41951- ANÁLISE DE SISTEMAS

Arquitetura do software e a UML - D. de pacotes, components e de instalação

Ilídio Oliveira

v2023/03/14

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

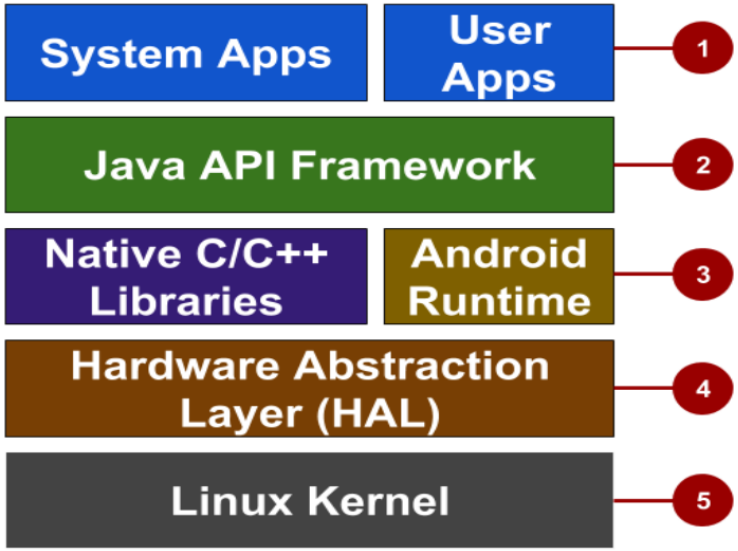
Objetivos de aprendizagem

- Identificar os elementos abstratos de uma arquitetura de software
- Descreva os conceitos de camadas e partições (numa arquitetura em camadas)
- Construir um diagrama de pacotes para ilustrar uma arquitetura lógica
- Interpretar um diagrama de componentes para descrever as partes tangíveis do software
- Construir um diagrama de instalação para descrever a configuração de um sistema

Android *stack* (visão geral)

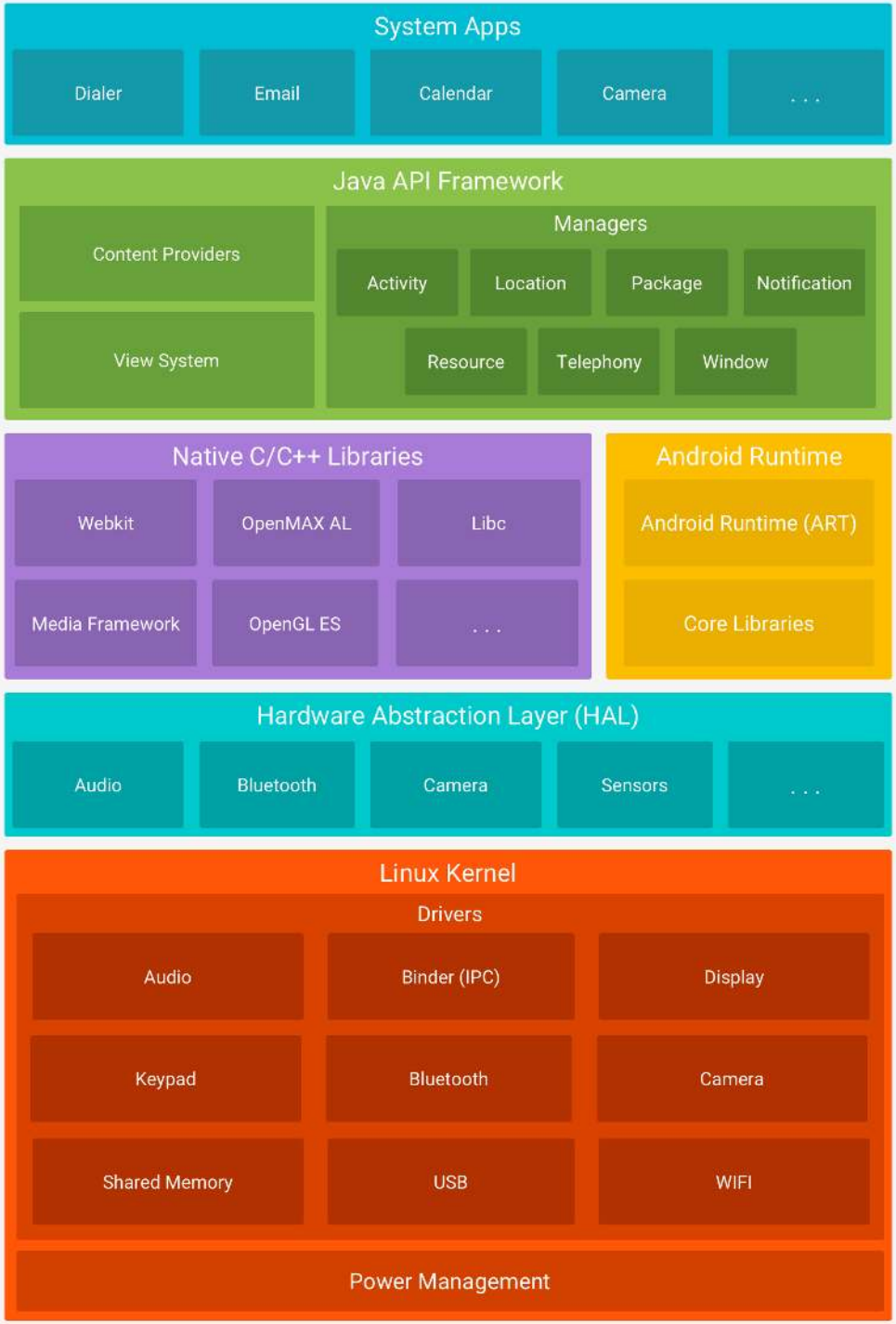
Android stack

1. System and user apps
2. Android OS API in Java framework
3. Expose native APIs; run apps
4. Expose device hardware capabilities
5. Linux Kernel

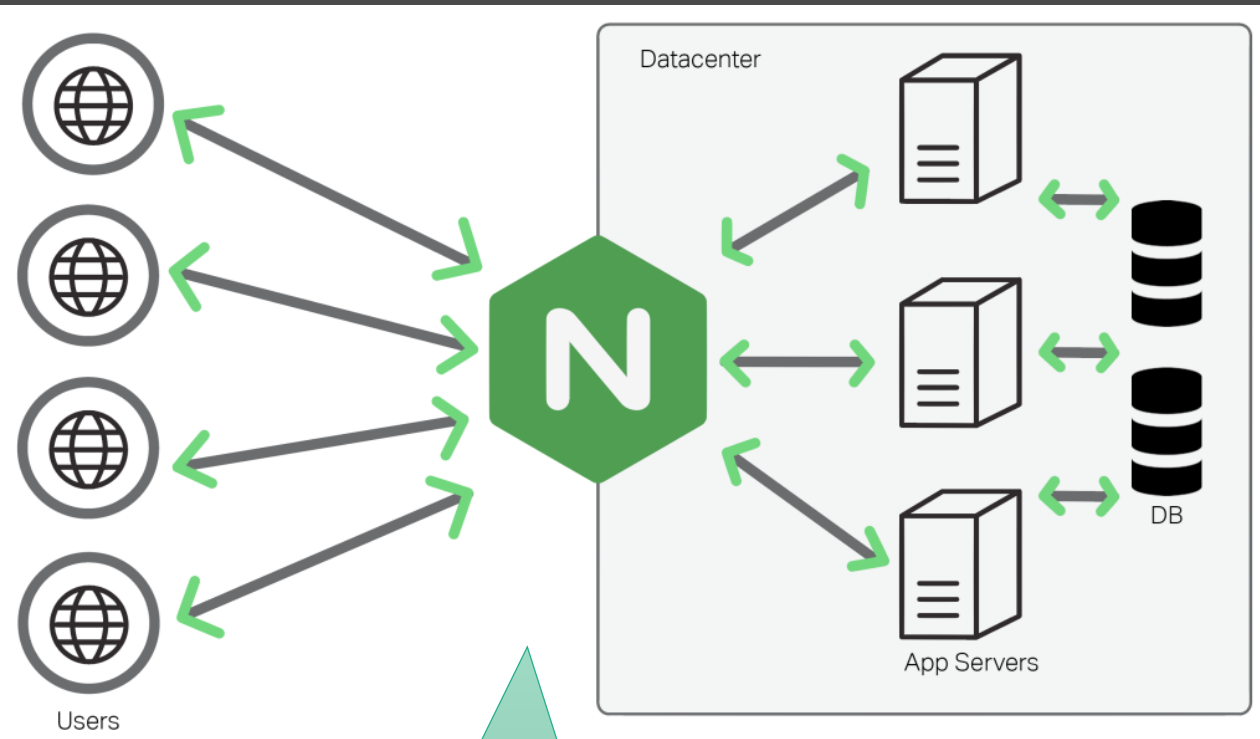


Um exemplo de arquitetura: a organização do sistema Android.

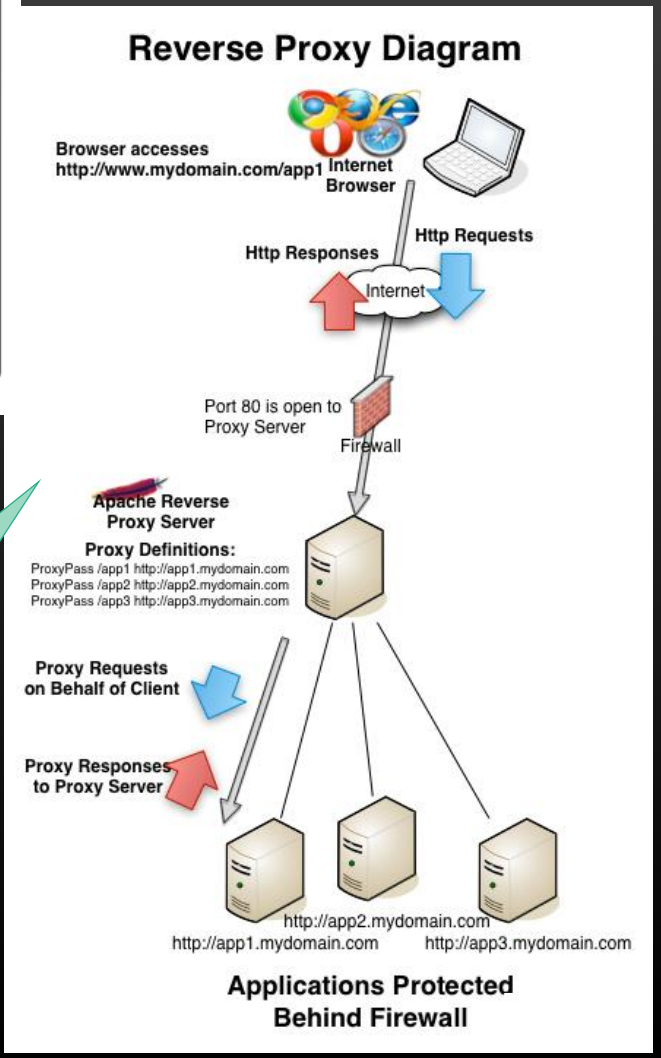
Android *stack*



Módulos em cada camada (partições).



Componentes organizados em 4 *tiers* numa solução de distribuição de carga (com Nginx).



Organização dos serviços de rede numa configuração de “reverse proxy”

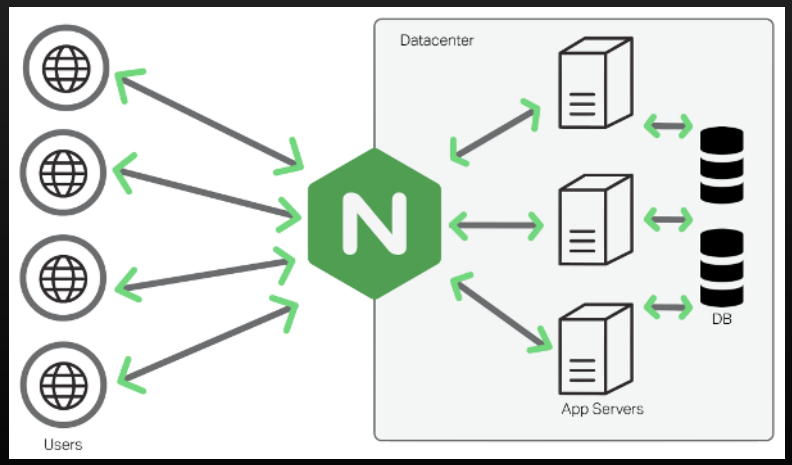
Elementos comuns

O que é que as "ilustrações" anteriores têm em comum?

- → Explicar a organização de uma solução, em termos dos seus "grandes peças" *(high-level)*, representando uma distribuição de responsabilidades
- → Mostrar as principais linhas de dependência (colaboração/comunicação) entre os módulos
- → Equilíbrio entre "Caixa aberta" (ver para dentro da solução) e "Caixa fechada" (sem mostrar a organização interna dos módulos)

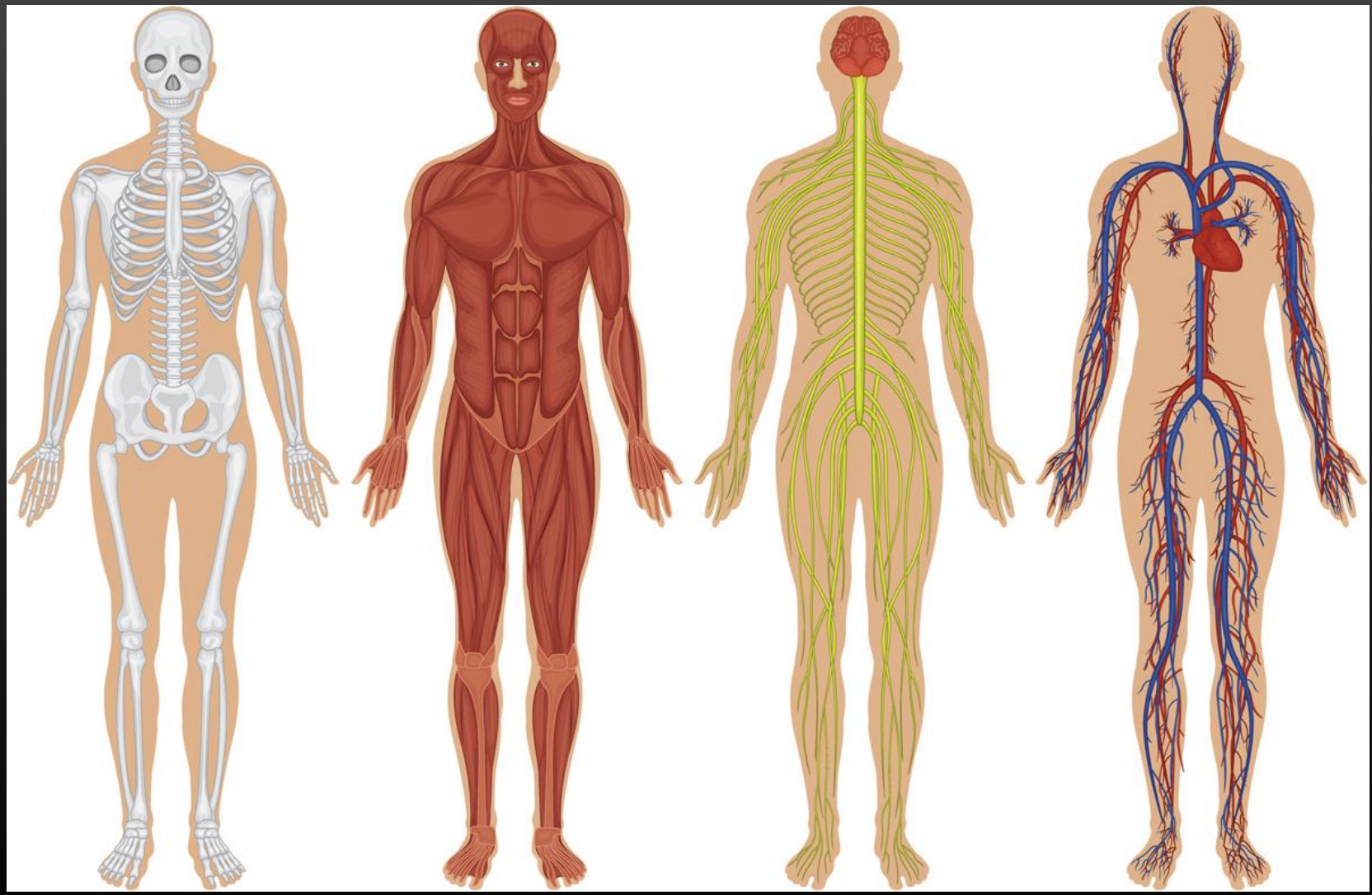
Mas também há diferenças:

→ Vista "lógica" (não mostra instalação) vs. vista de "sistema" (onde é que correm os componentes)



O que trata a arquitetura do software?

Analogia: sistemas feitos de estruturas… → ideia de partes, relações, comportamento.



O que é a arquitetura de software?

A Arquitectura é um Conjunto de Estruturas (de Software)

Partes/elementos relevantes ligados por alguma relação.

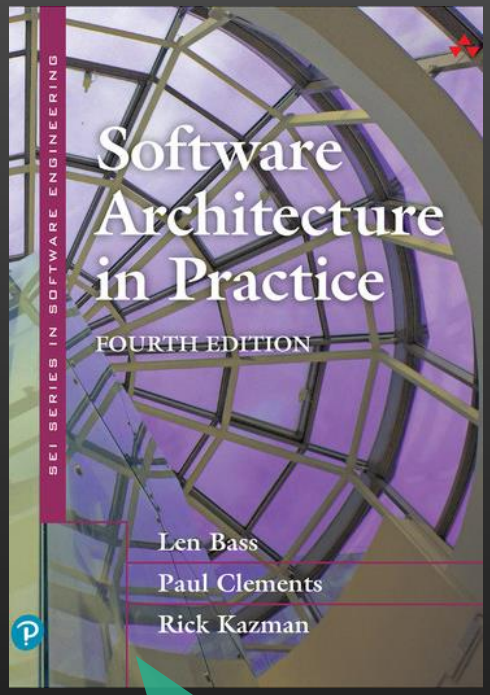
Arquitectura é uma Abstracção

Omite intencionalmente certas informações sobre os elementos que não são úteis para o raciocínio sobre o sistema

Detalhes privados dos elementos (têm a ver exclusivamente com a implementação interna) não são de arquitectura

Arquitectura Inclui Comportamento

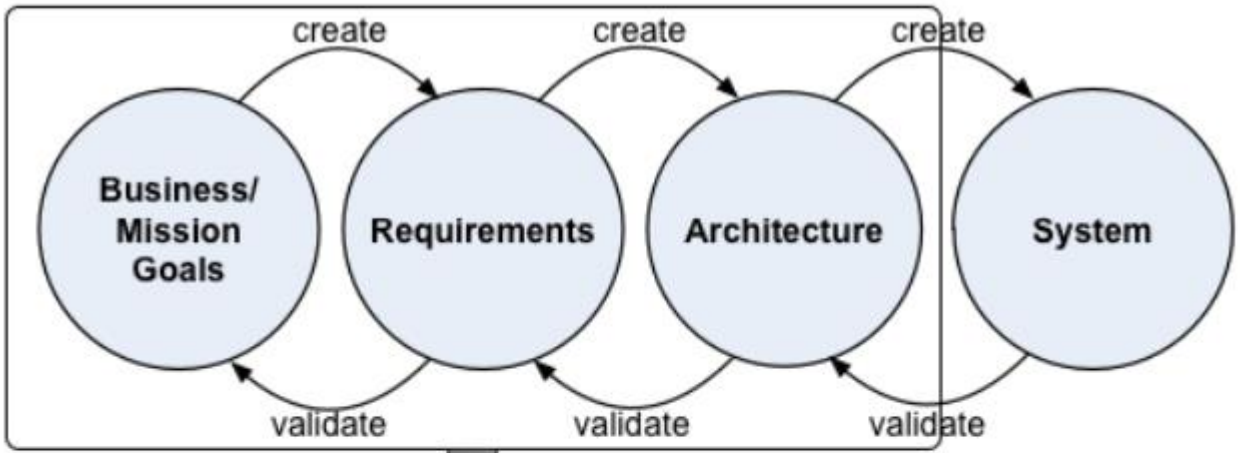
O comportamento dos elementos engloba a forma como interagem uns com os outros e com o ambiente.



The software architecture of a system is the set of structures needed to reason about the system. These structures comprise software elements, relations among them, and properties of both.

Architecture is about reasoning-enabling structures.

Papel do arquiteto (de software)



Core Skill Sets

- Design - create and evolve
- Analysis - will the design provide the needed functions and qualities?
- Models and representations - "documentation"
- Evaluation - are we satisfying stakeholders?

- Communication – with technical and business teams
- Technical Leadership

https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=455074

Elementos de uma arquitetura

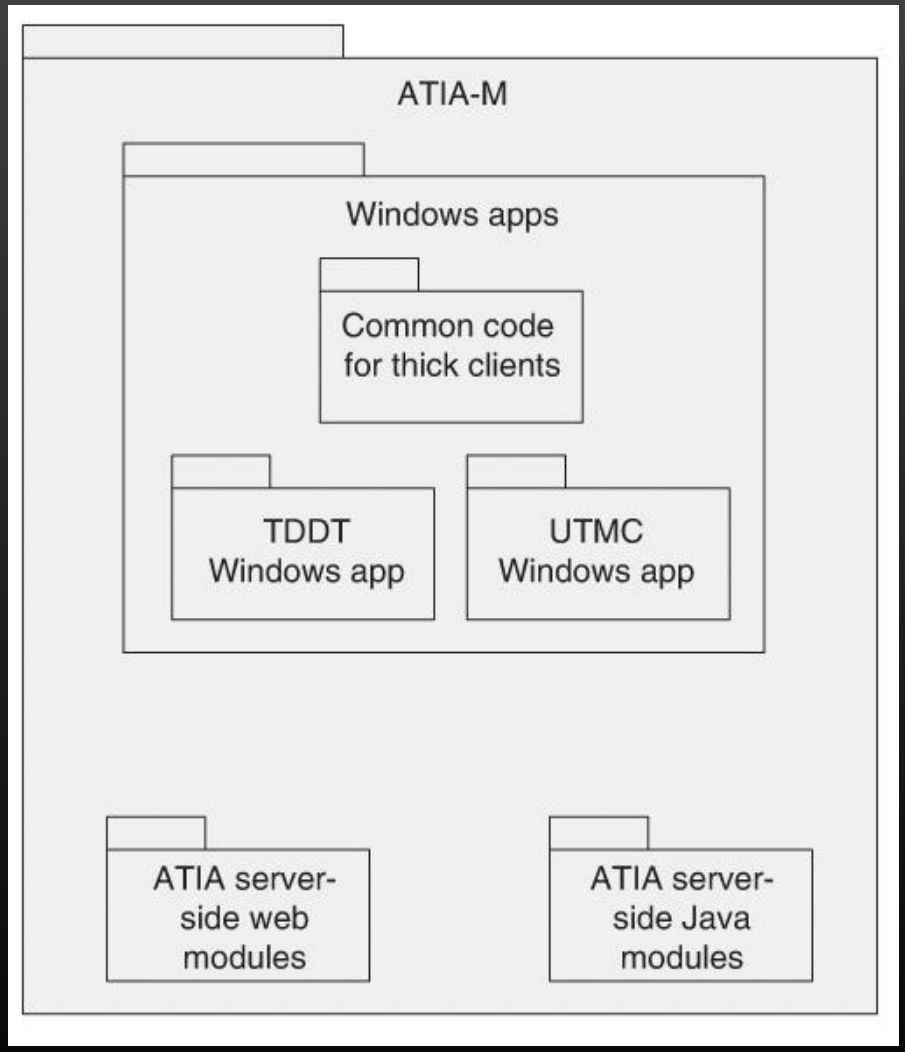
Tipos de estruturas 1(3)

1/ estruturas do tipo **módulo**

dividem os sistemas em unidades de implementação. Os módulos mostram como um sistema pode ser dividido como um conjunto de unidades de código ou de dados, que devem ser implementadas ou adquiridas.

Os módulos são usados para atribuir trabalho às equipas.

Elementos do tipo módulo podem ser classes, pacotes, camadas, ou meramente divisões de funcionalidade, todas elas refletindo unidades de implementação.

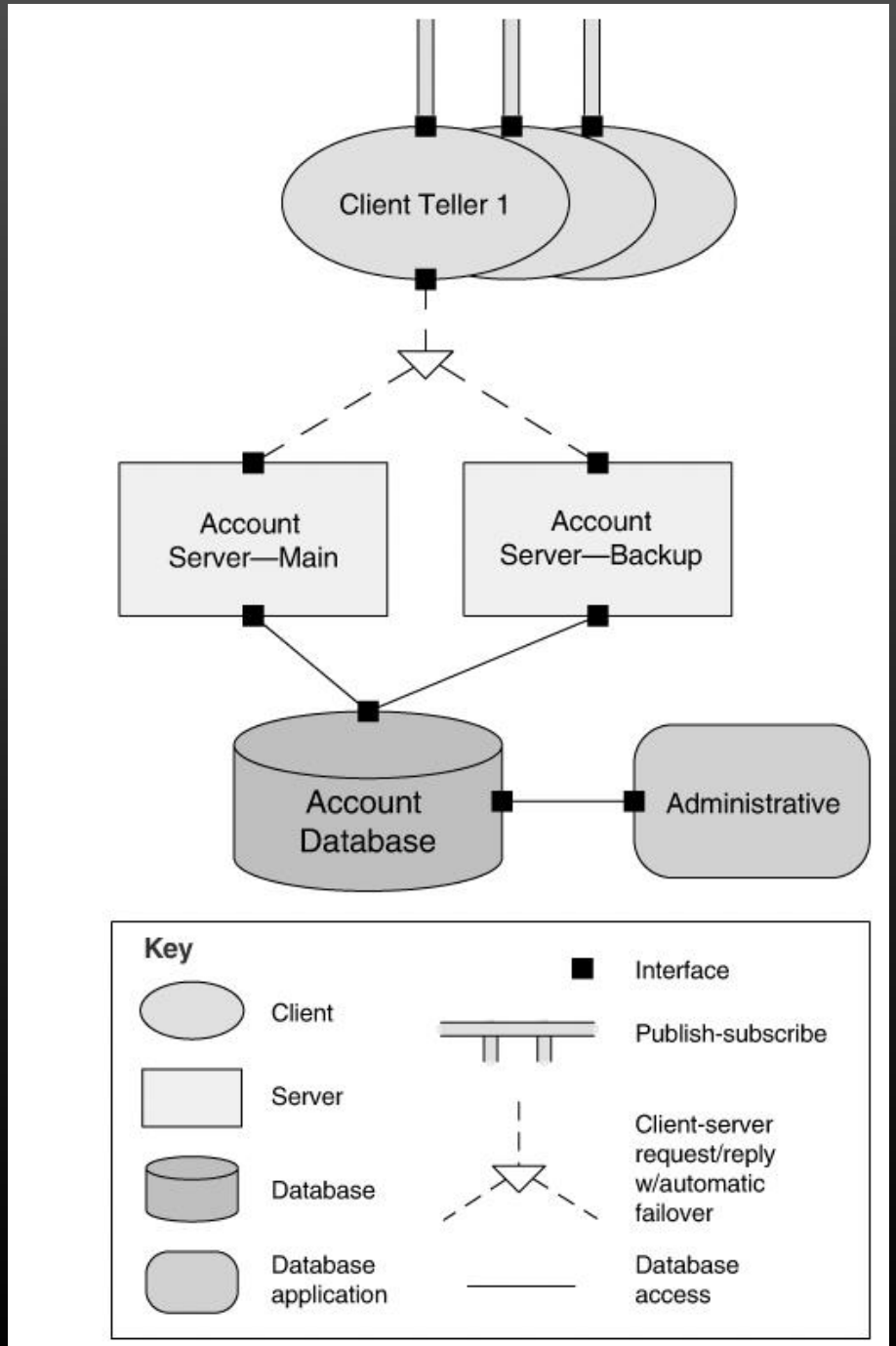


Três tipos de estruturas 2(3)

2/ estruturas de componentes e conectores (***Component-and-connector*** - C&C)

focam-se na forma como os elementos interagem uns com os outros em tempo de execução para realizar as funções do sistema.

Descrevem como o sistema é estruturado como um conjunto de elementos que têm comportamento em tempo de execução (componentes) e interacções (conectores).



Tipos de estruturas 3 (3)

3/ estruturas de alocação *(Allocation structures)*

estabelecem o mapeamento das estruturas de software nas estruturas de não-software do sistema, tais como os seus ambientes de desenvolvimento, teste, e execução.

As estruturas de alocação respondem a questões como as seguintes:

- Em que processador(es) cada elemento de software é executado?

- Em que directórios ou ficheiros é cada elemento armazenado durante o desenvolvimento, teste e construção do sistema?

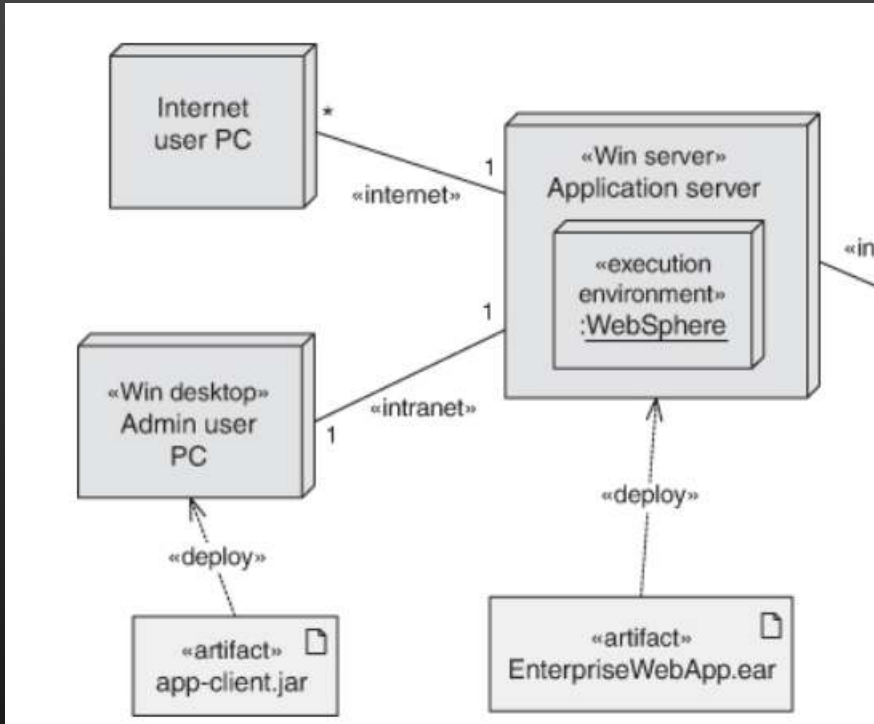
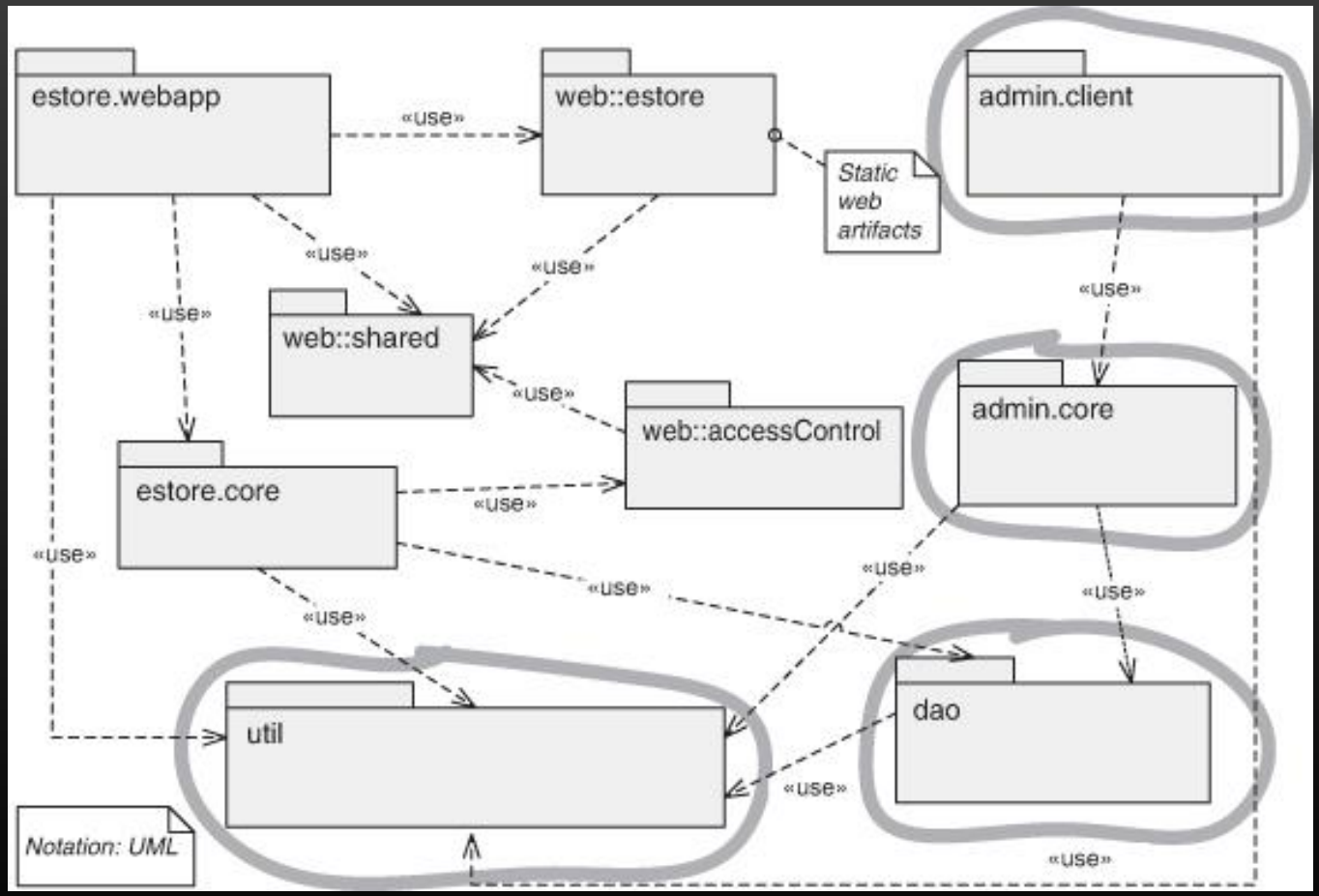


Figure 1.10 Deployment structure

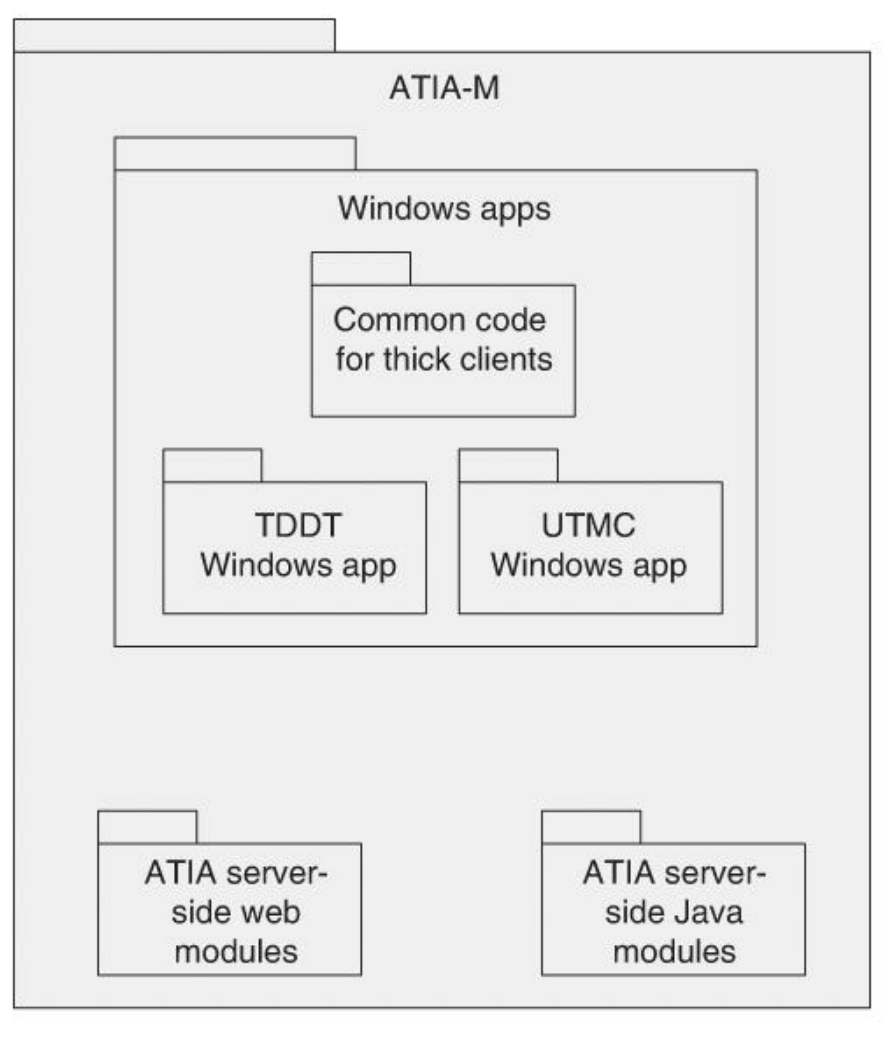
Exemplo de estrutura: módulos

Relação <<uses>>: as unidades estão relacionadas pela relação de *uses*, uma forma especializada de dependência.



Exemplo de estrutura: módulos

Decomposição (relação “is-a-submodule-of”)



Exemplo de estrutura: módulos

Layers: uma camada "virtualiza" os subsistemas subjacentes e fornece um conjunto coeso de serviços através de uma interface controlada.

the UNIX System V operating system.

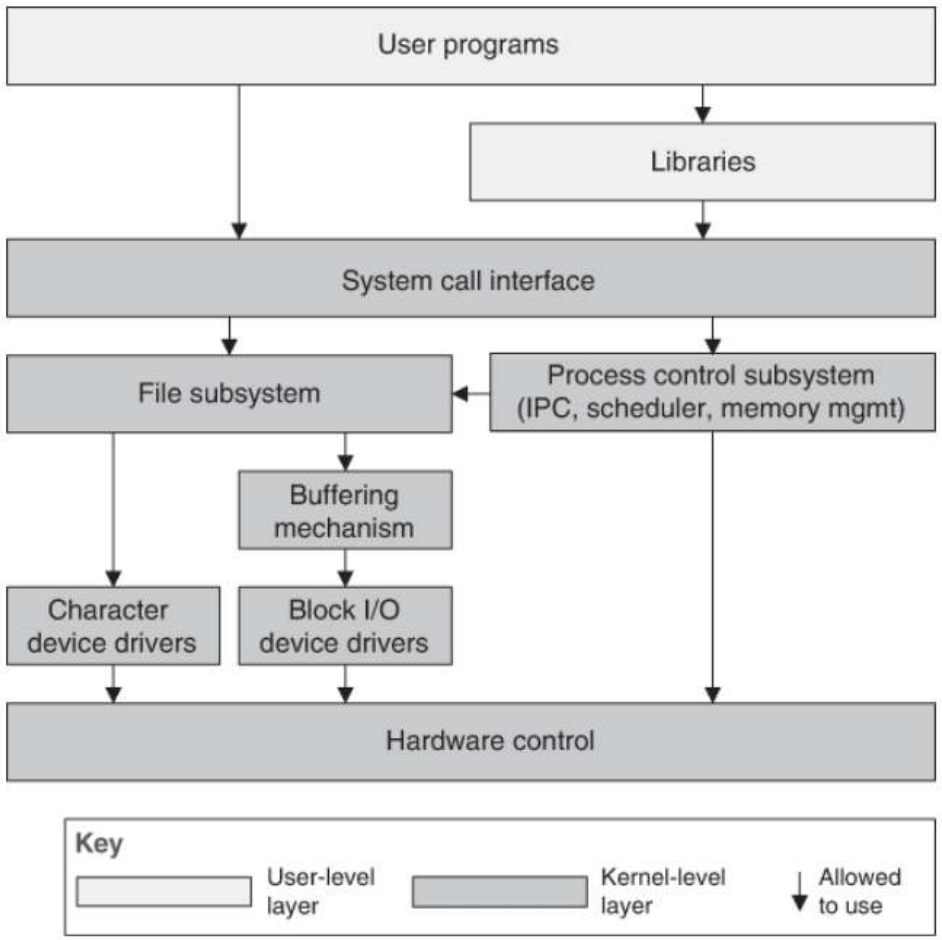
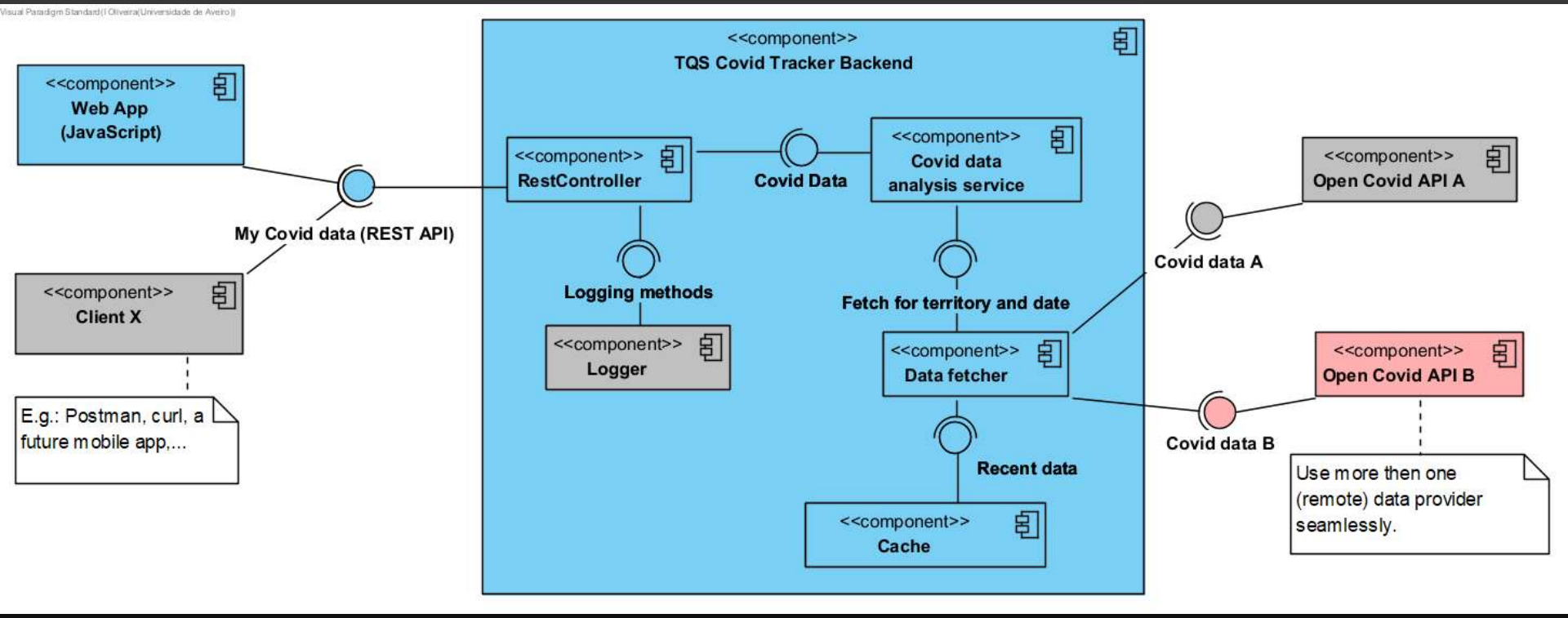


Figure 1.7 Layer structure

Caso de exemplo: componentes-conetores (C&C)

Service structure: as unidades aqui são serviços (em runtime) que interoperam através de um mecanismo de coordenação.



Casos de exemplo: alocação/atribuição

O diagrama de instalação mostra a alocação de unidades a nós computacionais.

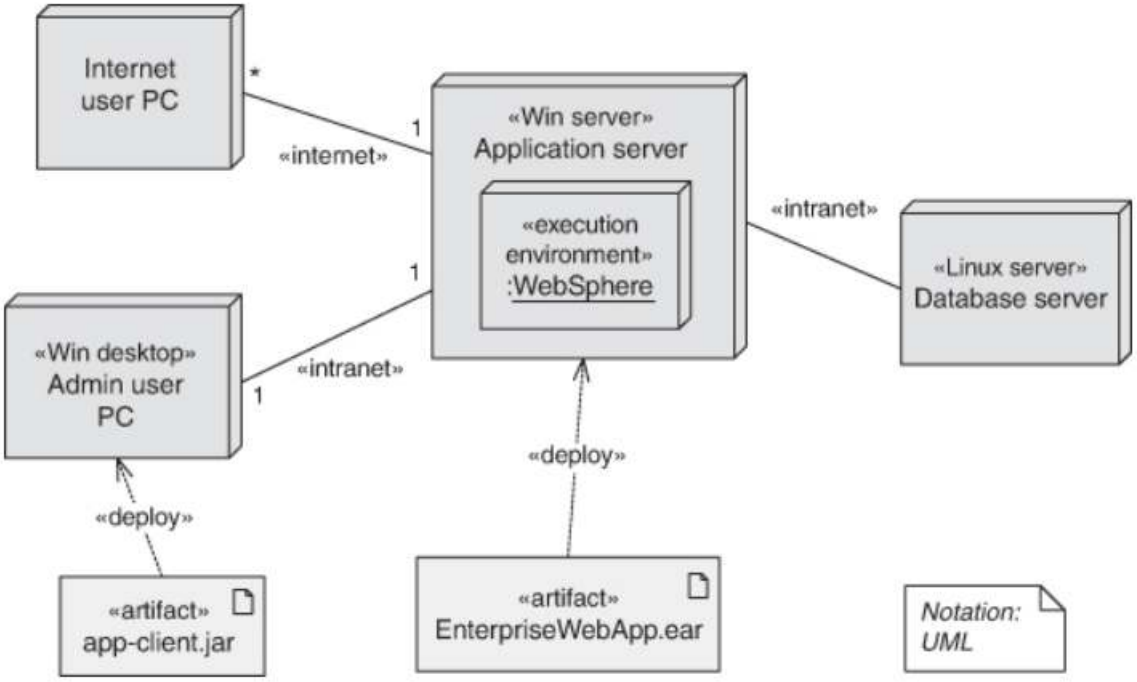
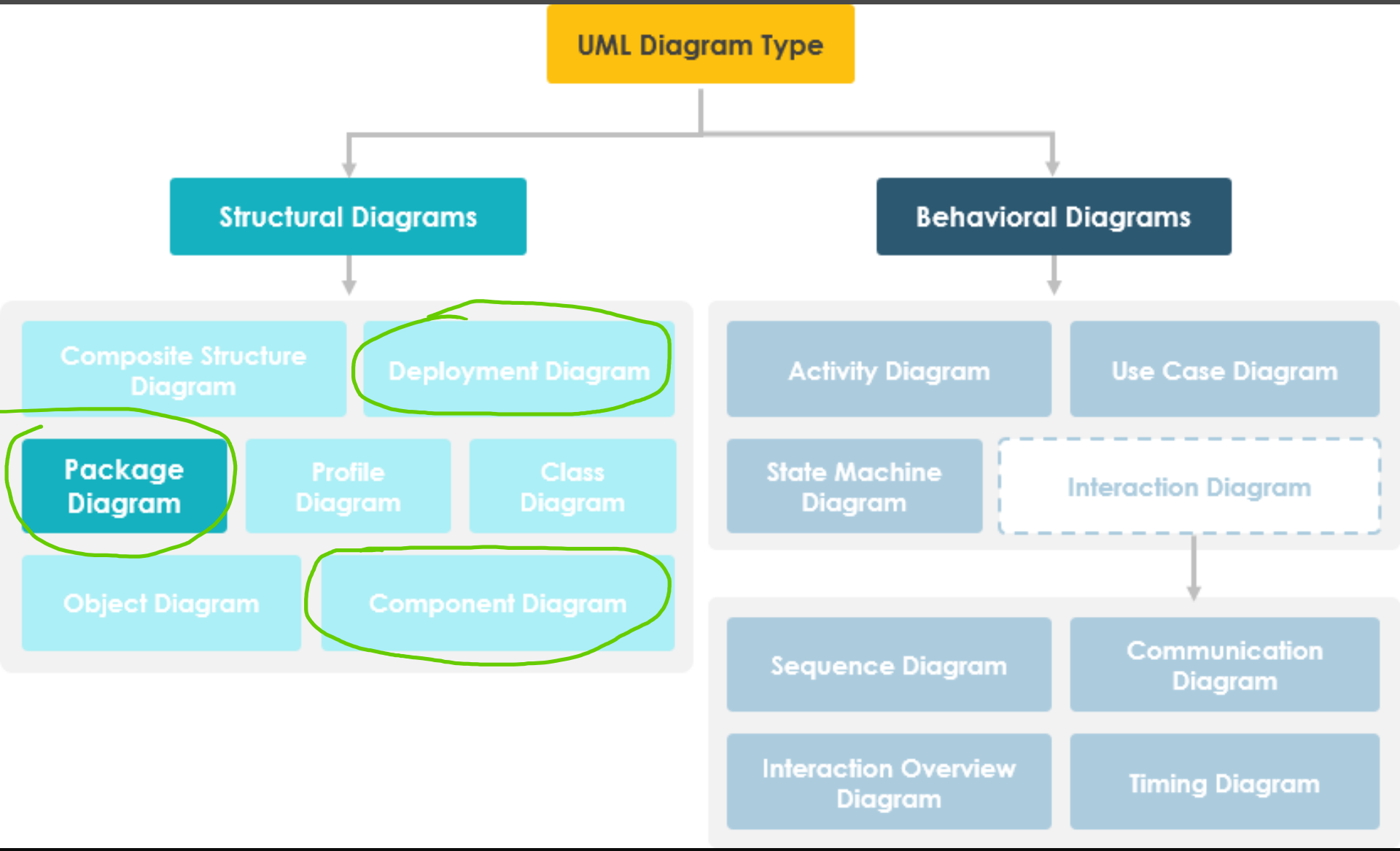


Figure 1.10 Deployment structure

VIEWS	Modules	C&C	Allocation
Key element	**Modules**, which are implementation units of software that provide a coherent set of responsibilities	**Components**: principal processing units and data stores. **Connectors**: pathways of interaction between components.	Software elements and environmental/processing elements.
Relations	**Is-part-of**, which defines a part/whole relationship between the submodule (the part) and the aggregate module (the whole) **Depends-on**, defines a dependency between two modules **Is-a**, which defines a generalization/specialization relationship between a more specific module (the child) and a more general module (the parent)	**Attachments** (connections): Components are associated with connectors to yield a graph.	**Allocated-to**: A software element is mapped (allocated to) an environmental element.
Sample usages	• Blueprint for construction of the code • Analysis of the impact of changes • Planning incremental develop. • Communicating the functionality of a system and the structure of its code base • Supporting the definition of work assignments, implementation schedules, and budget information	• Guide development by specifying the structure and behavior of runtime elements. • Reasoning of “redistributable” libraries/services. • Help reason about runtime system quality attributes, such as performance and availability.	For reasoning about performance, availability, security, and safety. For reasoning about the form and mechanisms of system installation.

Vistas de arquitetura na UML


UML Diagram Type
Structural Diagrams
Behavioral Diagrams
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Class Diagram
Object Diagram
Component Diagram
Activity Diagram
Use Case Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

A arquitetura do sistema aborda diferentes perspetivas de análise

❶ Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

❷Arquitetura de componentes do software

Peças construídas com uma tecnologia concreta

Construção “modular”

- E.g.: existem pré-feitos?

❸ Arquitetura de instalação

Visão dos equipamentos e configuração de produção (conectividade, distribuição,...)

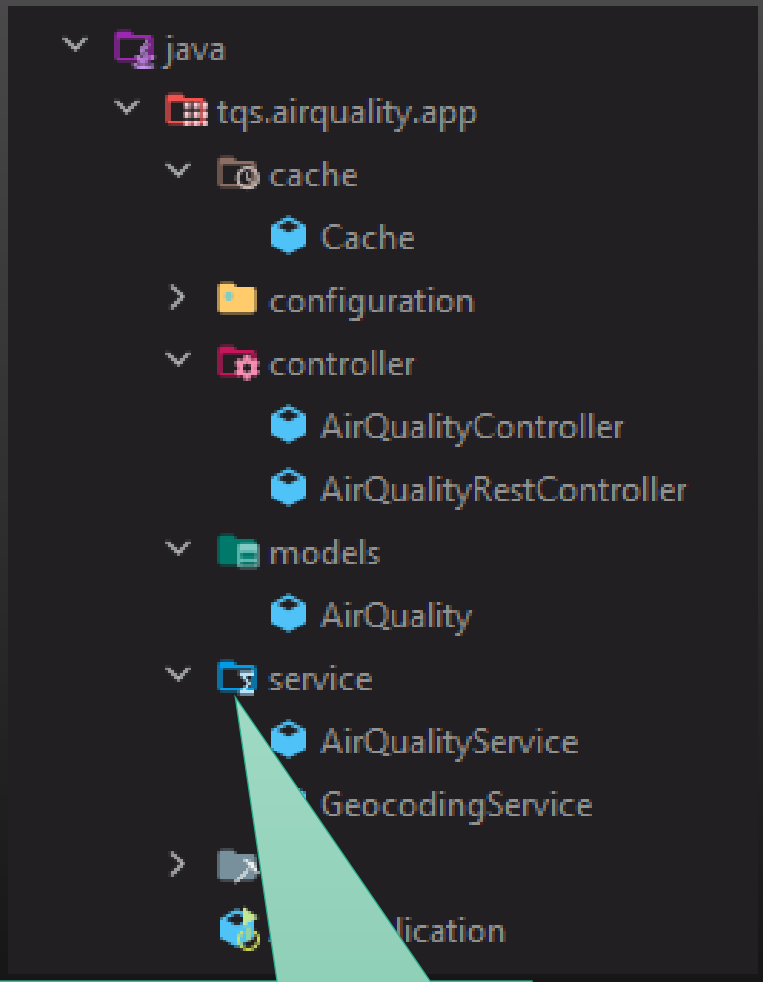
Arquitetura lógica

Organização geral da solução em blocos (*packages)*

Os *packages* podem representar agrupamentos muito diferentes.

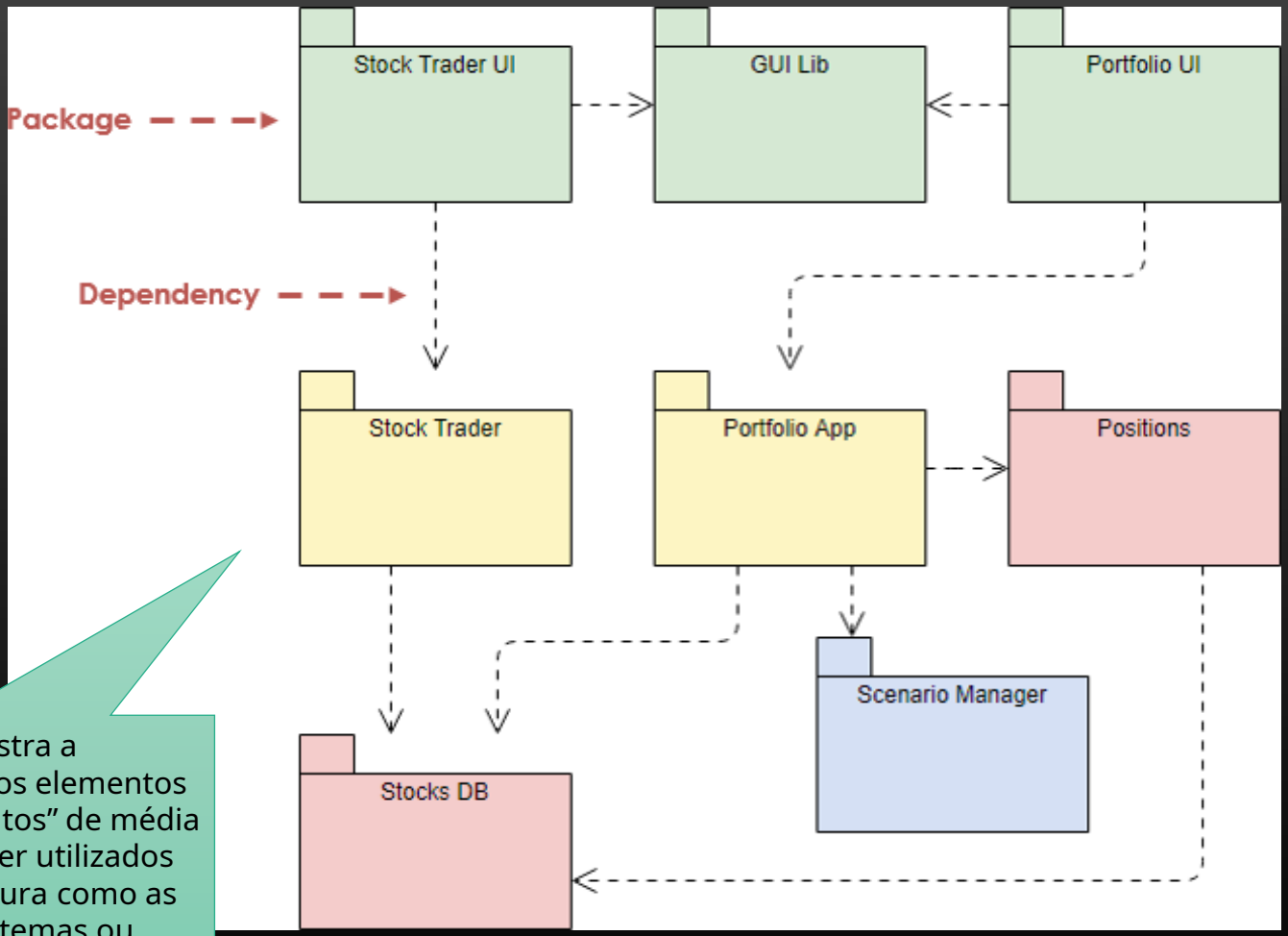
E.g.:

- *Packages* num programa em Java
- *Packages* num modelo UML
- Subsistemas/divisões do sistema sob especificação



A ideia de package é usada em Java para formar grupos de entidades relacionadas.
Os *packages* podem ser hierárquicos.

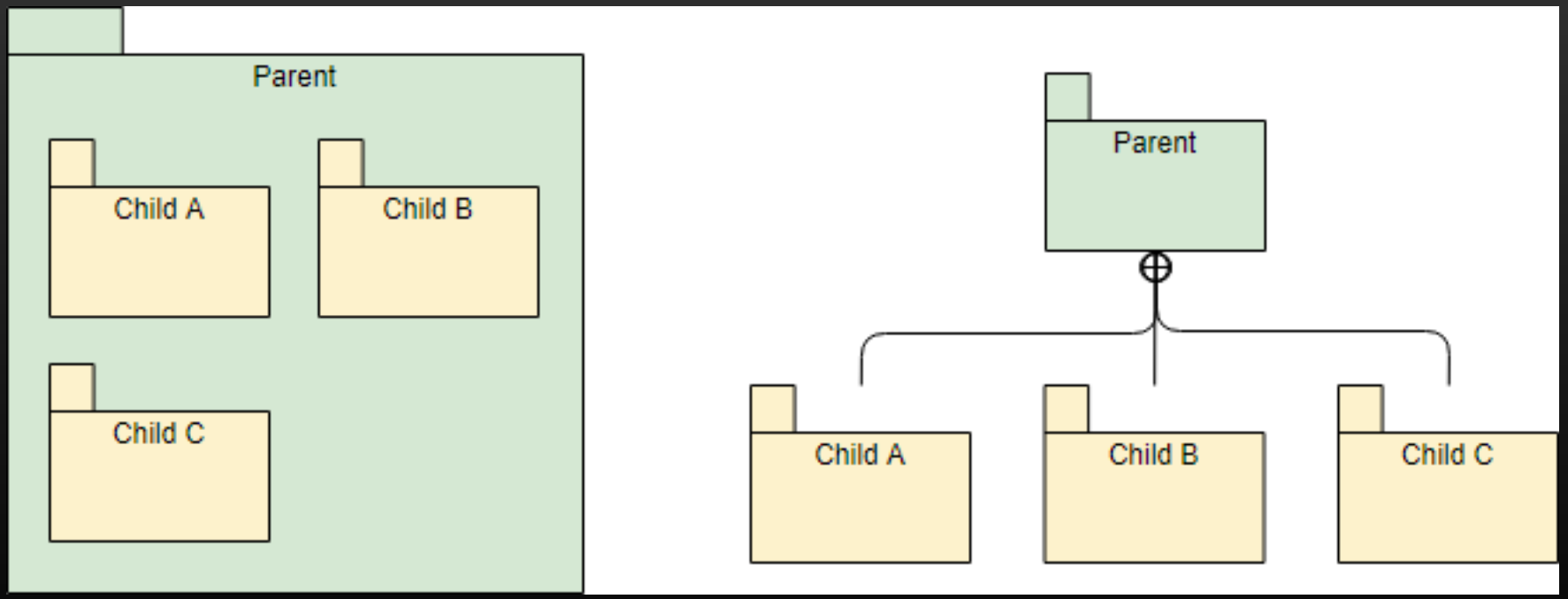
Elementos do diagrama de pacotes



O diagrama de pacotes mostra a disposição e organização dos elementos do modelo em "agrupamentos" de média a larga escala que podem ser utilizados para mostrar tanto a estrutura como as dependências entre sub-sistemas ou módulos.

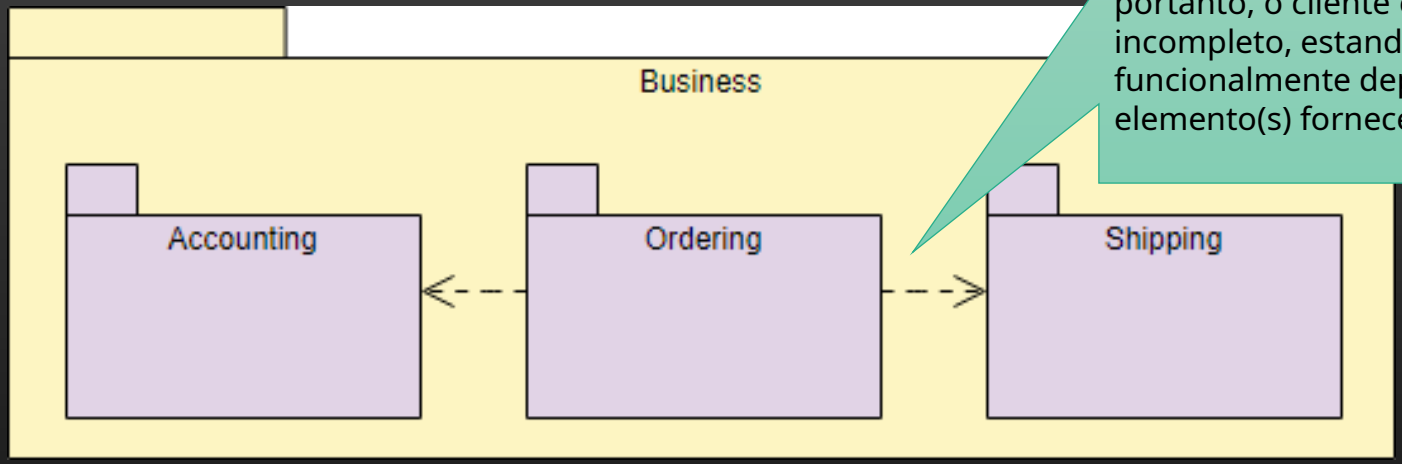
Comunicar a arquitetura lógica com pacotes (*packages)*

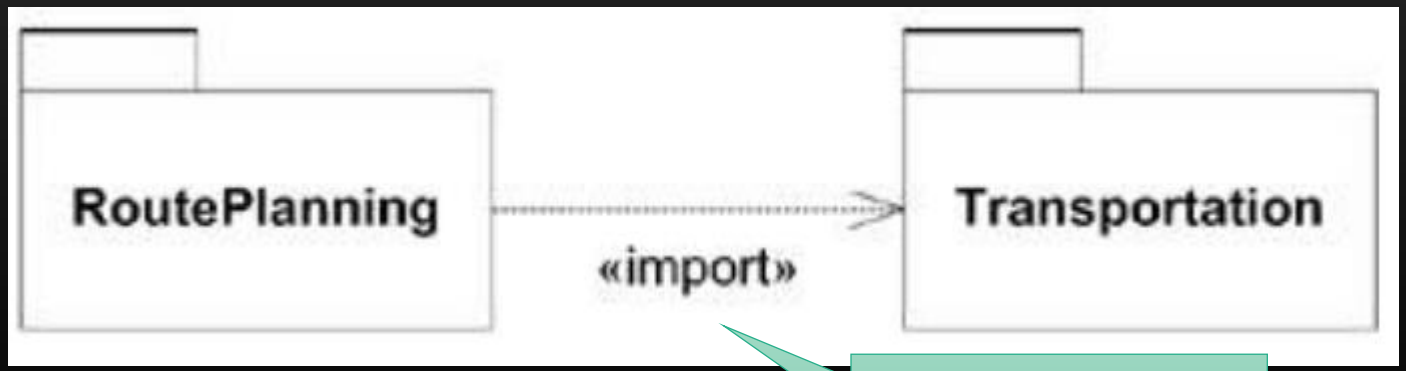
"Contido em"
- duas representações alternativas



Associações entre pacotes

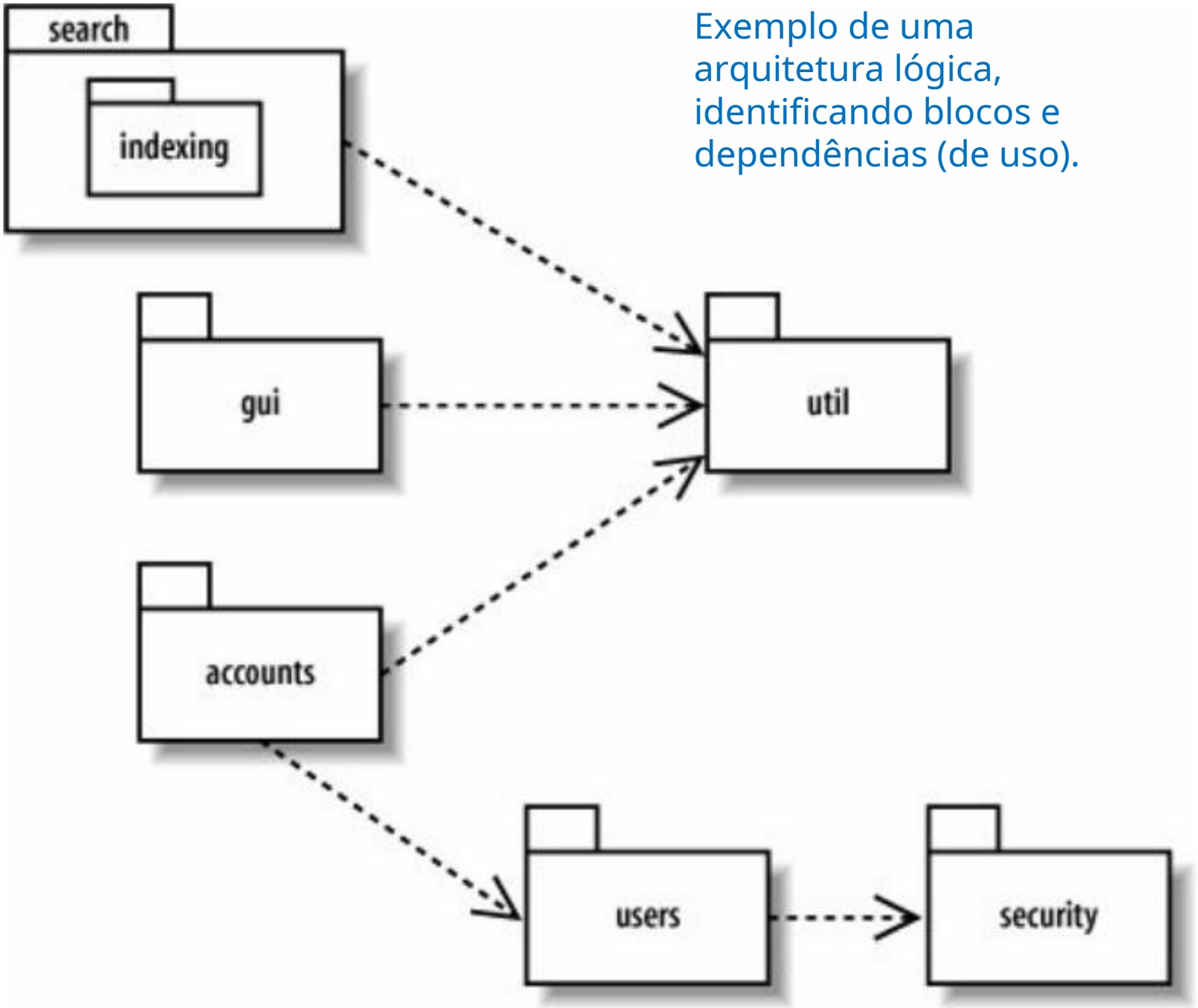
Dependência → transmite a ideia que partes de um elemento (pacote) precisam de (usar) partes de outro.
A dependência é designada como uma relação fornecedor - cliente, em que o fornecedor proporciona algo ao cliente e, portanto, o cliente é, em certo sentido, incompleto, estando semanticamente ou funcionalmente dependente do(s) elemento(s) fornecedor(es)





um pacote importa a funcionalidade de outro pacote

Exemplo de uma arquitetura lógica, identificando blocos e dependências (de uso).



A arquitetura do sistema aborda diferentes perspetivas de análise

❶ Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

❷Arquitetura de componentes do software

Peças construídas com uma tecnologia concreta

Construção “modular”

- E.g.: existem pré-feitos?

❸ Arquitetura de instalação

Visão dos equipamentos e configuração de produção (conectividade, distribuição,...)

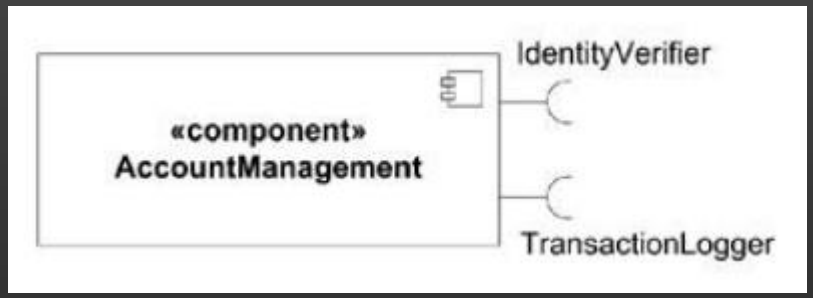
Componente

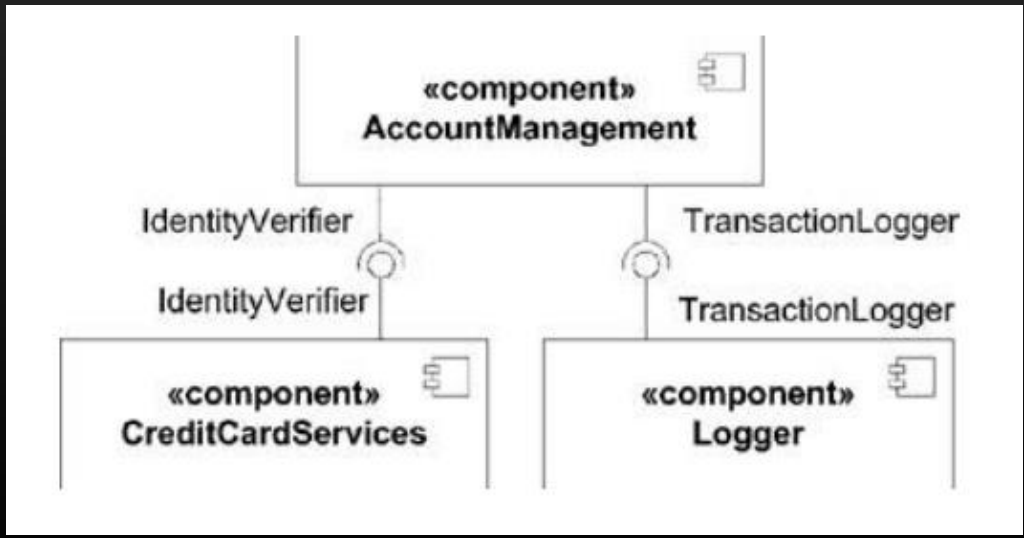
É normal dividir sistemas complexos em subsistemas mais geríveis

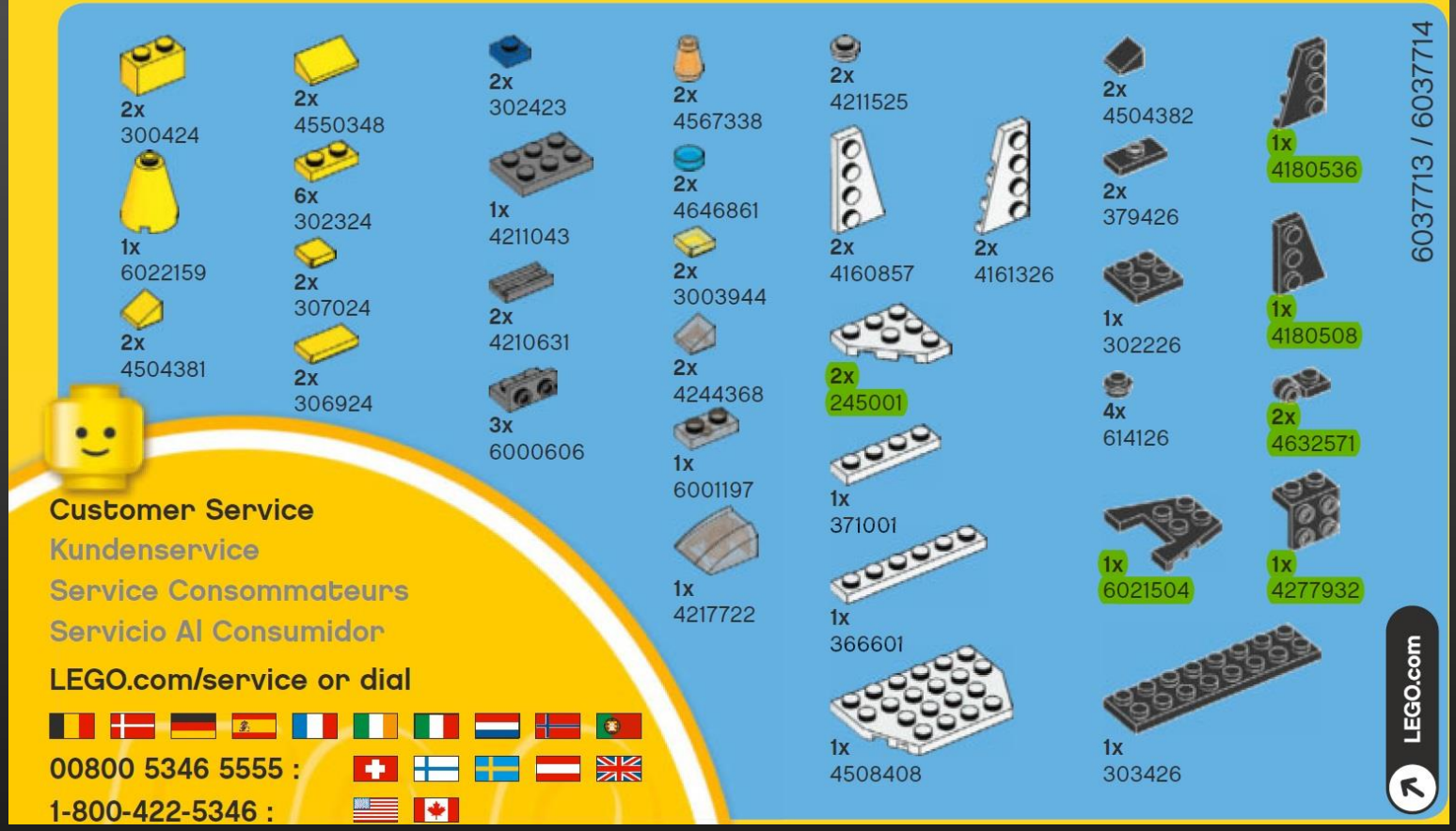
O componente é uma peça substituível, reusável de um sistema maior, cujos detalhes de implementação são abstraídos

A funcionalidade de um componente é descrita por um conjunto de interfaces fornecidos

Para além de implementar, o componente pode requerer funcionalidades de outros







A *component* is a self-contained, encapsulated piece of software that can be plugged into a system to provide a specific set of required functionalities. Today, there are many components available for purchase. A component has a well-defined *API* (application program interface). An API is essentially a set of method interfaces to the objects contained in the component. The internal workings of the component are hidden behind the API. Com-

Dennis, Alan, Barbara Wixom, David Tegarden. *Systems Analysis and Design: An Object Oriented Approach with UML, 5th Edition*. Wiley.

Arquitetura de componentes do software

Ao contrário do *package*, o componente é **uma peça tangível** da solução (e.g.: ficheiro, arquivo)

Os componentes são implementados com tecnologia concreta

Propriedades desejáveis:

Encapsulamento (da estrutura interna)

Reutilizável (em vários projetos)

Substituível

Candidatos naturais:

Aspetos recorrentes em vários projetos

Módulos que se podem obter pré-feitos ou disponibilizar

Módulos definidos para ir de encontro às regras dos ambientes de execução (e.g.: módulos para a*pplication servers*)

https://mvnrepository.com

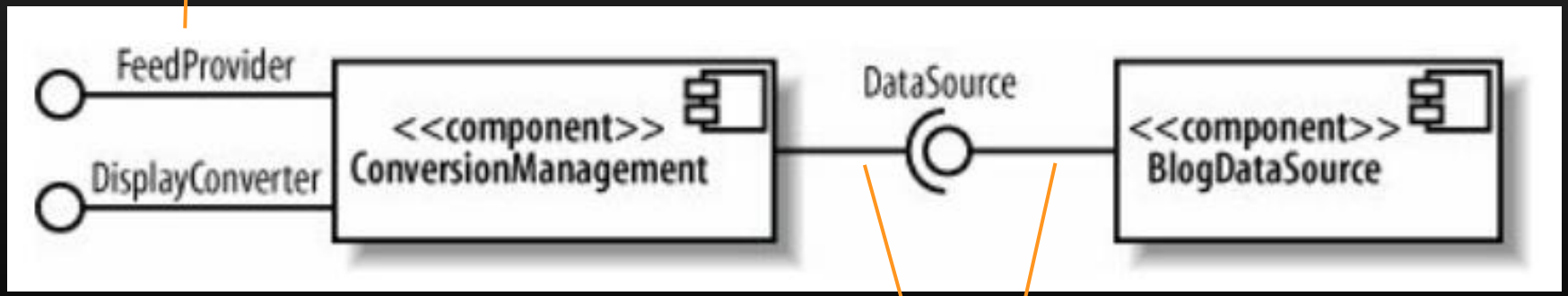
Uma “loja” de componentes, encapsulados, reutilizáveis e substituíveis.

Figure

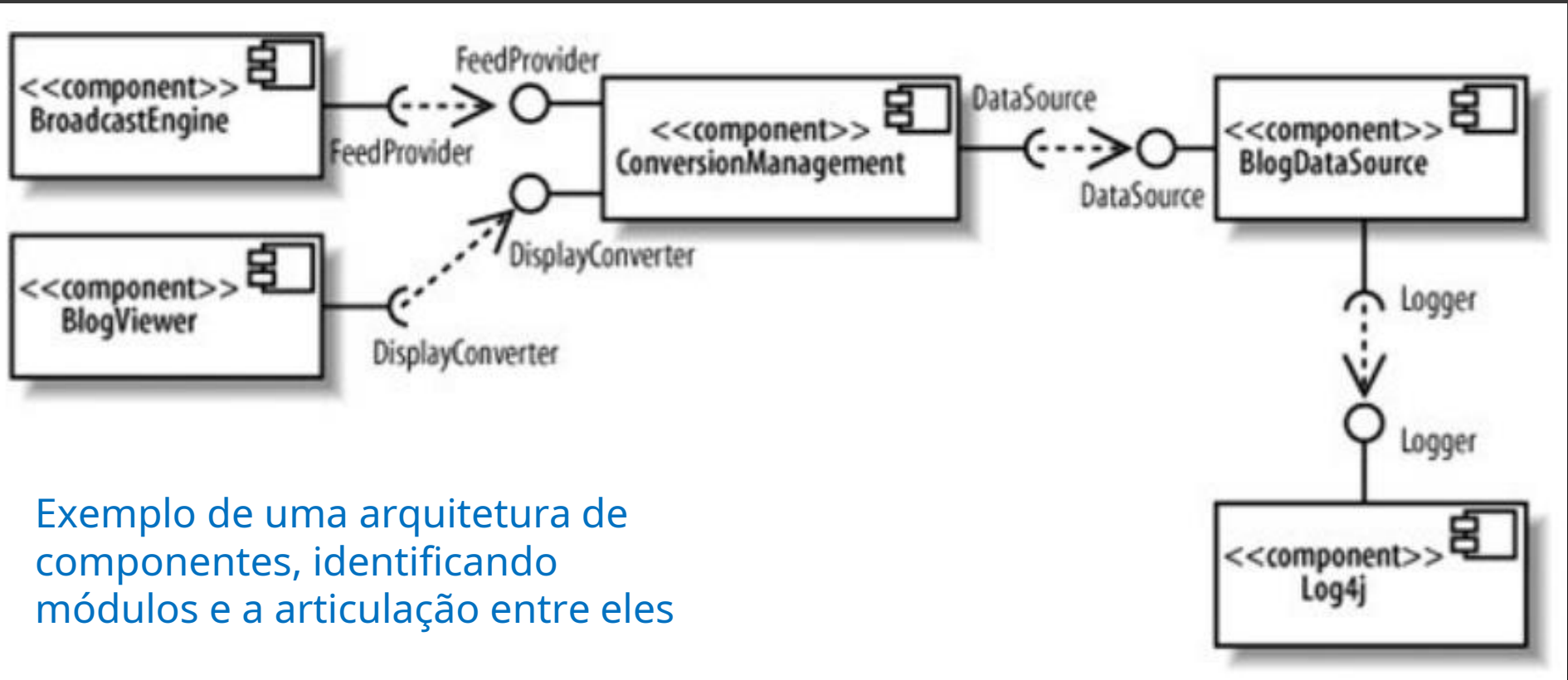
Solução modular com componentes

Com os componentes, pretende-se arquiteturas com baixo “coupling”

A exposição da funcionalidade através de interfaces ajuda a separar o contrato da implementação

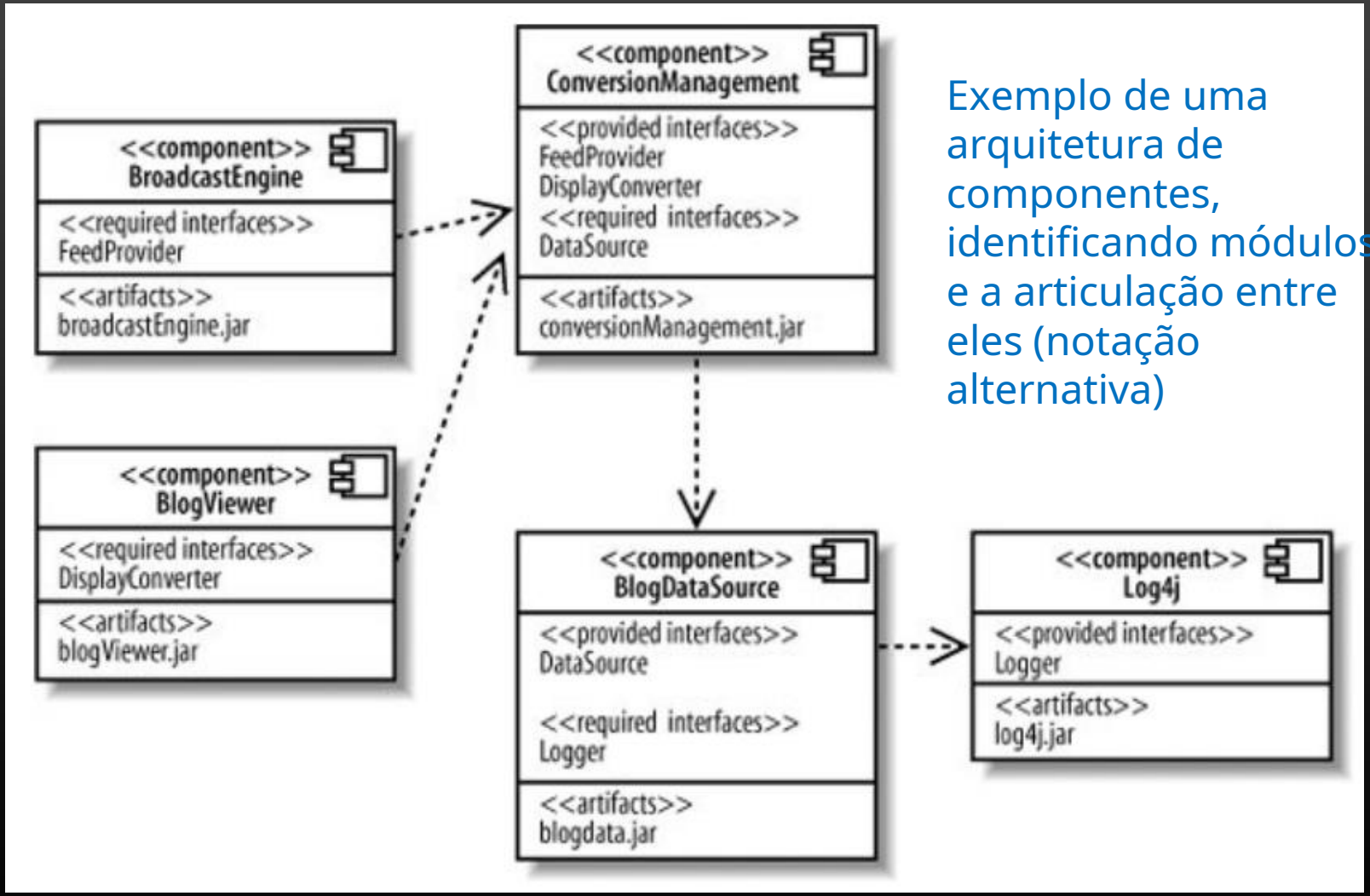


O mesmo modelo, notação ligeiramente diferente 1



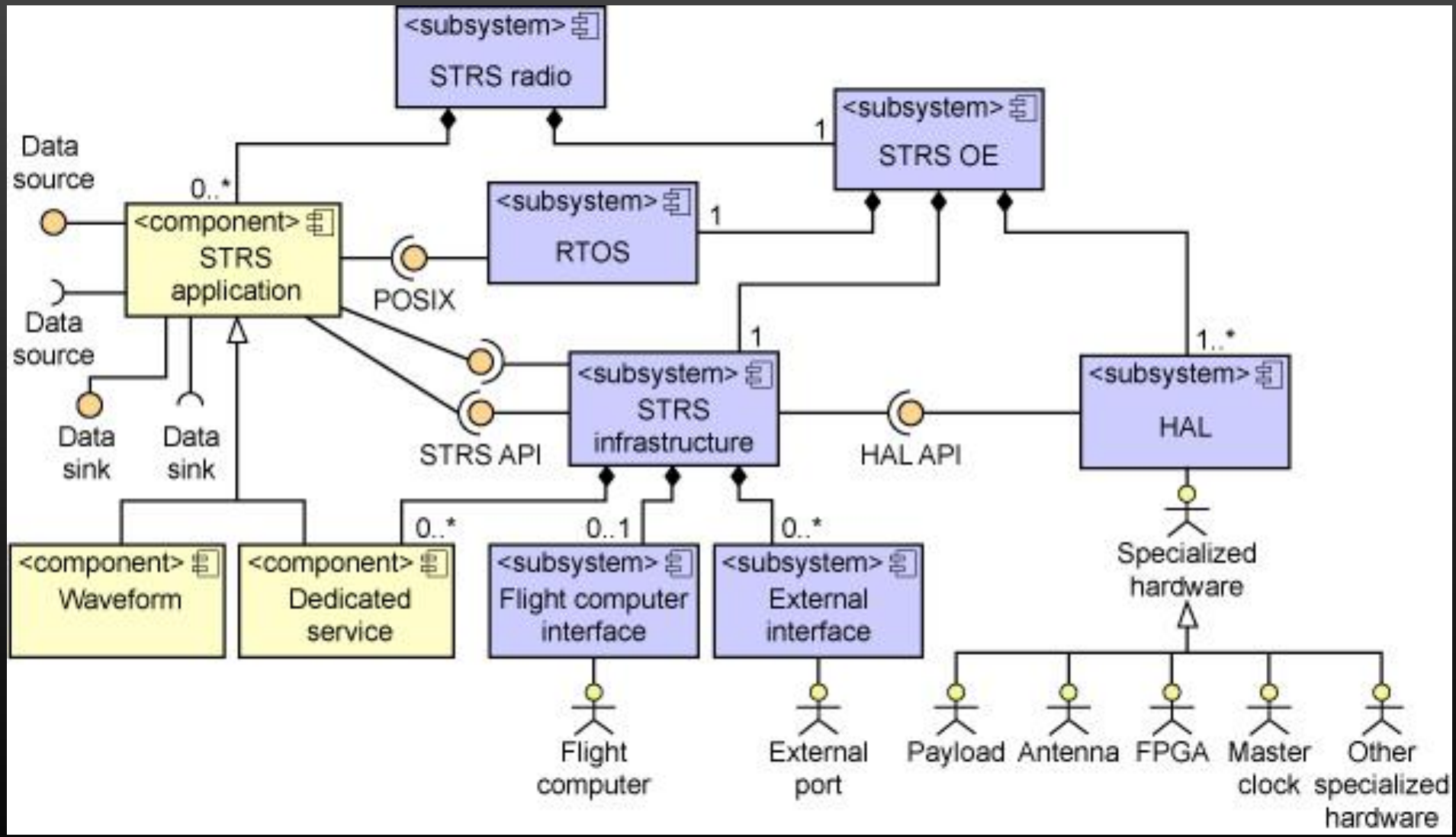
Exemplo de uma arquitetura de componentes, identificando módulos e a articulação entre eles

O mesmo modelo, notação ligeiramente diferente 2



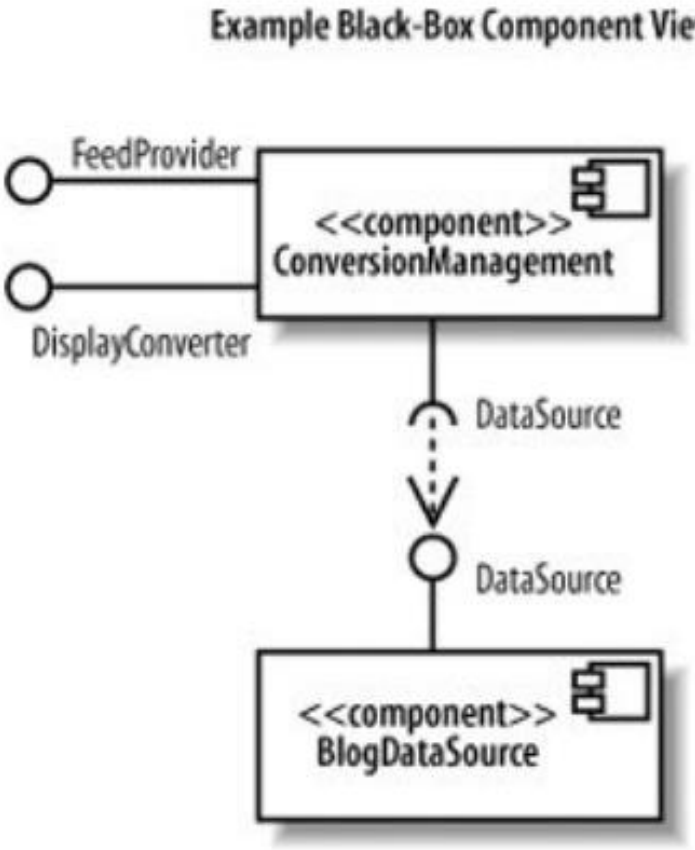
Exemplo de uma arquitetura de componentes, identificando módulos e a articulação entre eles (notação alternativa)

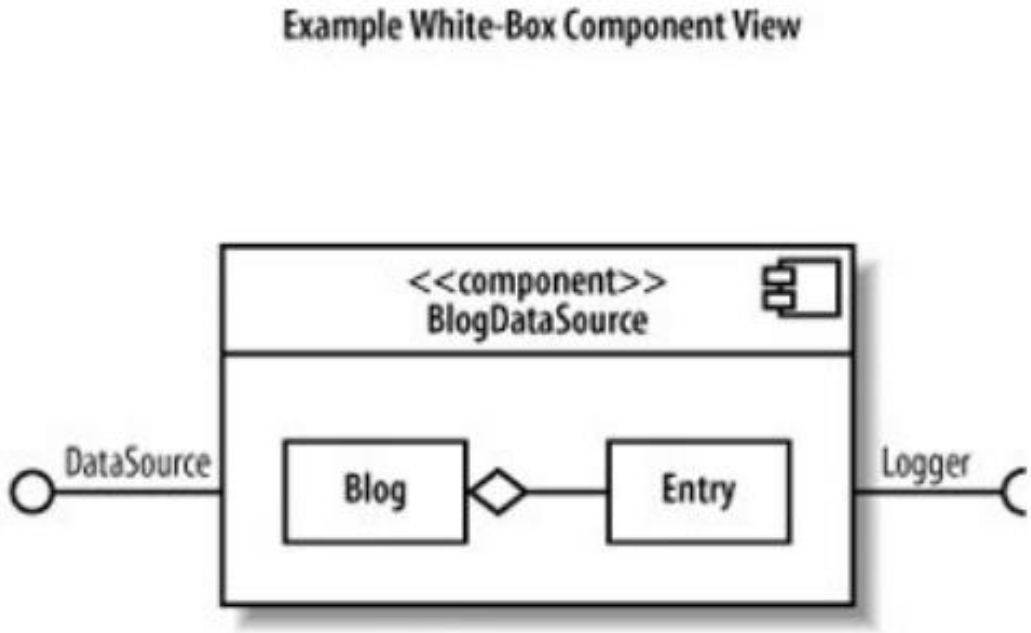
An example from NASA: Radio System



From http://www.grc.nasa.gov/WWW/RT/2006/RC/RCD-handler.html

Notação “caixa fechada” / “caixa aberta”





A arquitetura do sistema aborda diferentes perspetivas de análise

❶ Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

❷Arquitetura de componentes do software

Peças construídas com uma tecnologia concreta

Construção “modular”

- E.g.: existem pré-feitos?

❸ Arquitetura de instalação

Visão dos equipamentos e configuração de produção (conectividade, distribuição,...)

Diagramas de instalação da UML

Nós (*node*)

Um equipamento de hardware

Ambiente de execução

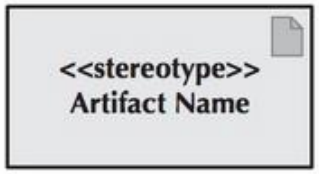
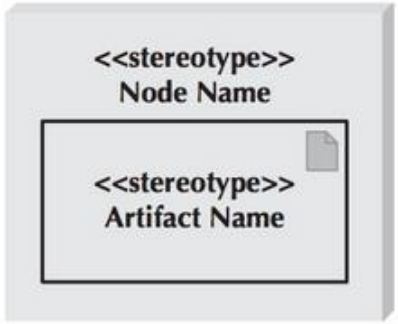
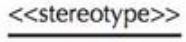
Um ambiente externo à solução que proporciona o contexto necessário à sua execução

E.g.: SO, servidor web, servidor aplicacional

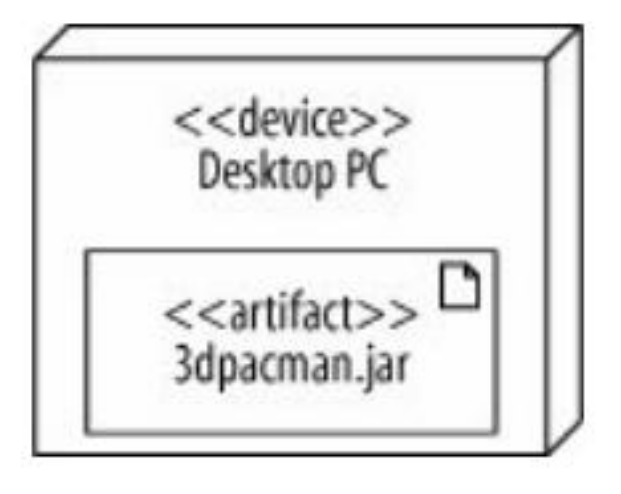
Artefactos (*artifact)*

Ficheiros concretos que são executados ou utilizados pela solução

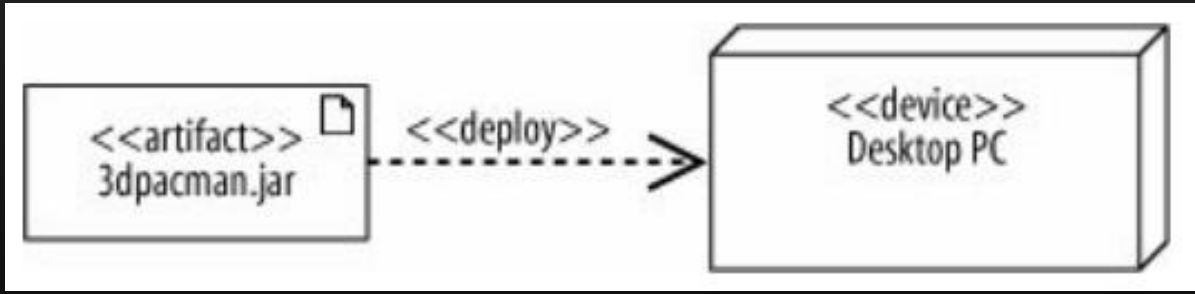
E.g.: executáveis, bibliotecas, configurações, scripts

A node: ■ Is a computational resource, e.g., a client computer, server, separate network, or individual network device. ■ Is labeled by its name. ■ May contain a stereotype to specifically label the type of node being represented, e.g., device, client workstation, application server, mobile device, etc.	
An artifact: ■ Is a specification of a piece of software or database, e.g., a database or a table or view of a database, a software component or layer. ■ Is labeled by its name. ■ May contain a stereotype to specifically label the type of artifact, e.g., source file, database table, executable file, etc.	
A node with a deployed artifact: ■ Portrays an artifact being placed on a physical node.	
A communication path: ■ Represents an association between two nodes. ■ Allows nodes to exchange messages. ■ May contain a stereotype to specifically label the type of communication path being represented, (e.g., LAN, Internet, serial, parallel).	

Os artefactos são executados em nós

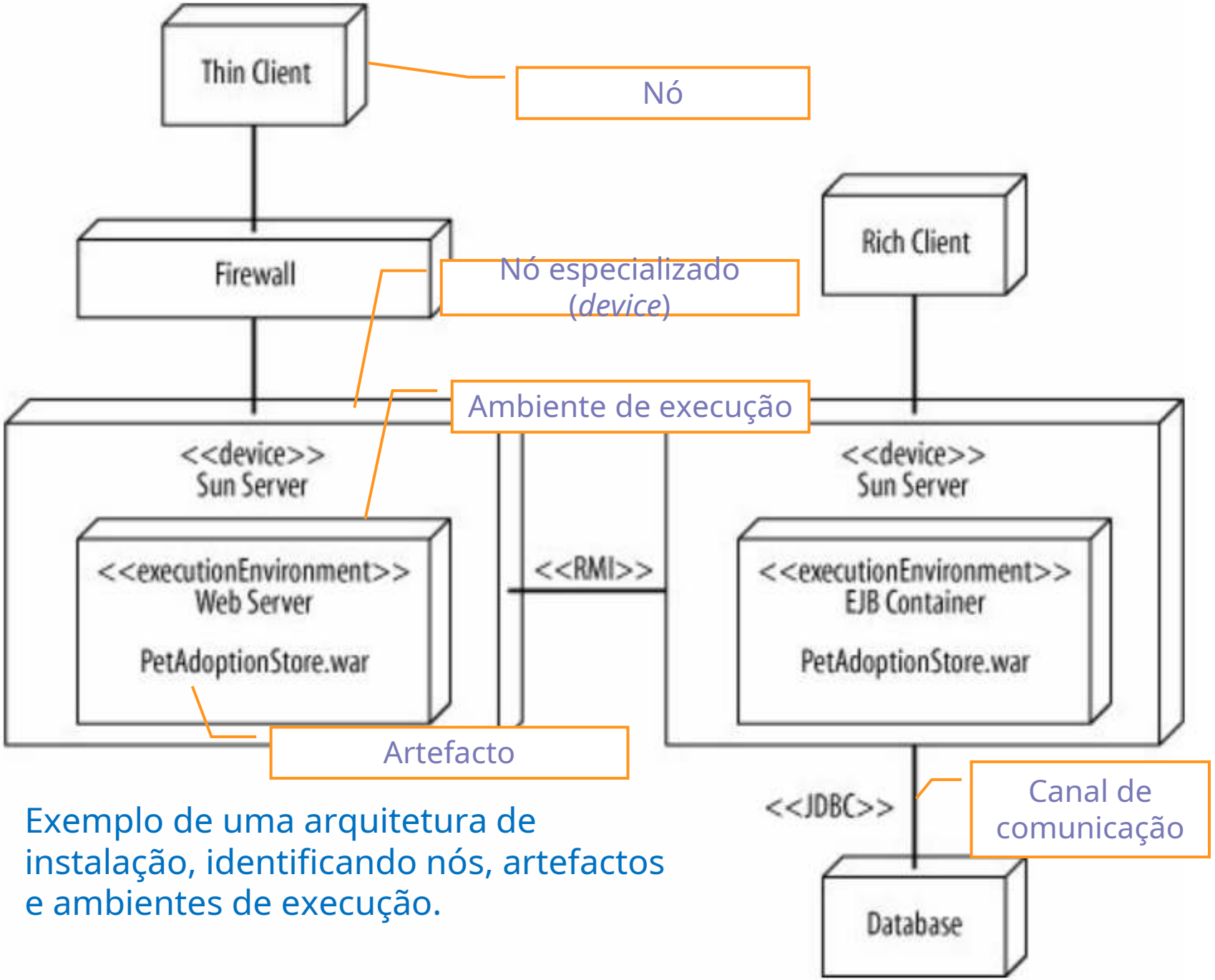


Notações alternativas.



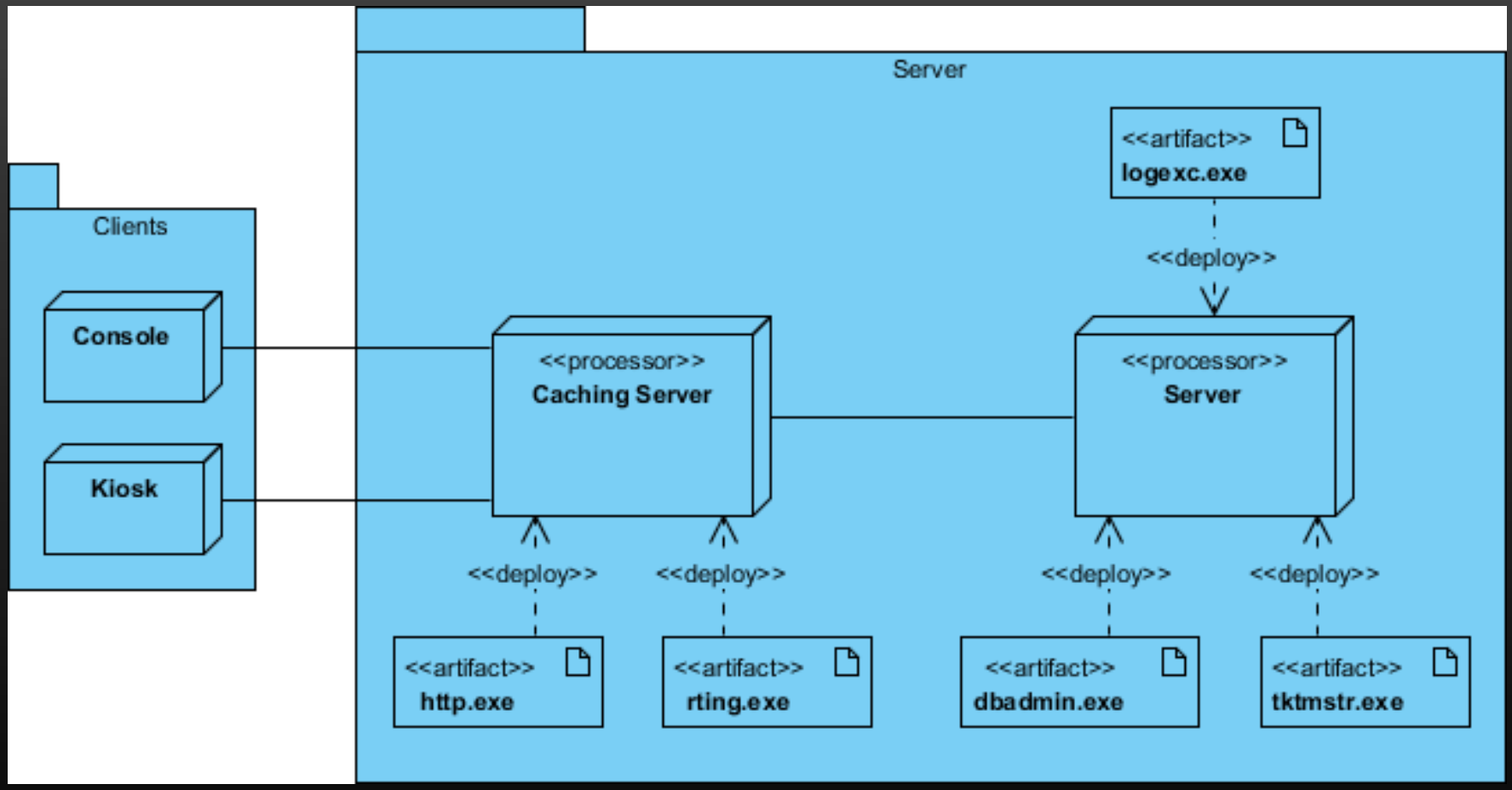
Rastreabilidade até aos componentes

A relação “manifest” permite associar componentes a artefactos.

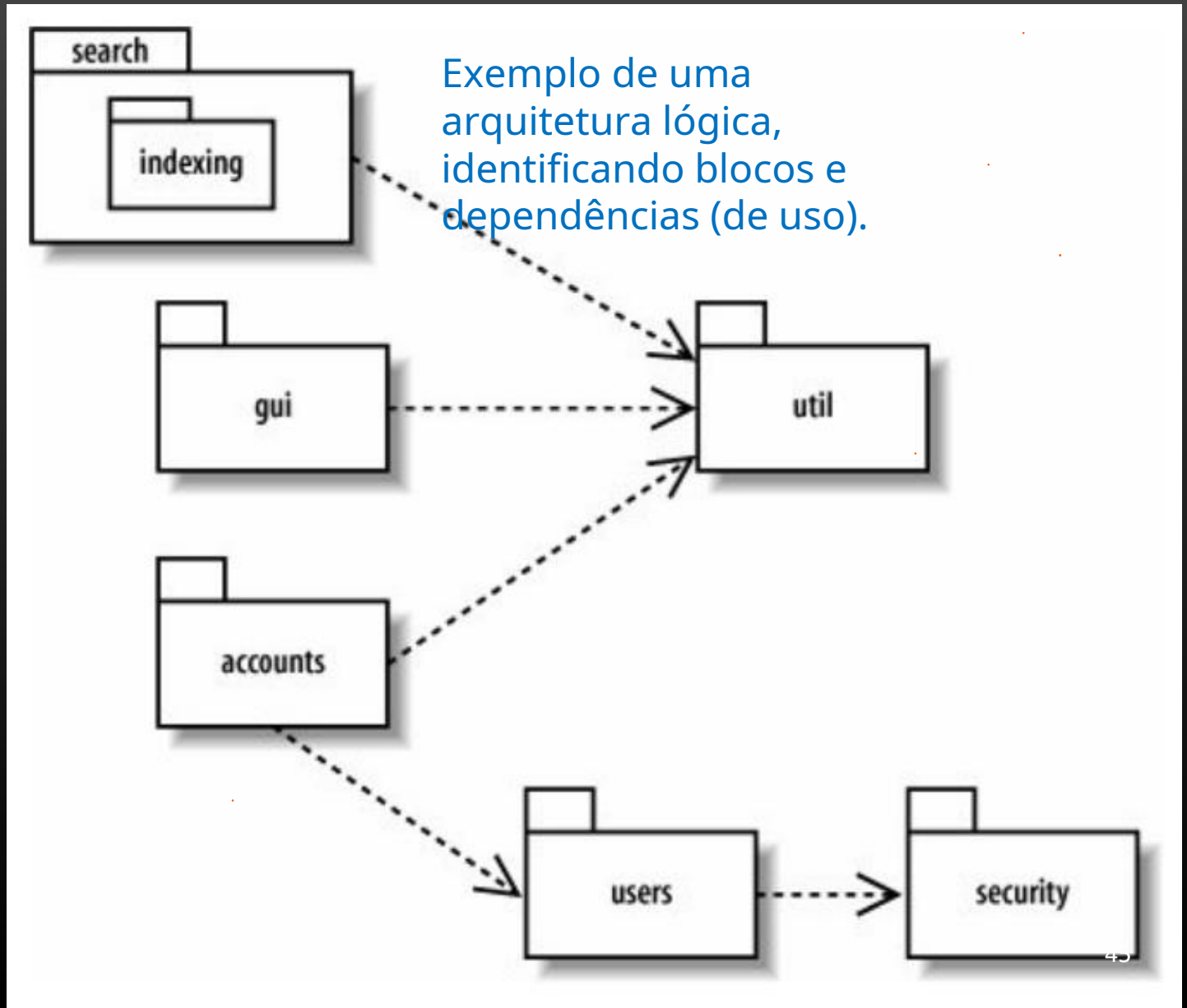


Exemplo de uma arquitetura de instalação, identificando nós, artefactos e ambientes de execução.

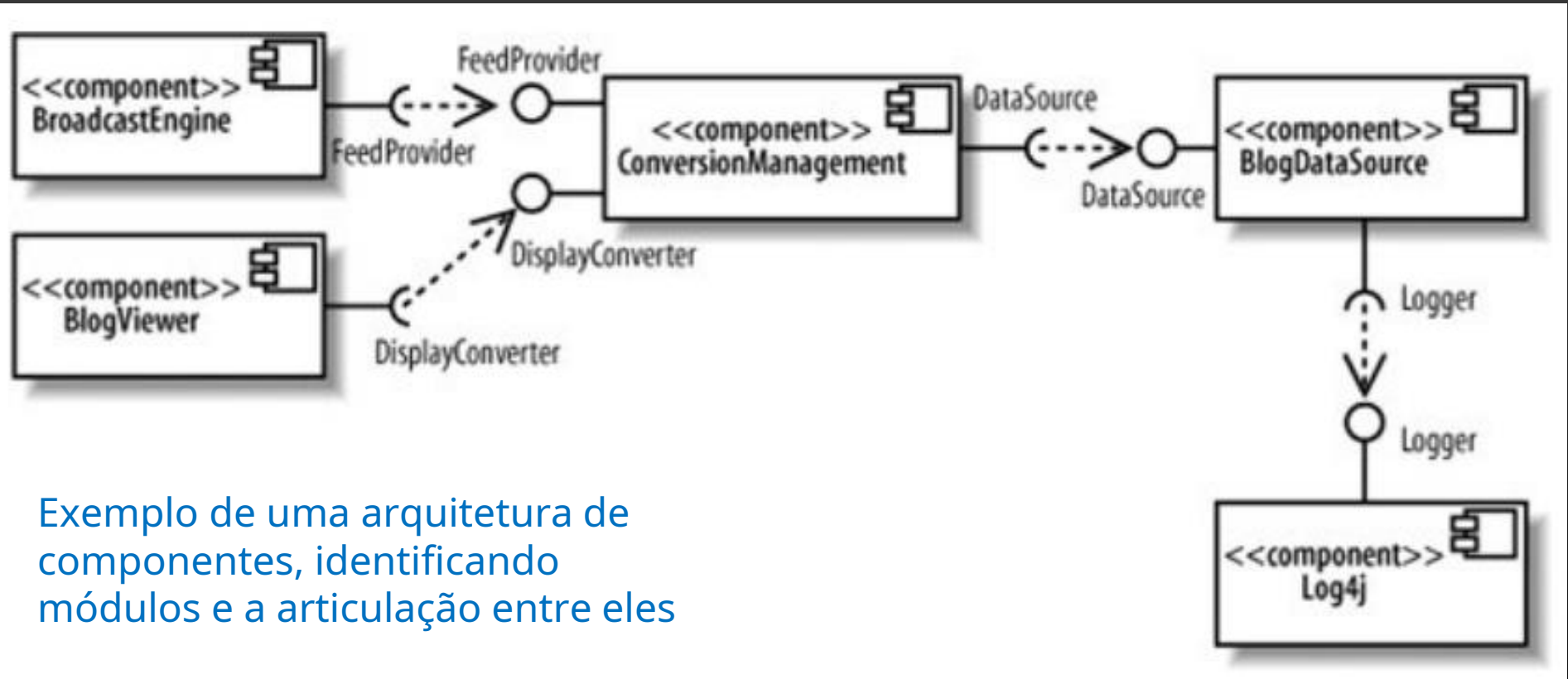
Exemplos (diagrama de instalação)



Vista de módulos → na UML: d. de pacotes

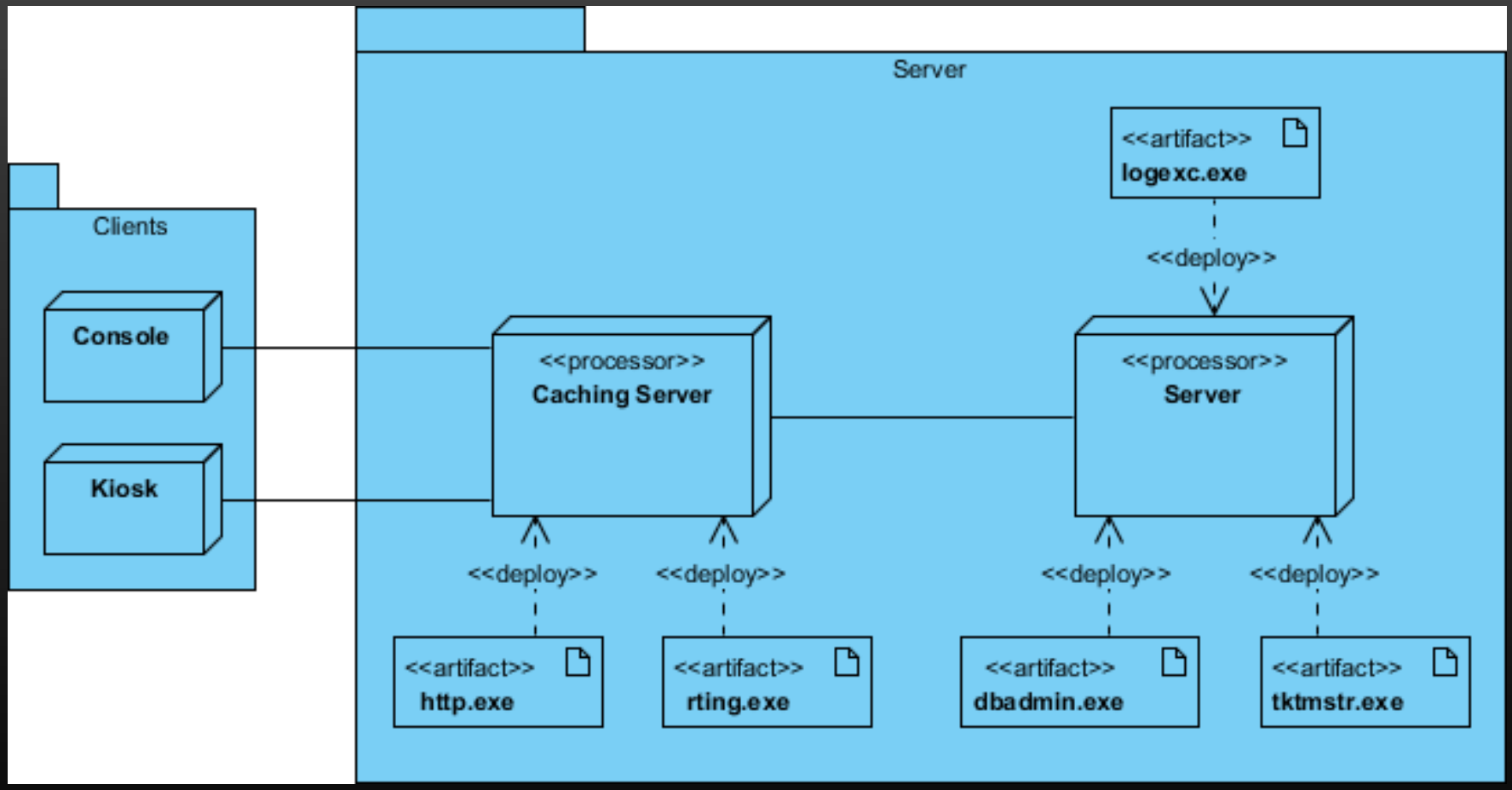


Vista C&C → na UML: d. componentes

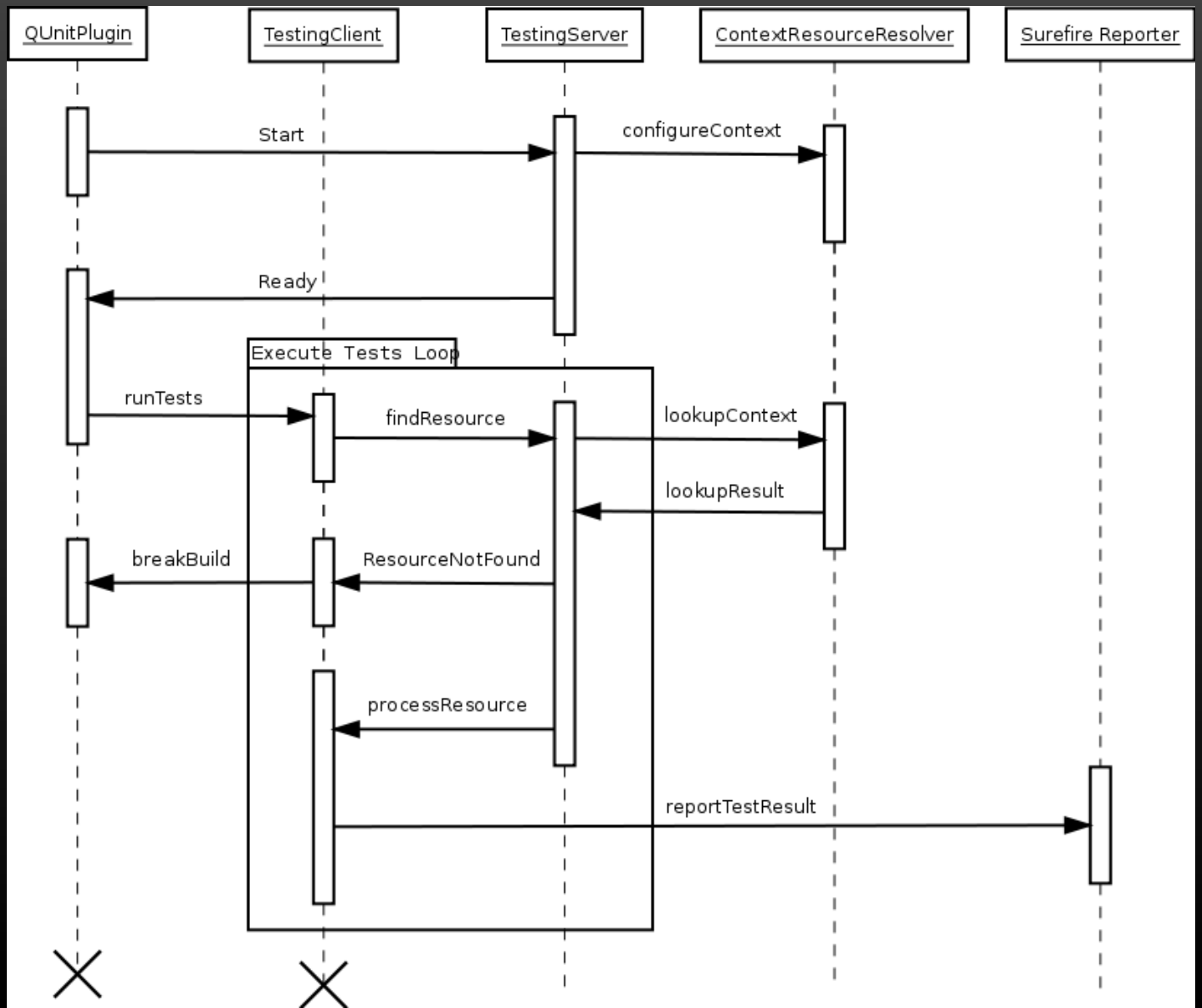


Exemplo de uma arquitetura de componentes, identificando módulos e a articulação entre eles

Vista de alocação → na UML: d. instalação/*deployment*



Vista dinâmica (interação). Notar o nível de abstração (subsistema, não objetos)



Estilos de arquitetura (padrões)

Estilos de arquitetura

Apesar da enorme diversidade dos produtos de software, é possível falar-se em certos "padrões de arquitetura" nas soluções empresariais

Um "padrão de arquitetura" reflete uma abordagem-tipo, i.e., uma maneira de organizar a solução que se provou adequada para certos tipos de projetos.

Exemplos:

- Microkernel Architecture
- Layered Architecture
- Event-Driven Architecture
- Service-oriented Architecture
- Microservices Architecture
- ...

Arquitetura por camadas

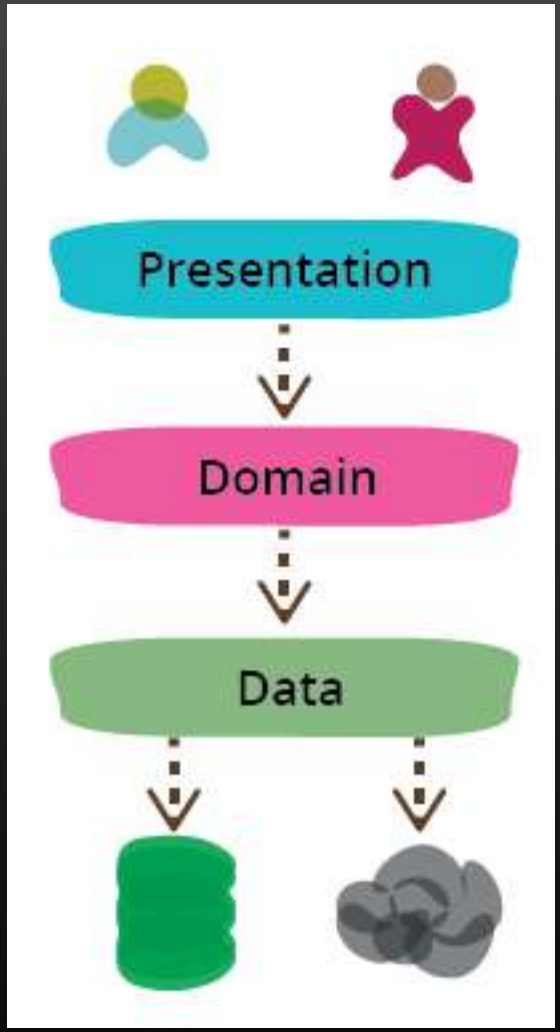
Divisão modular da solução de software em camadas/níveis de abstração.

As camadas são sobrepostas

Cada camada tem uma especialização

Camadas “em cima” pedem serviços às camadas “de baixo”

Não se pode saltar camadas: os componentes, em cada camada, “falam” com as camadas adjacentes.



https://martinfowler.com/bliki/PresentationDomainDataLayering.html

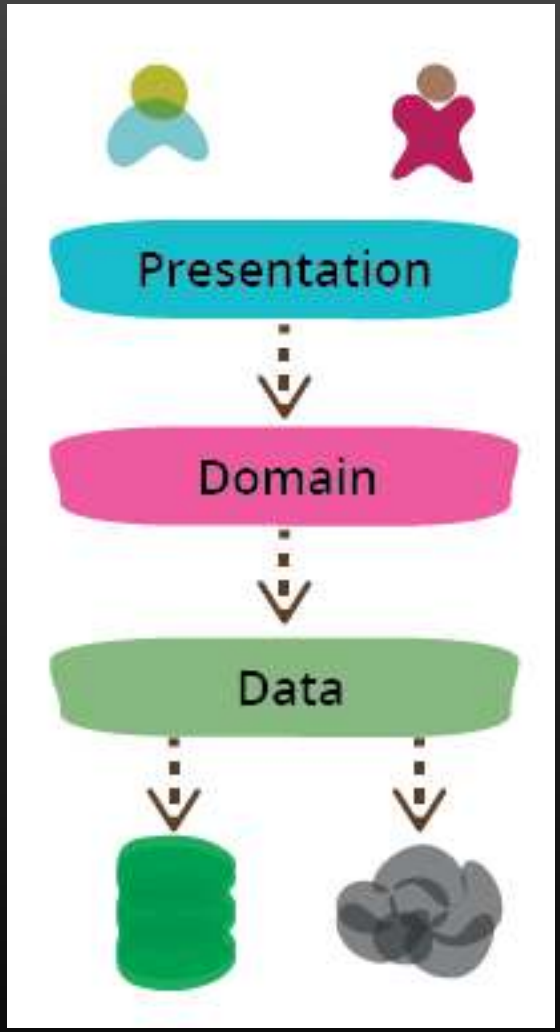
A arquitetura de 3 camadas

Uma das formas mais comuns de modularizar um programa orientado para a gestão de informação é separá-lo em três camadas principais:

1) apresentação (*user interface*, UI),
2) lógica de domínio (a.k.a. *business logic*)
3) acesso a dados.

Desta forma, é normal organizar uma aplicação web em três camadas:

- Uma camada web que sabe lidar com pedidos HTTP e preparar as páginas HTML,
- uma camada com a lógica de negócio, que contém regras (algoritmos), validações e cálculos,
- e uma camada de acesso de dados que assegura como gerir os dados de forma persistente, numa base de dados ou com integração de serviços remotos.



https://martinfowler.com/bliki/PresentationDomainDataLayering.html

Camadas e partições (modularização)

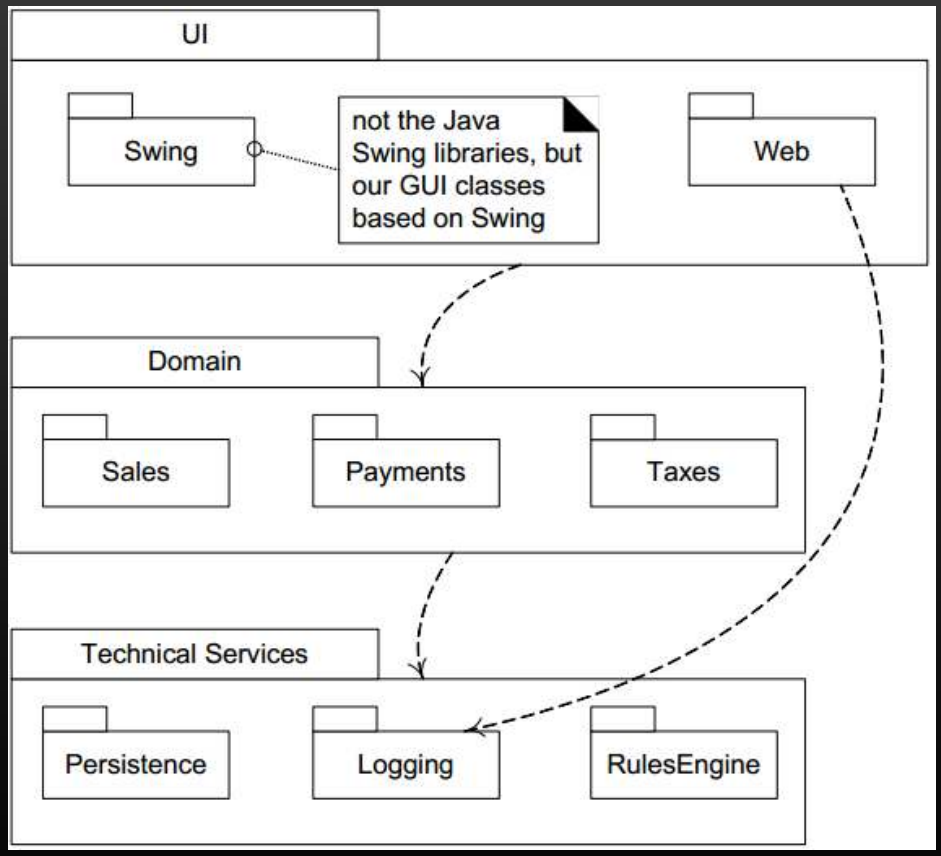
Camadas verticais:

- divisão por níveis de abstração

Partições horizontais:

- Módulos dentro de uma camada

E.g.: a lógica do domínio está dividida em grandes módulos funcionais especializados, agrupando as programação relativa às Vendas, aos Pagamentos e à Fiscalidade.



GUI windows
reports
speech interface
HTML, XML, XSLT, JSP, Javascript, ...

UI
(AKA Presentation, View)

handles presentation layer requests
workflow
session state
window/page transitions
consolidation/transformation of disparate data for presentation

Application
(AKA Workflow, Process, Mediation, App Controller)

handles application layer requests
implementation of domain rules
domain services (POS, Inventory)
- services may be used by just one application, but there is also the possibility of multi-application services

Domain
(AKA Business, Application Logic, Model)

very general low-level business services used in many business domains
CurrencyConverter

Business Infrastructure
(AKA Low-level Business Services)

(relatively) high-level technical services and frameworks
Persistence, Security

Technical Services
(AKA Technical Infrastructure, High-level Technical Services)

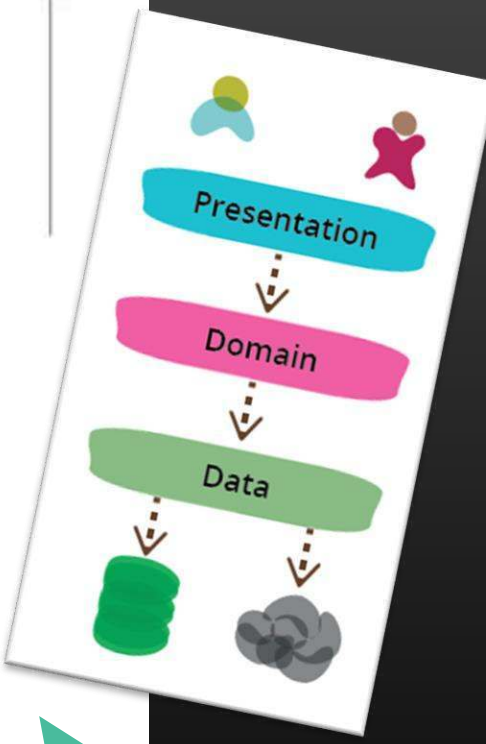
low-level technical services, utilities, and frameworks
data structures, threads, math, file, DB, and network I/O

Foundation
(AKA Core Services, Base Services, Low-level Technical Services/Infrastructure)

width implies range of applicability

more app specific

dependency



Algumas arquiteturas baseiam-se no modelo três camadas que ampliam para mostrar uma maior especialização de responsabilidades

[Larman'04]

Referências

Core readings	Suggested readings
• [Larman04] – Chap. 13 • [Dennis15] – Chap. 7 & 11	• Bass, Clements, Kazman, "Software Architecture in Practice", 4th ed. • Fowler, "Software Architecture Guide" Visual Paradigm tutorials on UML Diagrams: • https://online.visual-paradigm.com/diagrams/tutorials/