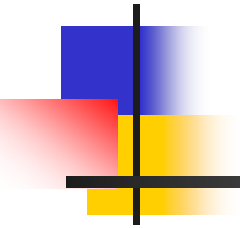
Desenvolvimento de Sistemas Orientados a Objetos com UML

UP/RUP: Projeto

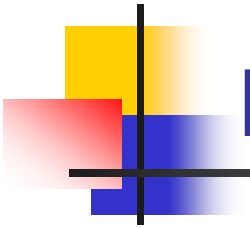


Engenharia de Software I

Informática

2009

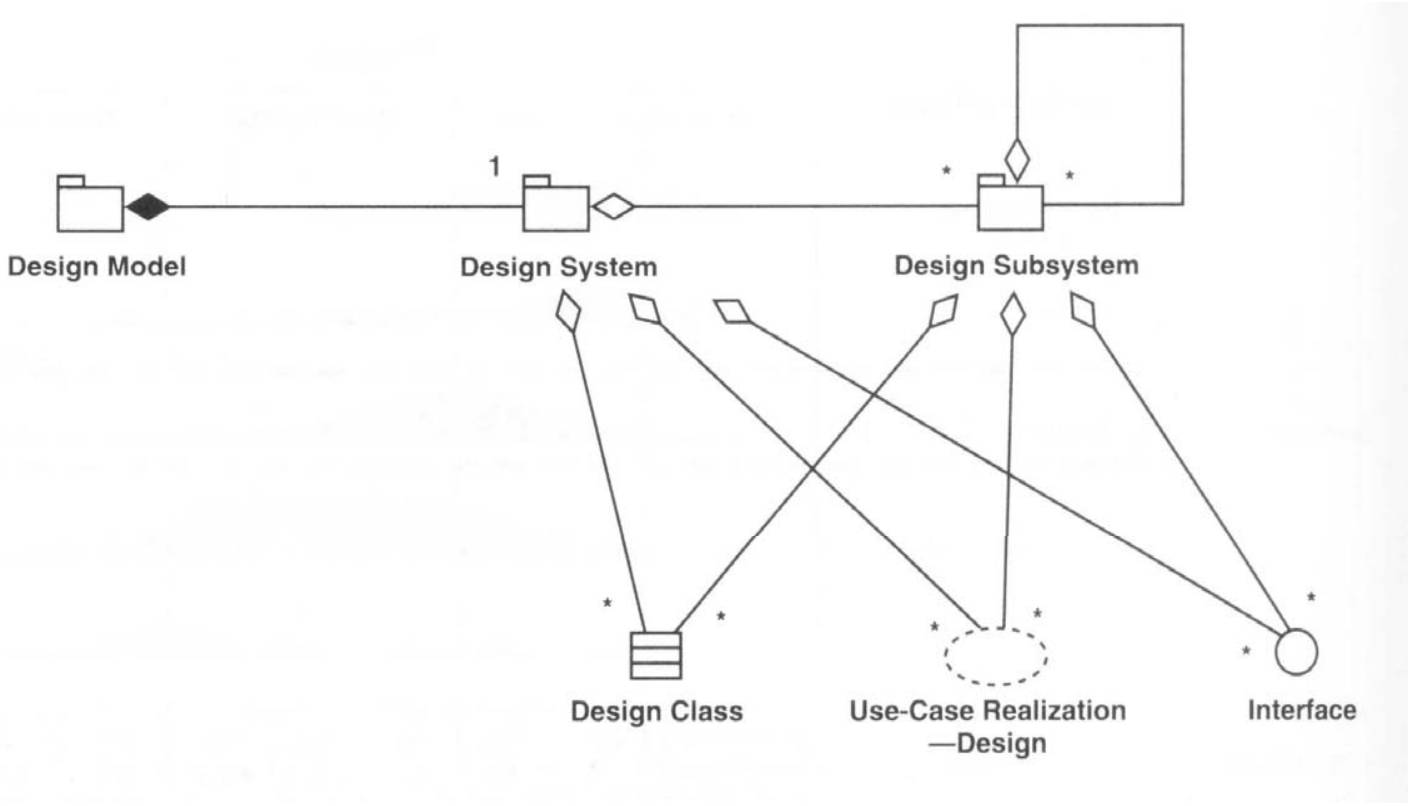
Profa. Dra. Itana Gimenes

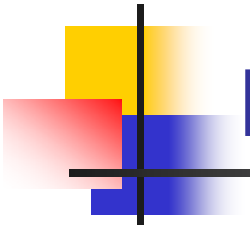


RUP: Artefatos de projeto

- Modelo de Projeto:
  - Use-Case Realization-projeto
    - Diagrama de classes
    - Diagrama de interação (sequência)
    - Diagramas de estado
  - Descrição da arquitetura
    - Subsistemas de projeto e serviço;
    - Interfaces;
  - Modelo de Instalação.

Modelo de Projeto



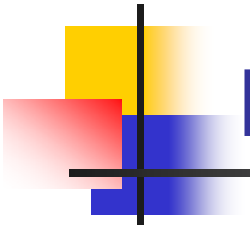


Modelo de Projeto

- Modelo de projeto (*design model*) – descreve a efetivação dos casos de usos focalizando em como os requisitos funcionais e não funcionais se relacionam com o ambiente de implementação.
- Modelo de projeto é representado por um sistema que denota o subsistema de mais alto nível, que por sua vez é composto de vários subsistemas.
- No modelo de projeto os casos de uso são efetivados por meio de classes de projeto e objetos.

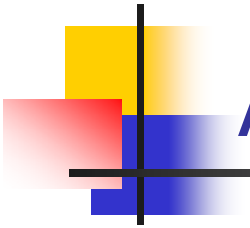
Modelo de Analysis versus Modelo de Projeto

Modelo de Análise	**Modelo de Projeto**
Modelo conceitual pois é uma abstração do sistema e evita questões de implementação.	Modelo físico pois é um esquema da implementação.
Projeto genérico (se aplica a vários projetos)	Não é genérico, é específico para uma implementação.
Três esteriótipos de classes: controle, entidade e interface.	Vários esteriótipos de classes, a depender da linguagem de programação.
Menos formal	Mais formal
Menor esforço de desenvolvimento (1:5 razão de projeto)	Maior esforço de desenvolvimento (5:1 razão para análise)
Poucas camadas	Muitas camadas
Dinâmica (mas não muito foco em seqüência)	Dinâmico (muito foco em seqüência)
Esquematiza o projeto do sistema, incluindo a arquitetura.	Manifesta o projeto do sistema, incluindo sua arquitetura (uma de suas visões).
Criado principalmente por trabalho braçal, em workshops ou similares.	Criado principalmente por programação visual no ambiente de desenvolvimento.
Pode não ser mantido ao longo do ciclo de vida do software.	Deve se mantido ao longo do ciclo de vida do software.
Define uma estrutura que se constitui uma entrada essencial para formulação do sistema.	Formula o sistema tentando preservar, tanto quanto possível, a estrutura definida no modelo de análise.



Papéis

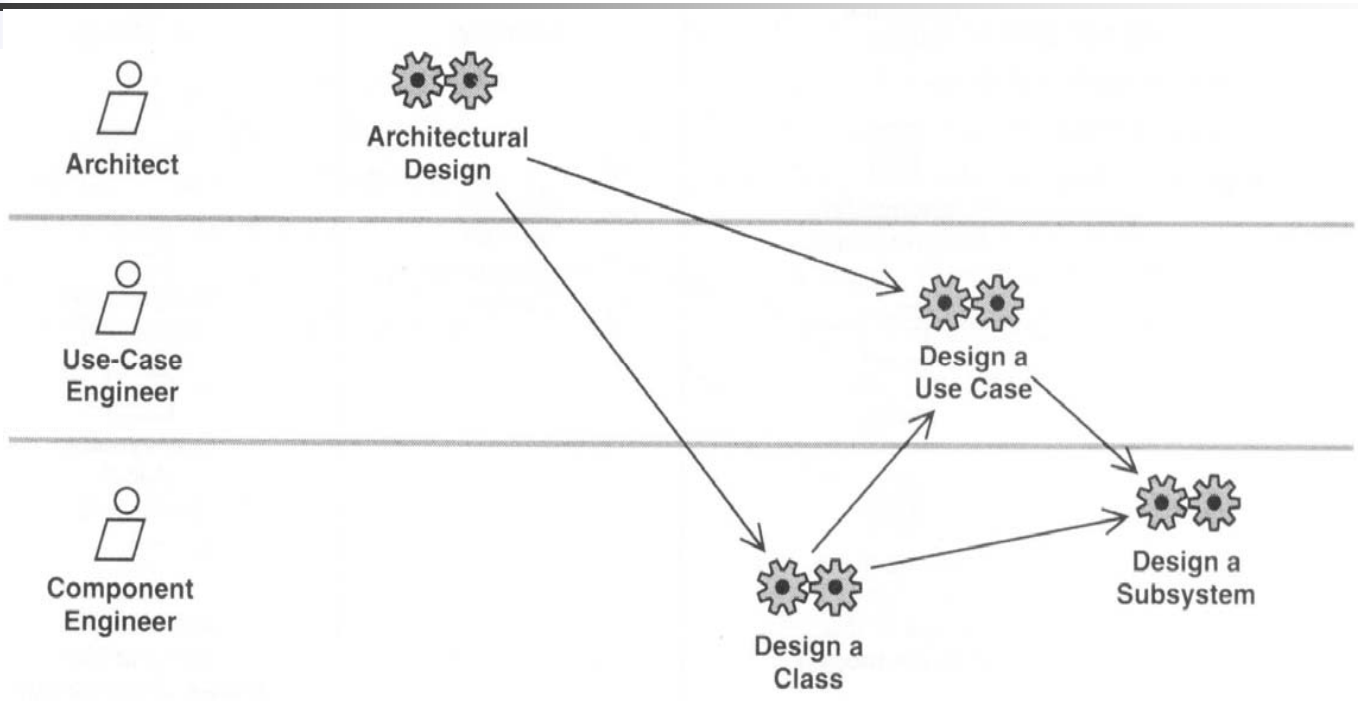
- Arquiteto
- Engenheiro de casos de uso
- Engenheiro de componentes



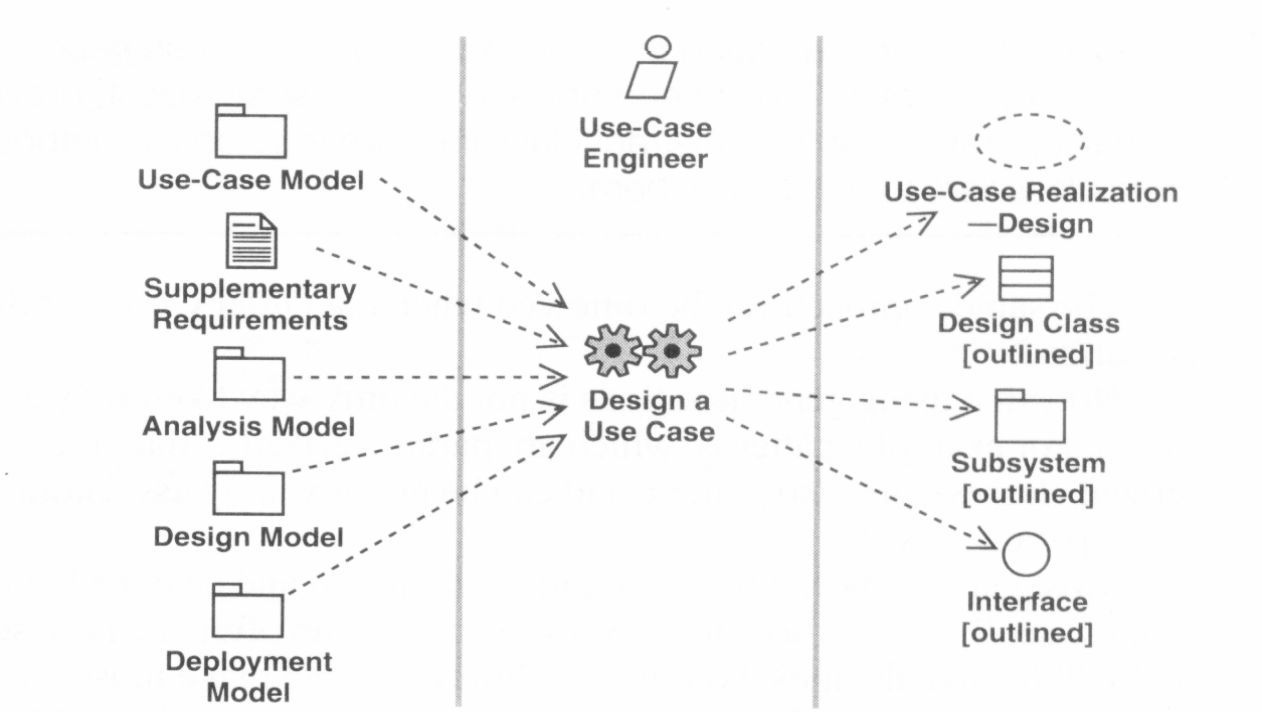
Atividades

- Projeto de casos de uso
  - Projeto de classes
  - Especificação de comportamento
- Projeto arquitetural
  - Projeto de subsistemas
  - Projeto do modelo de instalação

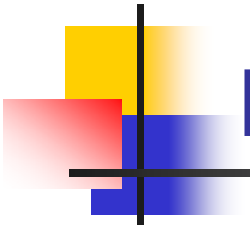
Workflow de projeto



Projeto de casos de uso (*Use Case Realization – Design*)

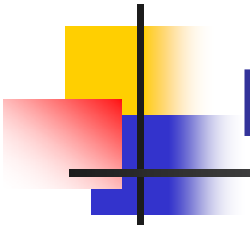


O Design model e o Deployment model aparecem na figura para mostrar que, nas iterações, eles são parte da entrada, mas eles não existem a primeira vez que se faz o modelo de projeto.



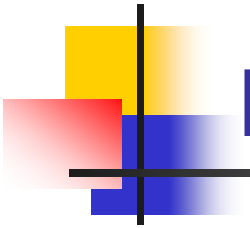
Projeto de casos de uso

- *Use case realization – design* é uma colaboração que faz parte do modelo de projeto descrevendo como um caso de uso específico é efetivado e executado em termos das classes de projeto e seus objetos.
- Um *Use case realization – design* é diretamento rastreado a partir de um *Use case realization – analysis*
- Conteúdo:
  - Descrição textual do fluxo de eventos
  - Diagrama de classes
  - Diagramas de interação (diagrama de seqüência)
  - Requisitos de implementação



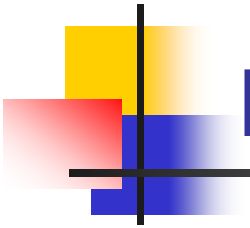
Projeto de casos de uso

- Identificar classes de projeto cujas instâncias são necessárias para efetivar o fluxo de eventos do caso de uso.
- Distribuir o comportamento do caso de uso entre os objetos de projeto que interagem ou participam do subsistema.
- Definir os requisitos das operações sobre as classes de projeto.
- Capturar os requisitos de implementação do caso de uso.



Projeto de classes

- Classes de projeto são abstrações da classe de implementação.
- A linguagem utilizada para especificar uma classe de projeto é a mesma da classe de implementação.
- A forma de acesso (public, private, protected) dos atributos e classes são normalmente especificados, mas também podem ser adiadas para o workflow de implementação.
- Os relacionamentos entre classes são geralmente mapeados diretamente do projeto para implementação.
- Métodos das classes de projeto são diretamente mapeados para implementação.
- Alguns requisitos são transferidos para a implementação (ex. relacionadas com codificação).



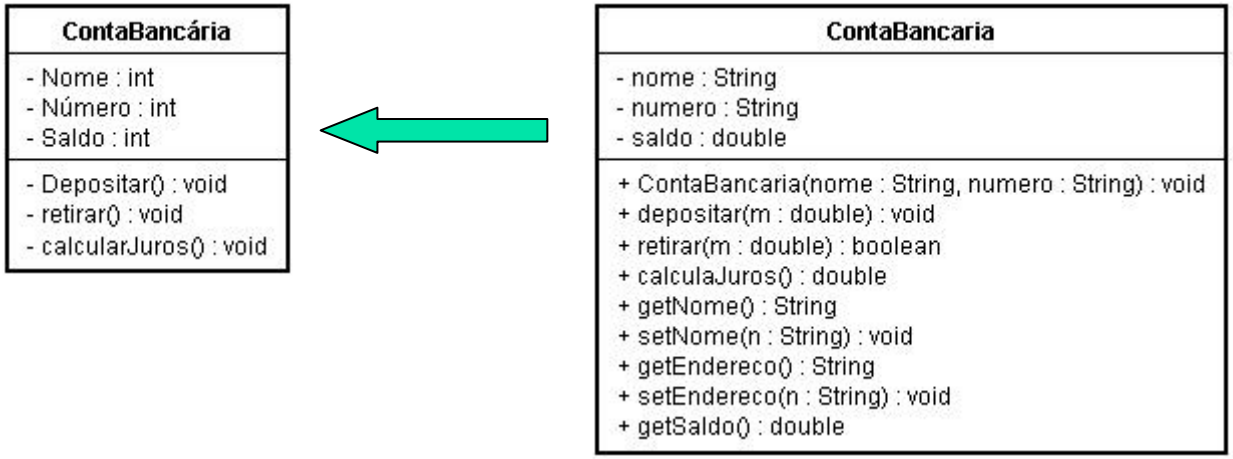
Projeto de classes

- Atributos
- Operações (métodos que implementam as operações)
- Relacionamentos que participa
- Estados (Ex. uma fatura pendente ou paga)
- Dependências de algum mecanismo genérico de projeto (Ex. exigência de um mecanismo de persistência)
- Requisitos relevantes para implementação

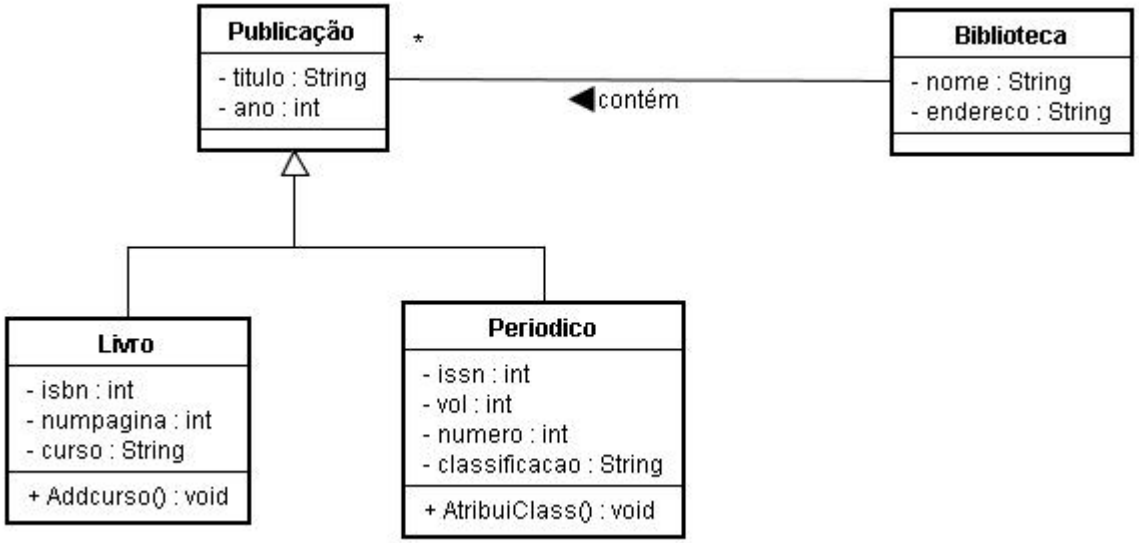
Exemplo Classes Análise x Projeto

Análise

Projeto

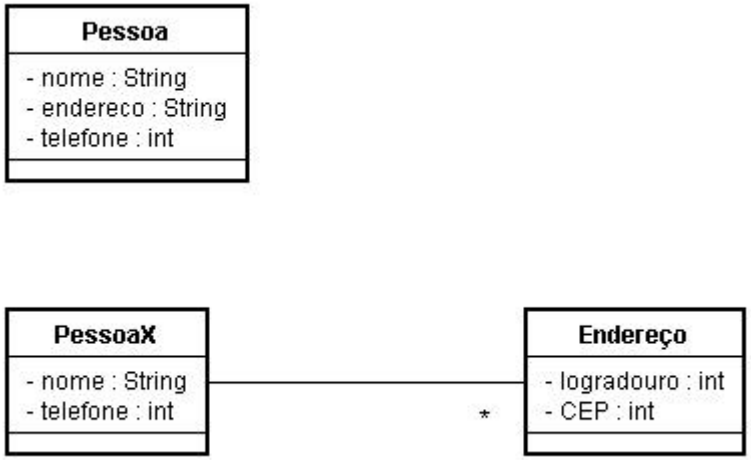


Exemplos de classes



Projeto de classes

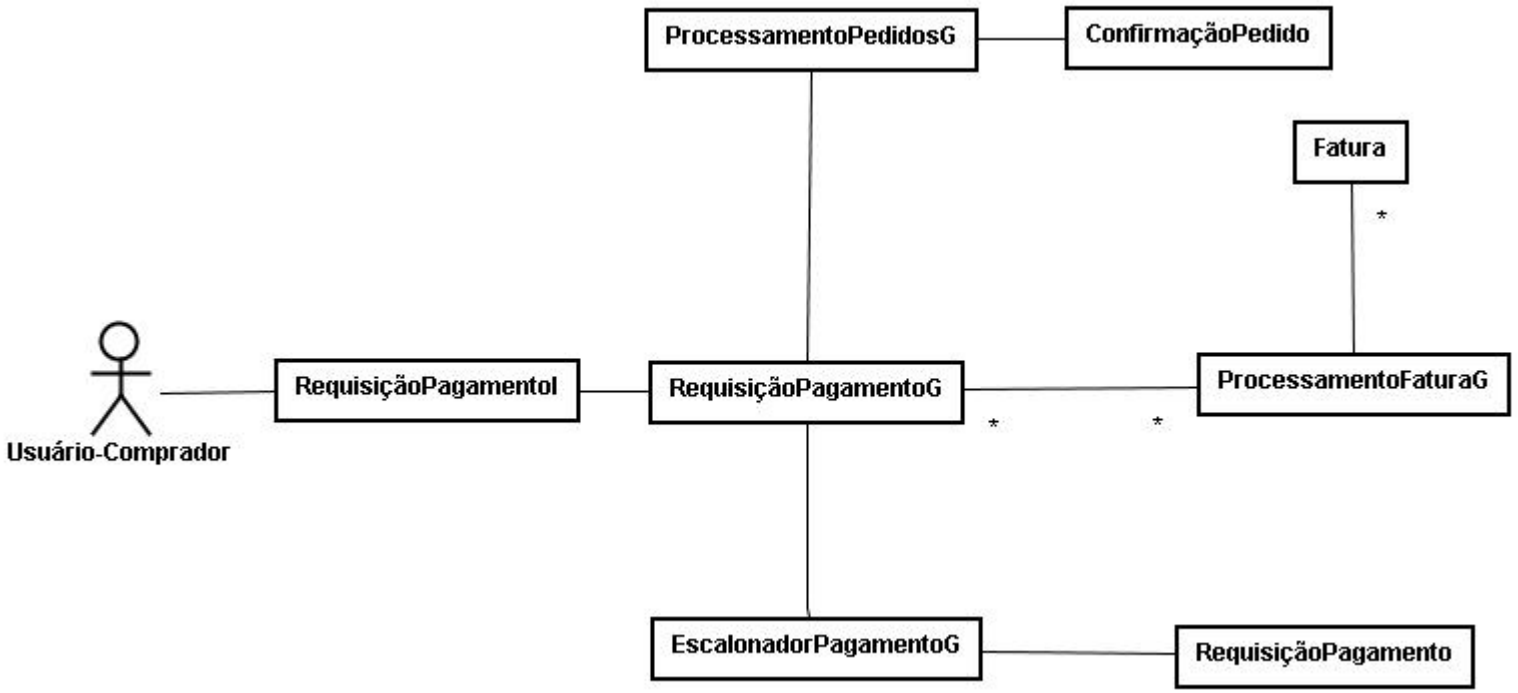
- Decisões de projeto afetam o projeto das classes principalmente requisitos não funcionais (eficiência, memória, uso de frameworks, etc.).



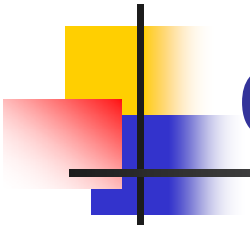
Capturando Requisitos de Implementação

- Exemplo: um objeto da classe (ativa) Payment Request Processing manipula 10 clientes compradores sem um atraso perceptível para nenhum deles.

Exemplo das classes participantes do caso de uso “Pagar Fatura”



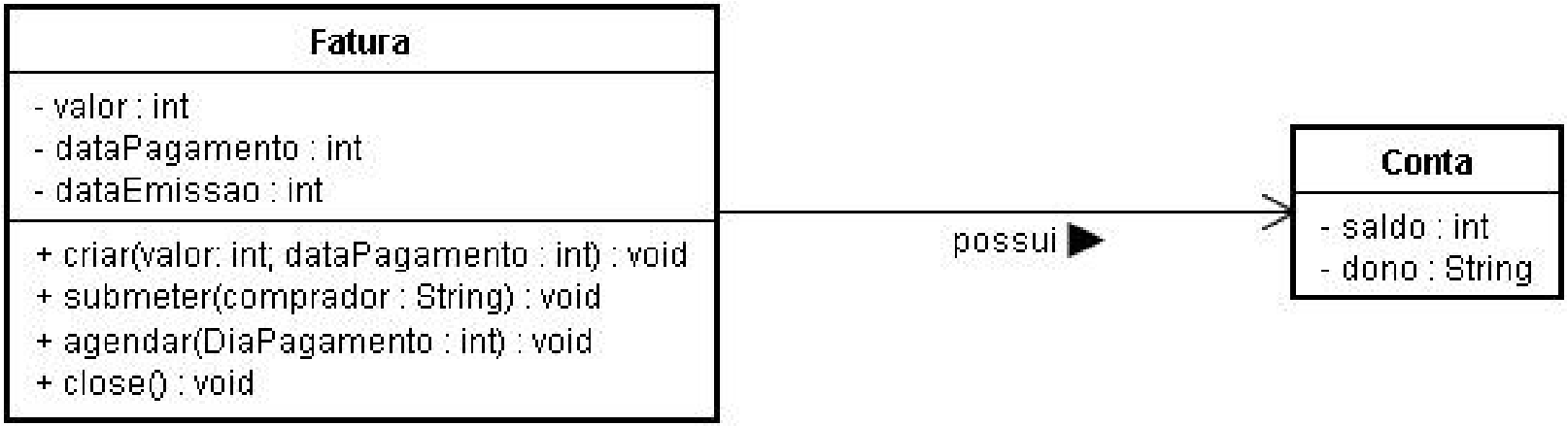
* As classes devem conter atributos e métodos.



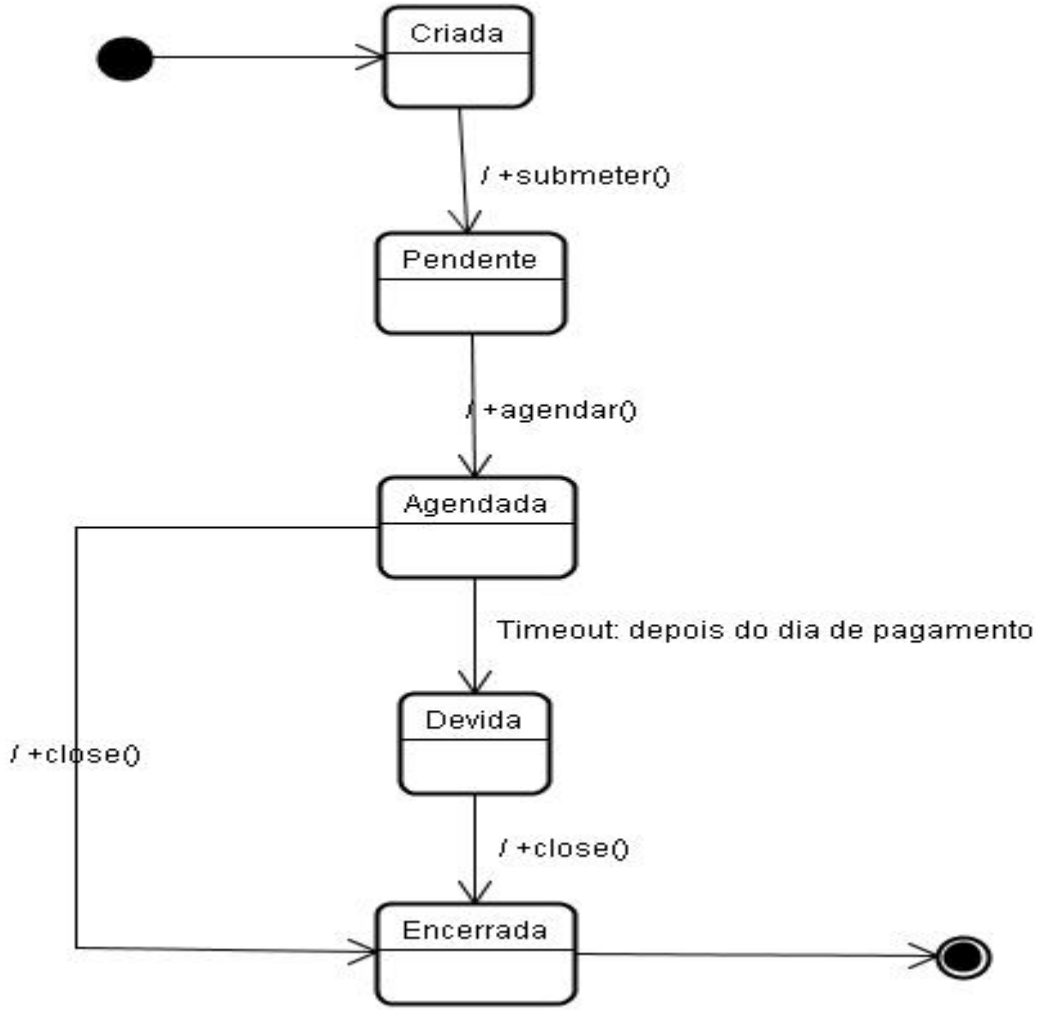
Comportamento das classes

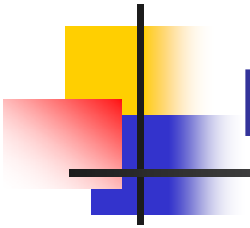
- Classes mais complexas demandam especificação de comportamento, ex. Classes controladora, classes de sistemas em tempo real.
- Objetos reativos respondem a estimulos externos, geram respostas aos estimulos, tem um ciclo de vida modelado com estados, eventos e transições, seu comportamente corrente depende do passado.
- O comportamento de uma classe pode ser especificado por um diagrama de estados UML.

Ex: Comportamento das classes – diagramas de estado



Ex: Comportamento das classes – diagramas de estado

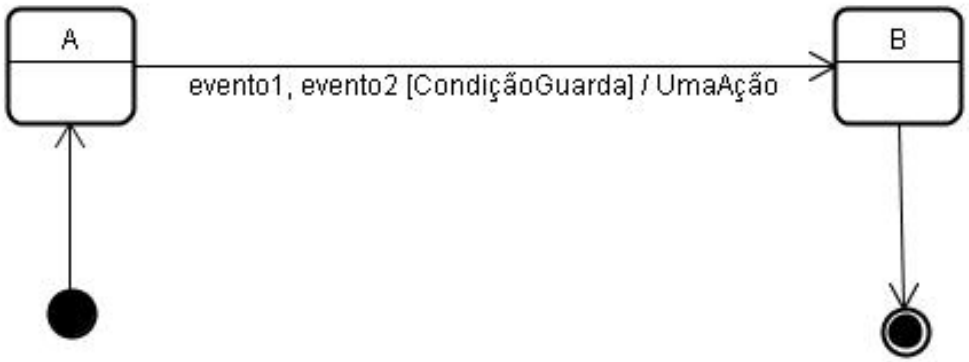




Diagramas de estado

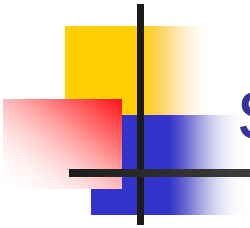
- Modela o ciclo de vida de um objeto como uma máquina de estados finitos. A máquina representa as transições entre estados em resposta a eventos.
- Elementos:
  - Estado: uma situação na vida do objeto que satisfaz uma condição, realiza uma atividade e espera por um evento.
  - Evento: a especificação de uma ocorrência relevante que tem tempo e espaço defindo.
  - Transição: o movimento de um estado para outro em resposta a um evento.

Ex: diagrama de estados



- Zero ou mais eventos – especificam as ocorrências externas que disparam uma transição.
- Zero ou mais condições de guarda – uma expressão booleana que deve ser avaliada antes que a transição ocorra. É colocada após o evento.
- Zero ou mais ações – são operações associadas com a transição que são executadas quando a transição é disparada.

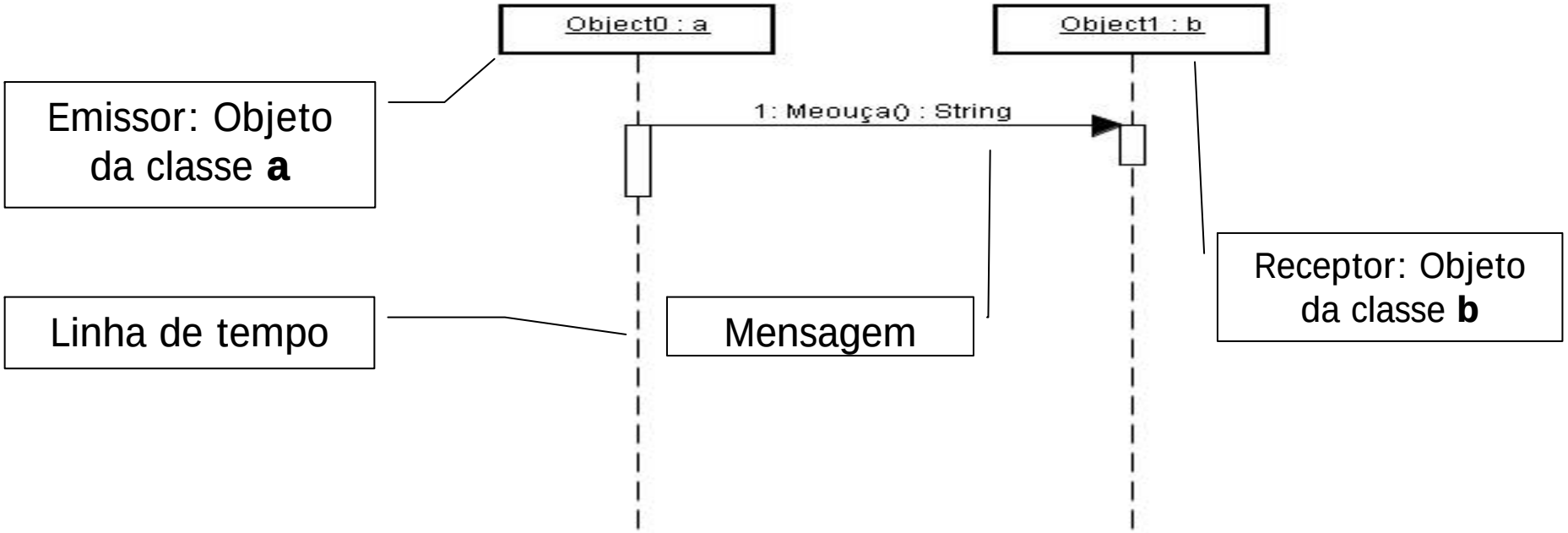
Interação entre objetos: diagrama de sequência



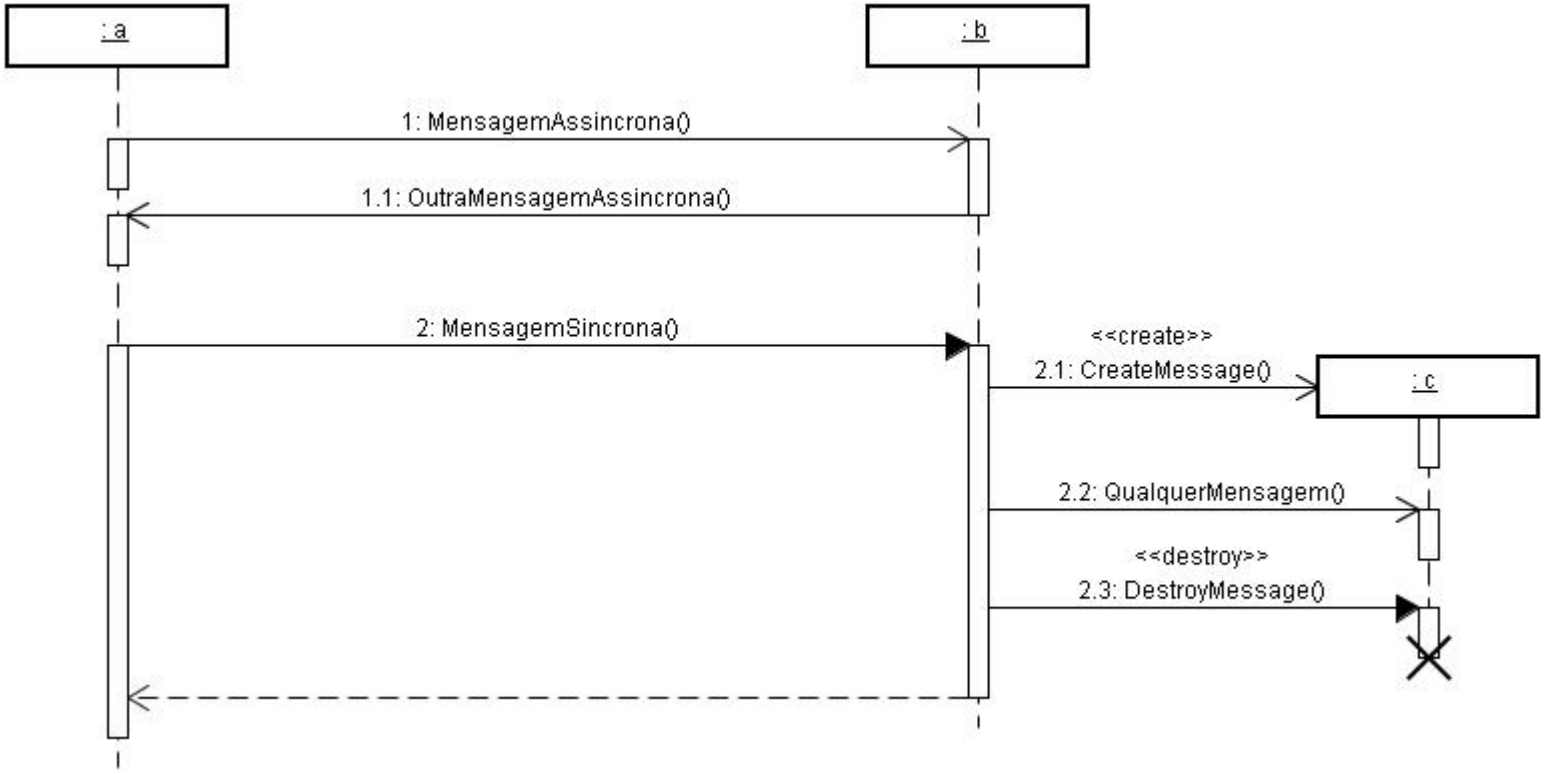
- Para representar a interação entre objetos podemos usar: diagrama de comunicação ou diagrama de sequência.
- O diagrama de sequência representa a troca de mensagens entre objetos para executar um caso de uso.
- O diagrama de sequência tem duas dimensões: tempo e papel dos objetos na colaboração.

Exemplos

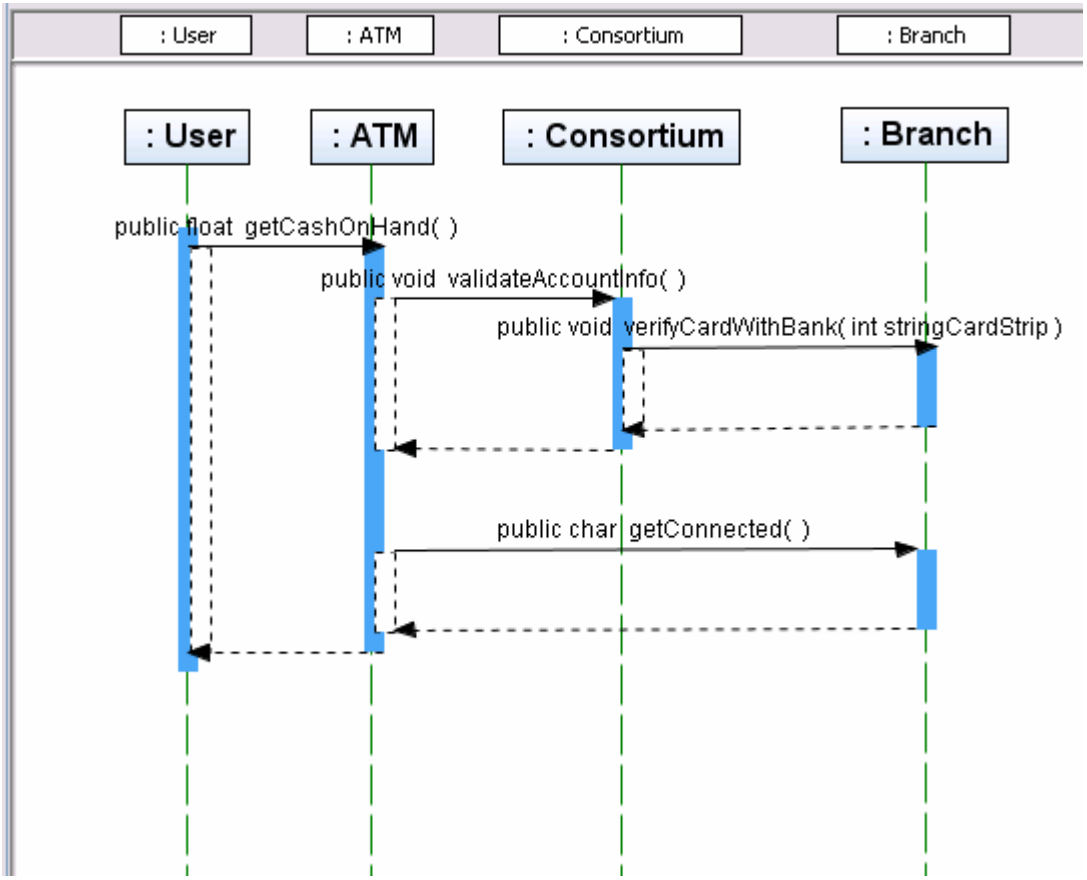

a
conversa
b
+ Meouça() : String

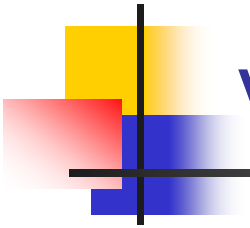

Object0 : a
Object1 : b
1: Meouça() : String
Emissor: Objeto da classe **a**
Linha de tempo
Mensagem
Receptor: Objeto da classe **b**

Elemento de um diagrama de sequência

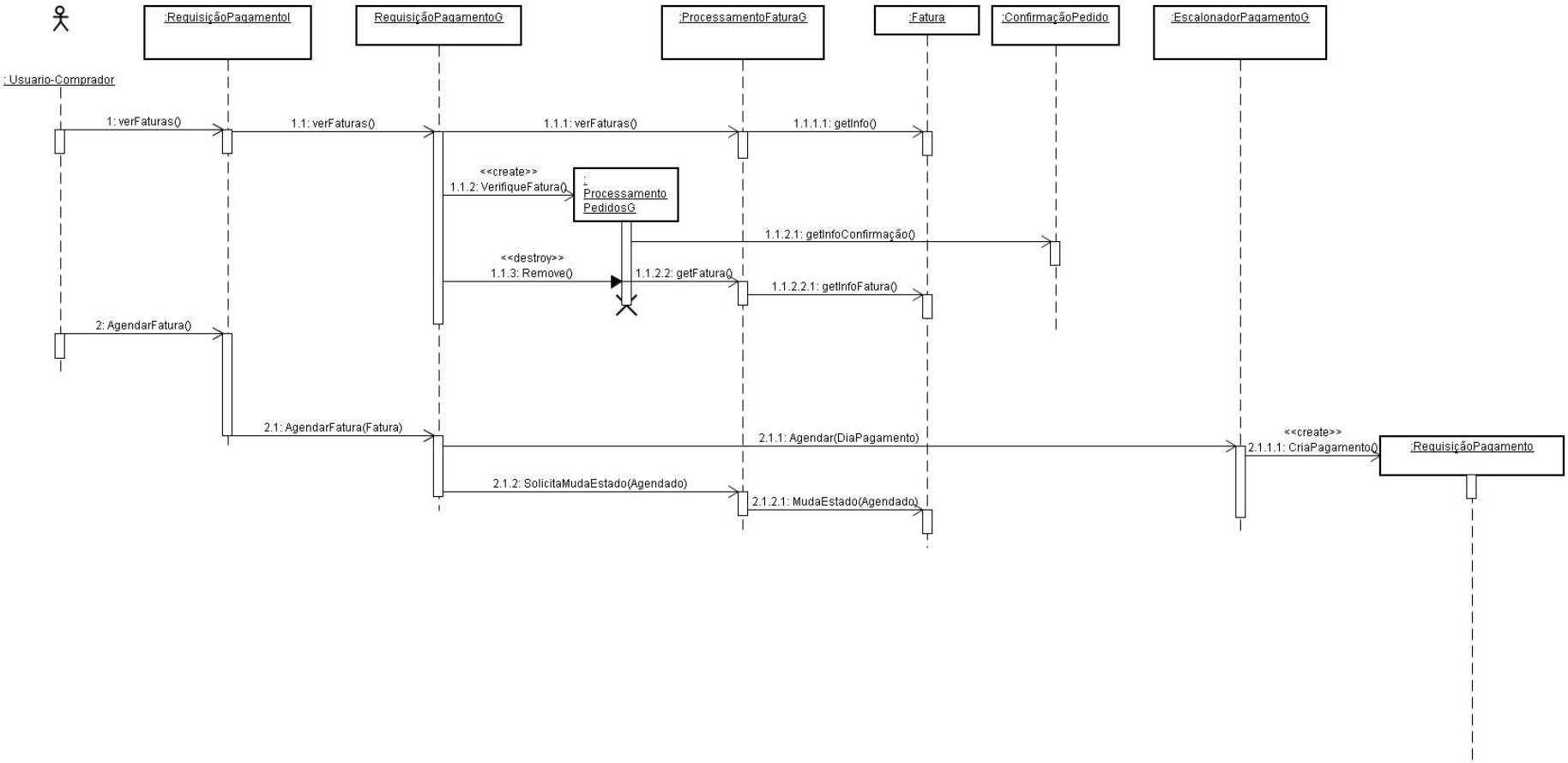


Exemplo: http://www.netbeans.org/kb/55/uml-sequence-diagram_pt_BR.html#using

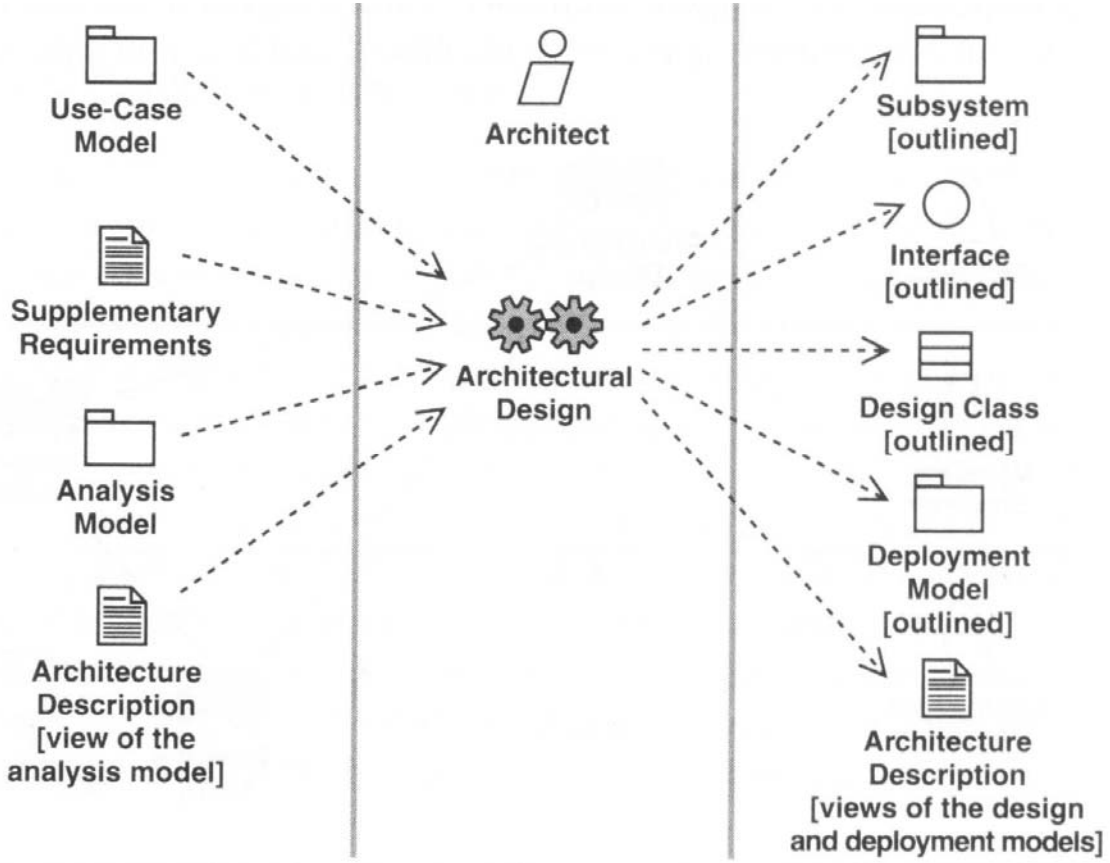




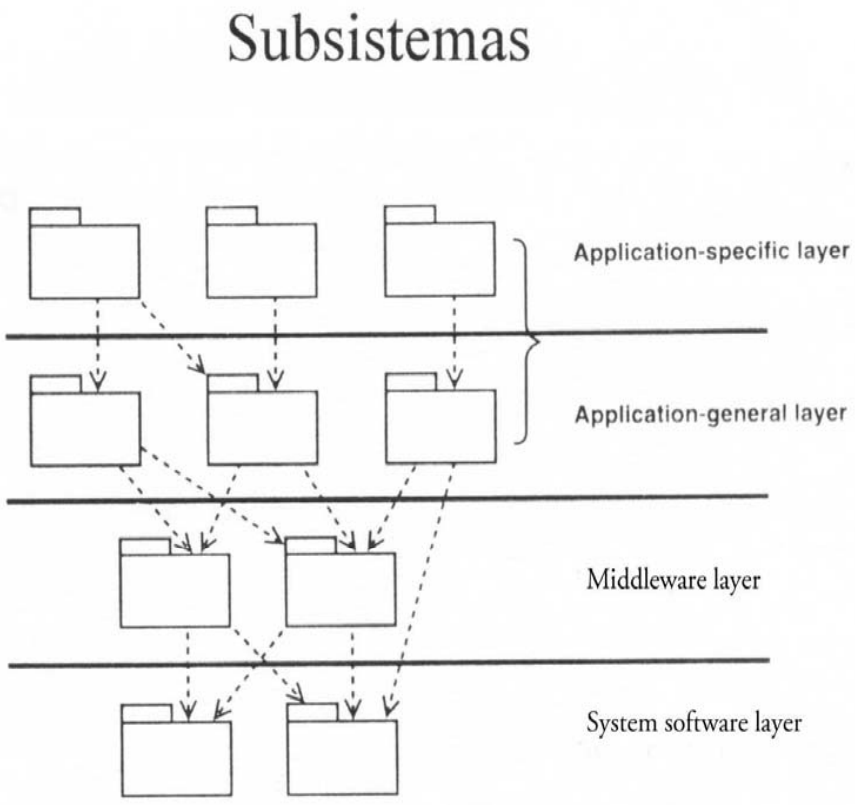
Ver Manual de referência

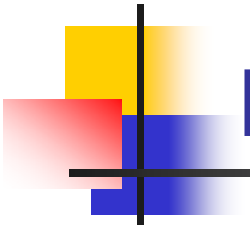

:RequisiçãoPagamentoI
RequisiçãoPagamentoG
:ProcessamentoFaturaG
:Fatura
:ConfirmaçãoPedido
:EscalonadorPagamentoG
:Usuario-Comprador
1: verFaturas()
1.1: verFaturas()
1.1.1: verFaturas()
1.1.1.1: getInfo()
<<create>>
1.1.2: VerifiqueFatura()
:
ProcessamentoPedidosG
1.1.2.1: getInfoConfirmação()
<<destroy>>
1.1.3: Remove()
1.1.2.2: getFatura()
1.1.2.2.1: getInfoFatura()
2: AgendarFatura()
2.1: AgendarFatura(Fatura)
2.1.1: Agendar(DiaPagamento)
<<create>>
2.1.1.1: CriaPagamento()
:RequisiçãoPagamento
2.1.2: SolicitaMudaEstado(Agendado)
2.1.2.1: MudaEstado(Agendado)

Projeto arquitetural



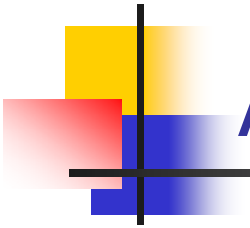
Camadas da arquitetura de projeto





Projeto arquitetural

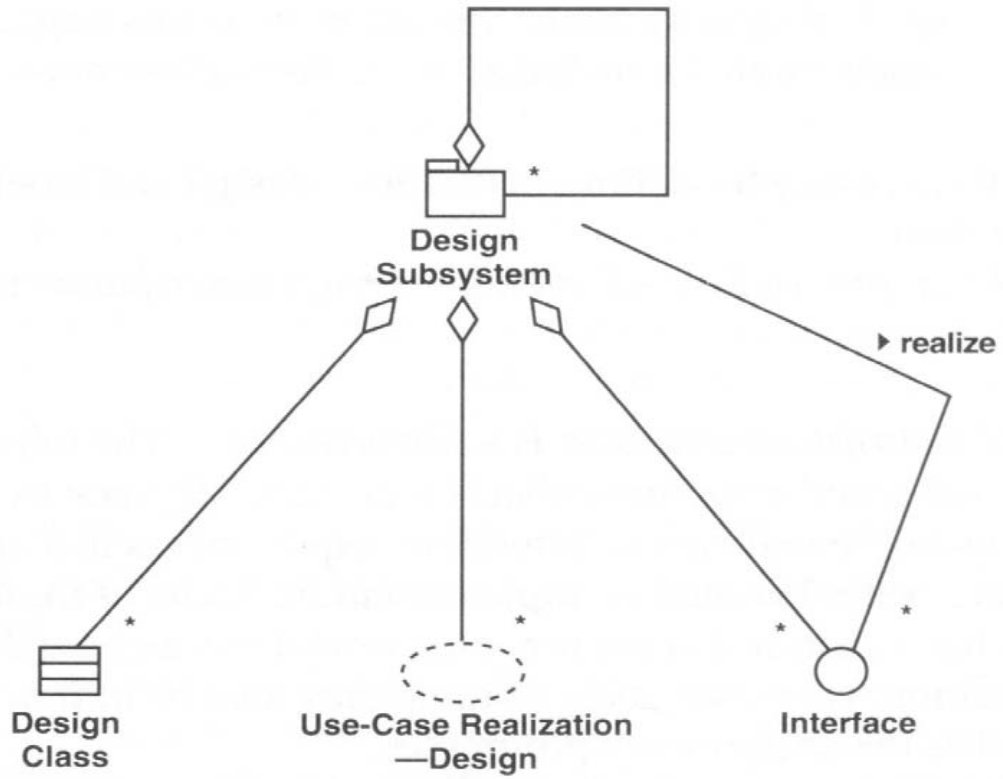
- Subsistemas e suas interfaces
- classes significantes para a arquitetura do sistema, ex. controladores
- mecanismos genéricos de projeto, ex. persistência, distribuição, desempenho, etc.
- possibilidades de reutilização são consideradas
- Nodos e configurações de rede

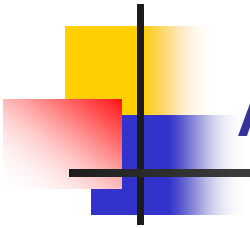


Arquitetura

- Visão arquitetural do modelo de projeto.
- Artefatos mais significantes do ponto de vista arquitetural:
  - Decomposição do modelo de projeto em subsistemas, suas interfaces e dependências.
  - Classes de projeto chaves: classes que correspondem a classes de analises significativas do ponto de vista arquitetural, classes ativas e classes de projeto que representam mecanismos de projeto e tem muitas relacionamentos com outras classes.
  - *Use case realization – design* – que representam funcionalidade criticas/chaves.

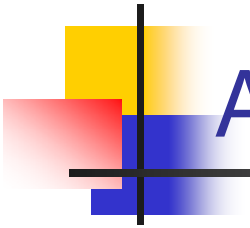
Artefato: Subsistema de projeto





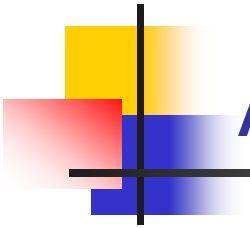
Arquitetura: subsistema de projeto

- Subsistemas fornecem um meio de organizar os artefatos do modelo de projeto em partes gerenciáveis.
- Subsistemas podem representar separação de subdomínios. Ex. Empréstimos de livros e cadastramento de usuários.
- As duas camadas de mais alto nível e seus subsistemas tem rastreamento direto para o modelo de análise.
- Um subsistema consiste de classes de projeto, *use case realizations*, interfaces e outros subsistemas recursivamente.
- Nem todos os subsistemas são desenvolvidos “in-house” pelo projeto corrente.
- Reutilizações podem se dar na forma de frameworks, componentes COTS, padrões de projeto.
- Subsistemas podem representar software legado.
- Subsistemas podem representar componentes de grandes granulosidades na implementação do sistema.



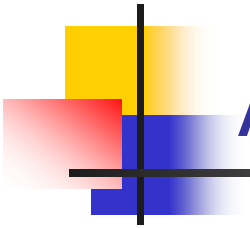
Arquitetura: subsistema de projeto

- Passos:
  - identificação de subsistemas da aplicação a partir da análise
  - identificação do middleware e sistemas de software básicos
  - definição das dependências de subsistemas
  - identificação das interfaces do subsistema



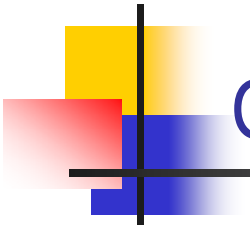
Arquitetura: subsistema de serviços

- Subsistemas de serviço são utilizados nos níveis mais baixos da hierarquia de subsistemas.
- A identificação dos subsistemas de serviços é baseada nos pacotes de análise do modelo de análise.
- Subsistemas de serviços podem fornecer serviços em termos de interfaces e suas operações.
- Subsistemas de serviço contém classes de projeto, portanto tratam requisitos não funcionais e outras restrições relacionadas ao ambiente de implementação. Contém mais classes do que os pacotes de serviço.
- Um subsistema de serviço geralmente, leva a um componente executável. Porém, pode ser precisar ser decomposto em susbsistemas menores.



Arquitetura: Interfaces

- Interfaces especificam as operações fornecidas pelas classes e subsistemas de projeto.
- Classes fornecem métodos que efetivam as funcionalidades da interface. Subsistemas contém classes que efetivam as funcionalidades dos subsistemas.
- Interfaces viabilizam a separação da especificação das funcionalidades da implementação.

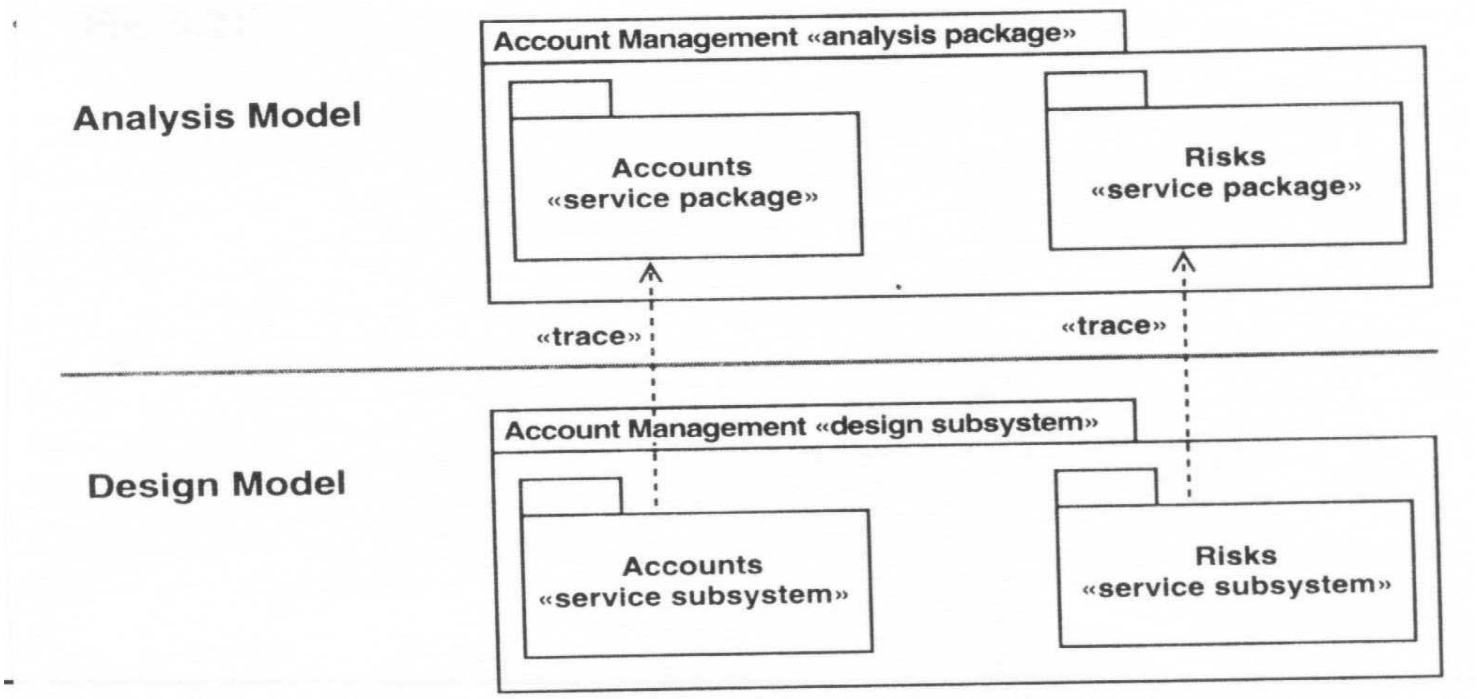


Considerações no projeto de Subsistema

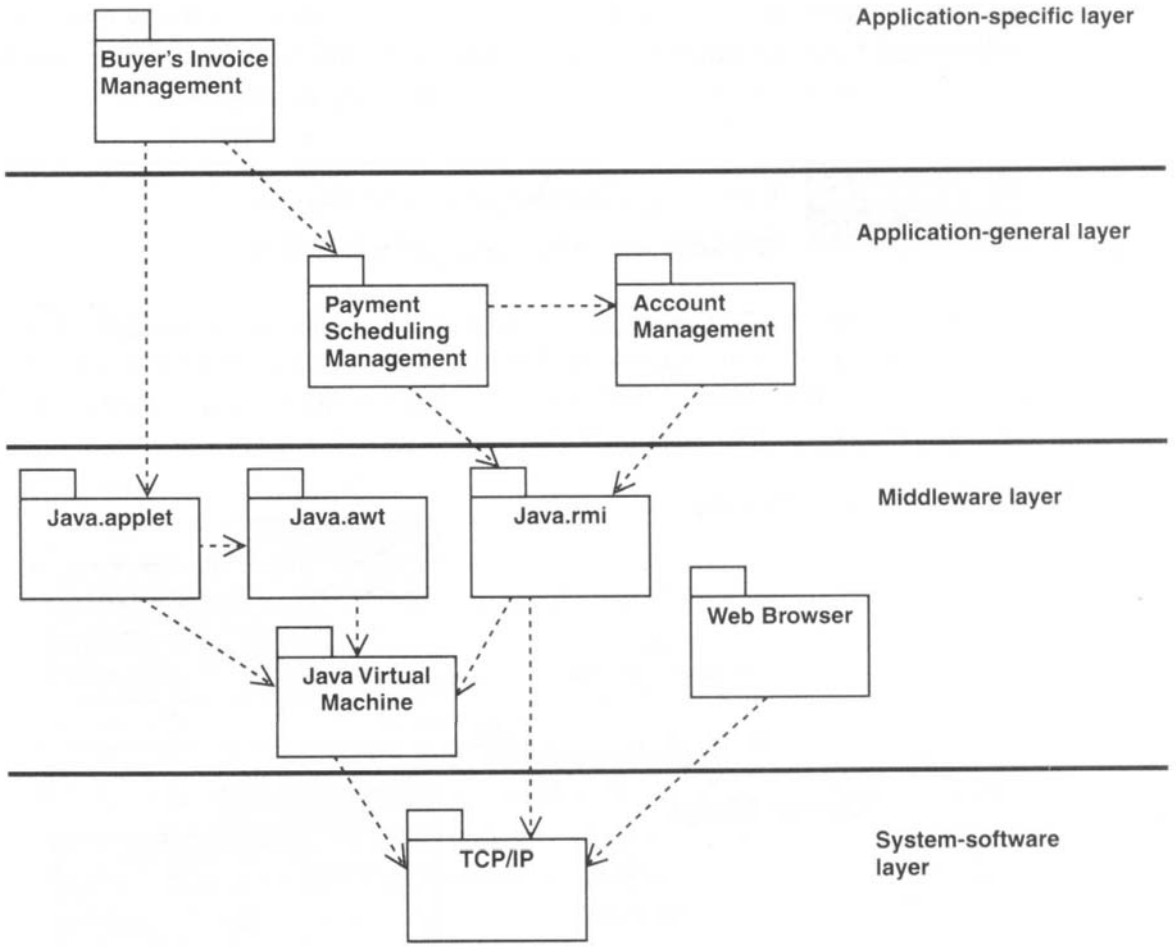
- Garantir que um subsistema é tão independente quanto possível: alta coesão e baixo acoplamento.
- Garantir que os subsistemas oferecem as interfaces corretas.
- Garantir que os subsistemas realizam as funcionalidades de forma satisfatória através de suas interfaces.

Exemplo de identificação de subsistemas

- Decomposição a partir dos pacotes de análise podem ser percebidas, por exemplo quando partes do subsistema podem ser compartilhadas por vários outros subsistemas.



Definindo dependências entre subsistemas



Identificando interfaces

- Dependências pois estas geralmente implicam em serviços oferecidos.
- Classes referenciadas dentro dos subsistemas.

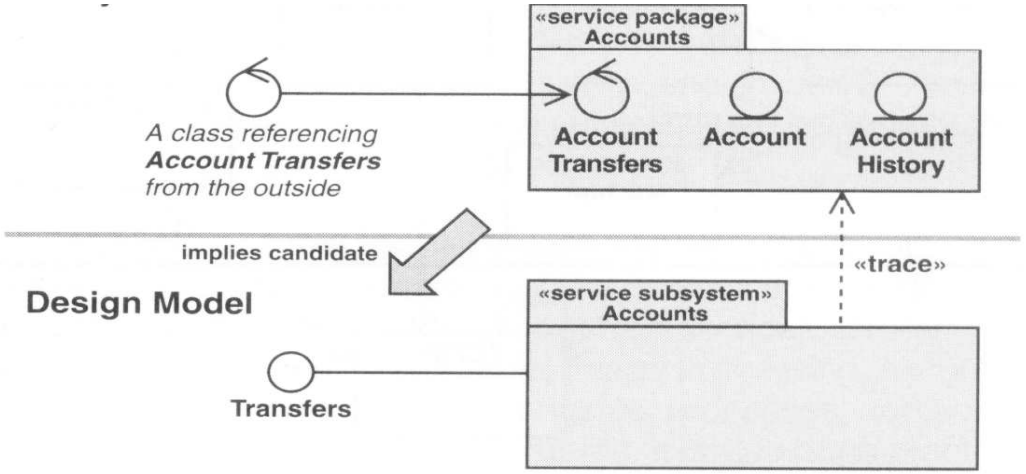


FIGURE 9.27 An interface initially identified based on the analysis model.

Identificando interfaces

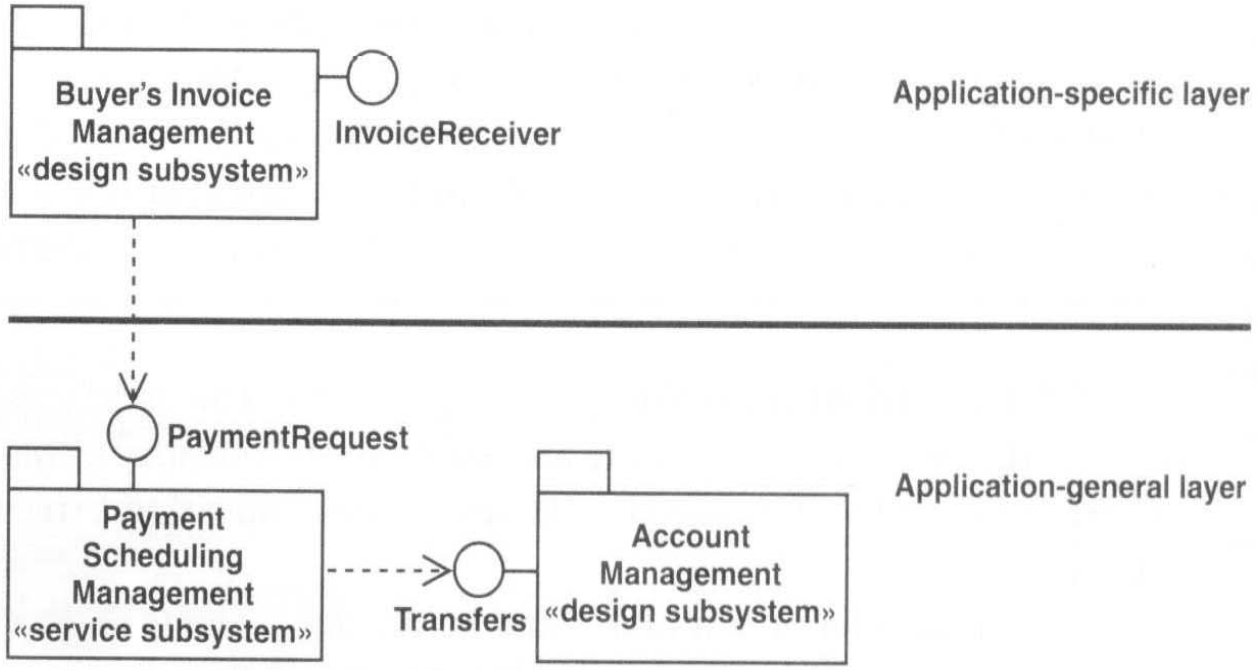
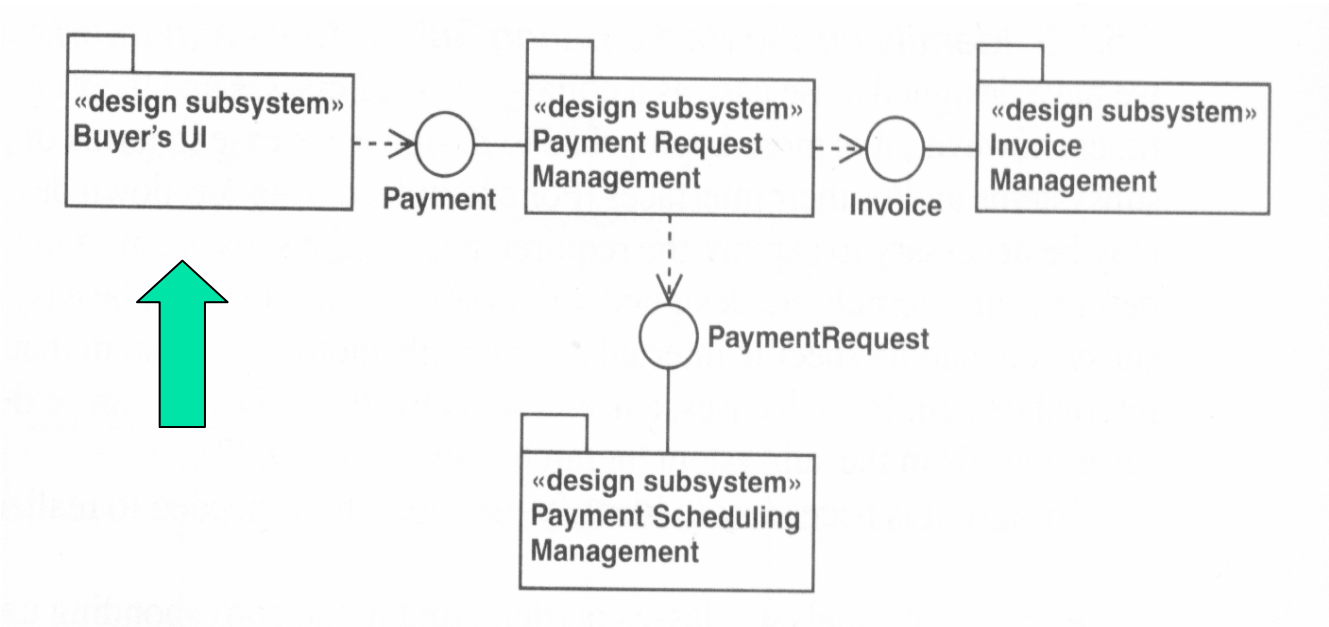
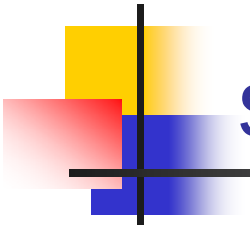


FIGURE 9.28 The interfaces in the two top layers of the design model.

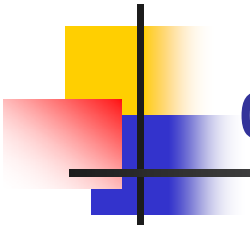
Identificando Subsistemas Participantes e Interfaces



Identificando Middleware e subsistemas de software

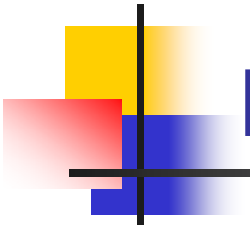


- Sistema operacional, gerenciador de banco de dados, software de comunicação, tecnologias de objetos distribuídos, kits de interface (GUI) e tecnologias para gerenciamento de transações.
- Escolher ou adquirir produtos de software?
- Controle da evolução de produtos adquiridos



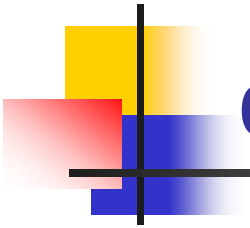
Identificando Middleware e subsistemas de software

- Persistência
- Distribuição transparente de objetos
- Segurança
- Detecção e recuperação de erros
- Gerenciamento de transações



Modelo de Instalação

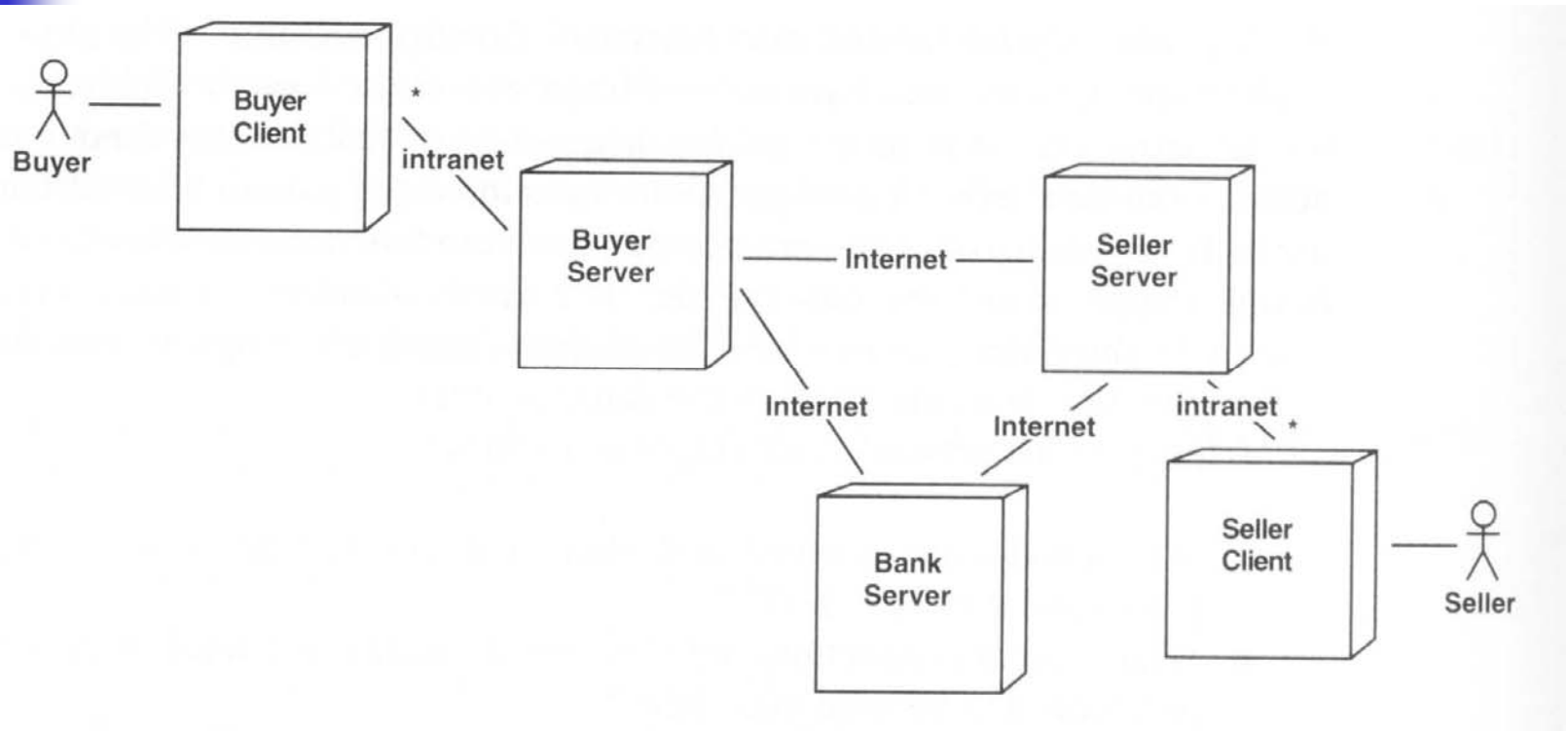
- É um modelo de objetos que descreve a distribuição física do sistema em termos de como a funcionalidade é distribuída entre os nodos computacionais.
- Exemplos de itens importantes relacionados ao modelo:
  - Cada nodo representa um recurso computacional, um processador ou dispositivo de hardware.
  - Nodos tem relacionamentos que representam meios de comunicação, como intranet, internet, bus.
  - Várias configurações de rede, incluindo teste e configurações podem ser incluidas.
  - Representa um mapeamento entre a arquitetura de software e a arquitetura do sistema.



Identificação de nodos e configurações de redes

- Quais são os nodos, suas capacidades de processamento e memória?
- Qual o tipo de conexões entre os nodos e quais são os protocolos de comunicação?
- Quais são as características das conexões e protocolos de comunicação, ex. bandwidth, disponibilidade, qualidade, etc.
- necessidade de redundância, tolerância a falhas, migração de processos, backup, etc. ?

Network configuration for the Interbank System



Distribuindo Subsistemas nos nodos