

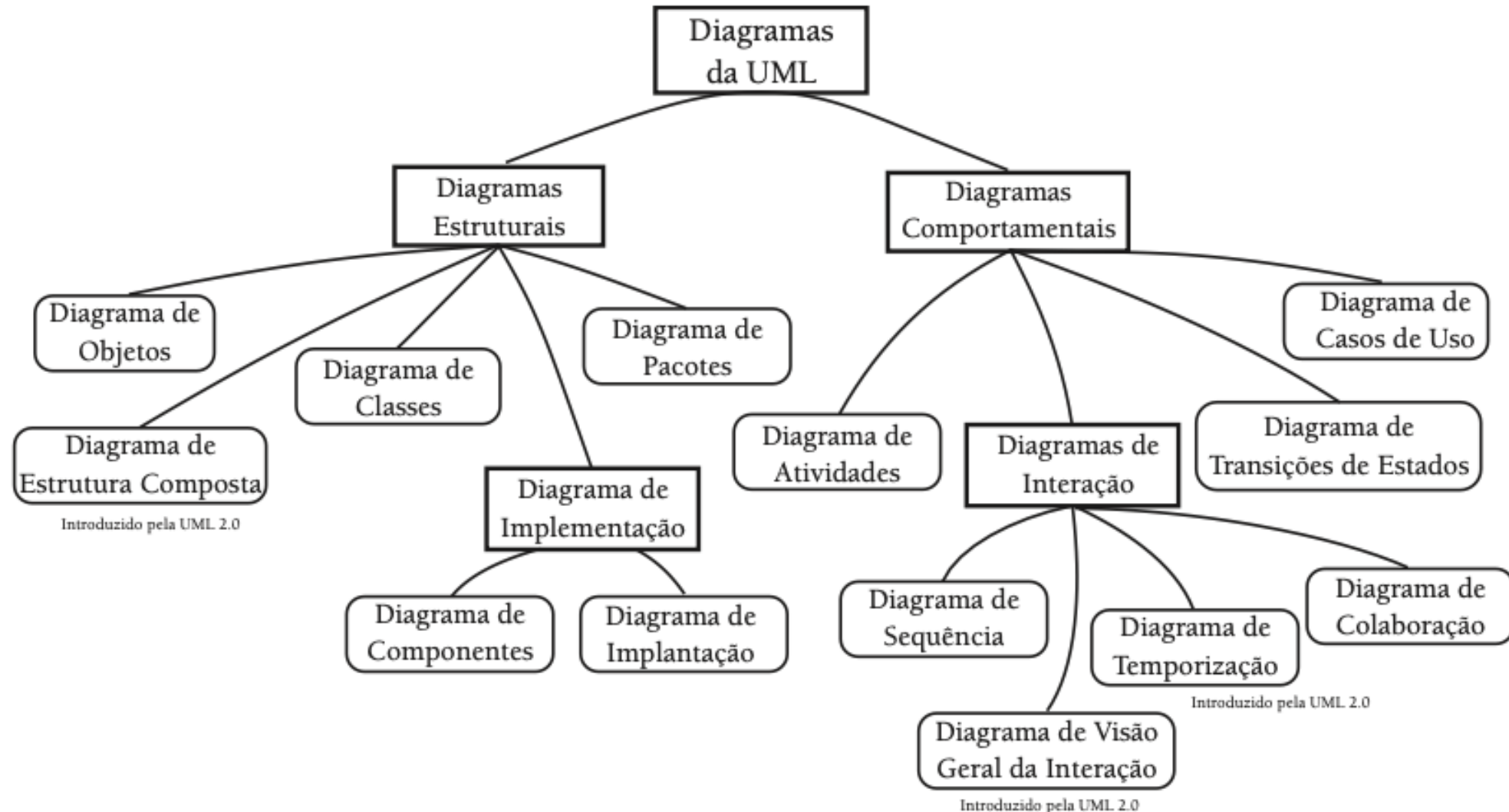
Modelagem Arquitetura de Software

João Pedro Oliveira Batisteli

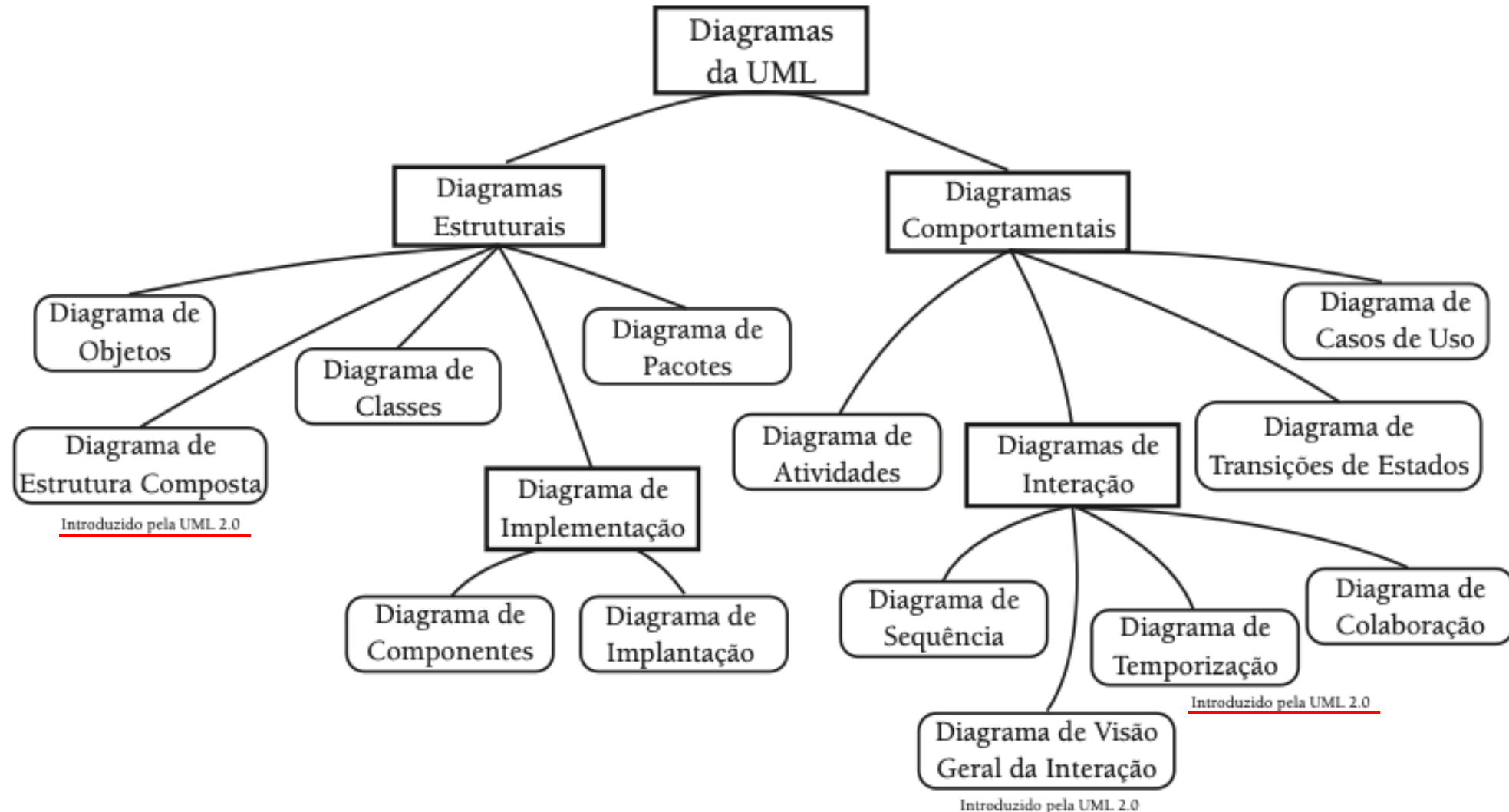
Bibliografia - UML

- LARMAN, Graig. Utilizando UML e Padroes: Uma introdução a análise e ao projeto orientados a objetos. Porto Alegre: Bookman, 3ª Edição, 2007. capítulo 39.
- BEZERRA, Eduardo. Princípios de analise e projeto de sistemas com UML. Rio de Janeiro: Campus, 2a Edição, 2007. capítulo 11.
- GUEDES, Gilleanes T. A. UML 2 : uma abordagem prática. São Paulo: Novatec, 1ª Edição, 2009.

Diagramas definidos pela UML

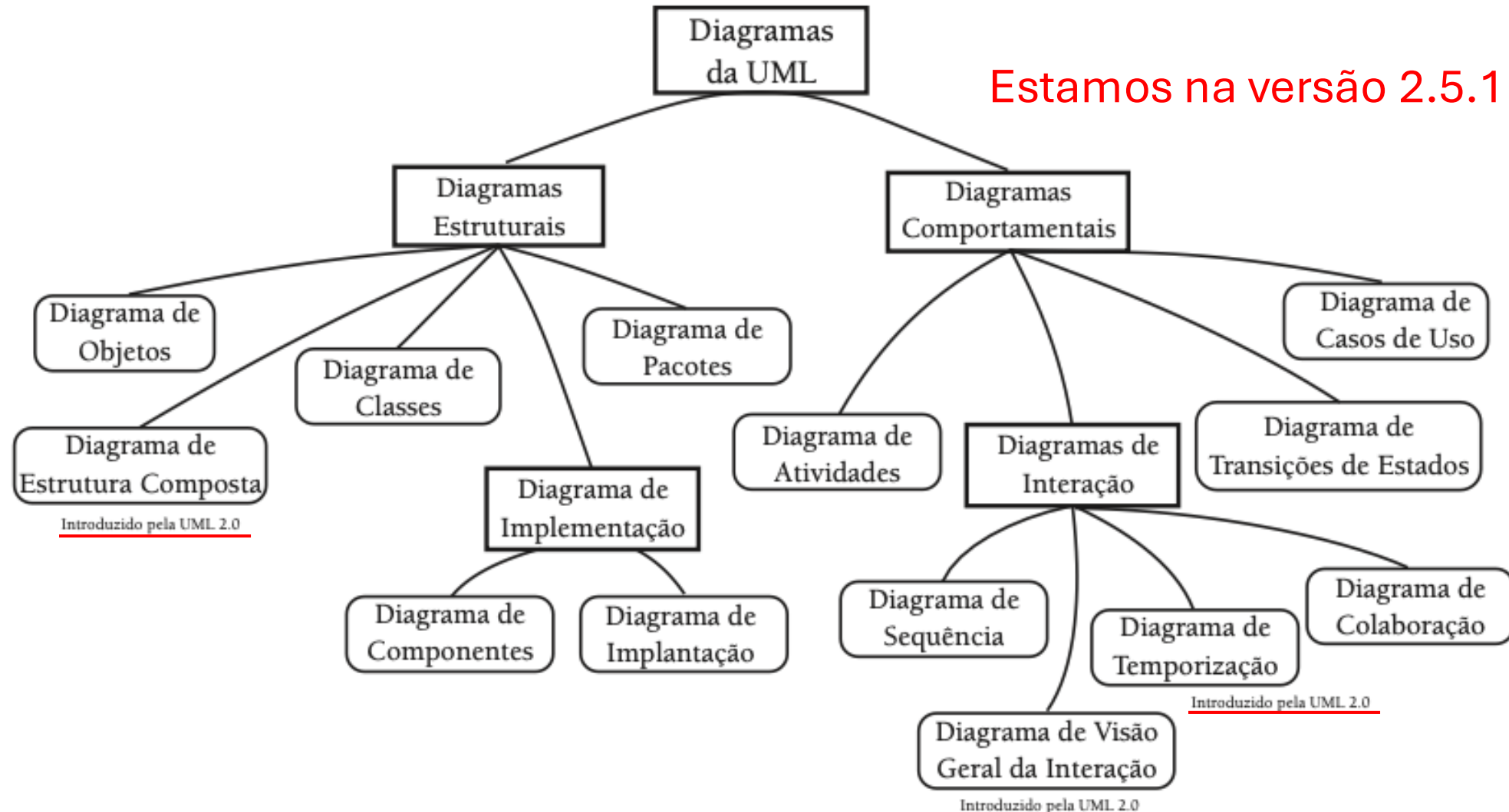


Diagramas definidos pela UML



Diagramas definidos pela UML

Estamos na versão 2.5.1 da UML!



Diagramas definidos pela UML

Diagramas

Qual a necessidade de
tantos diagramas?

Diagrama de Visão
Geral da Interação

Introduzido pela UML 2.0

Introduzido pela UML 2.0

Arquitetura de Software na UML

- Diagramas fornecem uma visão **bidimensional** do sistema que estamos desenvolvendo.
- Para compensar essa dimensão a menos, utilizam-se diversos diagramas para construir modelos de várias perspectivas do sistema.
- Cada um dos diagramas da UML fornece uma perspectiva parcial do sistema sendo modelado, consistente com as demais perspectivas.

Arquitetura de Software na UML

- Na UML os seguintes conceitos são usados para modelar a arquitetura:
 - **Visão Lógica (projeto)**
 - Pacotes
 - Subsistemas
 - Componentes
 - Interfaces
 - Camadas
 - **Visão de Implementação**
 - Diagrama de componentes
 - **Visão de Implantação**
 - Diagrama de implantação

Modelagem de Arquitetura Lógica

- **Objetivo:** Capturar a organização dos principais Elementos de um sistema e seus relacionamentos.
- **Diagramas Básicos:** Classes e Pacotes.

Diagrama de Classes

- As **classes** representam a forma básica de estruturação de um sistema orientado a objetos.
- Embora elas sejam úteis, é necessário algo mais para estruturar sistemas grandes, os quais podem ter **centenas de classes**.



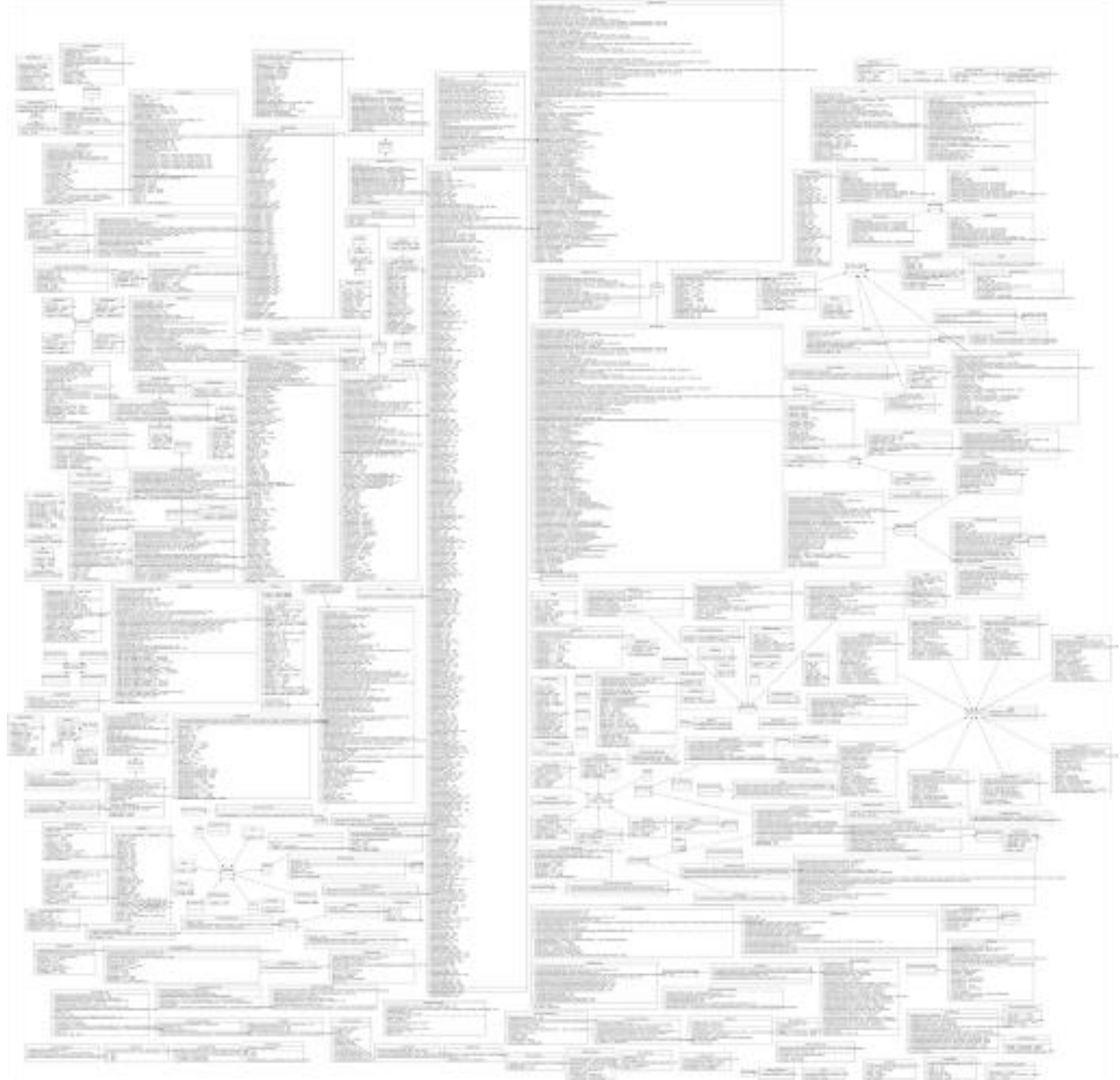


Diagrama de Pacotes

Diagrama de Pacotes

- Um pacote é uma construção de agrupamento que permite a você pegar **qualquer construção na UML** e **agrupar seus elementos** em **unidades de nível mais alto**.
- O uso mais **comum é o agrupamento de classes**, mas também pode-se usar pacotes para todos os outros elementos da UML.

Diagrama de Pacotes

- Em um modelo da UML, cada classe é membro de **um único pacote**.
- Os **pacotes** também podem ser **membros de outros pacotes**, de modo que você obtém uma **estrutura hierárquica** na qual os **pacotes de nível superior** são divididos em **subpacotes** que possuem seus próprios **subpacotes** e assim por diante, até que a hierarquia chegue nas classes.
- Um pacote pode conter subpacotes e classes.

Diagrama de Pacotes

- **Notação:** uma pasta com guia (ou aba).

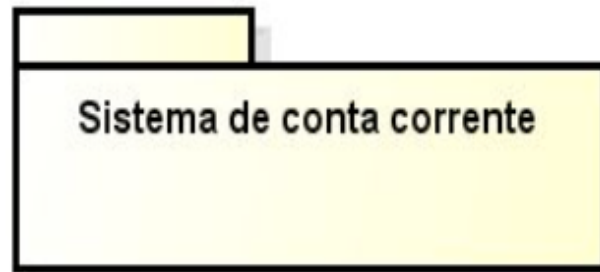


Diagrama de Pacotes

- **Notação:** uma pasta com guia (ou aba).

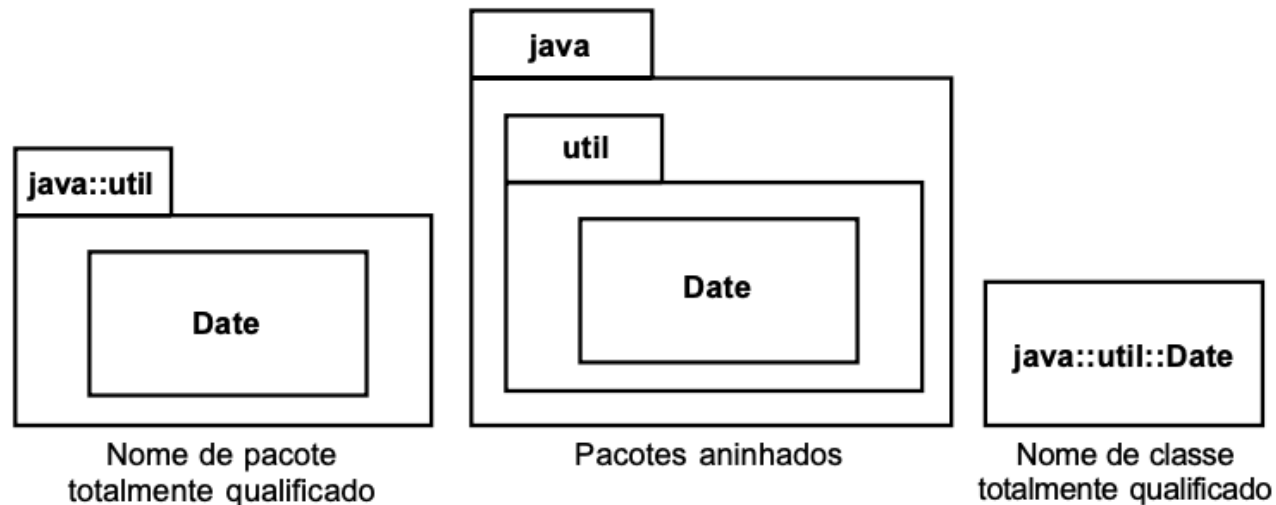
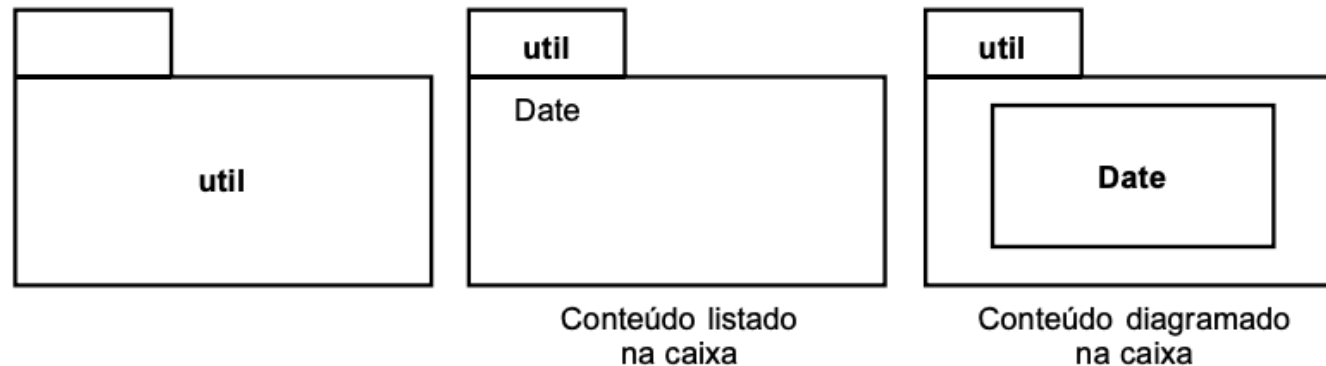
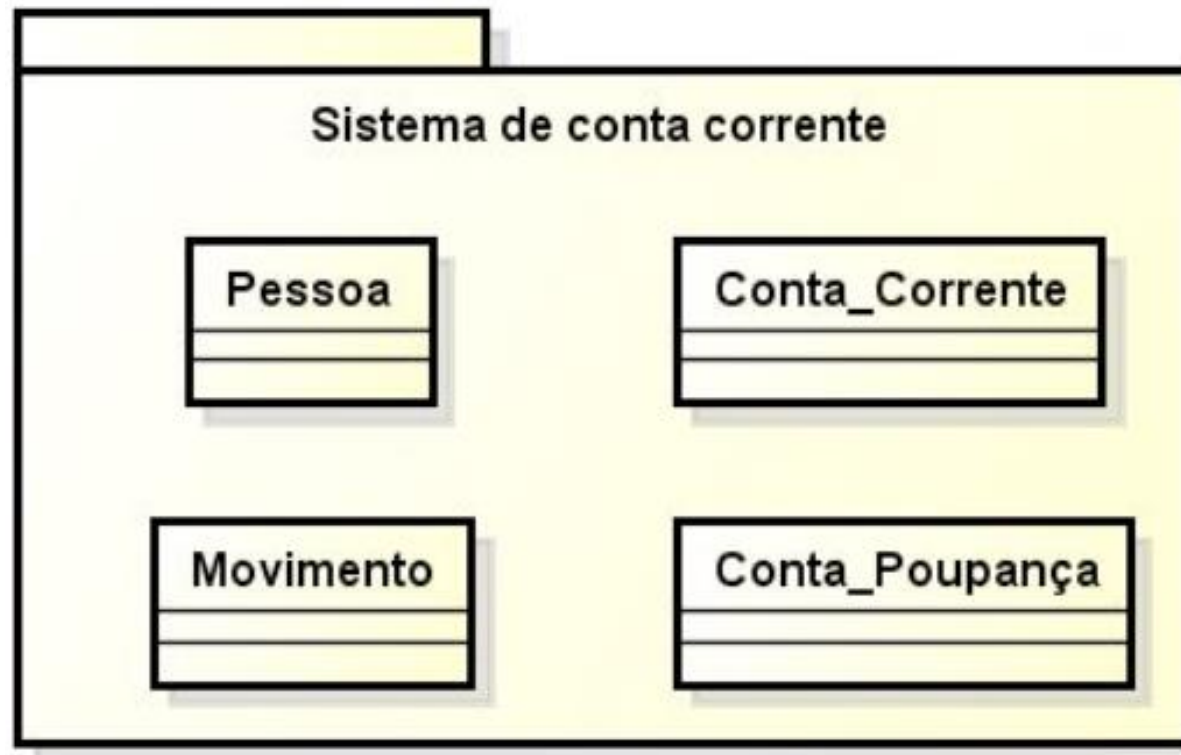


Diagrama de Pacotes – Representações Gráficas

- 1ª forma: exibir o conteúdo dentro do pacote



- Identificamos os elementos contidos pelo pacote, sem definir as características **ou**
- Definimos o diagrama completo

Diagrama de Pacotes – Representações Gráficas

- 2ª forma: exibir os membros do pacote por meio do conector de aninhamento.

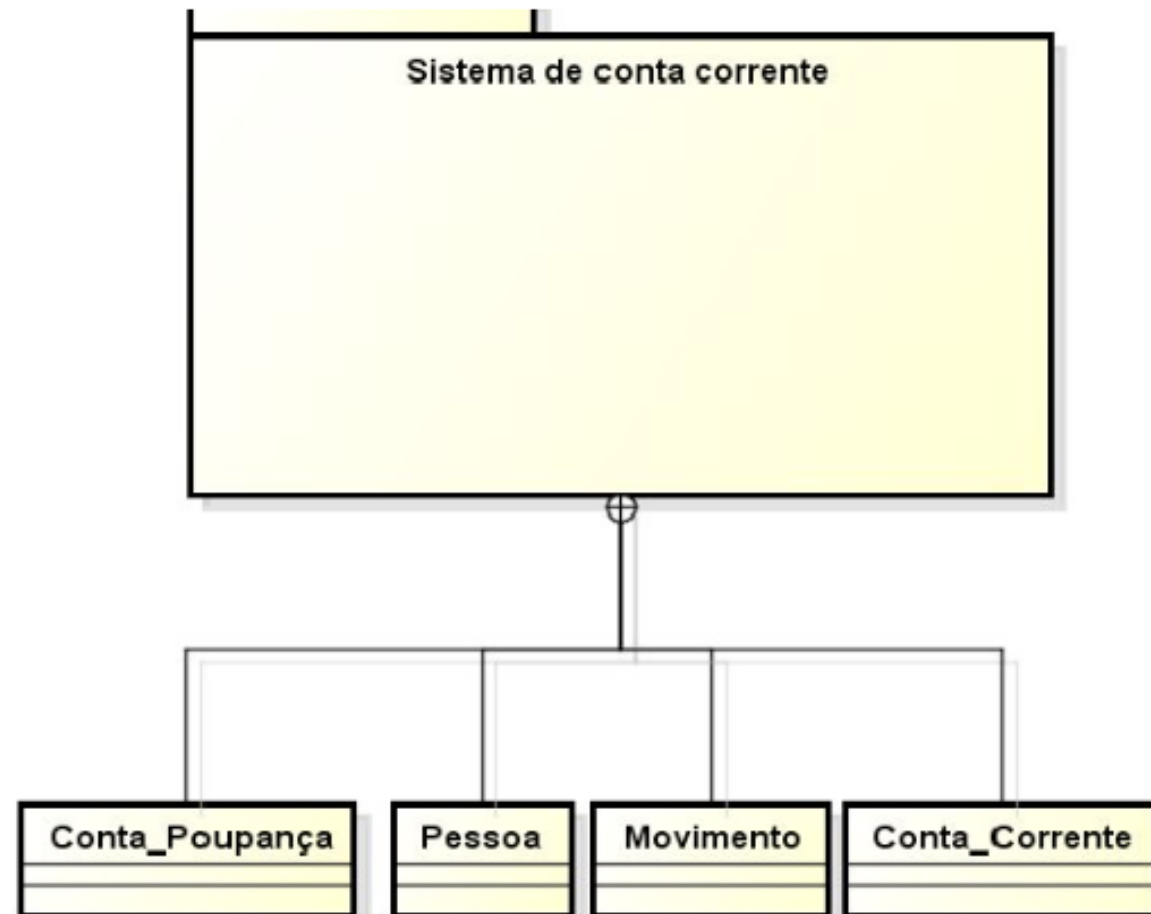


Diagrama de Pacotes – Representações Gráficas

- **Hierarquia:** Pacotes podem ser agrupados dentro de outros pacotes, formando uma hierarquia de contenção.

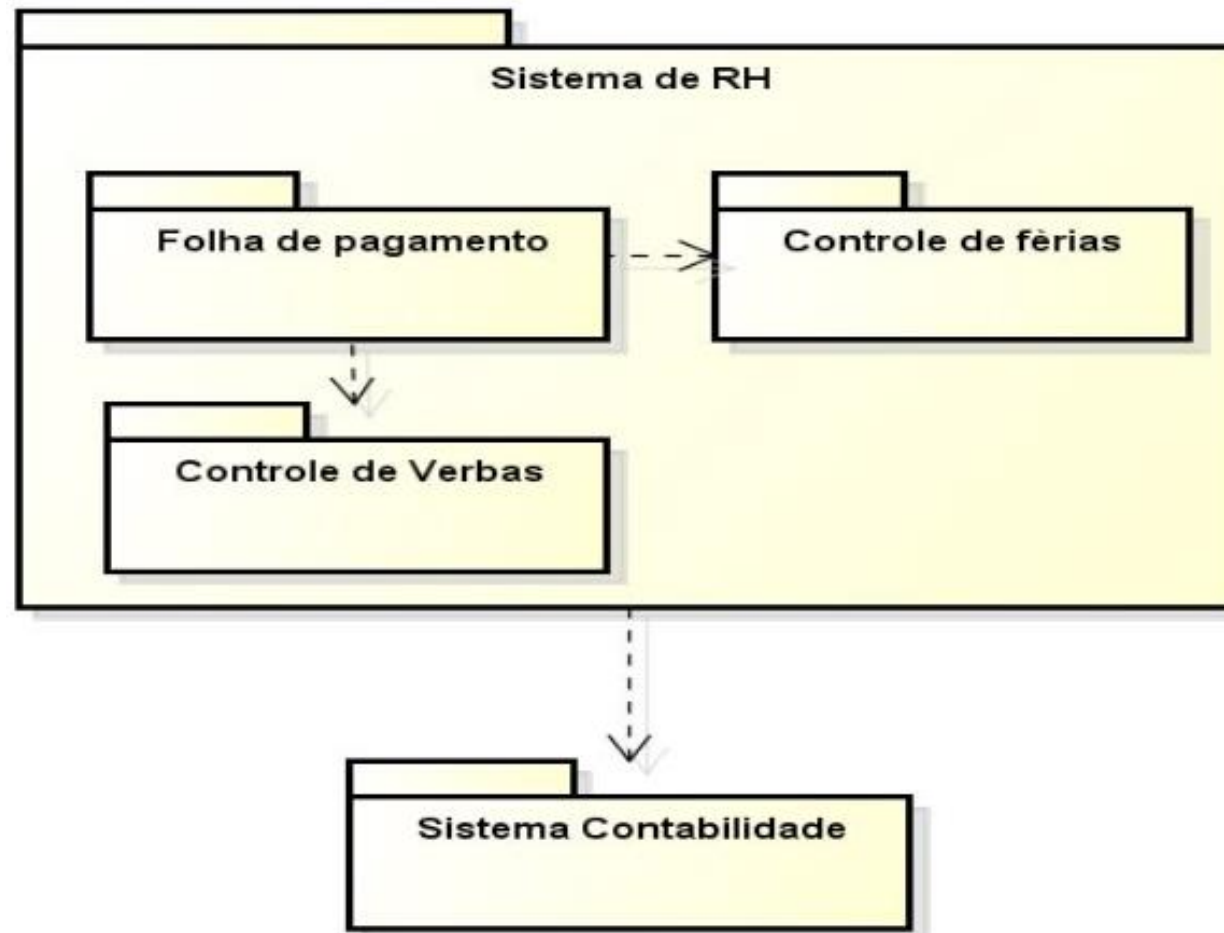


Diagrama de Pacotes – Dependências

- Um diagrama de pacotes mostra pacotes e **suas dependências**.
- **Ex:** se você tem pacotes de **apresentação** e de **domínio**, então tem uma **dependência** do pacote de apresentação para o pacote de domínio, caso **qualquer classe no pacote de apresentação tenha uma dependência** de qualquer classe no pacote de domínio.
- A UML tem muitas variedades de dependências, cada uma com uma semântica e um estereótipo em particular.

Diagrama de Pacotes – Dependências

- A UML tem muitas variedades de dependências, cada uma com uma semântica e um estereótipo em particular.

Associação



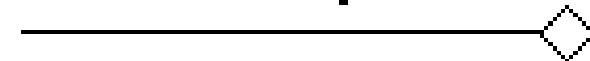
Herança



Dependência



Agregação



Composição

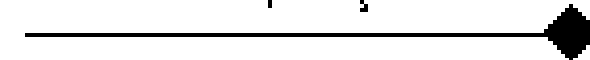
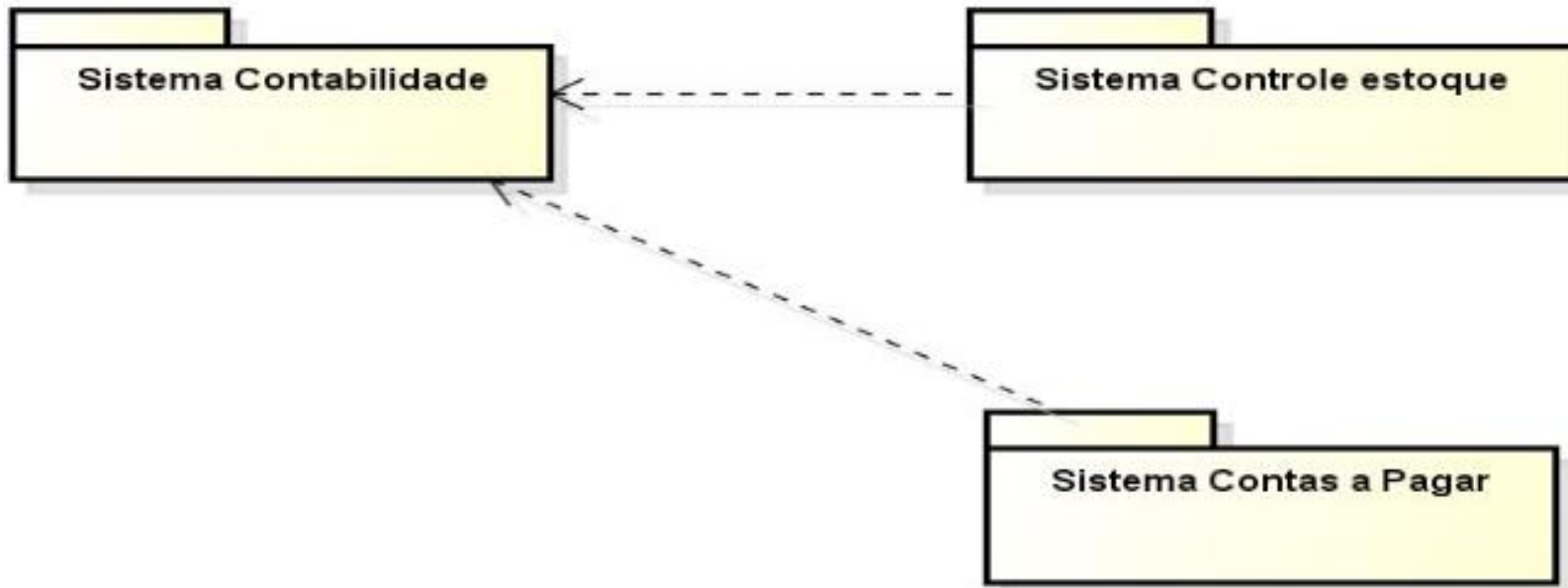


Diagrama de Pacotes – Dependências

Um pacote P1 é dependente do pacote P2, se houver qualquer dependência entre quaisquer elementos de P1 e P2.

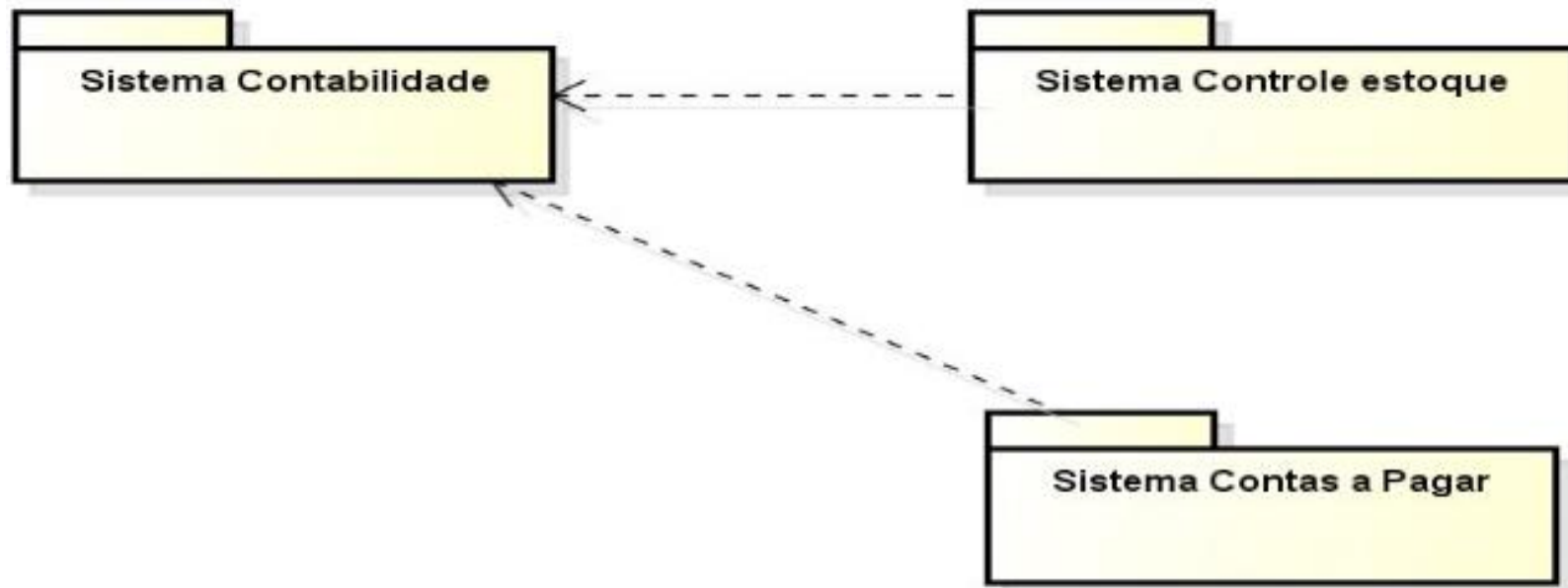


O relacionamento de dependência no diagrama de pacotes pode ter dois estereótipos:

- **<<merge>>**: significando que os elementos do pacote que utiliza essa dependência serão unidos aos elementos do outro pacote
- **<<import>>**: significando que o pacote que utiliza essa dependência está importando alguma característica ou elemento do outro pacote.

Diagrama de Pacotes – Dependências

Um pacote P1 é dependente do pacote P2, se houver qualquer dependência entre quaisquer elementos de P1 e P2.



O relacionamento de dependência no diagrama de pacotes pode ter dois estereótipos:

- **<<merge>>**: significando que os elementos do pacote que utiliza essa dependência serão unidos aos elementos do outro pacote
- **<<import>>**: significando que o pacote que utiliza essa dependência está importando alguma característica ou elemento do outro pacote.

Sistema de Controle estoque e Sistema Contas a pagar **dependem de Sistema Contabilidade**

Diagrama de Pacotes – Dependências

- É possível **identificar um fluxo claro** no diagrama ao lado, pois todas as dependências seguem uma única direção.
- Muitos autores dizem que não **devem existir ciclos nas dependências** (o Princípio da Dependência Acíclica [Martin]).
- Quanto **mais dependências entram em um pacote**, mais **estável a interface do pacote precisa ser**, pois qualquer **alteração em sua interface será propagada** para todos os pacotes que são dependentes dela.

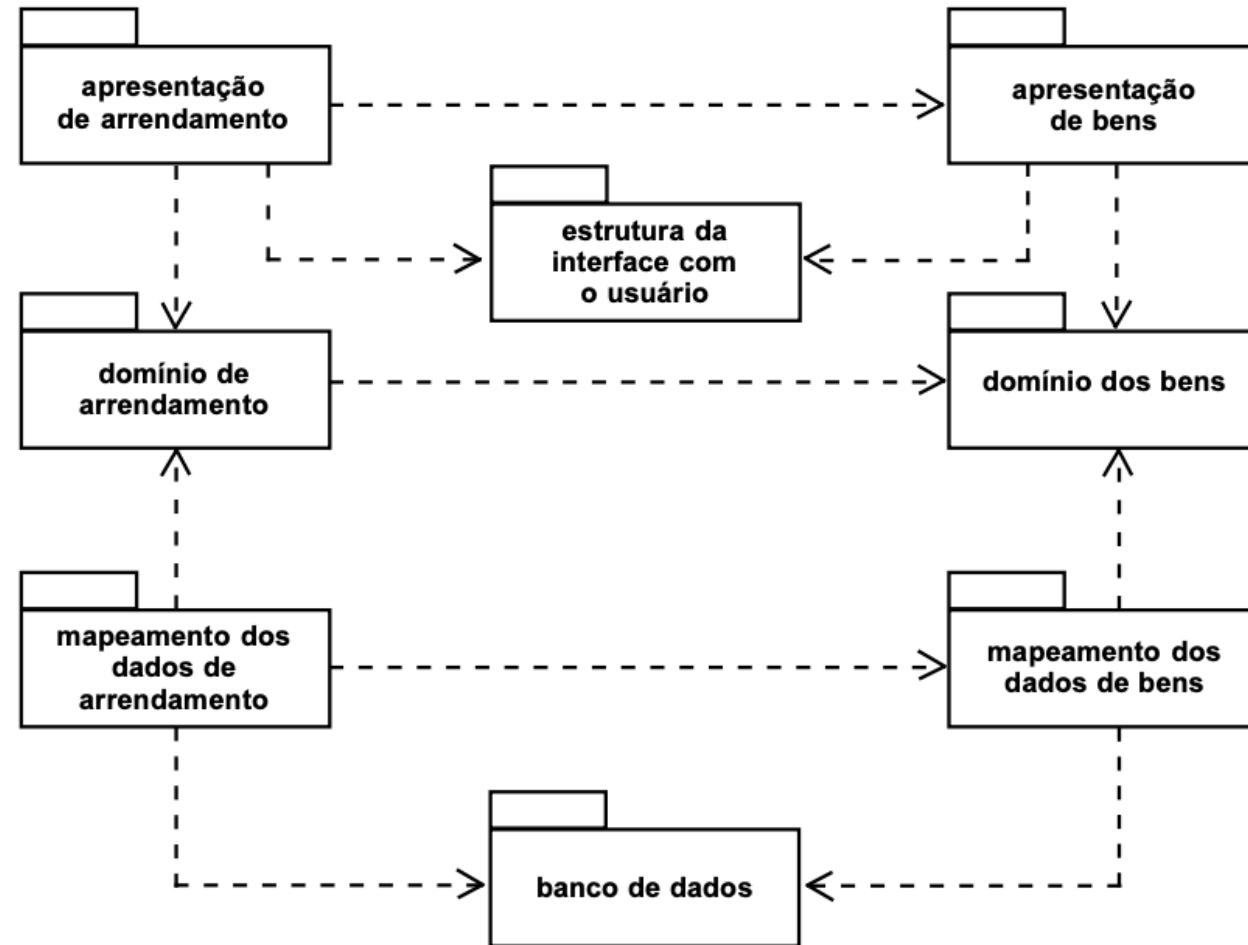


Diagrama de Pacotes – Dependências

- Os relacionamentos de dependência não são transitivos.
- **Ex:** Se uma classe no pacote do **domínio de bens** muda, talvez tenhamos que alterar as classes dentro do pacote do **domínio de arrendamento**. Mas essa alteração não se propaga necessariamente para a **apresentação de arrendamento**.
- Ela só se propaga se o domínio de arrendamento muda sua interface

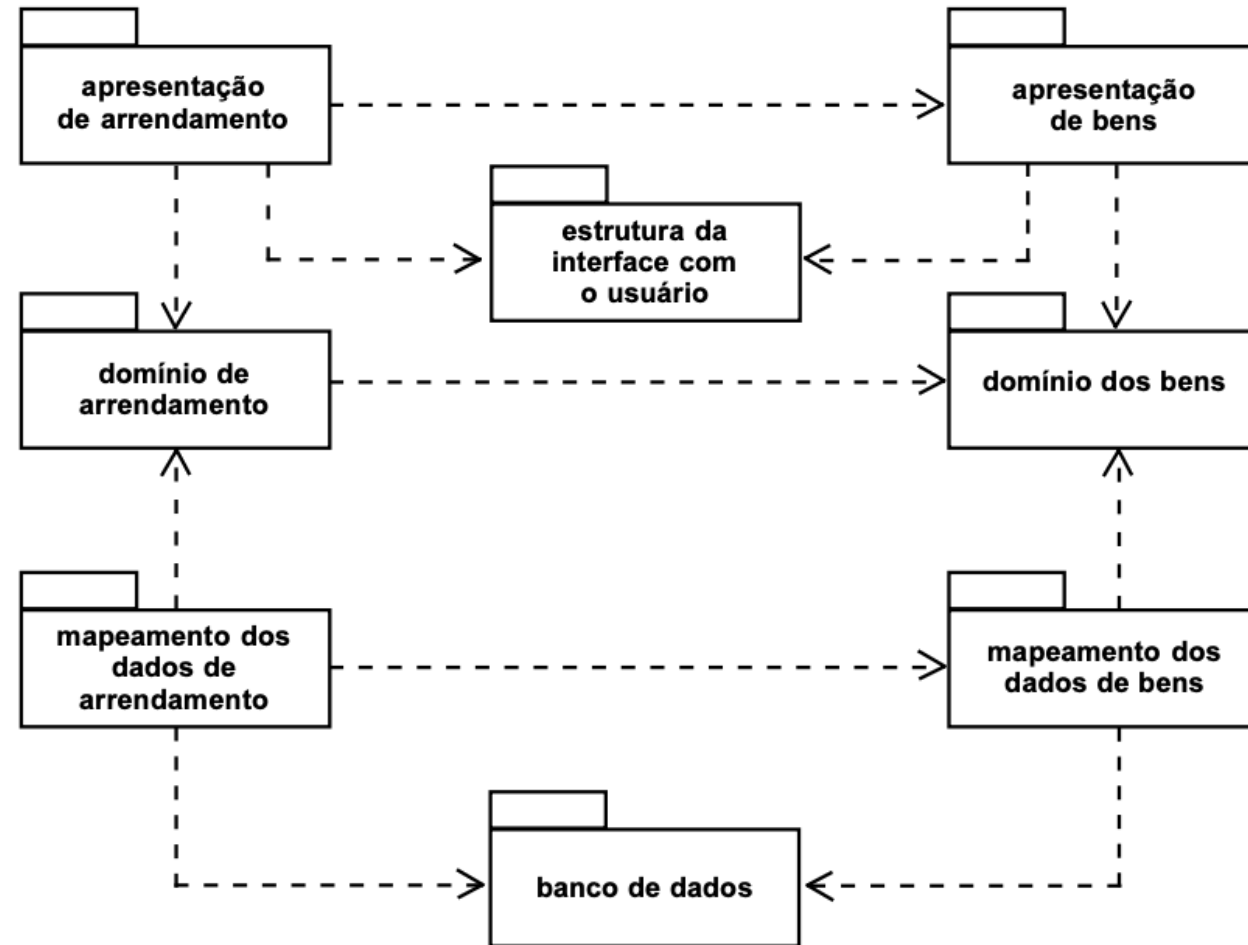


Diagrama de Pacotes – Dependências

- Alguns pacotes são usados em tantos lugares, que seria uma confusão desenhar todas as linhas de dependência para eles.
- Nesse caso, uma convenção é usar uma palavra-chave, como «global», no pacote.

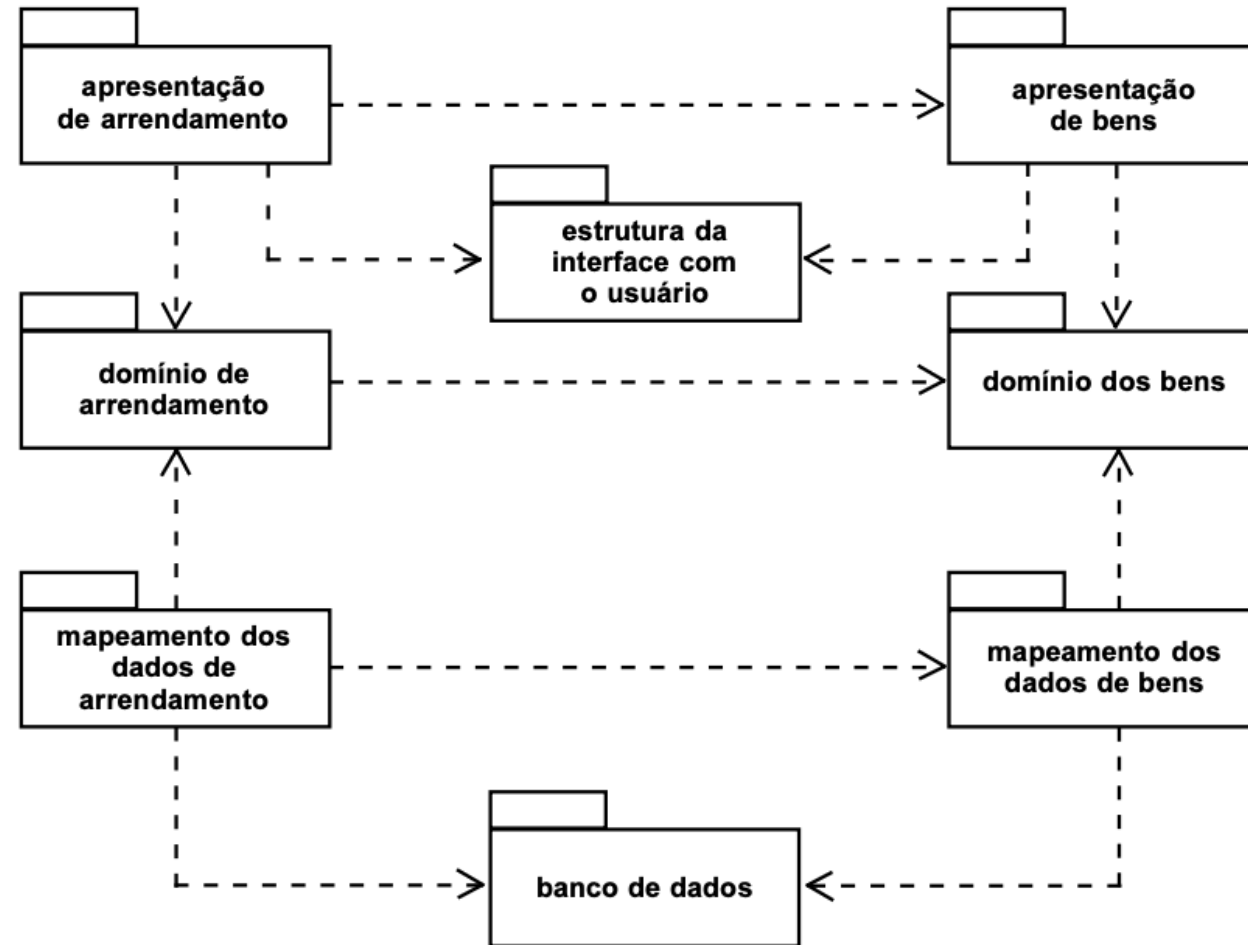


Diagrama de Pacotes – Dependências

- No diagrama ao lado as dependências indicam que o **gateway do banco de dados** define uma **interface** e que as outras classes de gateway fornecem uma implementação.

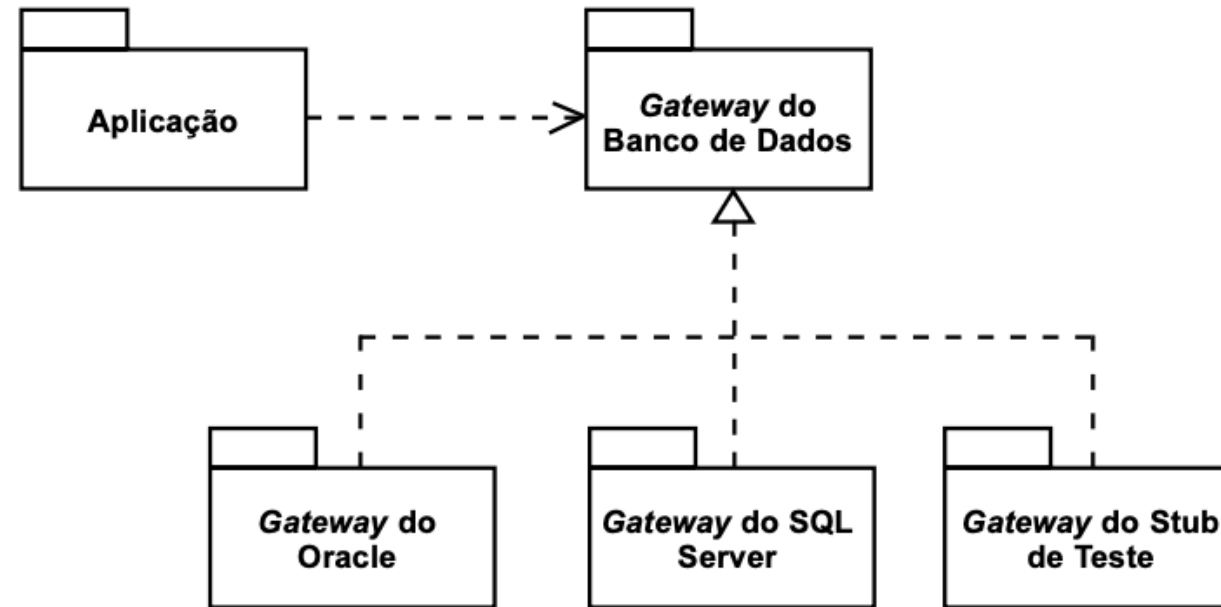


Diagrama de Pacotes – Dependências

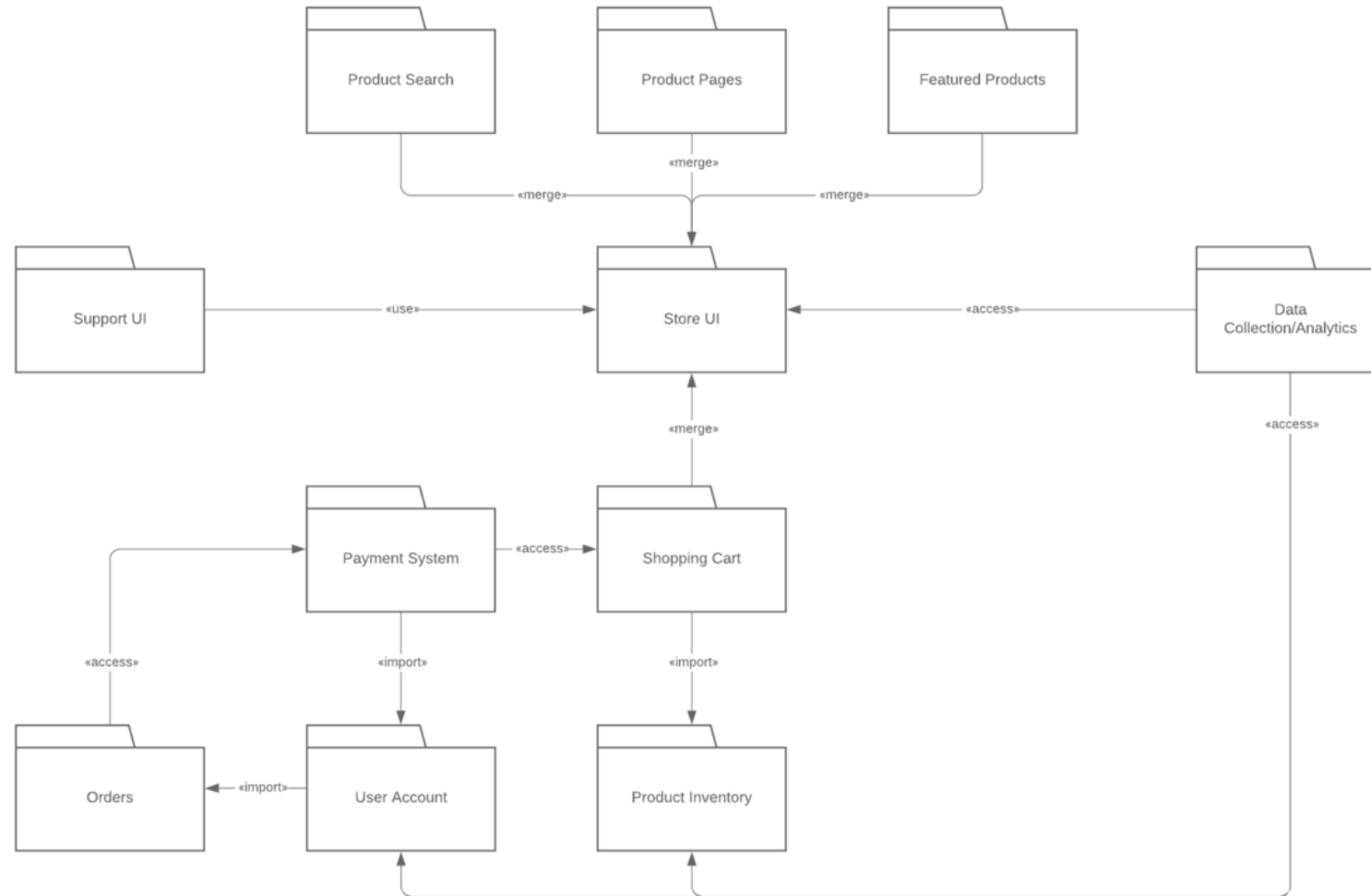


Diagrama de Pacotes – Dependências

- **Acesso:** indica que um pacote requer assistência das funções de outro pacote.
- **Importação:** indica que a funcionalidade foi importada de um pacote para outro.

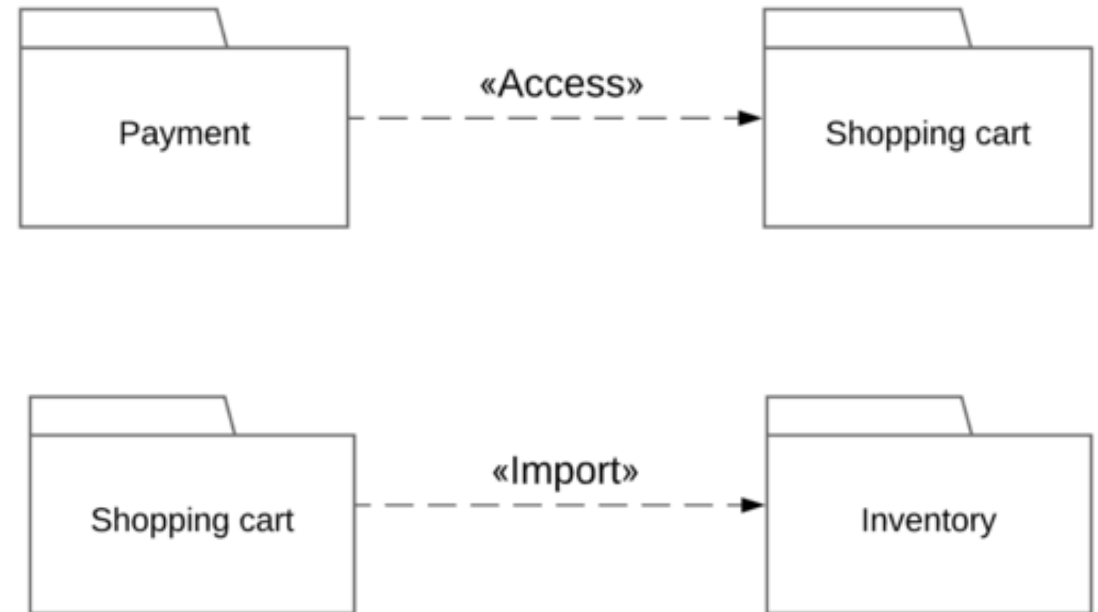


Diagrama de Pacotes – Dependências

- **Uso:** ocorre quando um determinado elemento nomeado requer outro para sua definição e implementação completa.
Exemplo: cliente e fornecedor.
- **Abstração:** relaciona dois elementos que representam o mesmo conceito em diferentes níveis de abstração no sistema (geralmente uma relação entre cliente e fornecedor).
- **Disponibilização:** mostra a implementação de um artefato em um alvo de implementação.

Diagrama de Pacotes – Estereótipos

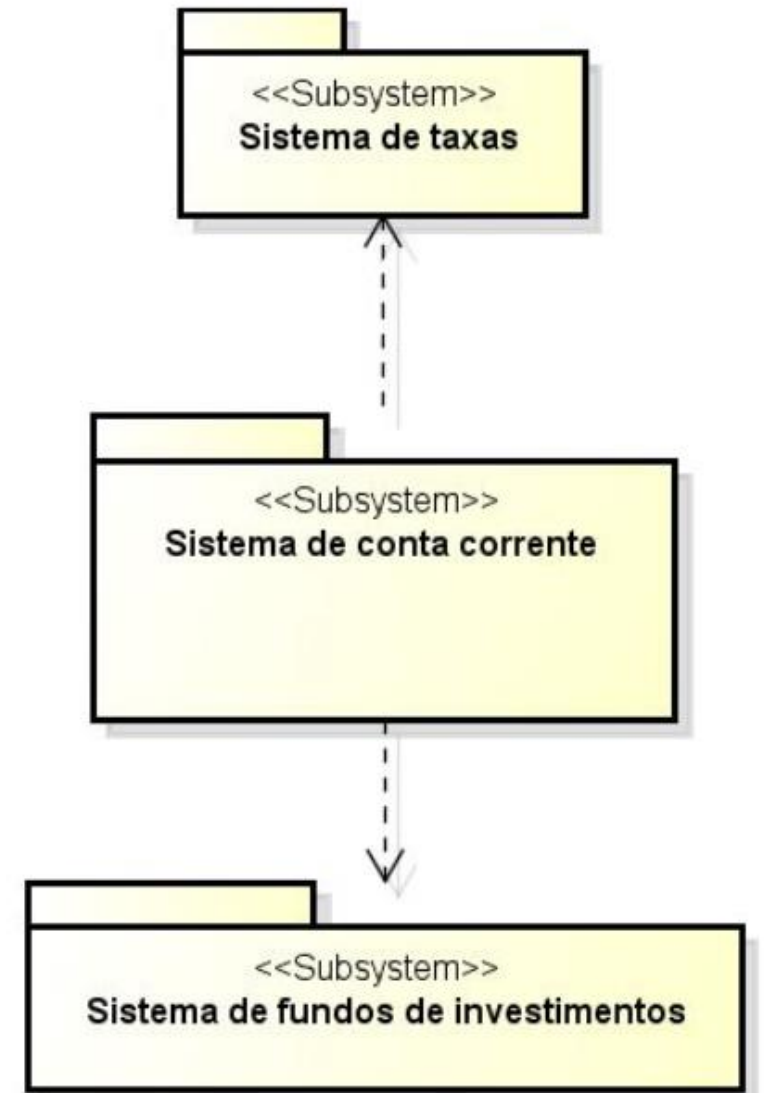
- É possível aplicar estereótipos aos pacotes, deixando claro o que cada um representa.
- Alguns tipos:
 - System
 - Subsystem
 - Model
 - Framework
 - Layer

Diagrama de Pacotes – Classificador

- É uma **entidade de software** que implementa os métodos dos serviços definidos em uma ou mais **interfaces**.
- Possui **comportamento**, ou seja:
 - **Define** um conjunto de operações.
 - **Executa** essas operações de acordo com sua especificação.
- Na UML, o termo classificador é genérico e pode representar:
 - Uma classe
 - Um subsistema
 - Um componente
 - Uma interface

Subsistema

- **Subsistema** é um **classificador** que representa um **agrupamento de classes relacionadas**.
- Como todo classificador, um subsistema:
 - Define comportamento por meio de suas classes.
 - Realiza uma ou mais interfaces, especificando os serviços que oferece.
- Na UML, um subsistema é representado como um pacote com o estereótipo **<<Subsystem>>**.
 - Importante: subsistemas podem realizar interfaces enquanto que pacotes são somente mecanismos de agrupamentos.



Alocando classes a subsistemas

Etapas:

1. Durante o desenvolvimento orientado a objetos, é necessário identificar os subsistemas e as interfaces entre eles.
2. Em seguida, cada classe do sistema é alocada ao respectivo subsistema.
3. A partir disso, é construído o diagrama de subsistemas.
4. Com os subsistemas e interfaces definidos, cada subsistema pode ser desenvolvido quase de forma independente.

Modelo de classes de domínio

Um **modelo de classes de domínio** é uma representação conceitual das principais entidades e relacionamentos de um **domínio de negócio** (ou seja, o problema que o sistema precisa resolver).

Ou seja:

- Ele não descreve ainda a solução de software (como classes de implementação em código).
 - Ele descreve a visão do mundo real, mostrando objetos, atributos e associações relevantes para o domínio.
-
- **Ex:** Em um sistema de biblioteca, o modelo de classes de domínio pode conter classes como **Livro, Usuário, Empréstimo, Autor**, e suas relações, sem falar ainda em métodos, bancos de dados ou detalhes de código.

Alocando classes a subsistemas

- **Modelo de Classes de Domínio e Subsistemas**
 - O **modelo de classes de domínio** é o ponto de partida para identificar os subsistemas.
 - As classes devem ser **agrupadas segundo critérios definidos**.
 - Um critério comum: **escolher as classes mais importantes** do modelo de domínio.
 - Para cada classe importante, cria-se um **subsistema**.
 - Classes menos importantes, mas relacionadas, são **incluídas no mesmo subsistema**.

Alocando classes a subsistemas

Boas Práticas no Projeto de Subsistemas:

- **Baixo acoplamento:** subsistemas devem ter o mínimo possível de dependências entre si.
- **Alta coesão:** cada subsistema deve ter responsabilidade clara e bem definida.
- **Evitar ciclos de dependência:**
 - Solução 1: dividir um subsistema do ciclo em dois ou mais.
 - Solução 2: combinar os subsistemas do ciclo em um único.
- **Definição de classes:**
 - Cada classe deve pertencer a **um único subsistema**.
 - O subsistema que define a classe apresenta **todas as suas propriedades**.
 - Outros subsistemas podem usar uma **versão simplificada** da classe.

Interfaces



Interfaces

Definição de Serviço:

- **Serviço:** tarefa que uma entidade de software realiza para outra.
- Um serviço é composto por:
 - **Especificação:** descreve **o que** o serviço faz e quais informações são necessárias.
 - **Método:** define **como** o serviço é realizado (podendo haver diferentes métodos para a mesma especificação).

Interfaces

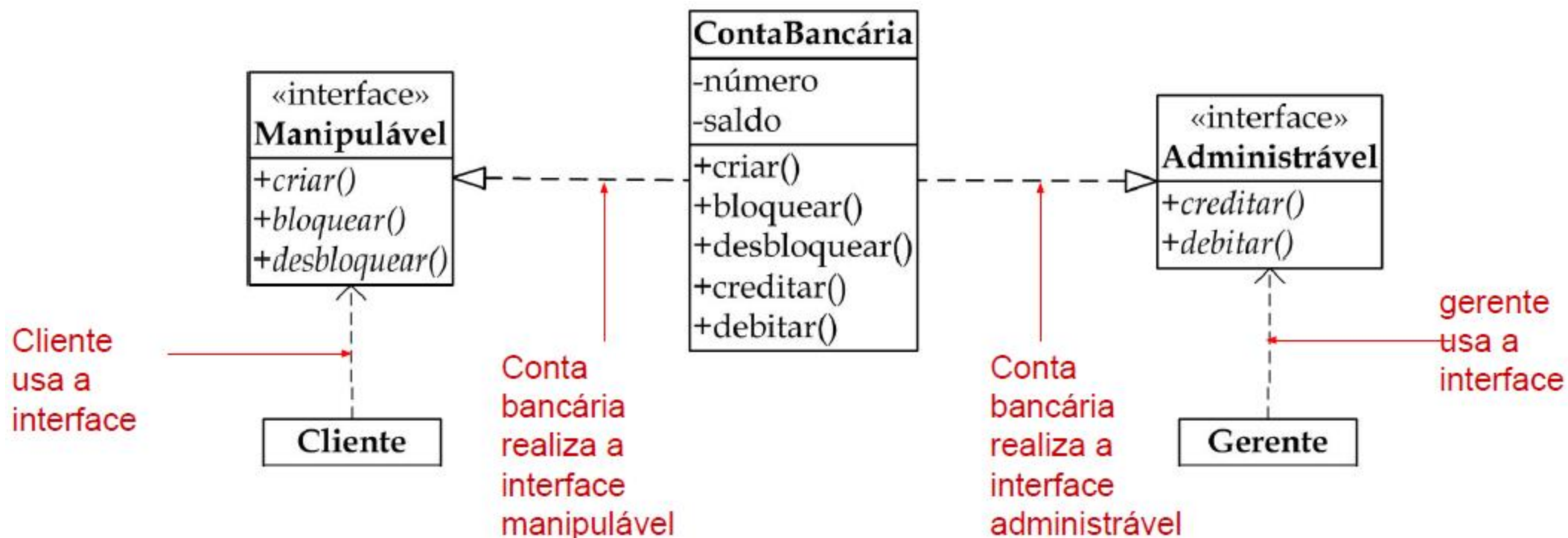
Interface na UML:

- **Interface**: conjunto de **especificações de serviços**.
- Semelhante a uma **classe abstrata** (tem nome, mas não gera instâncias).
- Diferenças principais:
 - Não possui **estrutura interna** (sem atributos ou associações).
 - Todas as operações têm apenas a **especificação** (o **método** não é definido na interface).

Interfaces - Notação

1ª notação: é a mesma para classes.

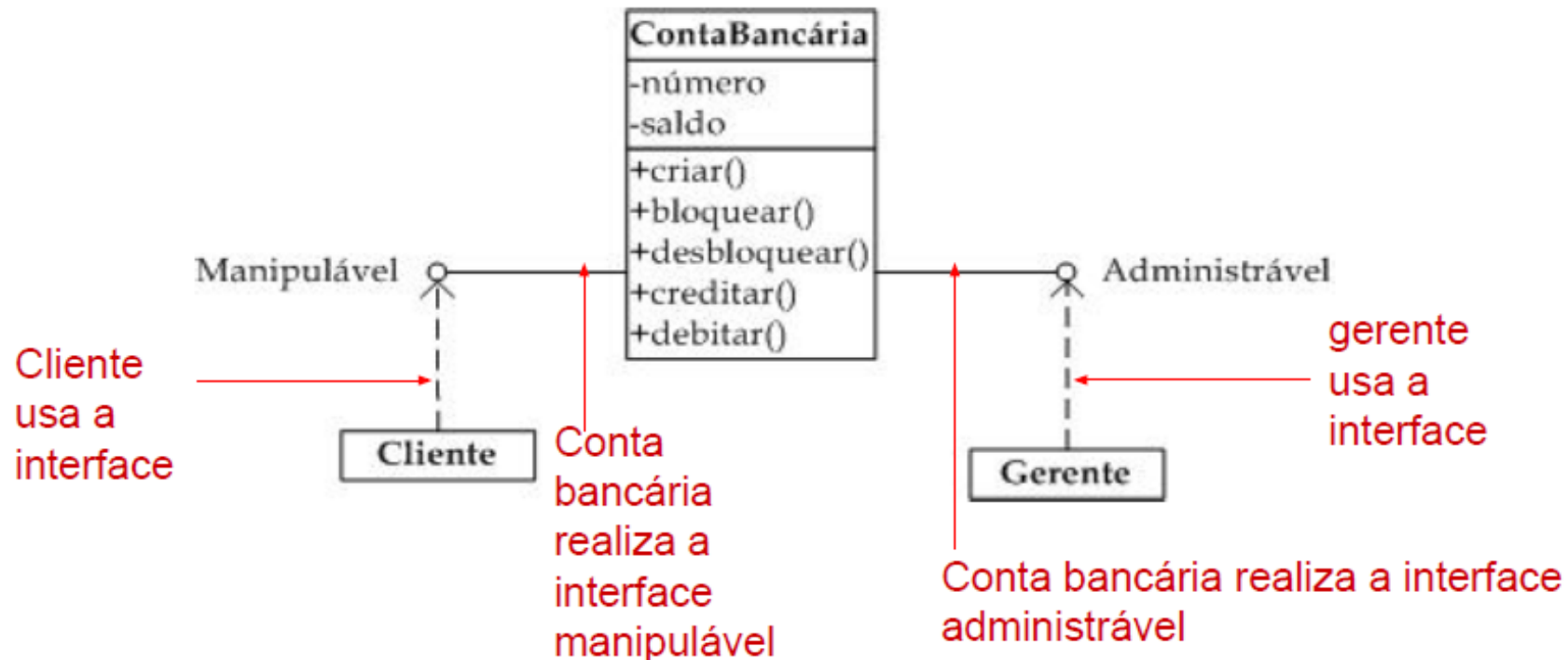
- São exibidas as operações que a interface especifica.
- Deve ser usado o estereótipo <<interface>>.



Interfaces - Notação

2ª notação:

- Representada por uma **linha reta com um pequeno círculo em uma das extremidades**.
- O círculo indica a **interface**.
- A outra extremidade da linha é ligada ao **classificador que realiza (implementa) a interface**.
- **Classes são conectadas à interface através de um relacionamento de dependência.**



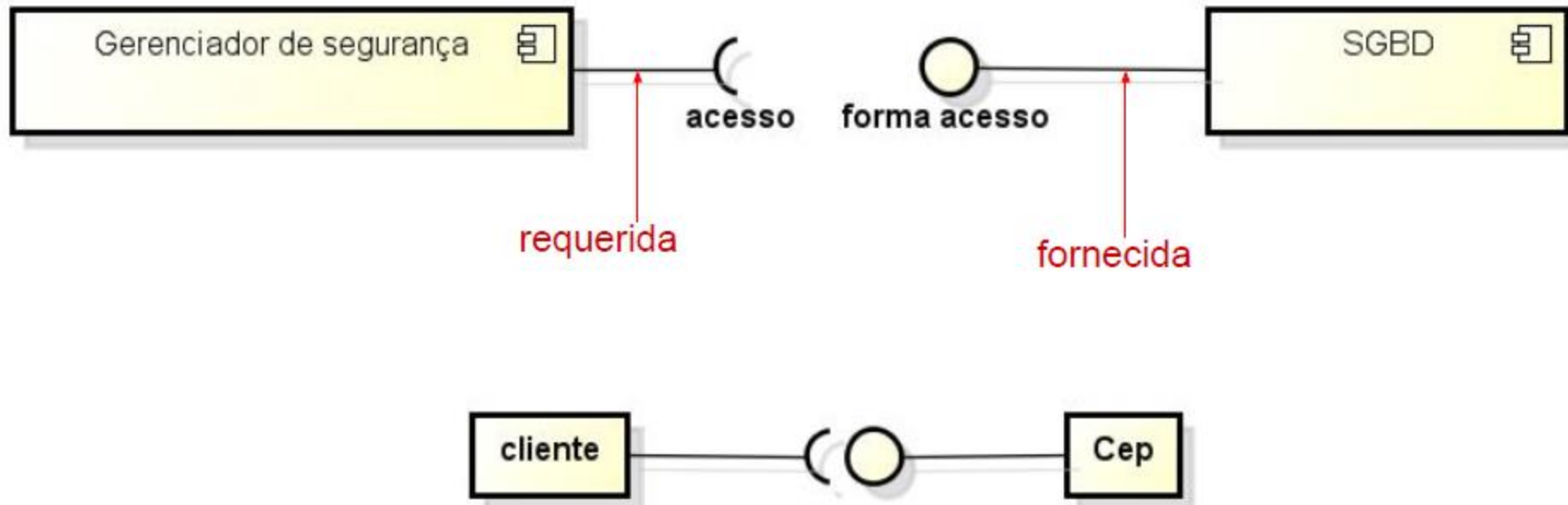
Interfaces - Tipos

Interface fornecida:

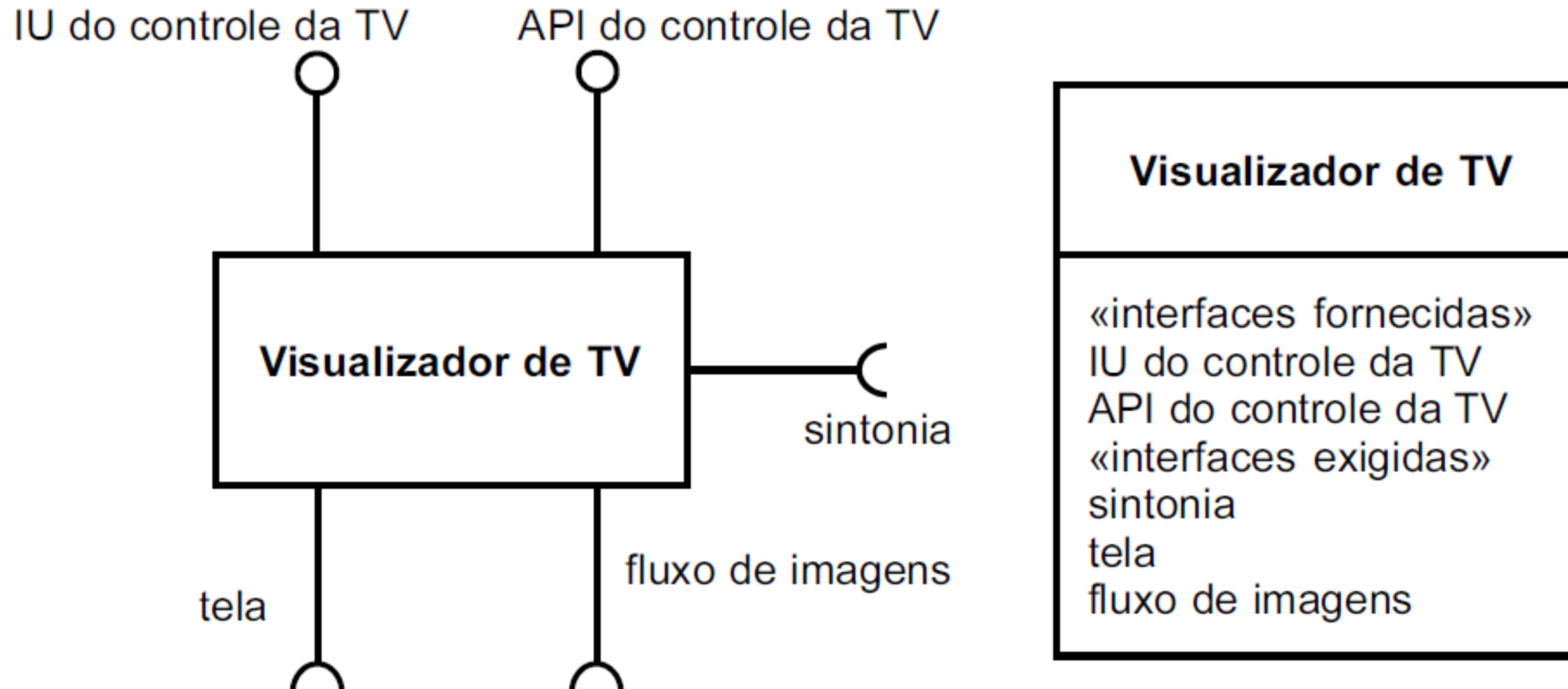
- Uma interface em que o componente fornece um serviço para outros componentes.

Interface requerida:

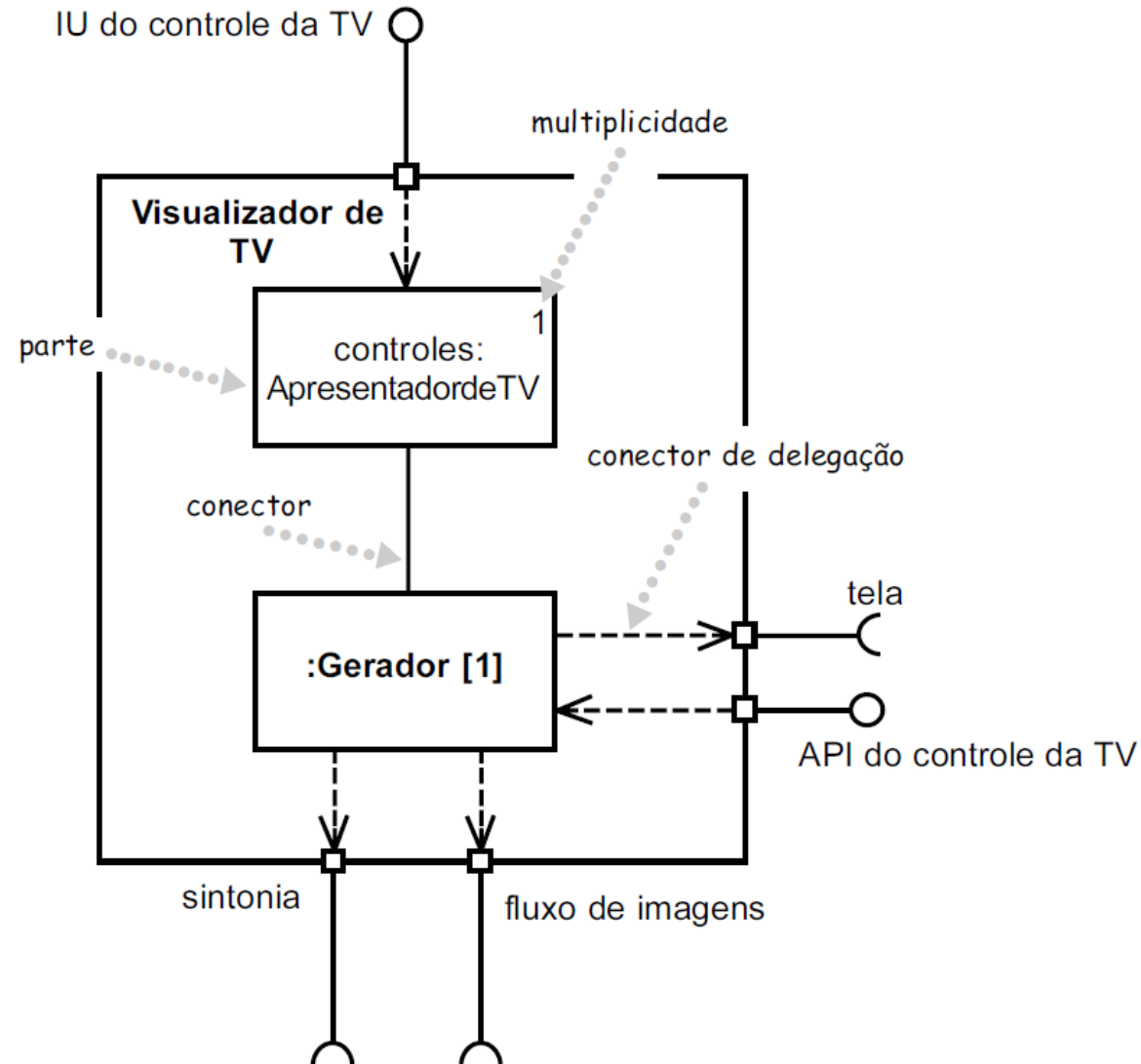
- Uma interface à qual o componente se **adapta** quando solicita serviços de outros componentes.



Interfaces – Estruturas compostas



Interfaces – Estruturas compostas (visão interna)



Camadas



Camadas

- Outra forma de **organizar a arquitetura** de um sistema complexo em partes menores é através de **camadas** de software.
- Uma camada é constituída por uma coleção de subsistemas.
 - Cada camadas corresponde a um conjunto de funcionalidades de um sistema de software.
 - **Dependência hierárquica**: funcionalidades de alto nível dependem das de baixo nível.
- Camadas fornecem abstração por meio do agrupamento lógico de subsistemas.

Exemplos de Arquitetura em Camadas

1. **Aplicação Web**

- Camada de Apresentação (UI)
- Camada de Lógica de Negócio
- Camada de Acesso a Dados
- Camada de Banco de Dados

2. **Sistema Operacional**

- Camada de Aplicações
- Camada de Gerenciamento de Recursos
- Camada de Kernel

3. **Aplicativo Mobile**

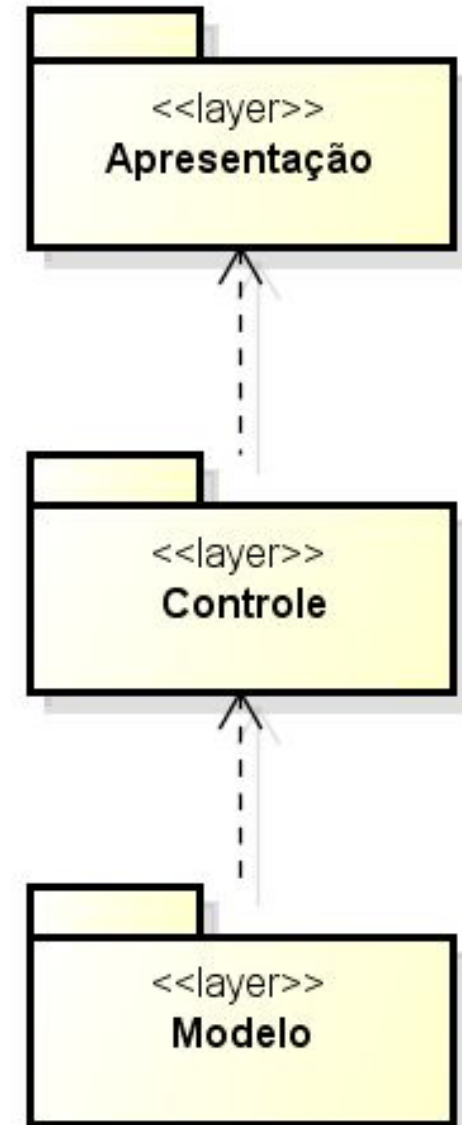
- Camada de Interface com o Usuário
- Camada de Serviços
- Camada de Persistência

Camadas

- **Regra geral:** camadas de abstração mais altas devem depender das camadas mais baixas, nunca o contrário.
- **Benefícios:**
 - Sistema mais portátil e fácil de modificar.
 - Mudanças em uma camada baixa sem alteração na interface não impactam camadas superiores.
 - Mudanças em camadas altas, que não exijam novos serviços das camadas inferiores, não afetam as camadas de baixo.

Camadas

- A UML **não possui** elemento gráfico específico para representar camadas de um sistema.
- No entanto, **pacotes** também podem ser utilizados para representar camadas.
 - O estereótipo <<camada>> (<<layer>>) pode ser utilizado no pacote que represente uma camada.
 - O fato de uma camada utilizar outra pode ser representado por um relacionamento de dependência.



Componentes



Componentes

- **Visualização de implementação** expressa ou mostra os **Componentes** de uma aplicação.
- **Diagrama de Componentes:**
 - Identifica os componentes de um sistema, subsistema ou até de um único componente.
 - Documenta a estrutura dos arquivos físicos do sistema.
 - Facilita a compreensão da arquitetura e promove a reutilização de componentes.

Componentes x Pacotes

- Pacote = agrupamento lógico de elementos do modelo (organização).
- Componente = unidade física implantável/executável do sistema (implementação).

Componentes

Analogia: Chip de Memória

- Componente eletrônico usado na construção de um computador.
- Pode ser substituído por outro mais poderoso, desde que mantenha a mesma especificação.

Componente de software:

- Um sistema pode ser visto como a composição de diversos componentes.
- Cada componente existe e interage em tempo de execução.

Componentes

Componente é uma unidade configurável de software:

- Pode ser reutilizada na construção de diferentes sistemas.
- Pode ser substituída por outra unidade que forneça a mesma funcionalidade.

Como classificador, um componente:

- Disponibiliza seus serviços por meio de interfaces.
- Pode fornecer serviços a outros componentes.
- Também pode requisitar serviços de outros.

Componentes x Classes

- **Existe/Existia** uma polêmica sobre qual é a diferença entre um componente e uma classe regular qualquer.

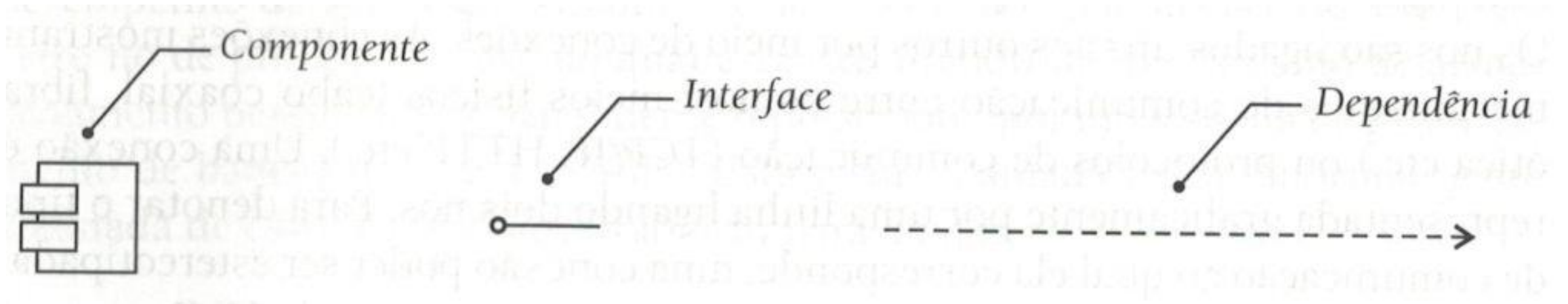
Classes:

- Representam abstrações lógicas.
- Possuem atributos e operações diretamente.

Componentes:

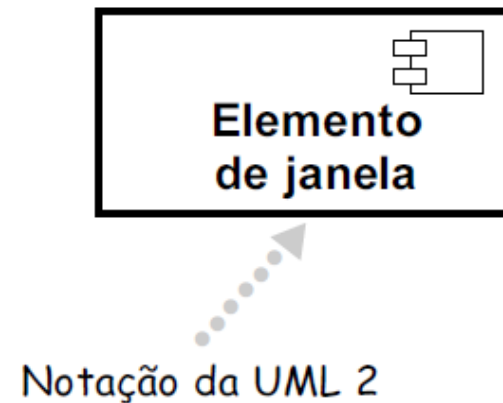
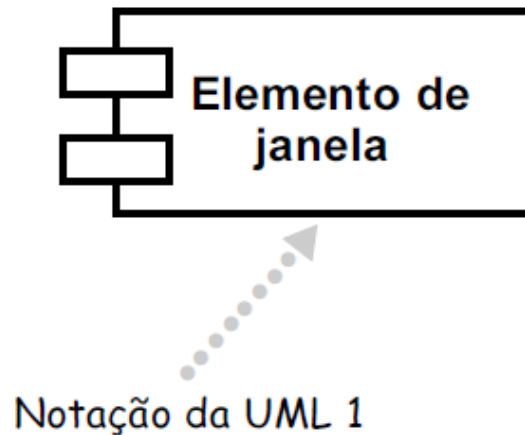
- Representam elementos no mundo físico do software (bits, arquivos executáveis, bibliotecas).
- Geralmente expõem apenas operações acessíveis via interfaces.
- São o pacote físico que encapsula classes e outros elementos lógicos.
- Estão em um nível de abstração diferente das classes.

Elementos do Diagrama de Componentes



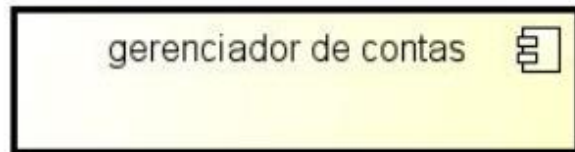
Componentes na UML

- UML 1 tinha um símbolo característico para um componente.
- A UML 2 retirou esse ícone, mas permite que se anote uma caixa de classe com um ícone de aparência semelhante.
- Como alternativa, utiliza-se a palavra-chave «component».

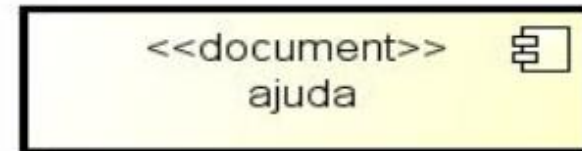
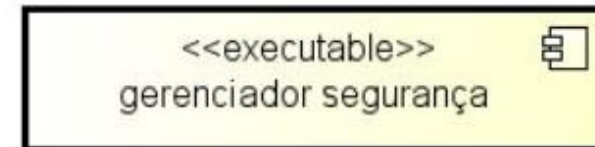


Elementos do Diagrama de Componentes

Nome simples



Nome simples com estereótipos



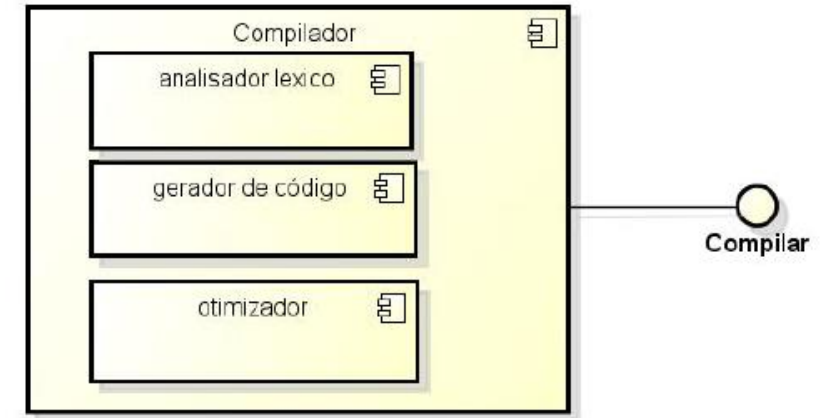
Com interfaces



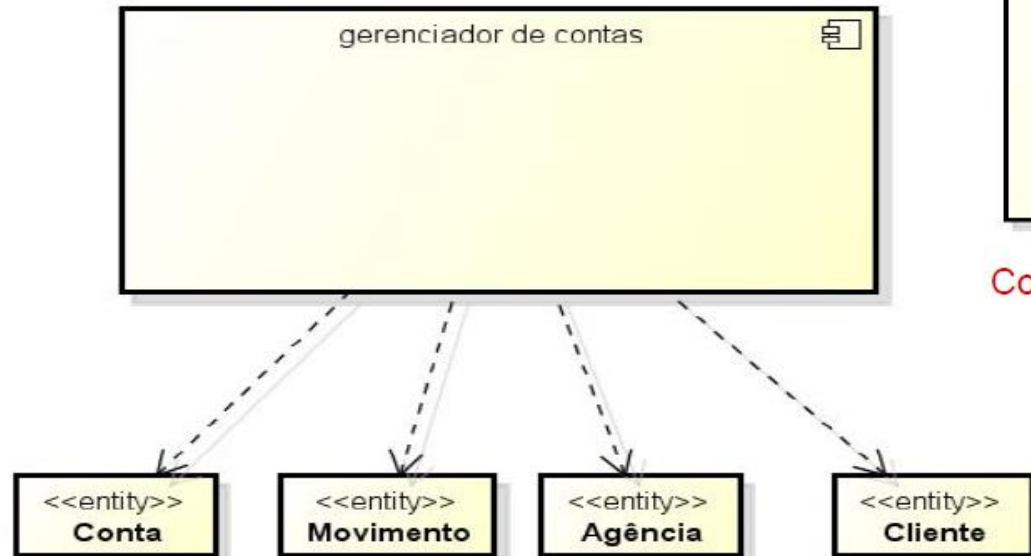
Diagrama de Componentes



Componentes com classes internas



Componentes com componentes internos



Classes internas relacionadas com dependência

Diagrama de Componentes

Portas x Interfaces

Interfaces:

- **Definem o comportamento geral de um componente.**
- Não possuem identidade própria.
- A **implementação** do componente **deve garantir** todas as **operações declaradas em suas interfaces** fornecidas.

Portas

- São **janelas explícitas** de interação em um componente encapsulado.
- Características estruturais que **intermediam a comunicação** entre o componente e seu ambiente.
- Seu comportamento é definido pelas **interfaces fornecidas** e **interfaces requeridas**.
- Permitem que a implementação interna seja modificada **sem impacto nos clientes externos**.
- Garantem que os clientes **não tenham visibilidade da estrutura interna**.
- Um componente pode possuir **múltiplas portas**, cada uma com seu próprio conjunto de interfaces.

Diagrama de Componentes

Em resumo:

- **Interface** = o contrato (o que é oferecido ou pedido)
- **Porta** = o ponto de acesso físico/lógico pelo qual esse contrato é exposto.

Diagrama de Componentes

Portas

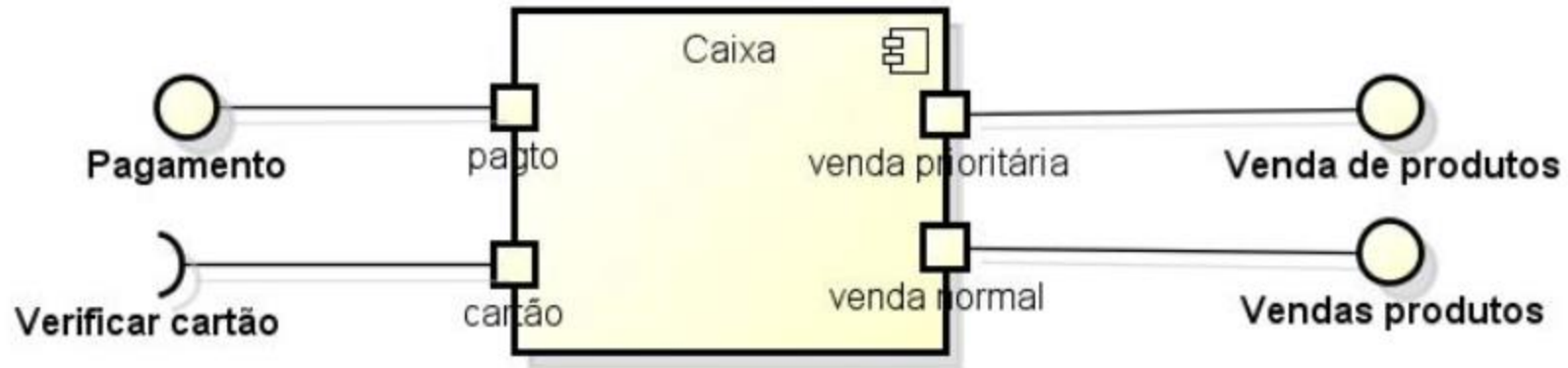


Diagrama de Componentes

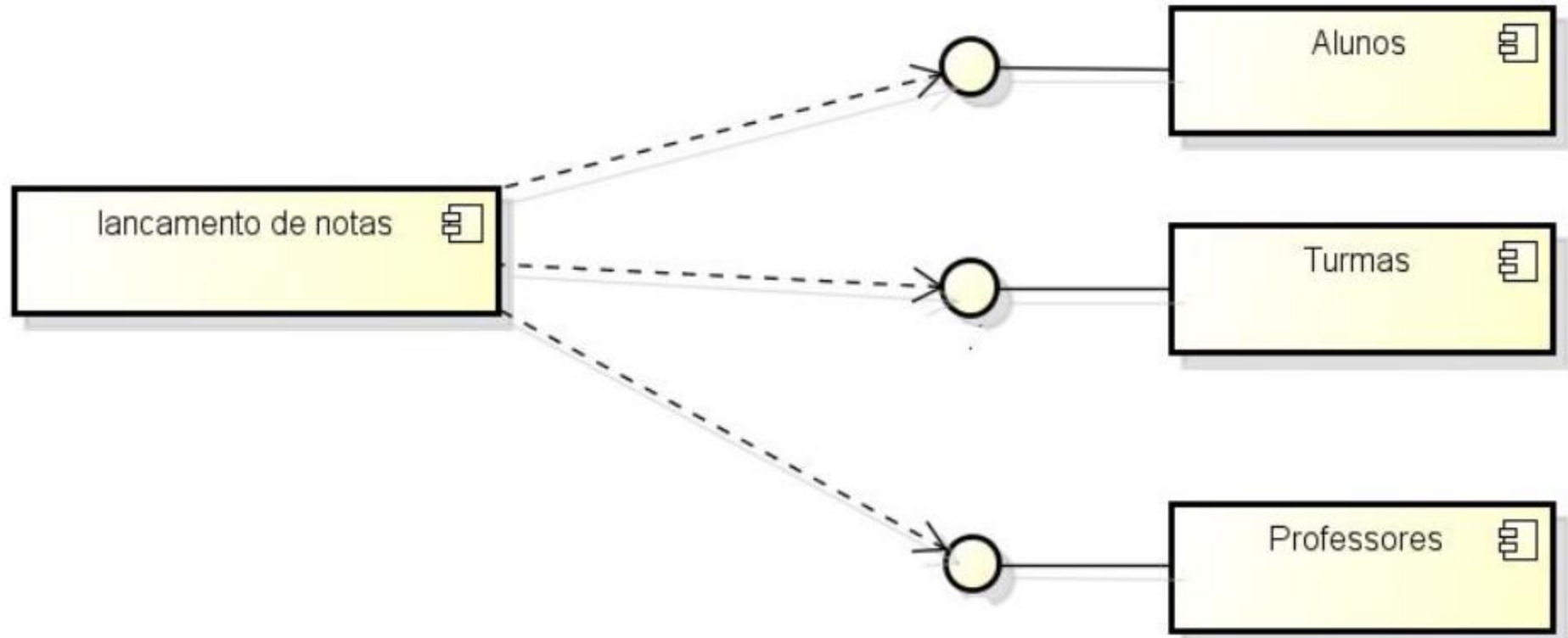
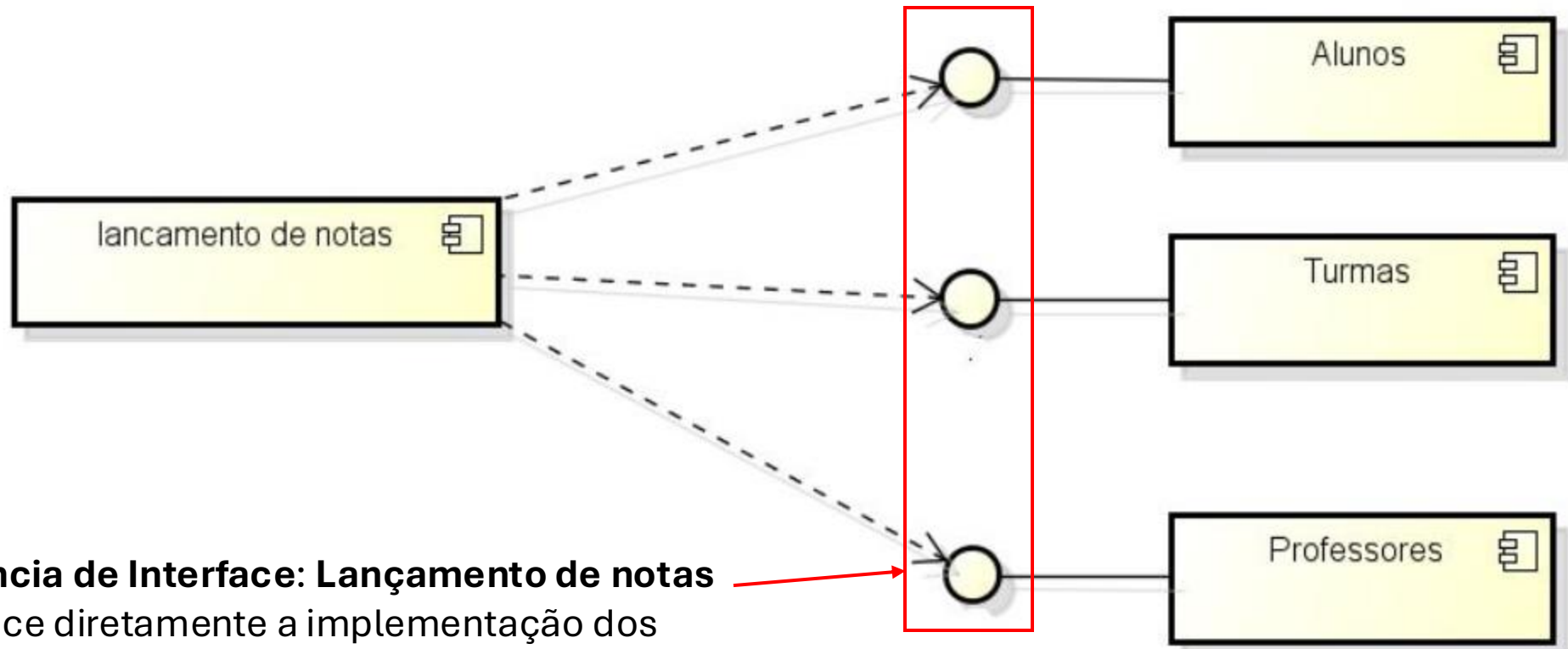


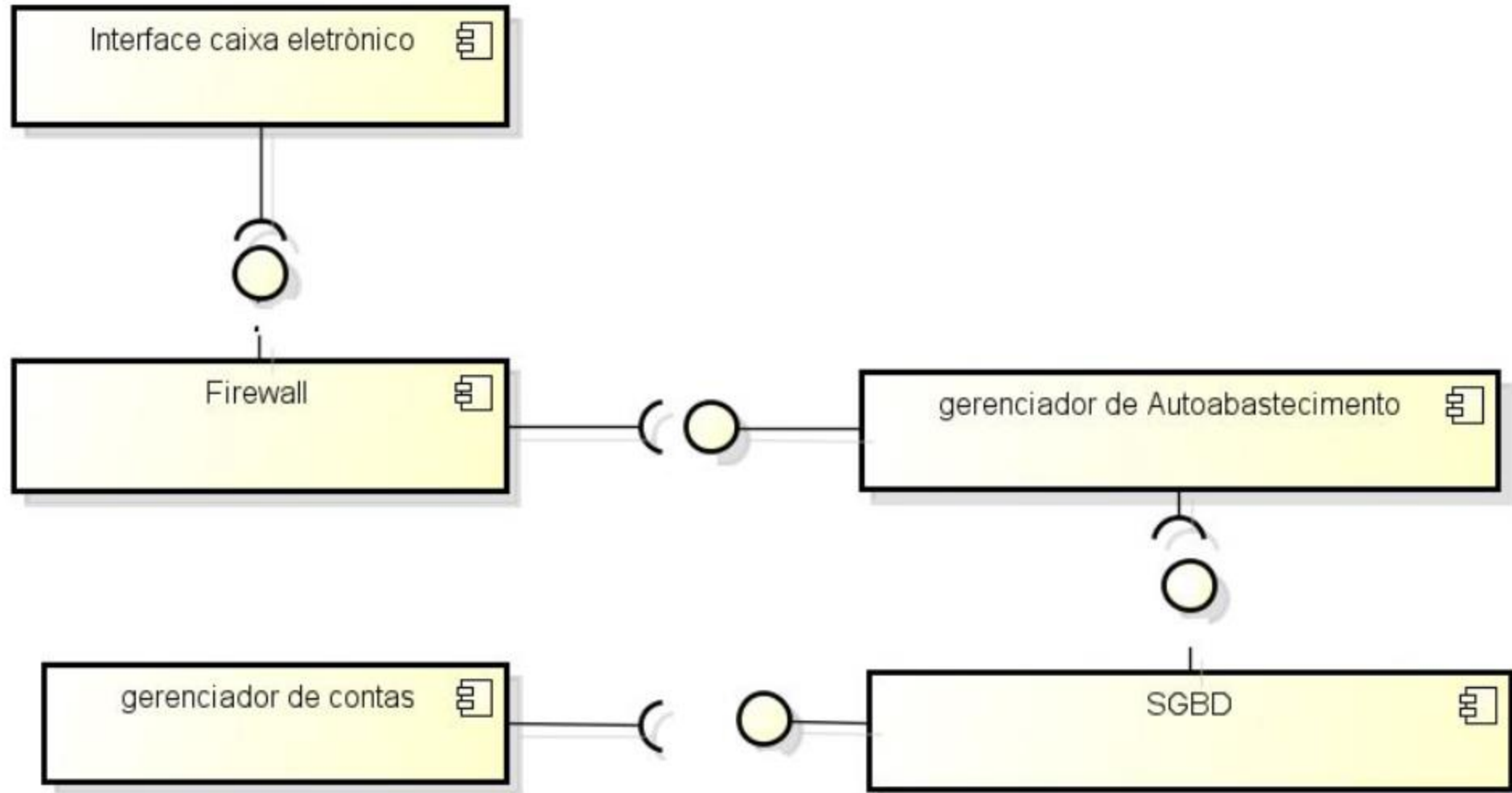
Diagrama de Componentes



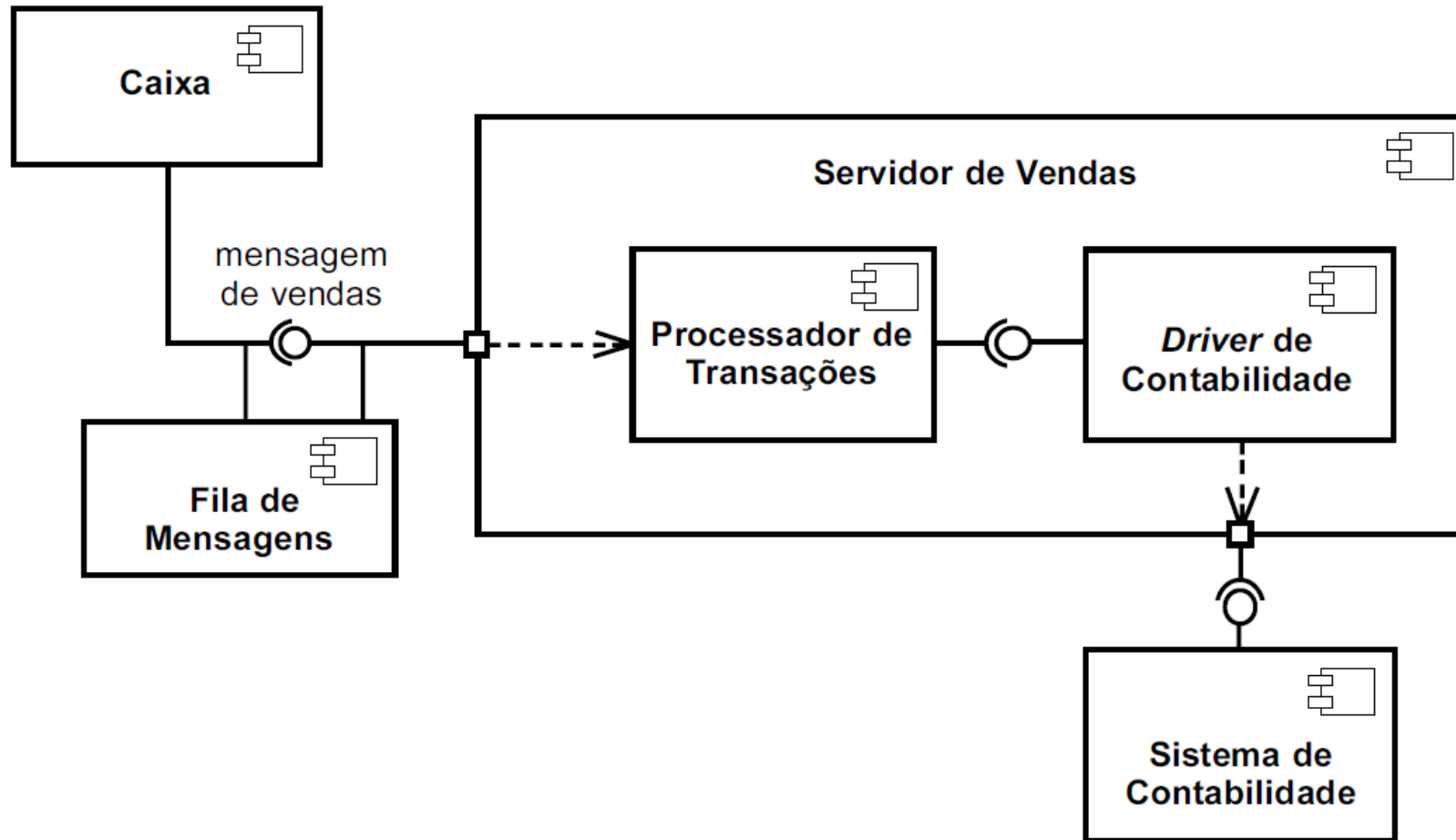
Dependência de Interface: Lançamento de notas

não conhece diretamente a implementação dos outros componentes, apenas interage com eles através de interfaces bem definidas.

Diagrama de Componentes

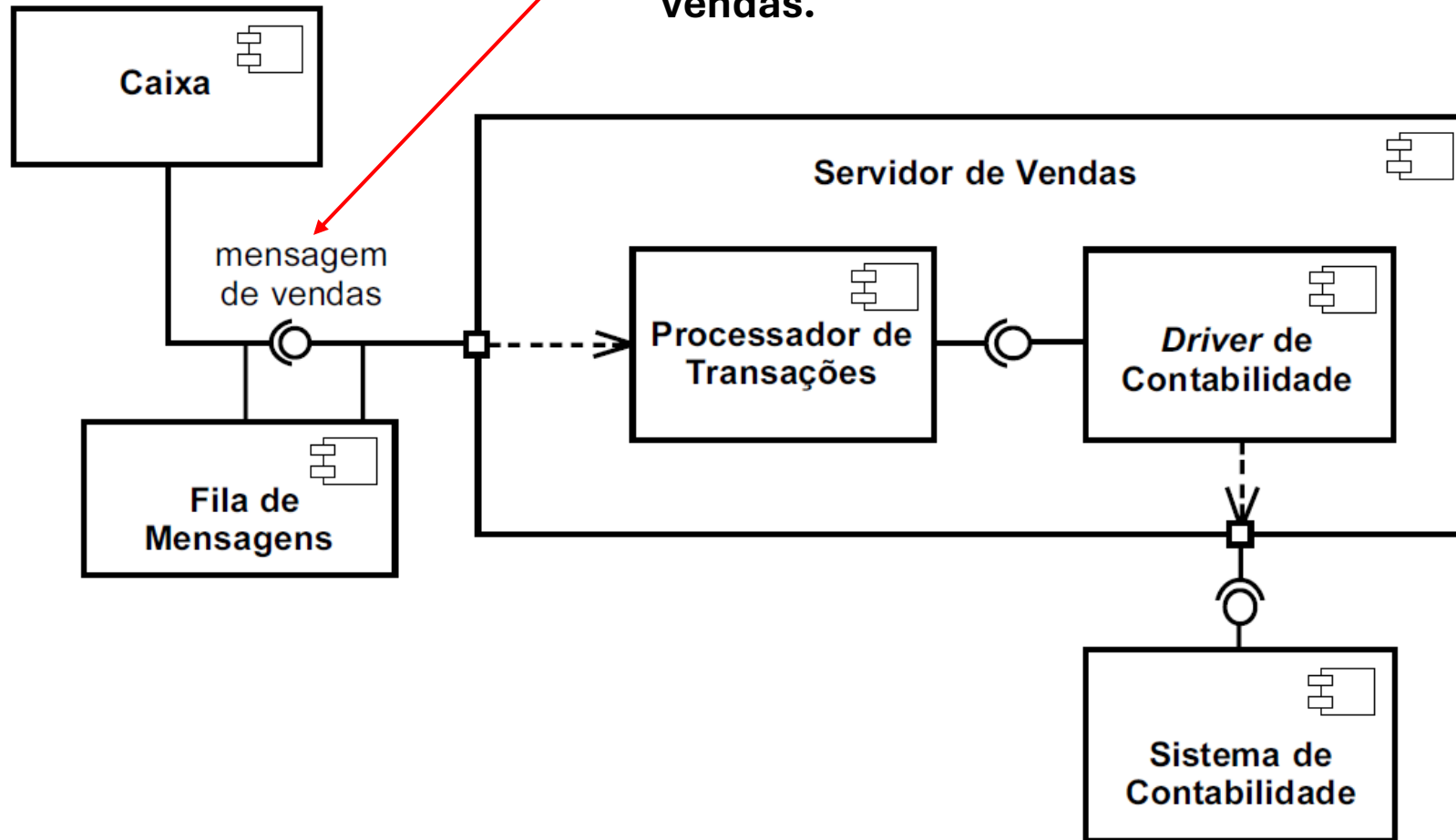


Exemplo de Diagrama

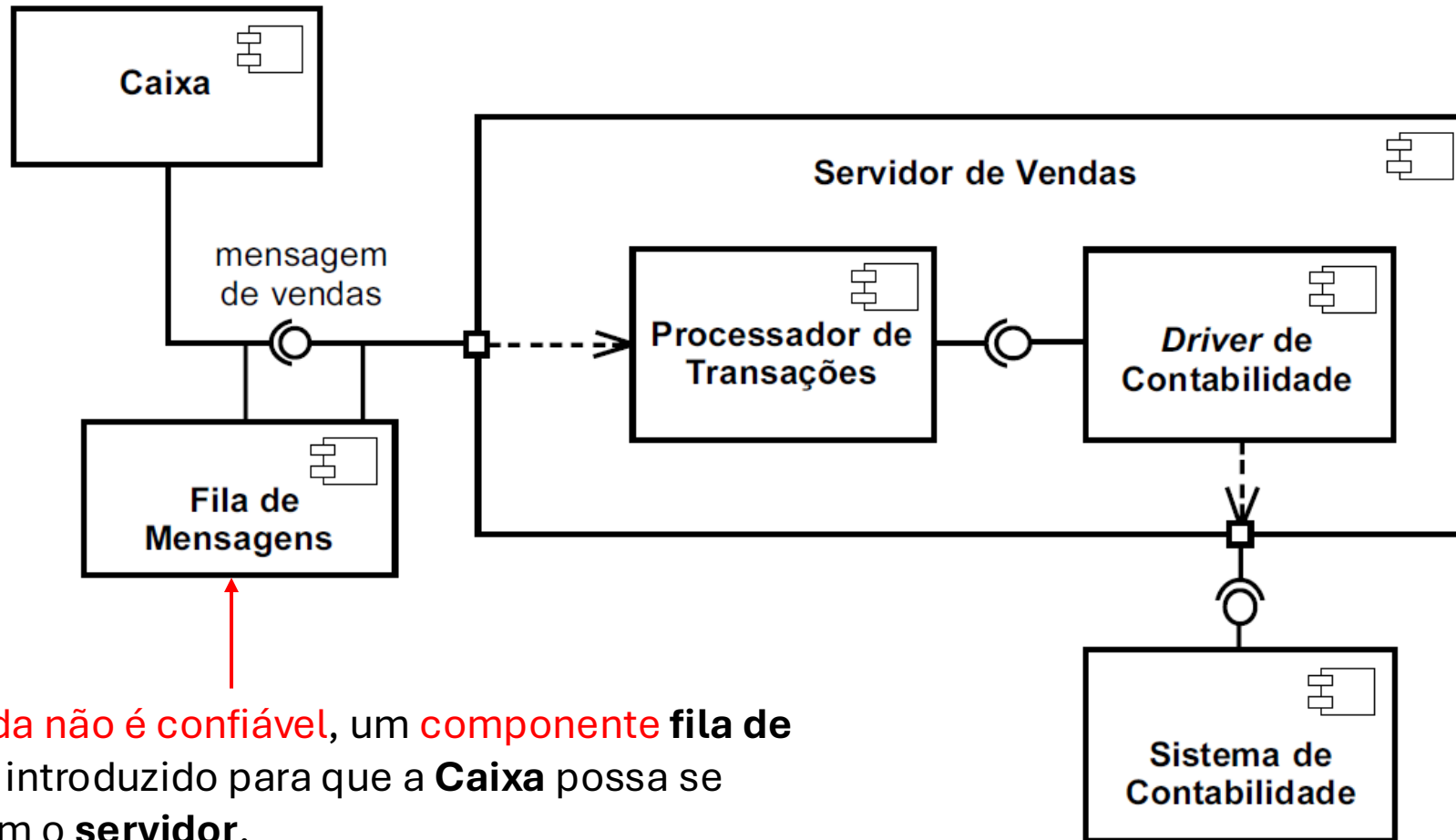


Exemplo de Diagrama

Uma **Caixa registradora** se conecta a um **componente servidor de vendas** usando a **interface mensagem de vendas**.

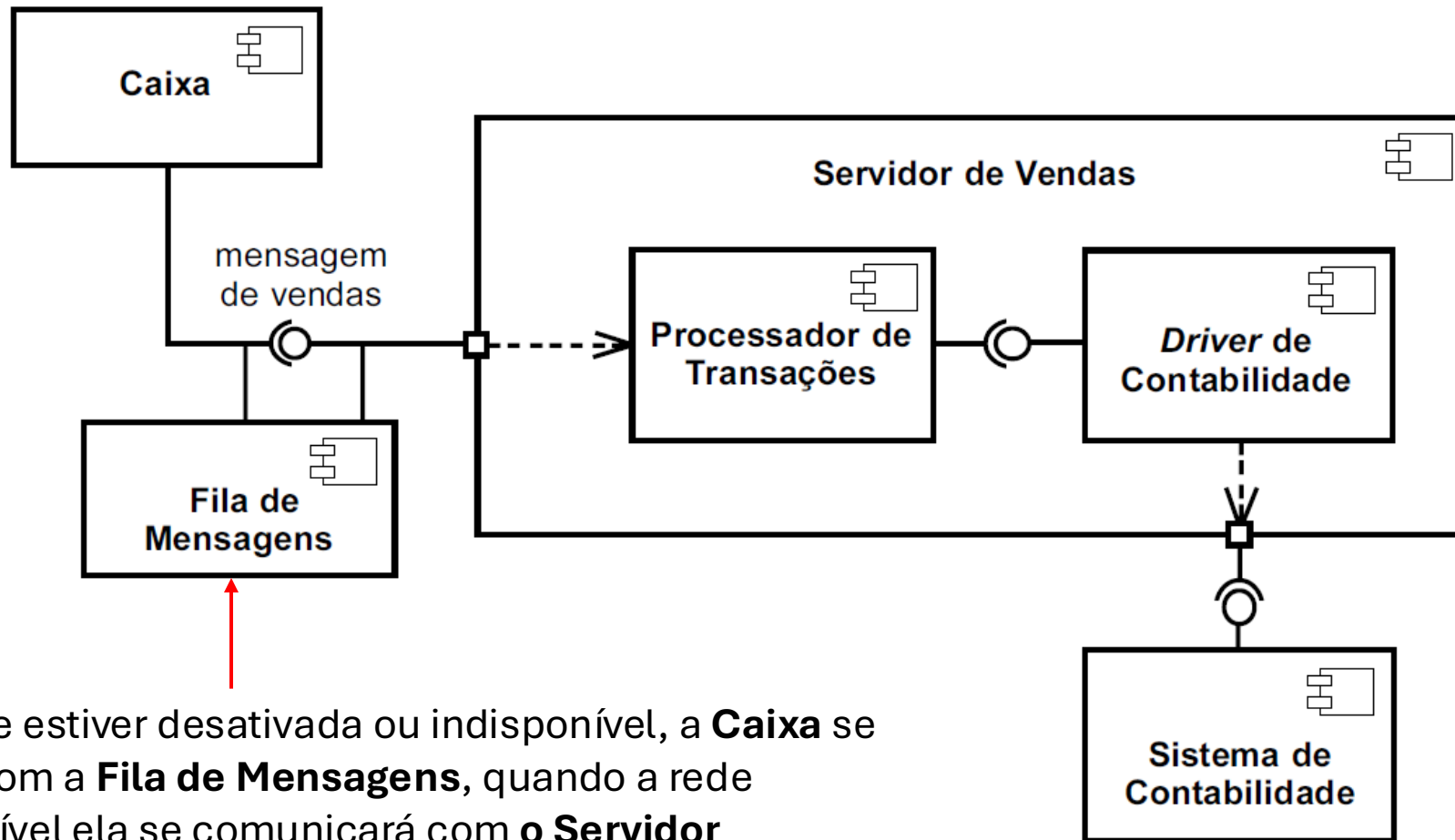


Exemplo de Diagrama



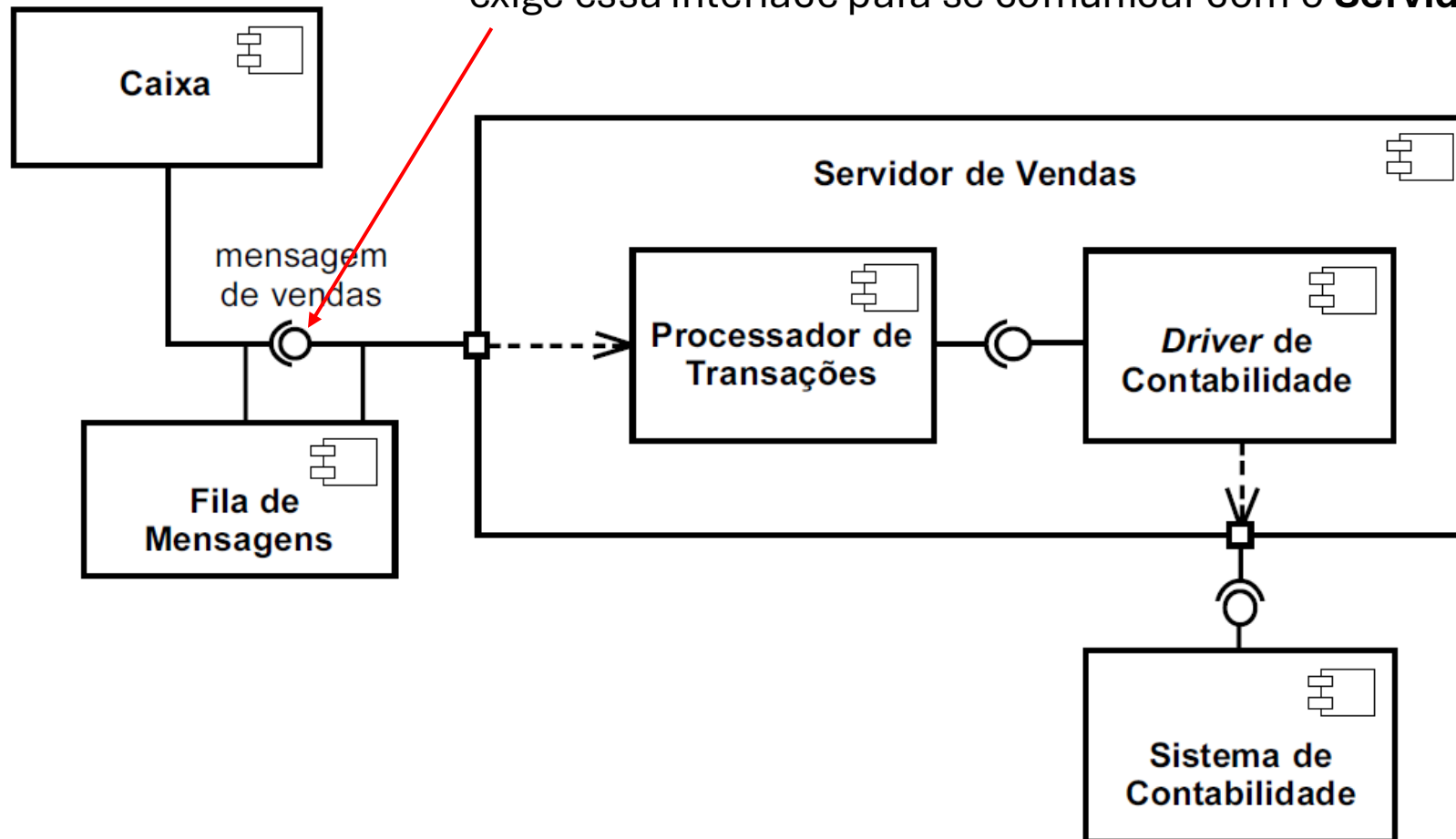
A rede utilizada não é confiável, um componente **fila de mensagens** é introduzido para que a **Caixa** possa se comunicar com o **servidor**.

Exemplo de Diagrama



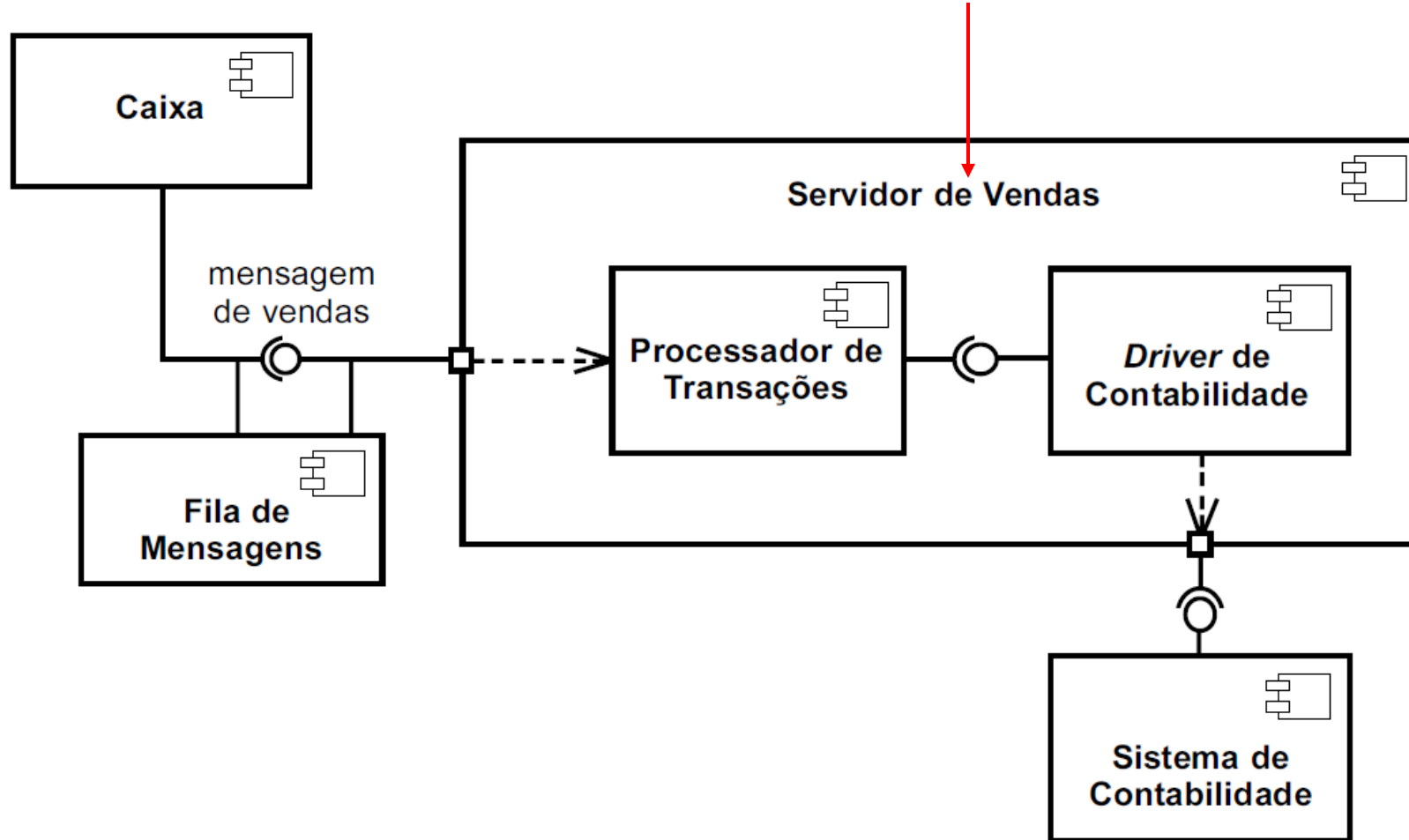
Exemplo de Diagrama

Como resultado, a **Fila de Mensagens** fornece a interface de mensagens de vendas para se comunicar com a **Caixa** e exige essa interface para se comunicar com o **Servidor**.



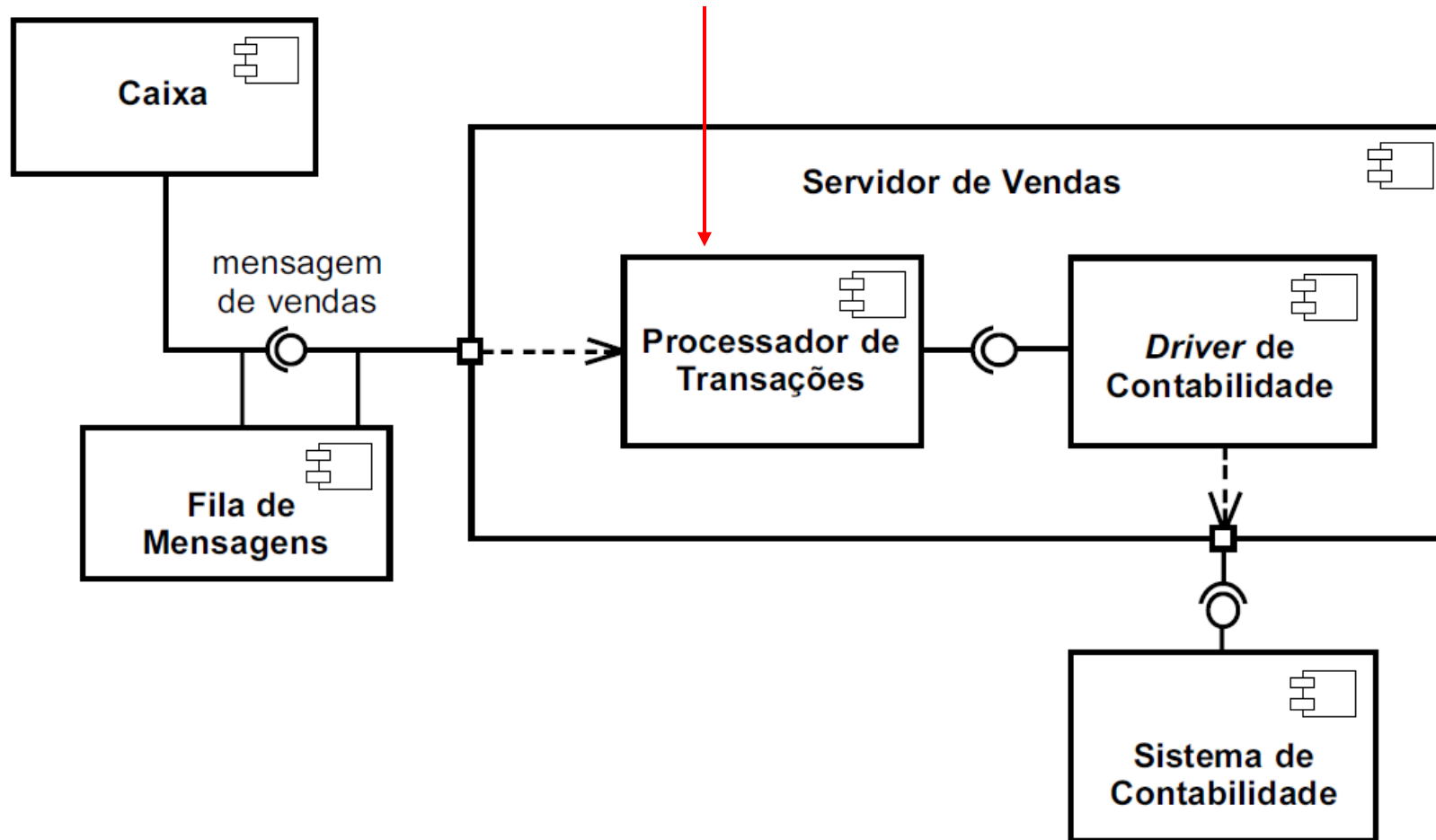
Exemplo de Diagrama

O **Servidor** é dividido em dois componentes principais.



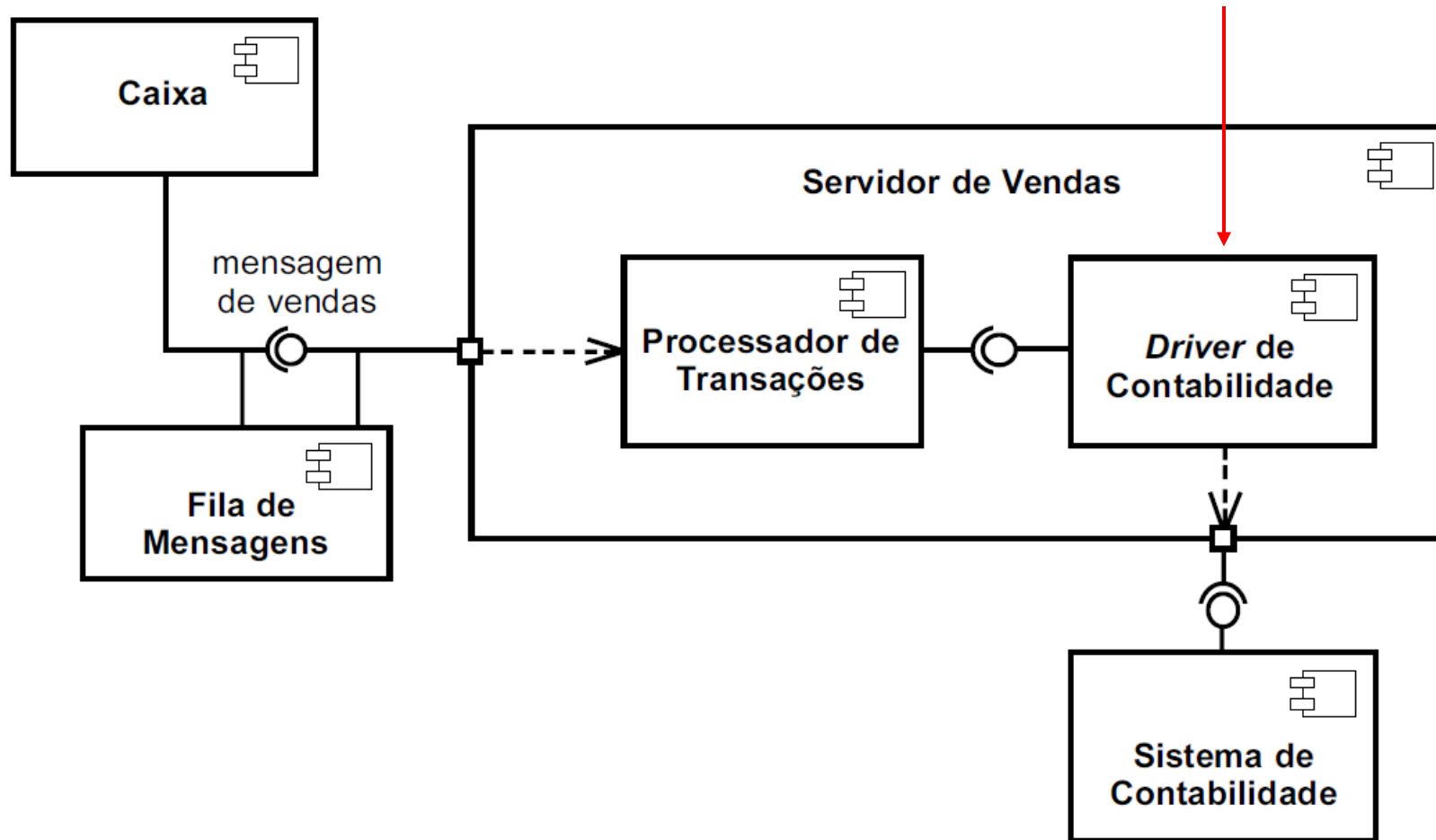
Exemplo de Diagrama

O **Processador de Transações** implementa a interface de mensagens de venda.

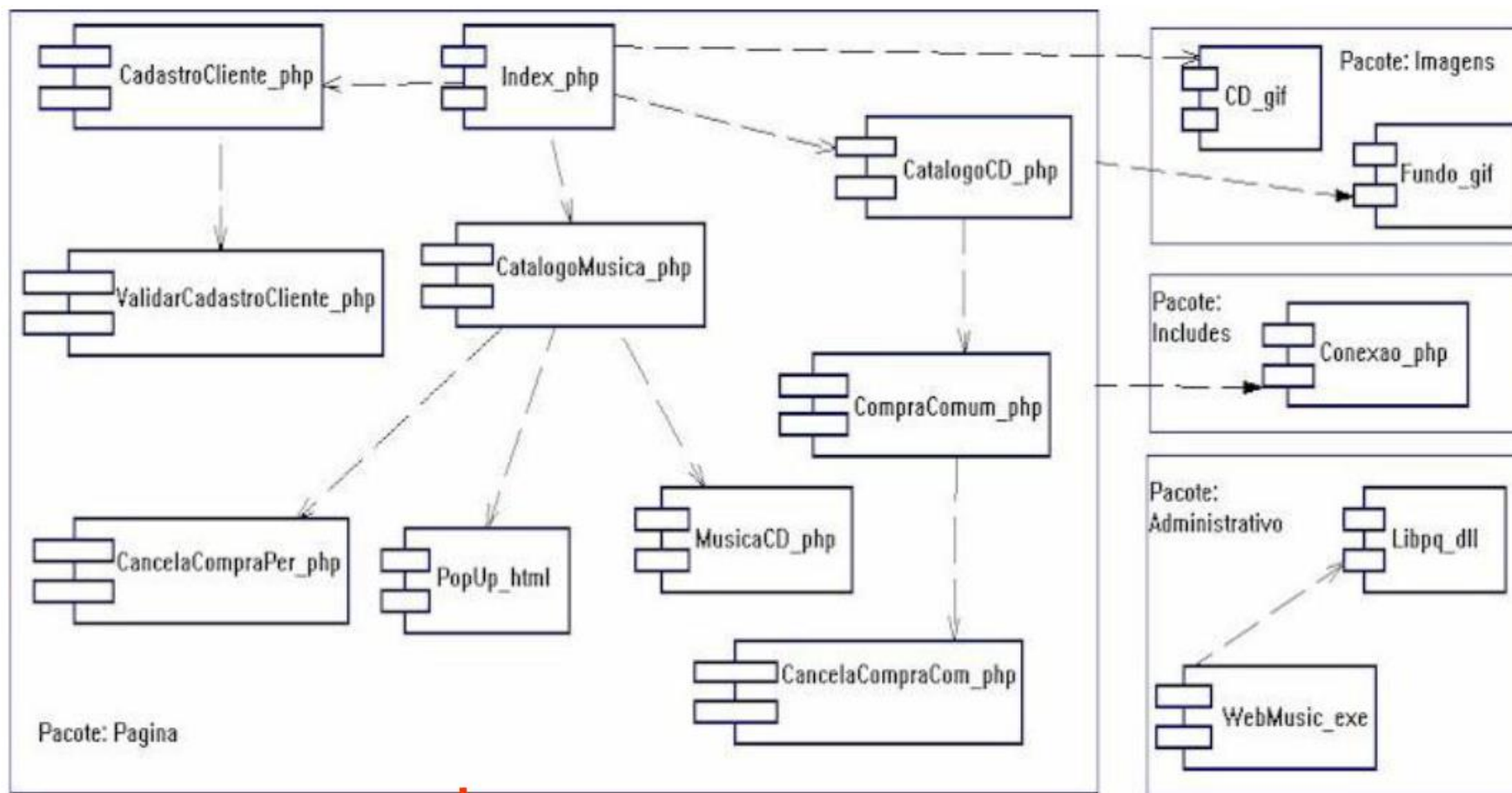


Exemplo de Diagrama

O **Driver** de contabilidade se comunica com o sistema de contabilidade.



Exemplo de Diagrama de Componentes



Pacotes (diretório)

OBS.: essa notação de componentes é da UML 1.5

Diagrama de Componentes - Dicas

- **Foco principal:** comunicar a visão **estática de implementação** do sistema.
- **Seja essencial:** inclua apenas os elementos necessários para o entendimento.
- **Consistência:** mantenha o nível de abstração adequado e coerente.
- **Clareza de propósito:** apresente informação suficiente para explicar o objetivo do diagrama.
- **Apoio visual:**
 - Utilize **estereótipos** para diferenciar papéis dos elementos.
 - Empregue **notas e cores** para destacar informações importantes e facilitar a leitura.