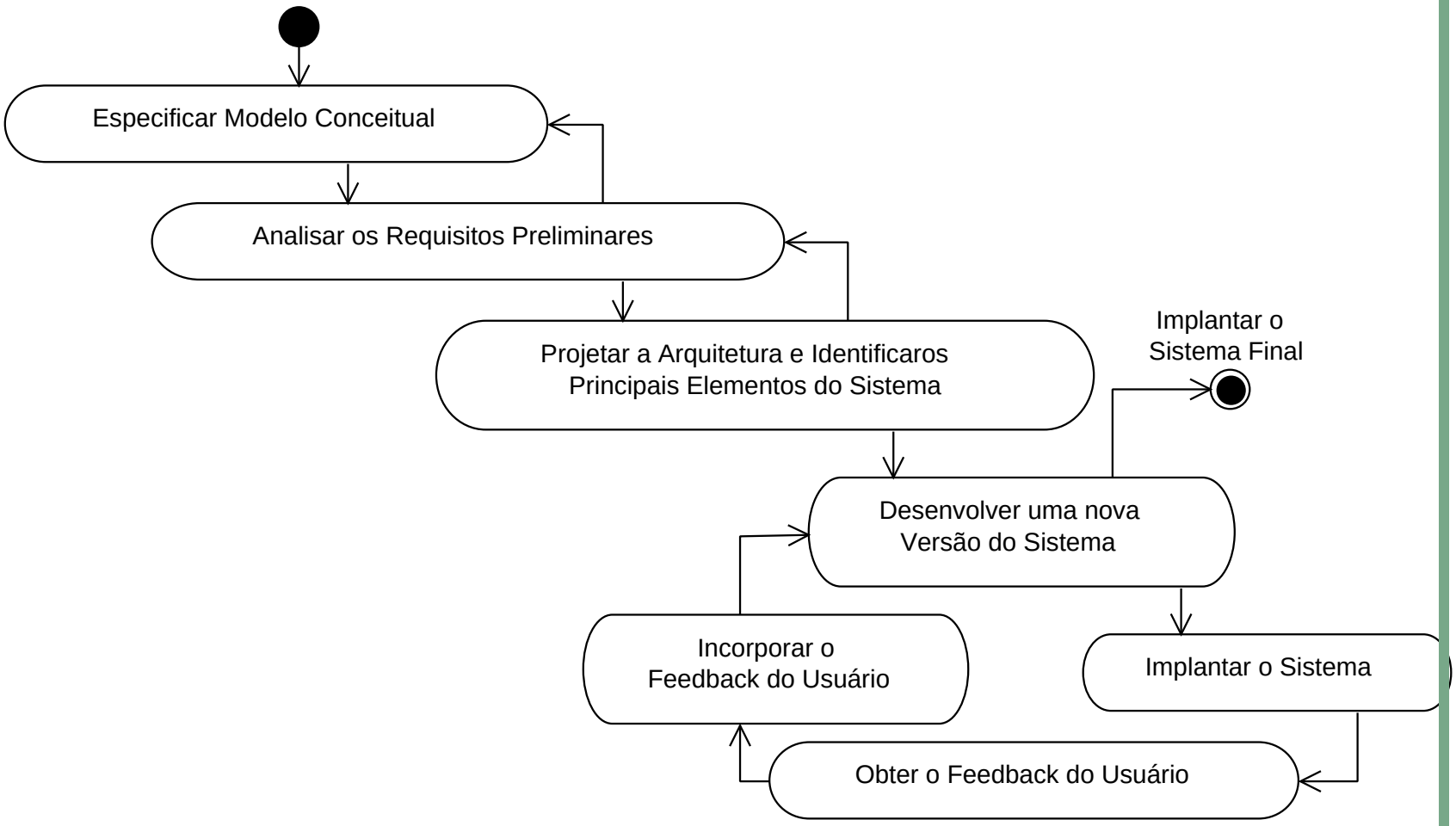
Uso de Padrões

- Estilos e padrões arquiteturais;
- Padrões de Projeto ("design patterns");
- Padrões de Codificação ("idioms").

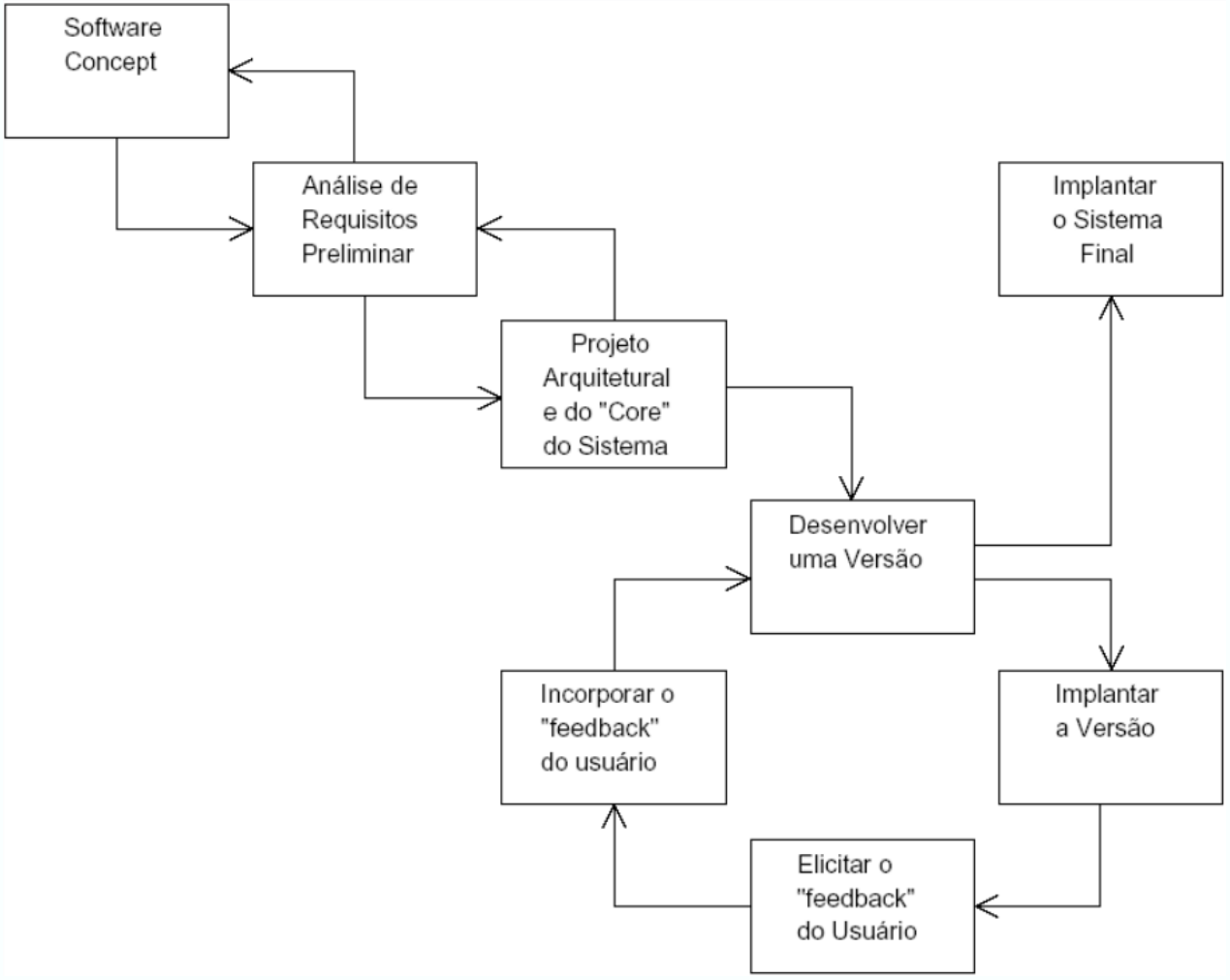
Garantir a Conformidade

- Documentação Atualizada;
- Engenharia Reversa.

Atividades Gerais de Processos Centrados na Arquitetura



Ciclo de Vida do Projeto Arquitetural



Recomendações Práticas para o Processo de Desenvolvimento (I)

- Líder da arquitetura: a arquitetura deve ser o produto de um único arquiteto ou um grupo pequeno com um líder definido;
- Requisitos técnicos e prioridades definidos: o arquiteto precisa dos requisitos funcionais e uma lista priorizada dos atributos de qualidade que a arquitetura deve realizar;
- Bem documentada: deve-se usar uma notação que os stakeholders sejam capazes de entender com esforço mínimo;
- Envolvimento dos usuários / clientes: a arquitetura deve ser circulada entre os *stakeholders*, que devem estar ativamente envolvidos na sua revisão.

Recomendações Práticas para o Processo de Desenvolvimento (II)

- Análise objetiva e independente: a arquitetura deve ser analisada para validar medidas quantitativas aplicáveis e deve ser formalmente revisada para validar suas propriedades qualitativas. (como por exemplo, modificabilidade) antes que seja muito tarde para se aplicar mudanças;
- Desenvolvimento incremental: desenvolvimento incremental via a criação de um esqueleto do sistema inicialmente com uma funcionalidade mínima;
- Regras simples de resolução de conflitos: a arquitetura deve resultar em um conjunto pequeno e específico de áreas de contenção de recursos.

O Projeto Arquitetural

Projeto Arquitetural Baseado em Atributos de Qualidade

- De acordo com o *Software Engineering Institute* (SEI), o promeiro passo do projeto arquitetual deve ser a escolha de uma **arquitetura de referência**, baseada nos atributos de qualidade desejados para o sistema;
- Em seguida, essa arquitetura deve ser refinada, de acordo com as prioridades entre os atributos de qualidade;
- Para a escolha da arquitetura de referência, o SEI propõe o uso de um catálogo de estilos arquiteturais, denominado ABAS;
- Para o refinamento da arquitetura de referência, o SEI propõe o uso de operações de unidade;
- Esses dois conceitos serão detalhados a seguir.

Estilo Arquitetural Baseado em Atributos (ABAS)

Estilo Arquitetural Baseado em Atributos (ABAS)

- É um "catálogo" que relaciona atributos de qualidade a estilos arquiteturais e diretrizes para definir as visões arquiteturais;
- Utiliza os estilos e padrões arquiteturais como base para um projeto arquitetural preciso;
- Cada organização pode desenvolver os seus próprios catálogos, relativos a atributos de qualidade específicos e relevantes para o domínio;
- Considera um catálogo por vez (normalmente um requisito não-funcional):
  - É possível considerar mais de um requisito ao mesmo tempo; por exemplo, um estilo arquitetural que satisfaz o par "desempenho e escalabilidade".

Por que Utilizar o ABAS?

- Torna o projeto arquitetural mais previsível;
- Padroniza o critério de análise dos estilos arquiteturais associados ao sistema;
- Facilita o mapeamento entre análise e projeto (requisitos $\rightarrow$) arquitetura;
- Facilita a atualização da arquitetura após evoluções do sistema (consulta ao catálogo).

Exemplo de um Catálogo

Facilidade de Modificação	**Desempenho**
Centrado em Dados	Pipes and filters concorrentes
Camadas	Componentes Indemendentes
Baseado em eventos	
Testabilidade	**Disponibilidade**
Auto teste (componentes auto-testáveis)	Blocos de recuperação (componente tolerante a falhas ideal)
	Redundância
	Segurança (security)
	Firewall
	Rede Privada Virtual
	Criptografia

Operações de Unidade

Operações de Unidade (I)

- A arquitetura de software é altamente influenciada pelos requisitos não-funcionais;
- Durante o projeto arquitetural, é necessário priorizar os atributos de qualidade e encontrar soluções para especificá-la de acordo com as prioridades estabelecidas;
- O SEI definiu seis *unit operations*, categorizadas de acordo com os requisitos não-funcionais que se propõem a resolver:
  - *Separation*, *virtual machine*, *compression*, *uniform composition*, *replication*, e *resource sharing*.

Operações de Unidade (II)

- **Separation:** separação dos sub-módulos funcionais menos coesos em componentes distintos.
  - Aumenta a coesão dos componentes;
  - Aumenta a indireção (menor desempenho).
- **Virtual Machine/Abstraction:** abstração de detalhes desnecessários, por exemplo, criação de máquinas virtuais, objetos, componentes, camadas, interfaces.
  - Flexibiliza a evolução das regras de execução;
  - Aumenta a indireção (menor desempenho).

Operações de Unidade (III)

- **Compression:** remoção de camadas ou interfaces que separam as funcionalidades do sistema.
  - Agiliza o desenvolvimento das aplicações;
  - Melhora o desempenho do sistema;
  - Dificulta a manutenção.
- **Uniform Composition:** combinação de dois ou mais componentes em um componente maior.
  - Limita a composição de um componente a um número pequeno de sub-componentes;
  - Pode estipular regras de composição para os sub-componentes formar o componente maior;
  - Facilita a compreensão do sistema;
  - Facilita a integração dos componentes.

Operações de Unidade (IV)

- **Replication:** replicação de componentes da arquitetura.
  - Melhora a confiabilidade (*reliability*), a disponibilidade e o desempenho;
  - Pode melhorar a escalabilidade (distribuição de carga);
  - Pode aumentar consideravelmente o custo da aplicação (redundância de projeto).
- **Resource Sharing:** encapsulamento dos dados e serviços que são compartilhados por vários consumidores independentes.
  - Facilita o gerenciamento dos recursos compartilhados (ponto único de manutenção);
  - Facilita a integração, a portabilidade e a manutenibilidade;
  - Aumenta o custo de desenvolvimento (desproporção entre o repositório e os demais elementos).

Operações de Unidade (V)

UNIT OPERATIONS	Fatores de Qualidade do Software							
	ESCALA-BILIDADE	MODIFICA-BILIDADE	INTEGRA-BILIDADE	PORTA-BILIDADE	DESEM-PENHO	CONFIABILIDADE (reliability)	DE FÁCIL CRIAÇÃO	REUSA-BILIDADE
SEPARATION	+	+	+	+	"+/-"		"+/-"	+
ABSTRACTION	+	+	+	+	-		+	+
COMPRESSION	-	-	-	-	+		"+/-"	-
UNIFORM COMPOSITION	+		+				+	
REPLICATION	-	-		-	"+/-"	+	-	-
RESOURCE SHARING		+	+	+	"+/-"	-	+	"+/-"

Recomendações Práticas para a Estrutura da Arquitetura (I)

- Ocultação da informação e encapsulamento: a arquitetura deve definir módulos bem definidos cujas responsabilidades funcionais sejam alocadas de acordo com os princípios de ocultação da informação e separação de interesses.

  Cada componente deve ter uma interface bem definida e explícita que encapsula detalhes de implementação;

- Divisão de responsabilidades: os componentes devem refletir a separação de interesses permitindo que seus times de desenvolvimento respectivos trabalhem independentemente entre si.

Recomendações Práticas para a Estrutura da Arquitetura (II)

- Isolamento da infraestrutura: a arquitetura deve separar interesses relacionados com a plataforma de implementação de tal forma que se ela mudar, o esforço de mudança seja minimizado;
- Independência de outros produtos: a arquitetura não deve ser dependente de versões específicas de componentes;
- Módulos produtores X consumidores de dados;
- Processos bem definidos e realocáveis.

Recomendações Práticas para a Estrutura da Arquitetura (III)

- Consistência nas interações entre componentes: a arquitetura deve empregar um número pequeno de padrões simples de interação entre componentes. Isso demonstra a integridade conceitual da arquitetura. Isto ajuda no entendimento da arquitetura, reduz o tempo de desenvolvimento, aumenta a confiabilidade, e facilita a evolução do sistema.

O Processo Unificado (RUP)

- Primeira versão em 1999.
- Surgiu da união das metodologias de Booch (Booch), Rumbaugh (OMT) e Jacobson (OOSE).
- Utiliza a notação UML.
- Imenso conjunto de métodos, técnicas, documentos e procedimentos que visa ser o mais genérico possível.
- Muito complexo para ser usado diretamente. é necessário eliminar partes que não sejam relevantes para as necessidades da organização.

Características do Desenvolvimento no RUP

- UML como linguagem de modelagem;
- Direcionado por casos de uso;
- Centrado na arquitetura;
- Adota o modelo de visões arquiteturais 4+1;
- Iterativo;
- Incremental.

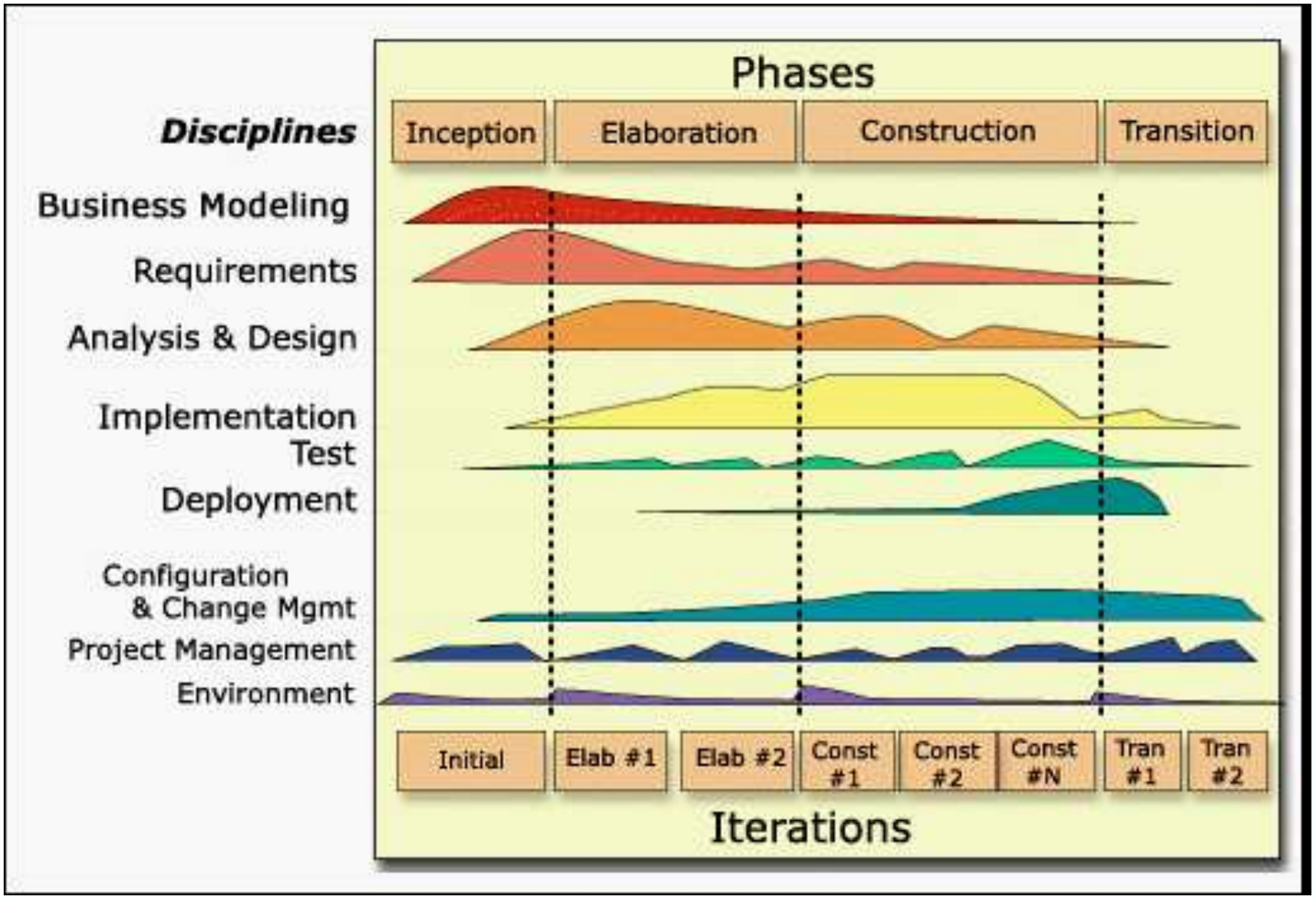
Diretrizes Arquiteturais do RUP

- Modularizar o sistema – componentes coesos;
- Destacar projeto de interfaces;
- Posicionar os componentes menores dentro de componentes maiores (especificação interna dos componentes arquiteturais);
- Reduzir o acoplamento entre componentes.

Conceitos do RUP

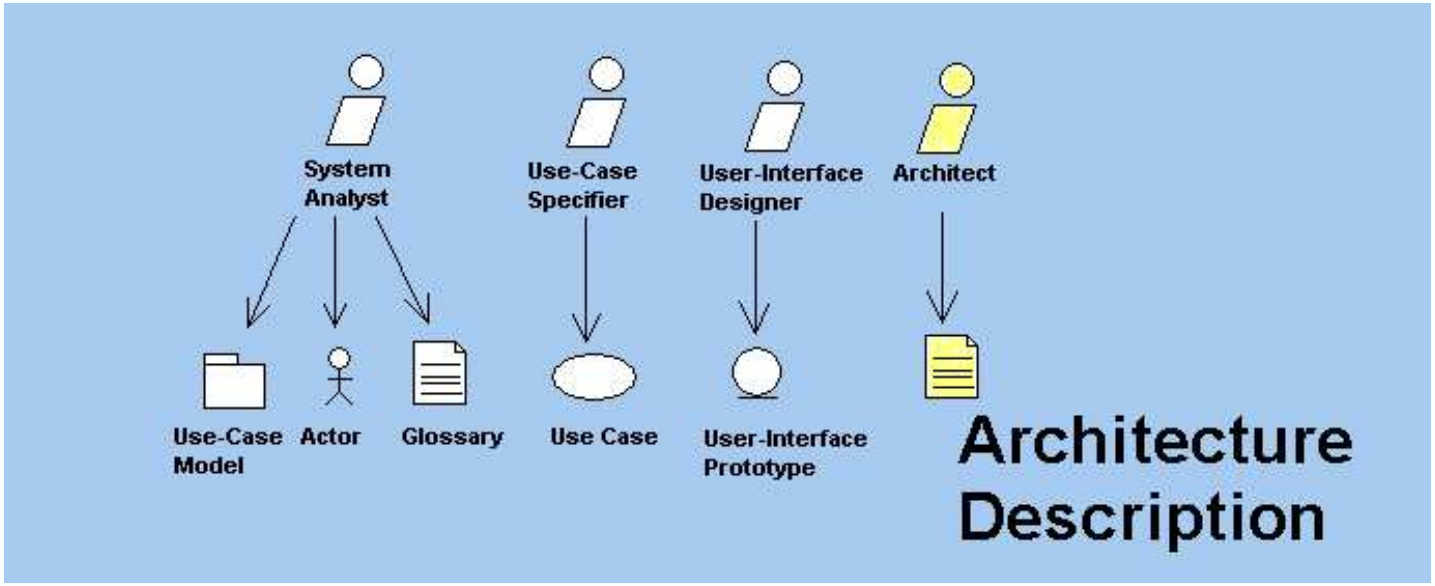
- *Architecture Baseline* (esqueleto);
- Descrição da Arquitetura (visões);
- Padrões Arquitetônicos (estilos);
- Padrões de Projeto.

Fases e Fluxos do RUP

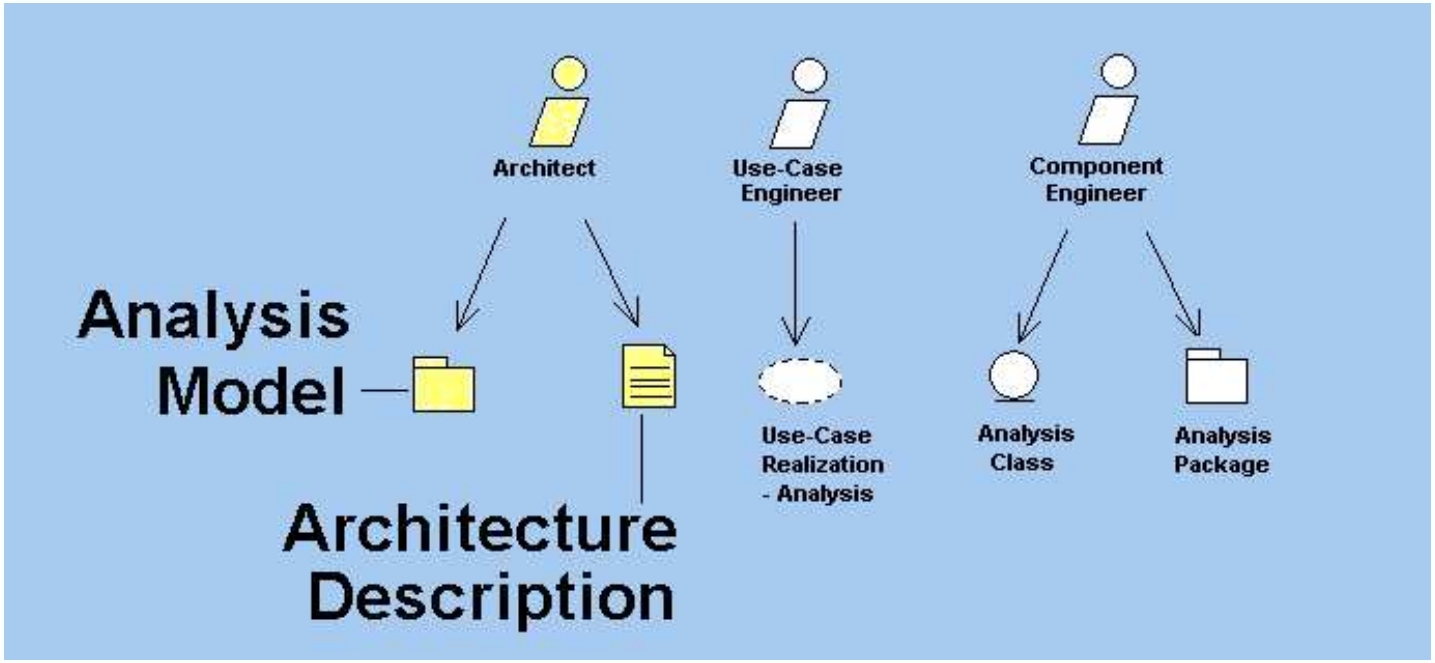


Como torná-lo um processo de DBC?

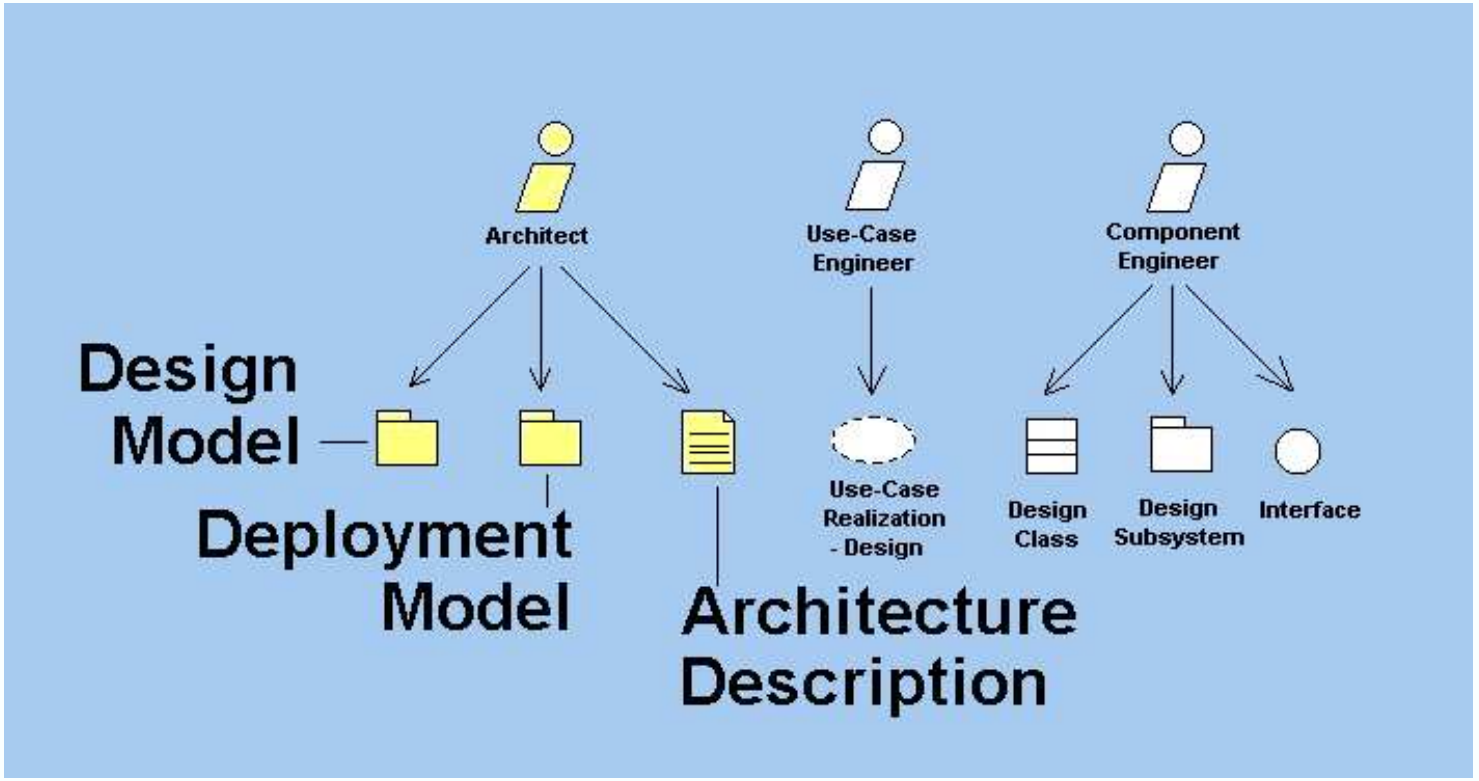
Especificação de Requisitos



Análise


Architect
Use-Case Engineer
Component Engineer
Analysis Model
Use-Case Realization - Analysis
Analysis Class
Analysis Package
Architecture Description

Projeto


Architect
Use-Case Engineer
Component Engineer
Design Model
Deployment Model
Use-Case Realization - Design
Design Class
Design Subsystem
Interface
Architecture Description

Implementação