

40431: Modelação e Análise de Sistemas

Arquitetura Evolutiva

Ilídio Oliveira

v2022-12-02

Objetivos de aprendizagem

Explicar as atividades do desenvolvimento associadas à arquitetura de software

Definir a prática de Arquitetura Evolutiva, conforme especificado no OpenUP

Identificar os elementos abstratos de uma arquitetura de software

Identificar requisitos de arquitetura significativos num domínio e relacionar com atributos de qualidade

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

Descrever a organização de uma arquitetura por camadas, em particular o estilo Apresentação + Lógica do domínio + Dados

OpenUP e a arquitetura do sistema

Prática do OpenUP : arquitetura evolutiva

Practices > Technical Practices > Evolutionary Architecture

Practice: Evolutionary Architecture



Analyze the major technical concerns that affect the solution, and capture those architectural decisions to ensure that those decisions are assessed and communicated.

Expand All Sections Collapse All Sections

Relationships

Content References

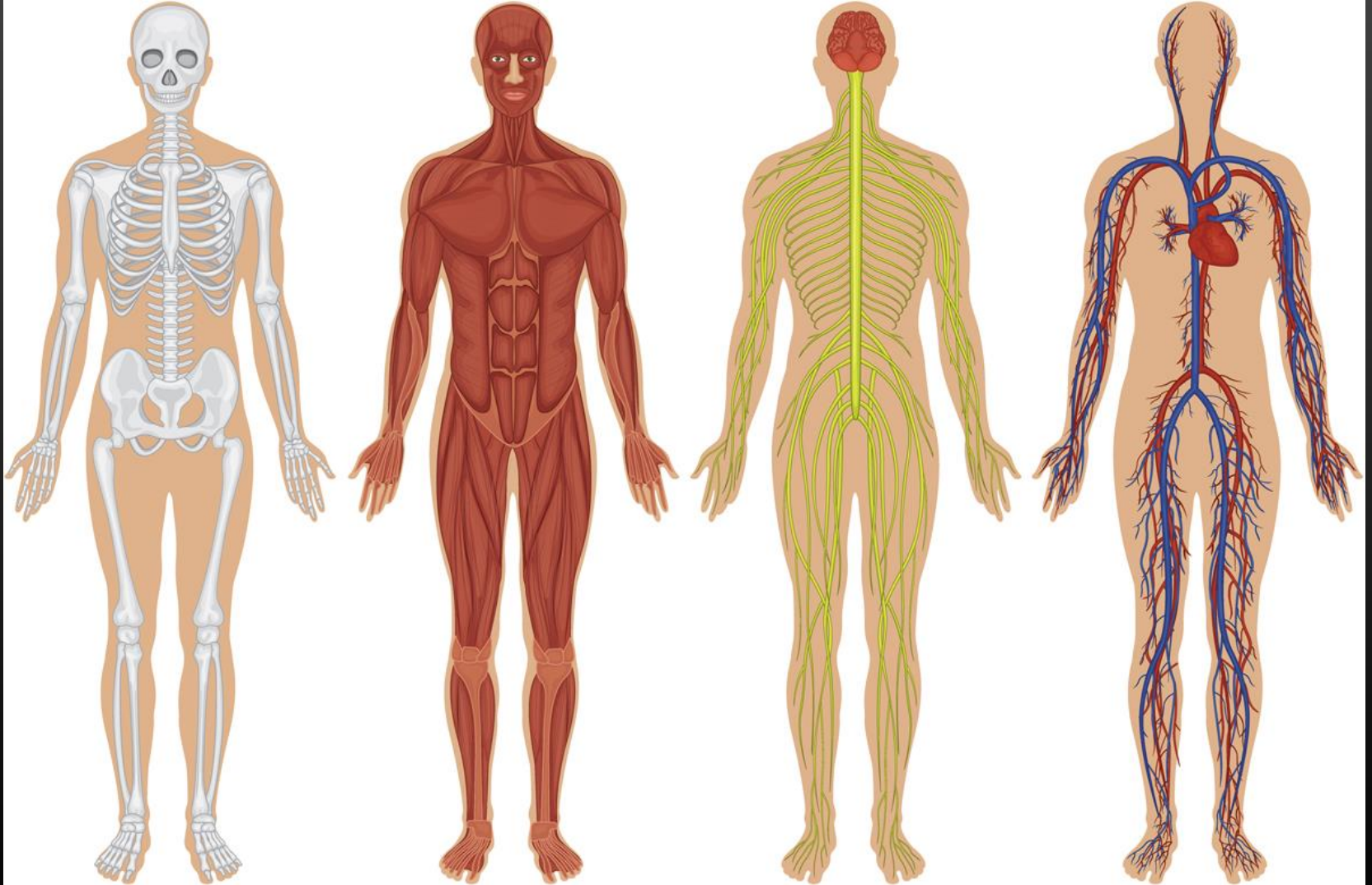
- How to adopt the Evolutionary Architecture practice
- Key Concepts
 - Architectural Mechanism
 - Architectural Views and Viewpoints
 - **Software Architecture**
- Architecture Notebook
- Envision the Architecture
- Refine the Architecture
- Guidance
 - Guidelines
 - Abstract Away Complexity
 - Modeling the Architecture
 - Software Reuse

Analisar as principais preocupações técnicas; definir estratégias; avaliar, documentar e comunicar decisões.

Inputs

- [Technical Design]

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



O que é a arquitetura de software?

A Arquitetura é um Conjunto de Estruturas (de Software)

Partes/elementos relevantes ligados por alguma relação.

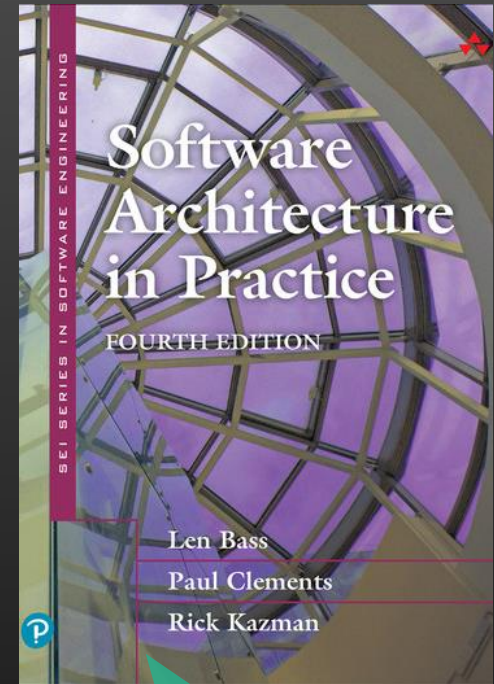
Arquitetura é uma Abstracção

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

Arquitetura Inclui Comportamento

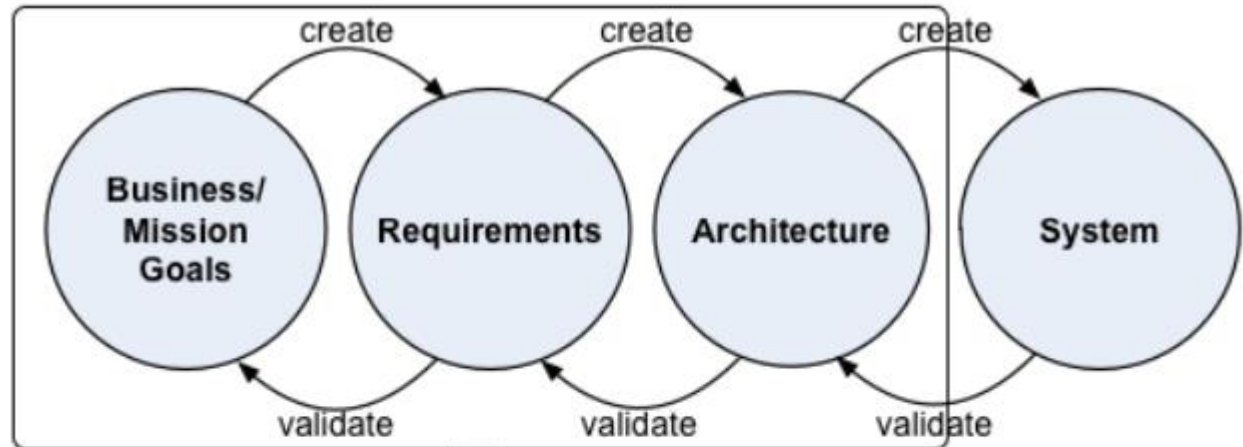
O comportamento dos elementos engloba a forma como interagem uns com os outros e com o ambiente.



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

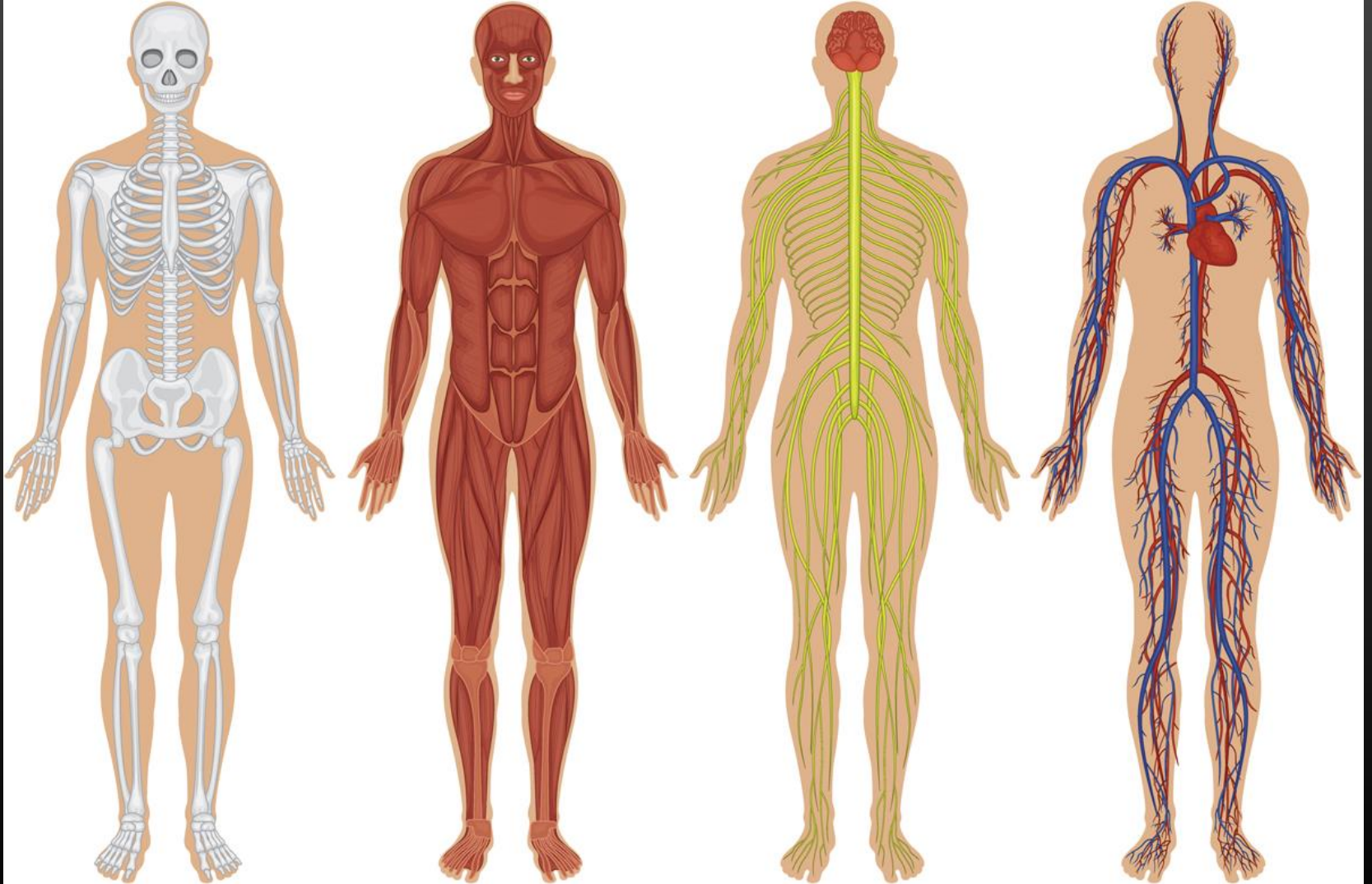
- Design - create and evolve
- Analysis - will the design provide the needed functions and qualities?
- Models and representations - "documentation"
- Evaluation - are we satisfying stakeholders?

- Communication – with technical and business teams
- Technical Leadership

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

Requisitos com impacto nas decisões de arquitetura

O que terá influenciado o “desenho”?



As decisões de arquitetura são conduzidas pelos requisitos

Para o arquitecto, nem todos os requisitos são idênticos.

Alguns têm um impacto muito mais profundo na arquitectura do que outros.

Um requisito de arquitectura significativo é um requisito que terá um efeito profundo na arquitectura - ou seja, a arquitectura pode ser substancialmente diferente conforme a presença/ausência de tal requisito.

Exemplo de um sistema complexo: Feedzai



PLATFORM

SOLUTIONS

INDUSTRIES

RESOURCES

COMPANY

CONTACT

LIFE OF TRANSACTION IN 3 MILLISECONDS

Feedzai uses advanced machine learning to minimize friction so you can maximize revenue

SEE FEEDZAI IN ACTION

- Número muito elevado de transações financeiras que devem ser analisadas em paralelo (intensidade variável)
- Processamento de eventos em larga escala em janelas temporais (solução otimizada para o processamento de *feeds*, i.e., séries de dados, e não interrogação de bases de dados convencionais)
- Respostas de muito baixa latência (indicação de fraude em <0,5s)
- Natureza sensível da informação: canais seguros e invioláveis.
- Há clientes que preferem a solução na cloud e outros que querem usar apenas instalações no seu datacenter

<https://feedzai.com>

Os requisitos conduzem a arquitetura

E.g.: desempenho como atributo de qualidade

Figure 9.1 gives an example concrete performance scenario: *Five hundred users initiate 2,000 requests in a 30-second interval, under normal operations. The system processes all of the requests with an average latency of two seconds.*

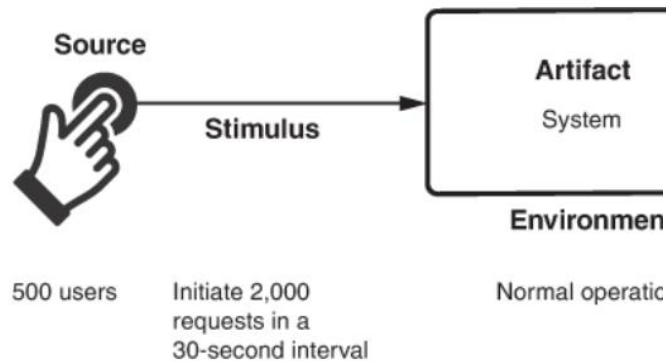


Figure 9.1 Sample performance scenario

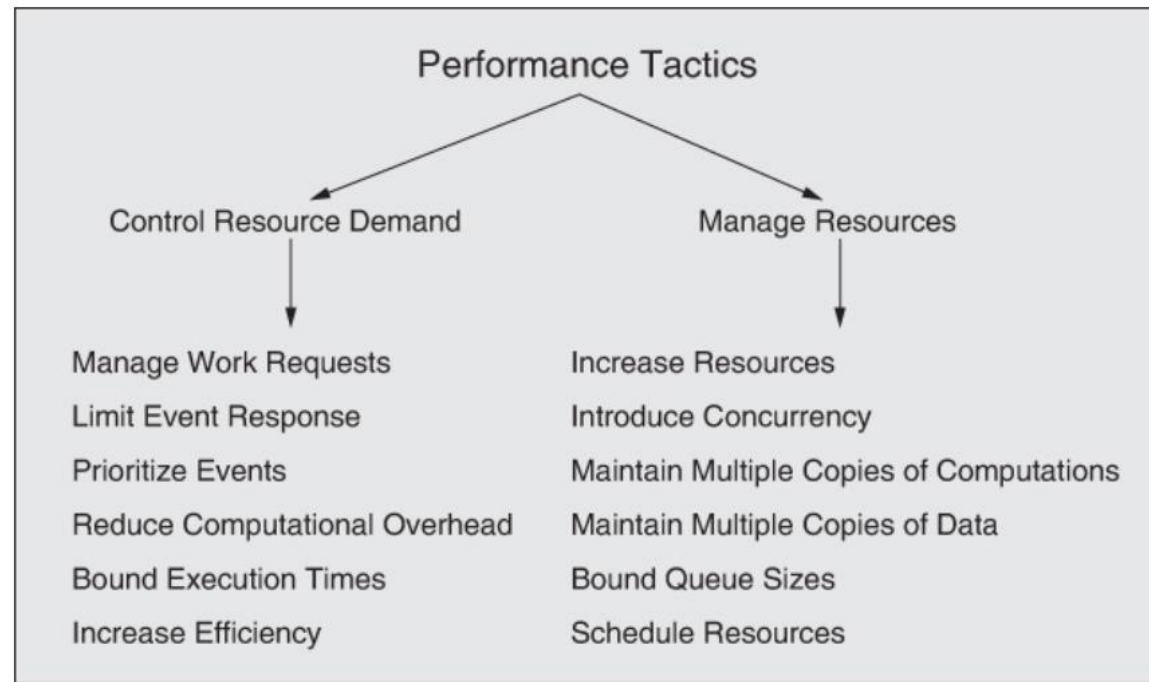


Figure 9.3 Performance tactics

Os requisitos conduzem a arquitetura

E.g.: disponibilidade como atributo de qualidade

Table 4.2 is shown in Figure 4.1. The scenario is this: *A server in a server farm fails during normal operation, and the system informs the operator and continues to operate with no downtime.*

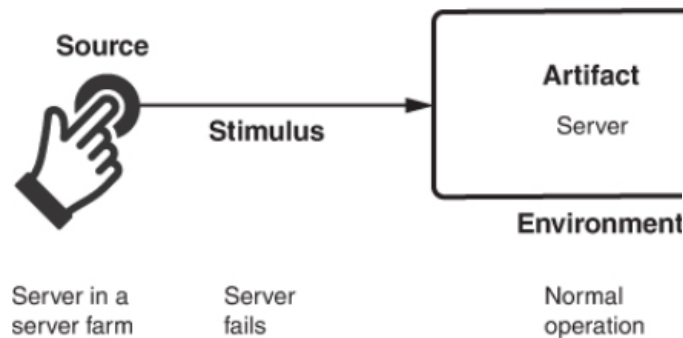
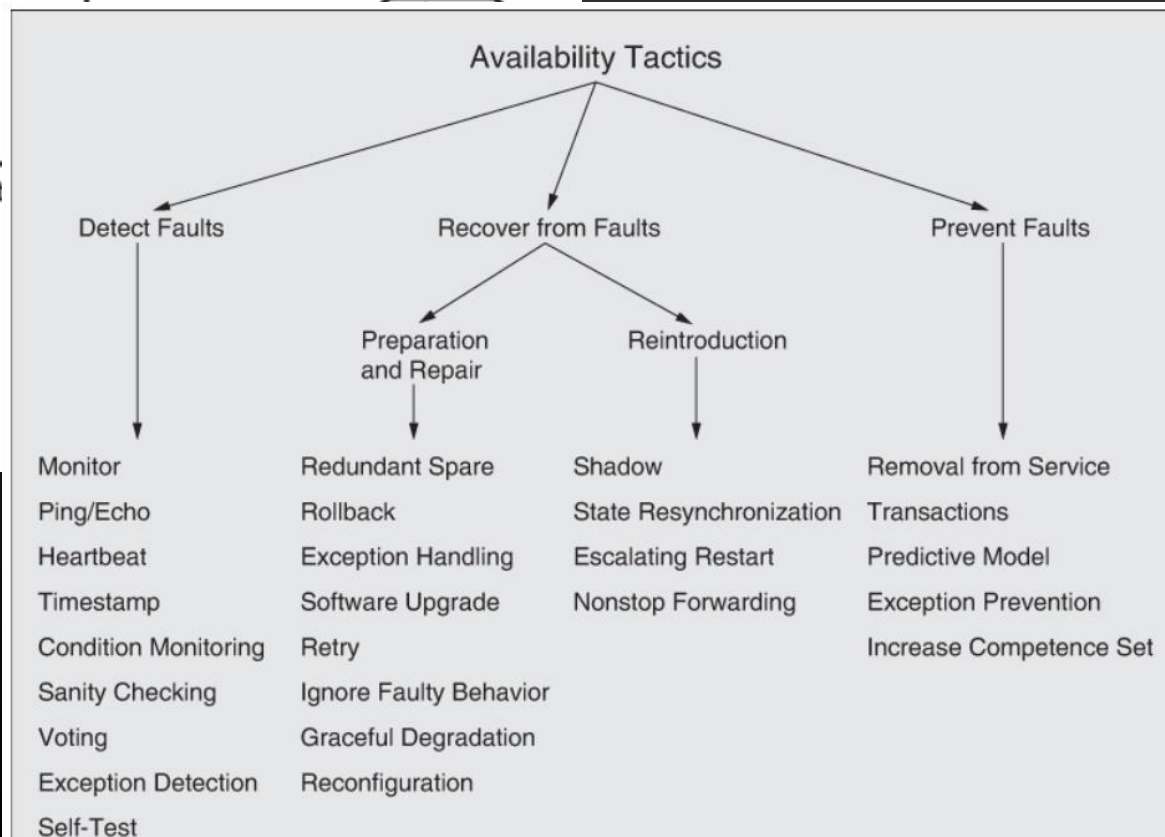


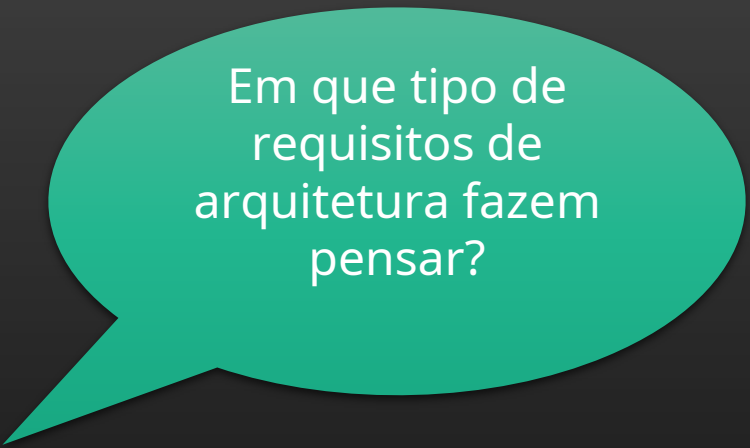
Figure 4.1 Sample concrete availability scenario



Precisamos de pensar na arquitetura de sistemas complexos

Alguns exemplos de sistemas complexos:

- a) Wikipedia
- b) Multi-player online RPG
- c) Amazon web store
- d) Twitter
- e) Netflix



Em que tipo de requisitos de arquitetura fazem pensar?

Exemplos

Wikipedia

Enorme quantidade de documentos de texto

Gerir milhões de documentos: armazenamento, recuperação

Pesquisar de conteúdo em documentos

Como criar e editar documentos de forma distribuída? Direitos de acesso dos utilizador?

Multi-player online RPG

Grande quantidade de jogadores interagindo entre si (por exemplo, milhares de utilizadores)

Como distribuir a carga? Como otimizar a latência?

Há utilizadores banidos?

Integrar faturação (em compras de jogos), etc?

Como prevenir hackers/ batoteiros?

Exemplos de sistemas complexos e requisitos significativos

Amazon web store

Lidar com picos de utilização (por exemplo: *black friday*)

Sistema de recomendação de produtos (AI)

Quais os utilizadores que têm interesses semelhantes aos de outros utilizadores?

O que deve ser rastreado? Cliques? Compras? Comentários?

Há problemas de privacidade?

Twitter

Grande número de utilizadores, enorme quantidade de eventos, interações complexas

Integrações complexas: redes, redes sociais, etc.

Sistemas de entrega fiáveis. Comprovativo de entrega?

Basear-se em protocolos Web

Exemplos

Netflix

Rede de distribuição de conteúdos em larga escala (CDN): equilíbrio de carga, réplicas,...

Utilização interativa, conteúdos multimédia (latência muito baixa)

Proteção de direitos (DRM)

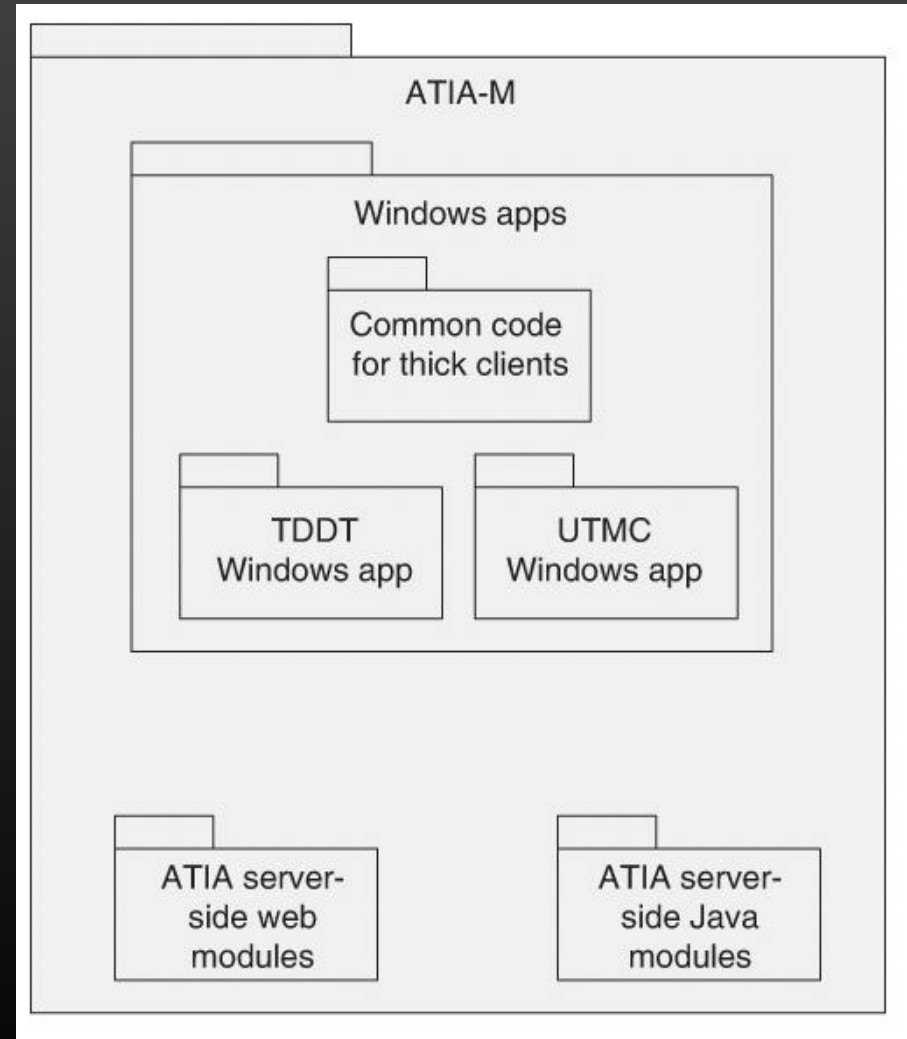
Elementos de uma arquitetura

Tipos de estruturas 1(3)

1/ As 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.

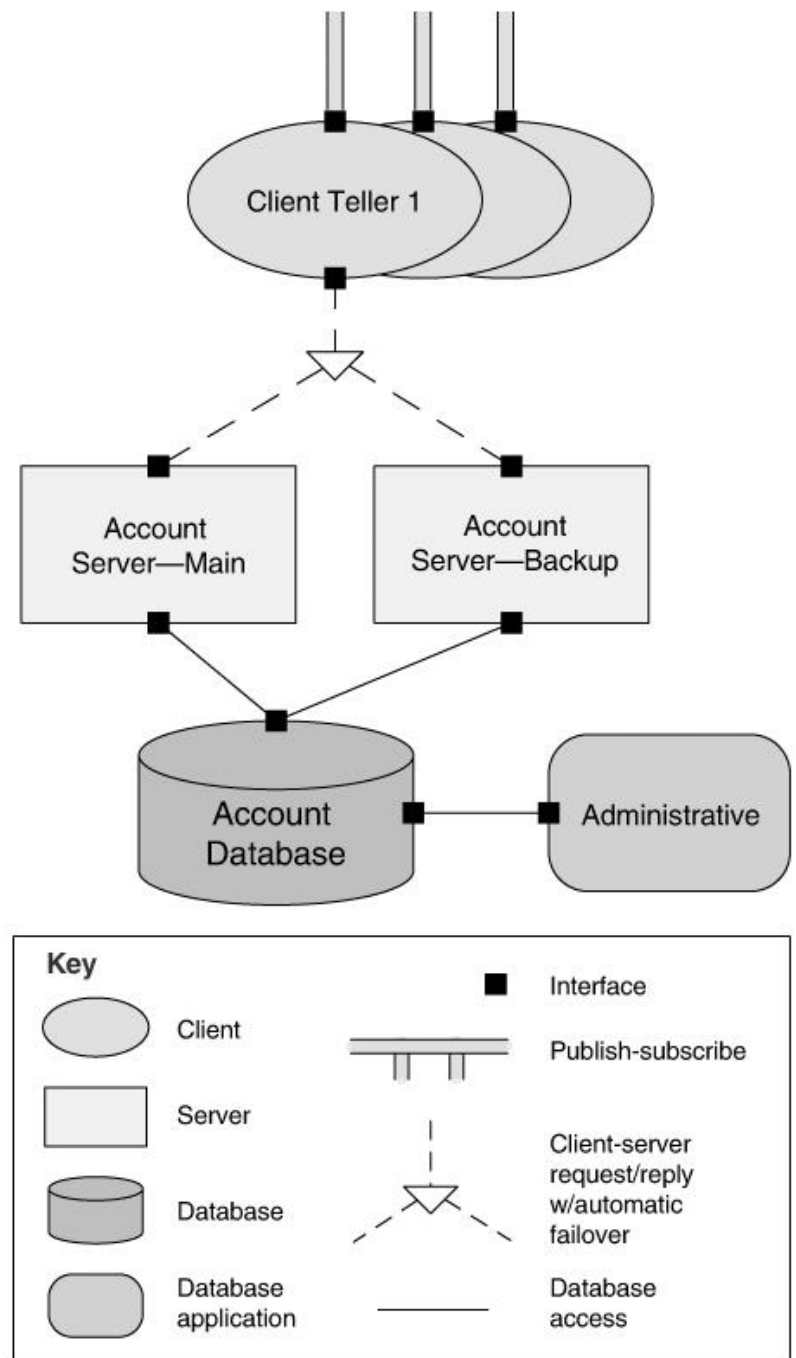
Elementos do tipo módulo podem ser classes, pacotes, camadas, ou meramente divisões de funcionalidade, todas elas refletindo unidades de implementação.



Três tipos de estruturas 2(3)

2/ As 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 interações (conectores).



Tipos de estruturas 3 (3)

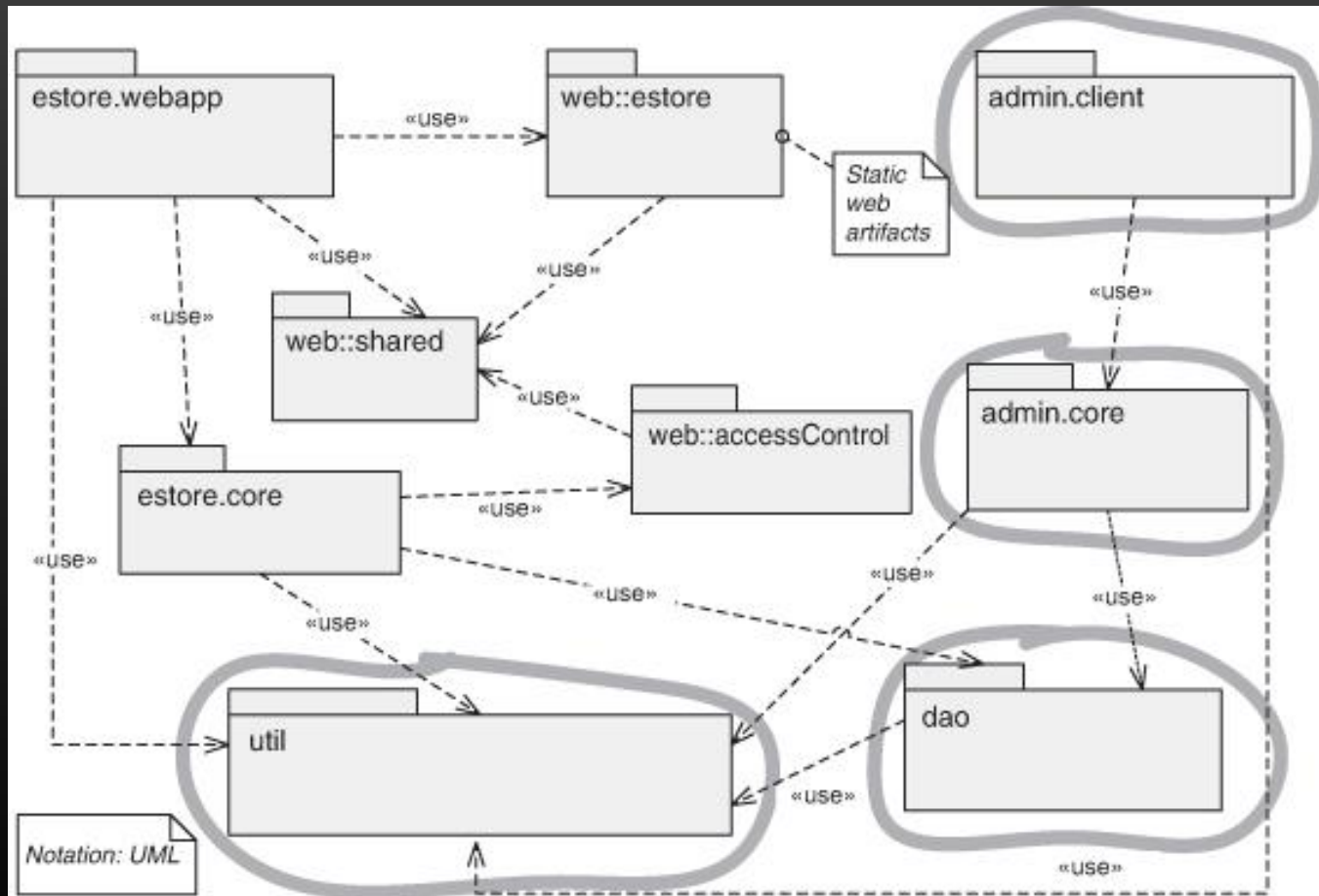
3/ As **estruturas de alocação** (*Allocation structures*) estabelecem o mapeamento das estruturas de software nas estruturas de não-software do sistema, tais como os seus ambientes de desenvolvimento, teste, e execução.

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?

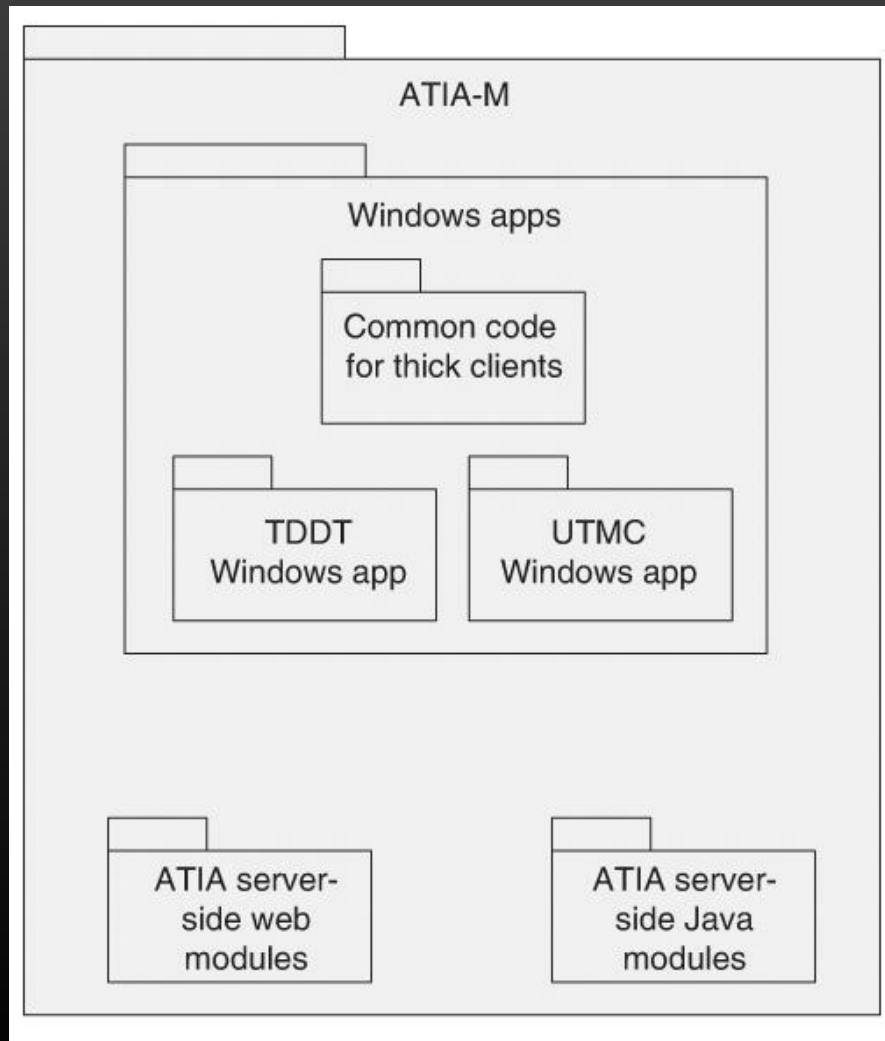
Exemplo de estrutura: módulos

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.

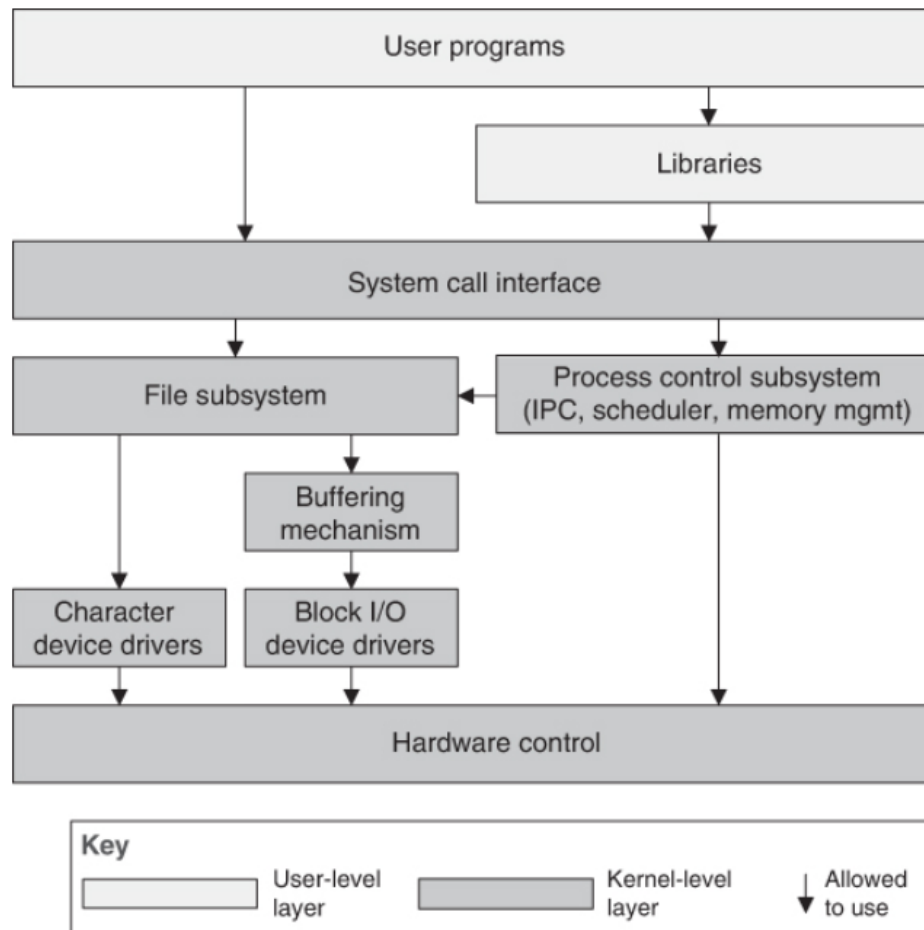
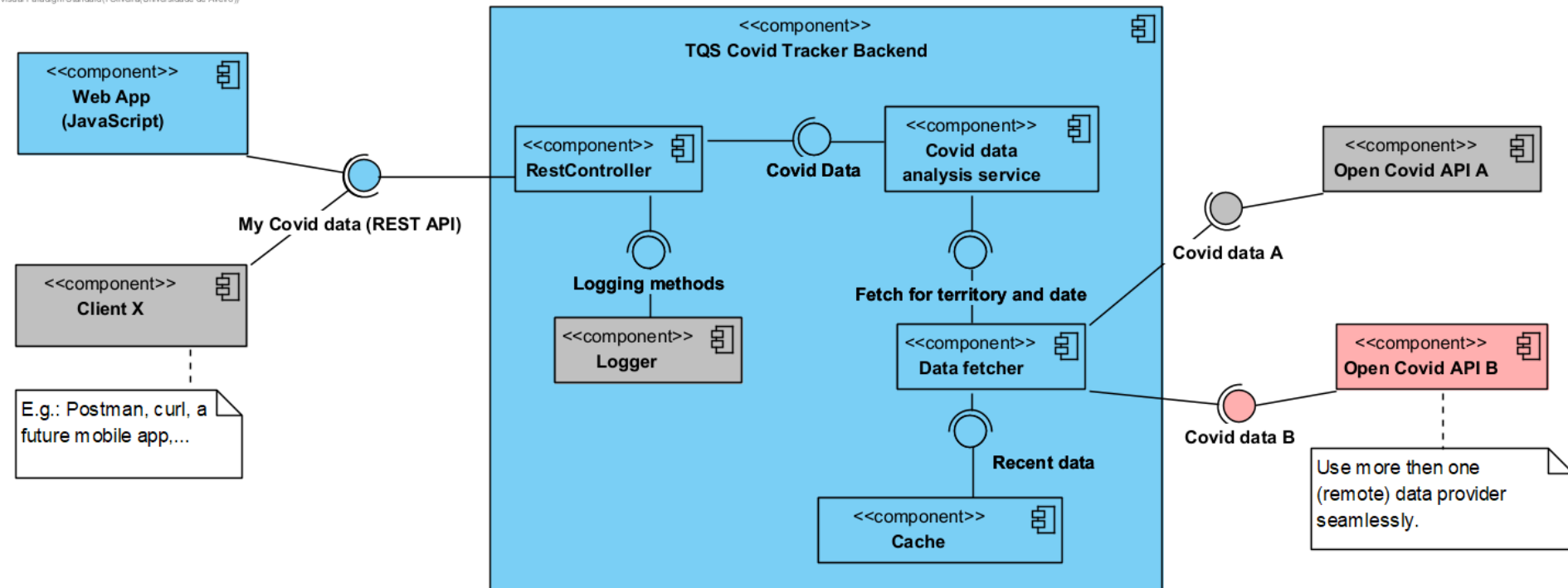


Figure 1.7 Layer structure

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

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

Visual Paradigm Standard [I Oliveira(Universidade de Aveiro)]



Casos de exemplo: alocação/atribuição

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

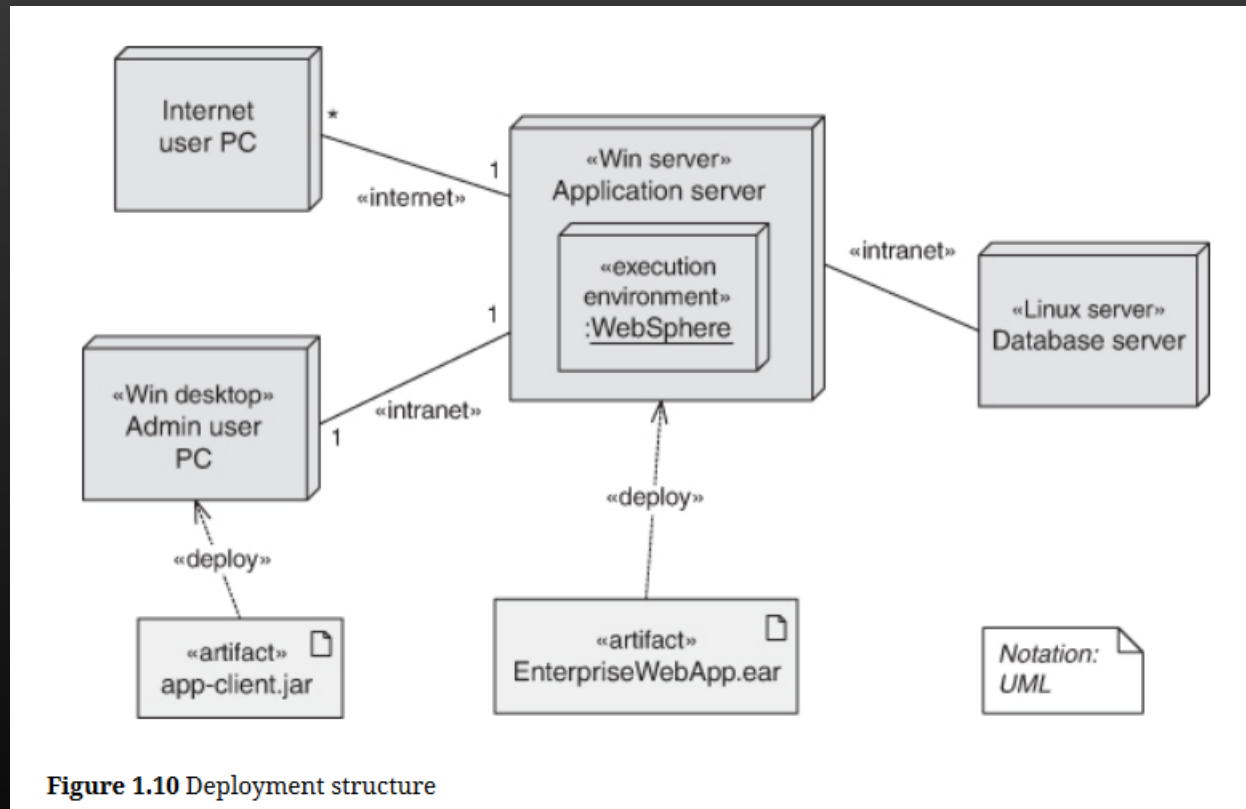
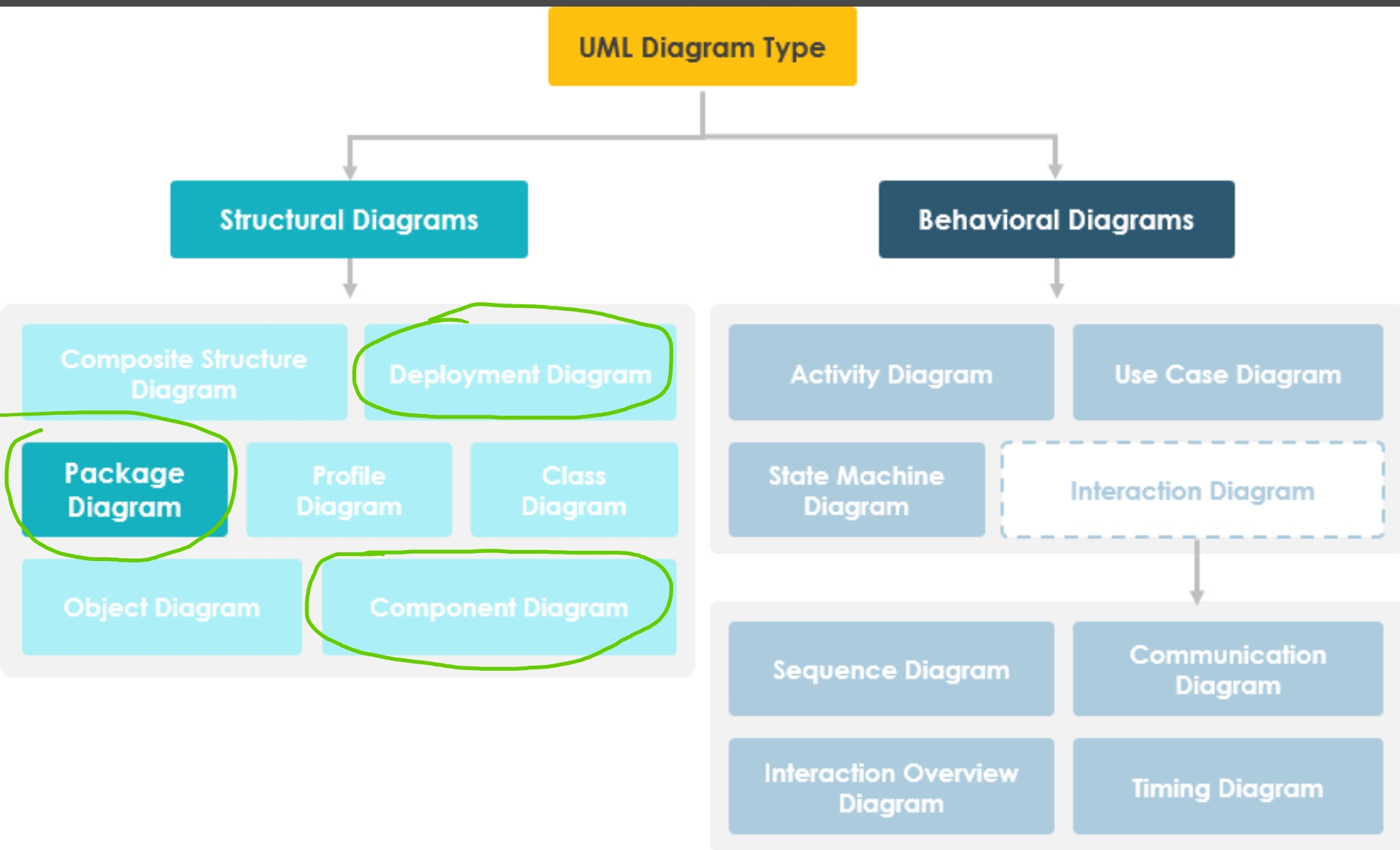


Figure 1.10 Deployment structure

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 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 : 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 development 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. 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



A arquitetura do sistema aborda diferentes perspectivas de análise

❶ Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

❷ Arquitetura 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,...)

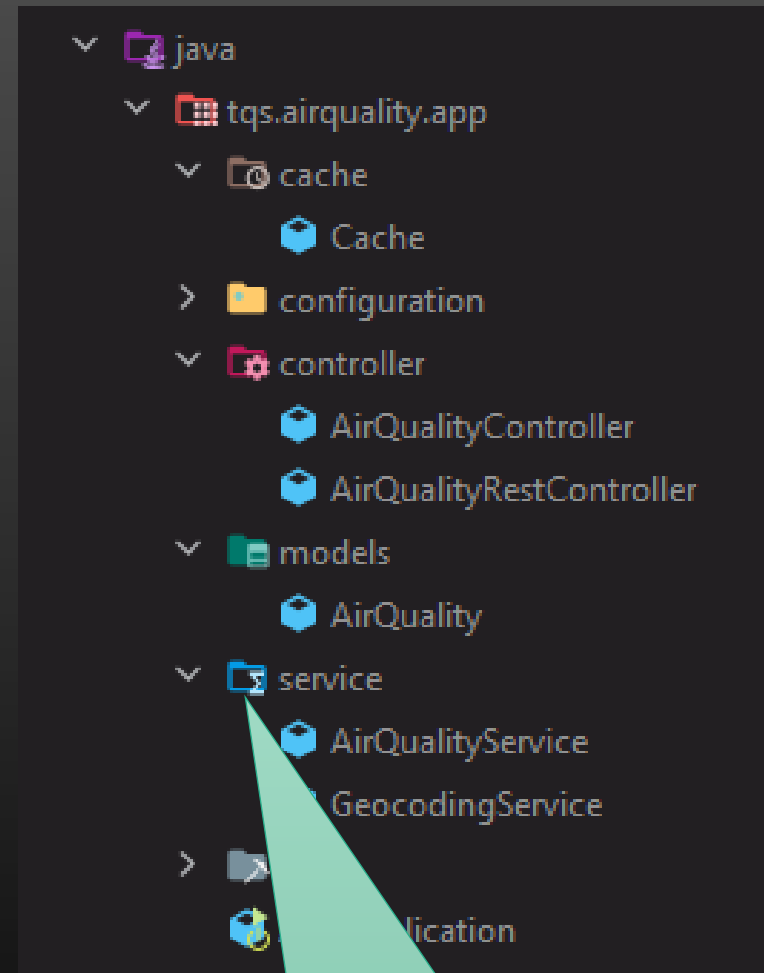
Arquitetura lógica

Organização geral da solução em blocos (*packages*)

Os *packages* podem representar agrupamentos muito diferentes.

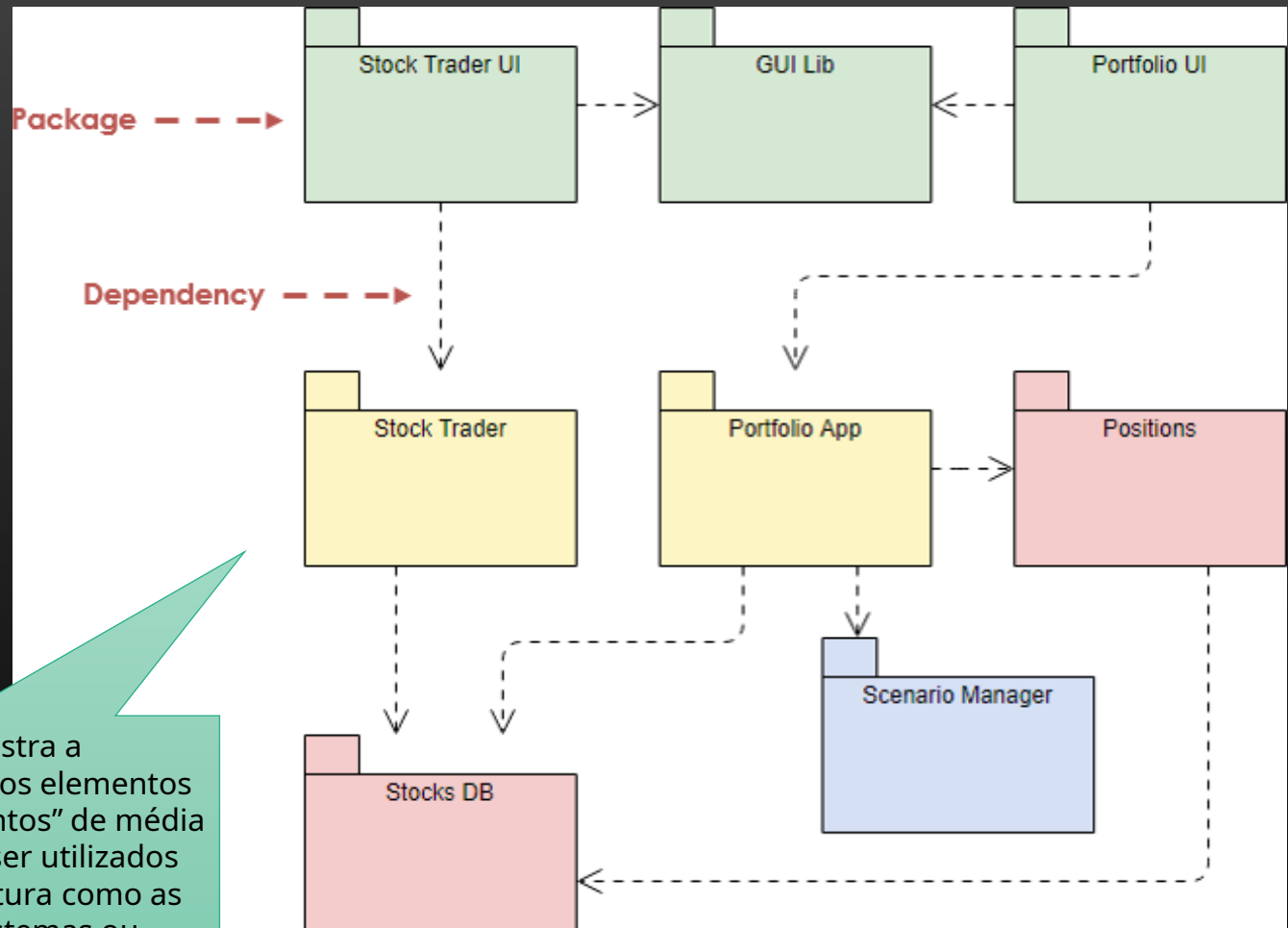
E.g.:

- *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

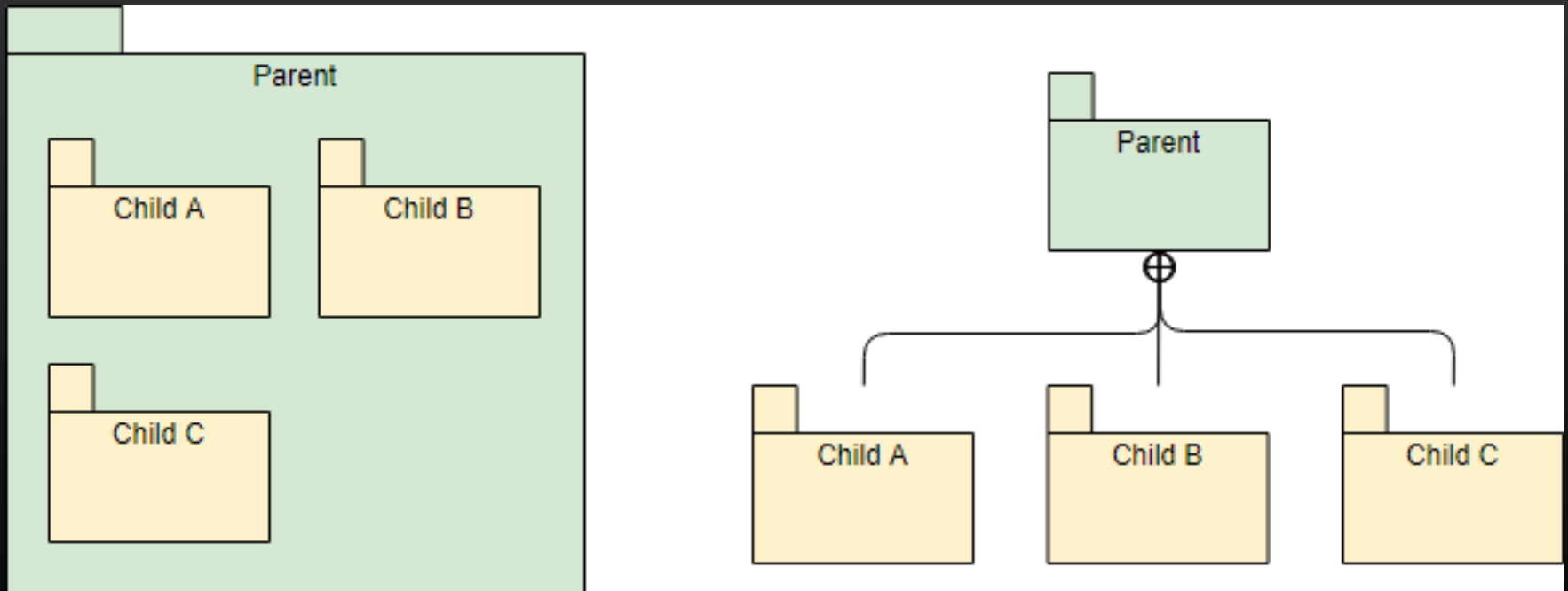


O diagrama de pacotes mostra a disposição e organização dos elementos do modelo em "agrupamentos" de média a larga escala que podem ser utilizados para mostrar tanto a estrutura como as dependências entre sub-sistemas ou módulos.

Comunicar a arquitetura lógica com pacotes (*packages*)

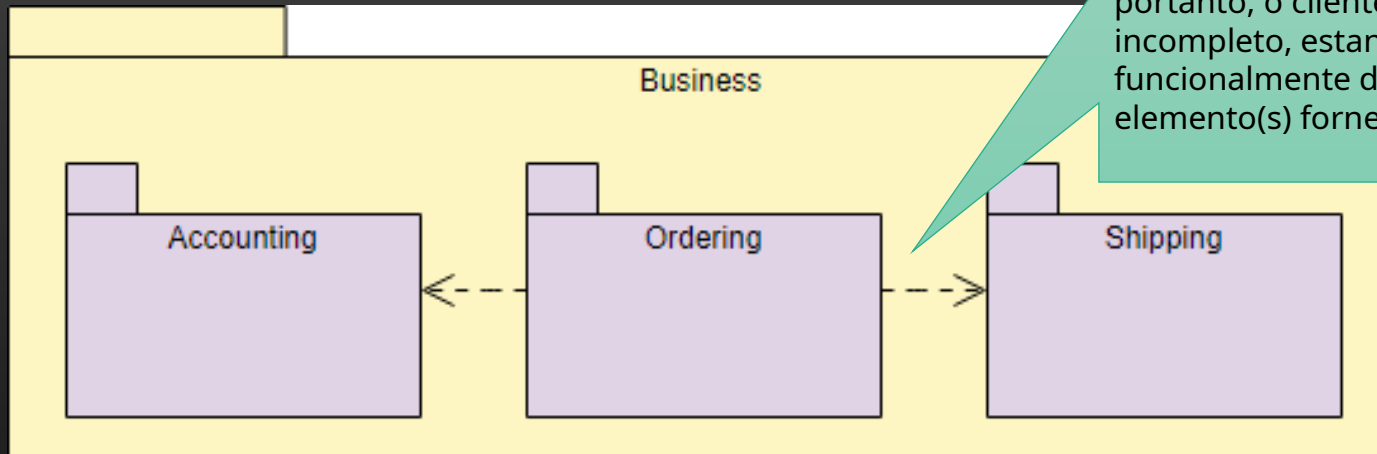
“Contido em”

- duas representações alternativas

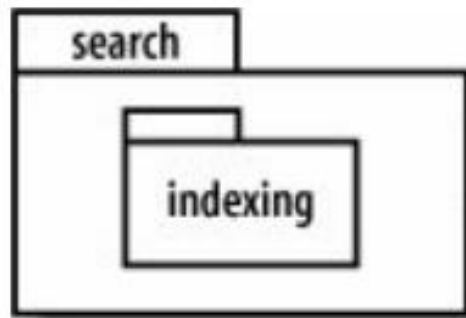


Associações entre pacotes

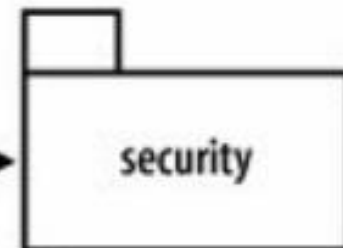
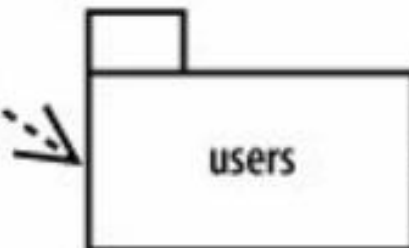
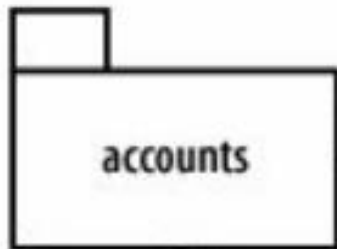
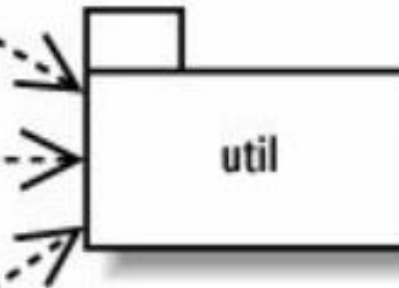
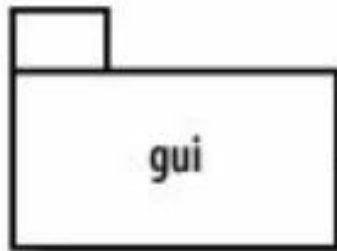
Dependência → transmite a ideia que partes de um elemento (pacote) precisam de (usar) partes de outro.
A dependência é designada como uma relação fornecedor - cliente, em que o fornecedor proporciona algo ao cliente e, portanto, o cliente é, em certo sentido, incompleto, estando semanticamente ou funcionalmente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

❶ Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

❷ Arquitetura 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,...)

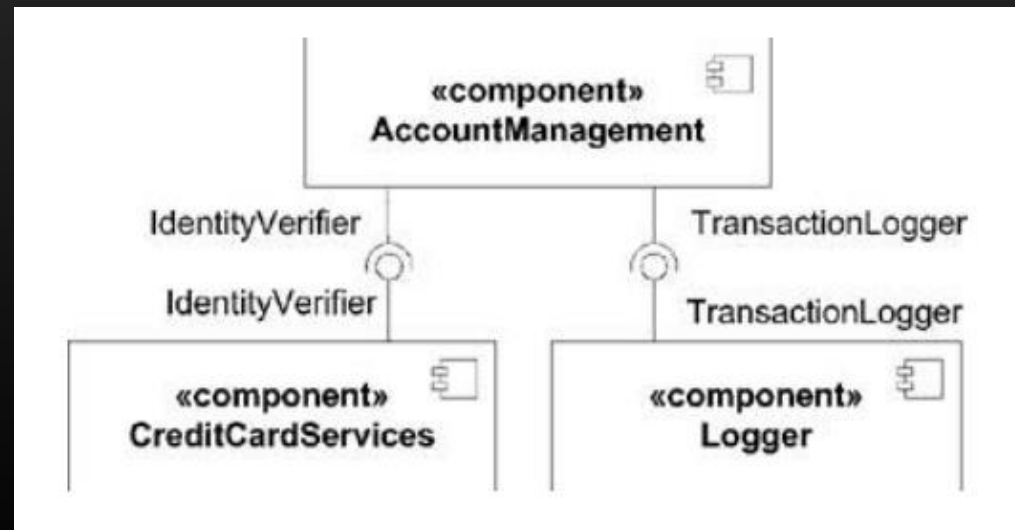
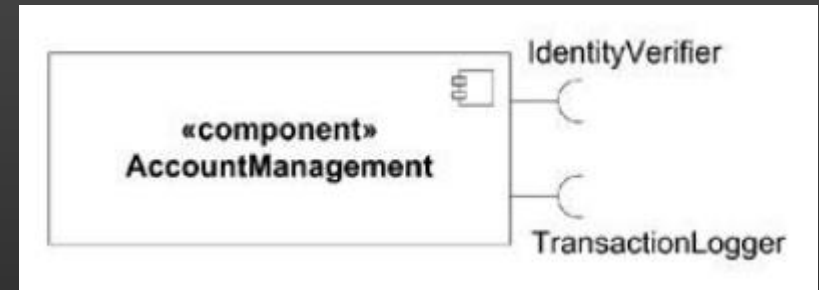
Componente


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








Customer Service
Kundenservice
Service Consommateurs
Servicio Al Consumidor
LEGO.com/service or dial

00800 5346 5555 :     

1-800-422-5346 :  



2x 300424
1x 6022159
2x 4504381

2x 4550348
6x 302324
2x 307024
2x 306924

2x 302423
1x 4211043
2x 4210631
3x 6000606

2x 4567338
2x 4646861
2x 3003944
2x 4244368
1x 6001197
1x 4217722

2x 4211525
2x 4160857
2x 245001
1x 371001
1x 366601
1x 4508408

2x 4504382
2x 379426
1x 302226
4x 614126
1x 6021504
1x 303426

1x 4180536
1x 4180508
2x 4632571
1x 4277932

6037713 / 6037714

LEGO.com

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 *application servers*)

The logo for MVNREPOSITORY, with 'MVN' in a stylized blue font and 'REPOSITORY' in a standard blue font, all within a dark blue rounded rectangle.

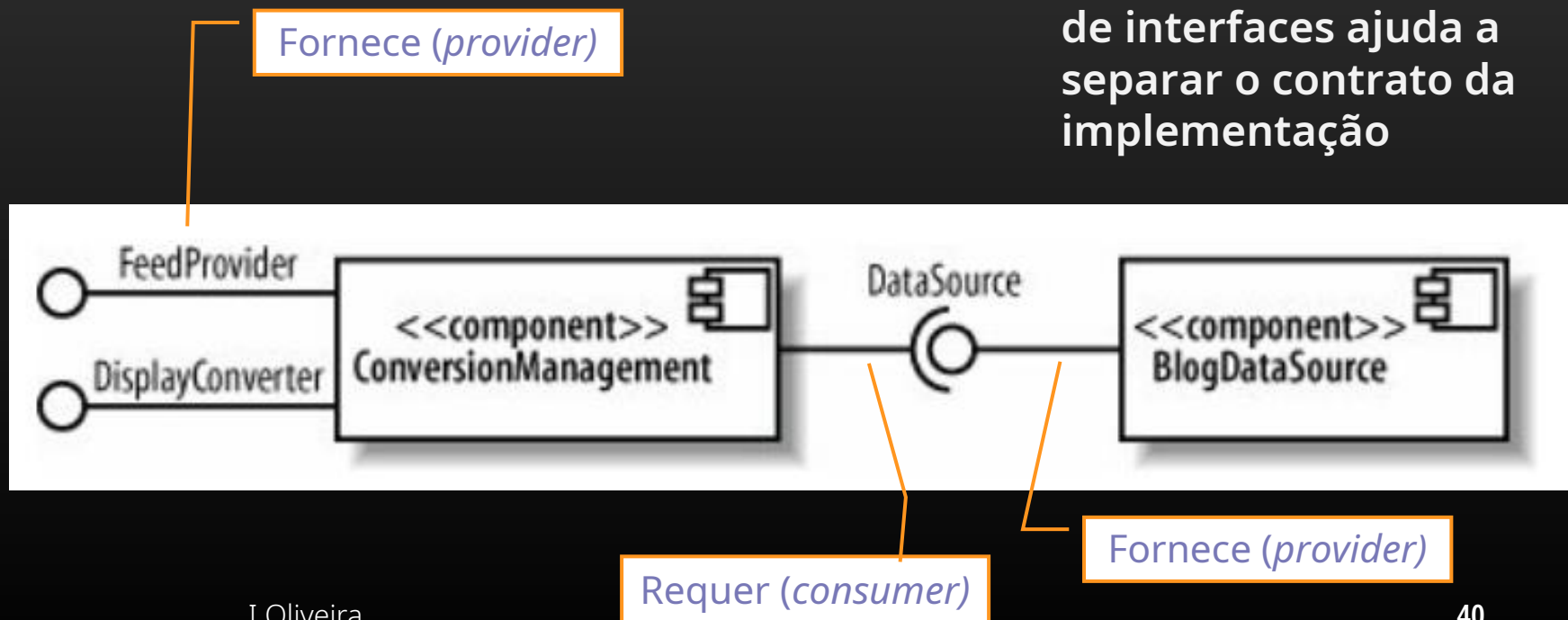
<https://mvnrepository.com>

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

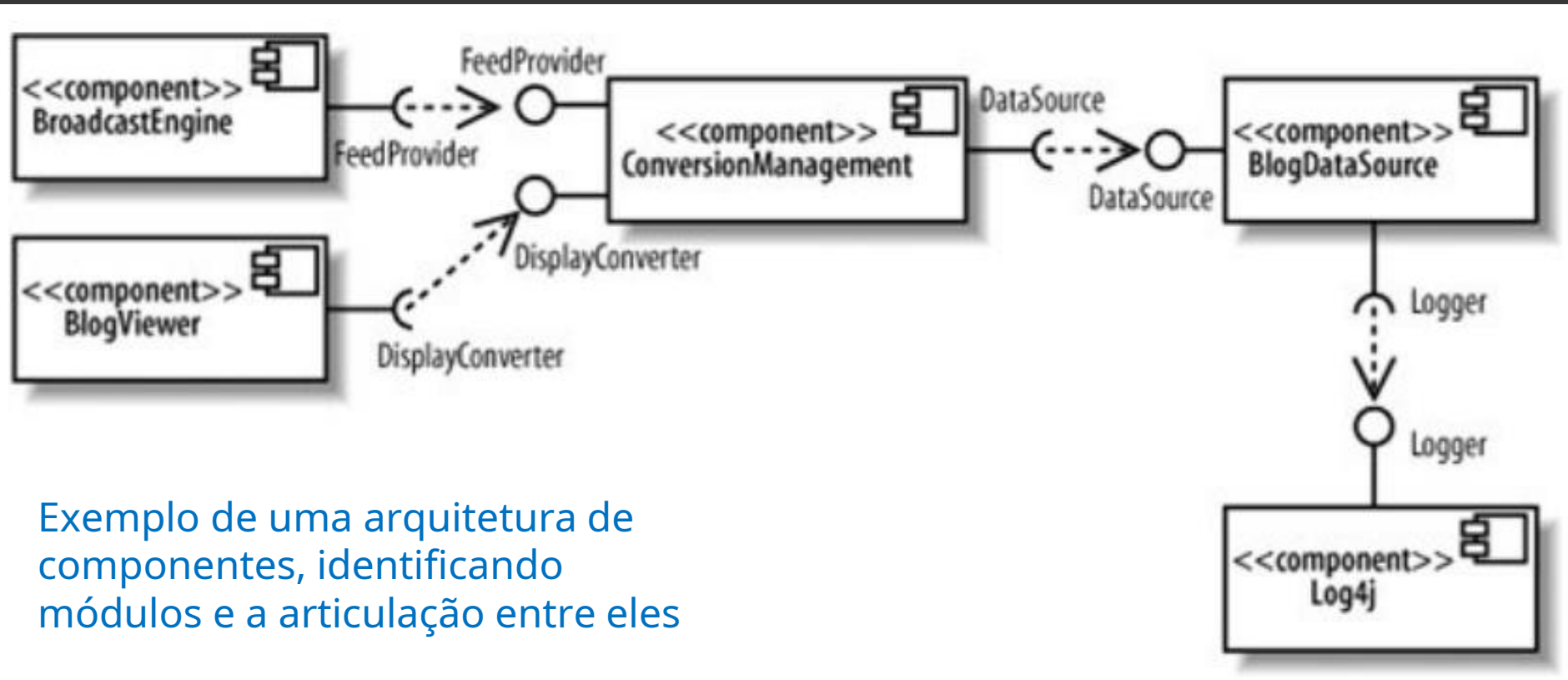
Solução modular com componentes

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

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

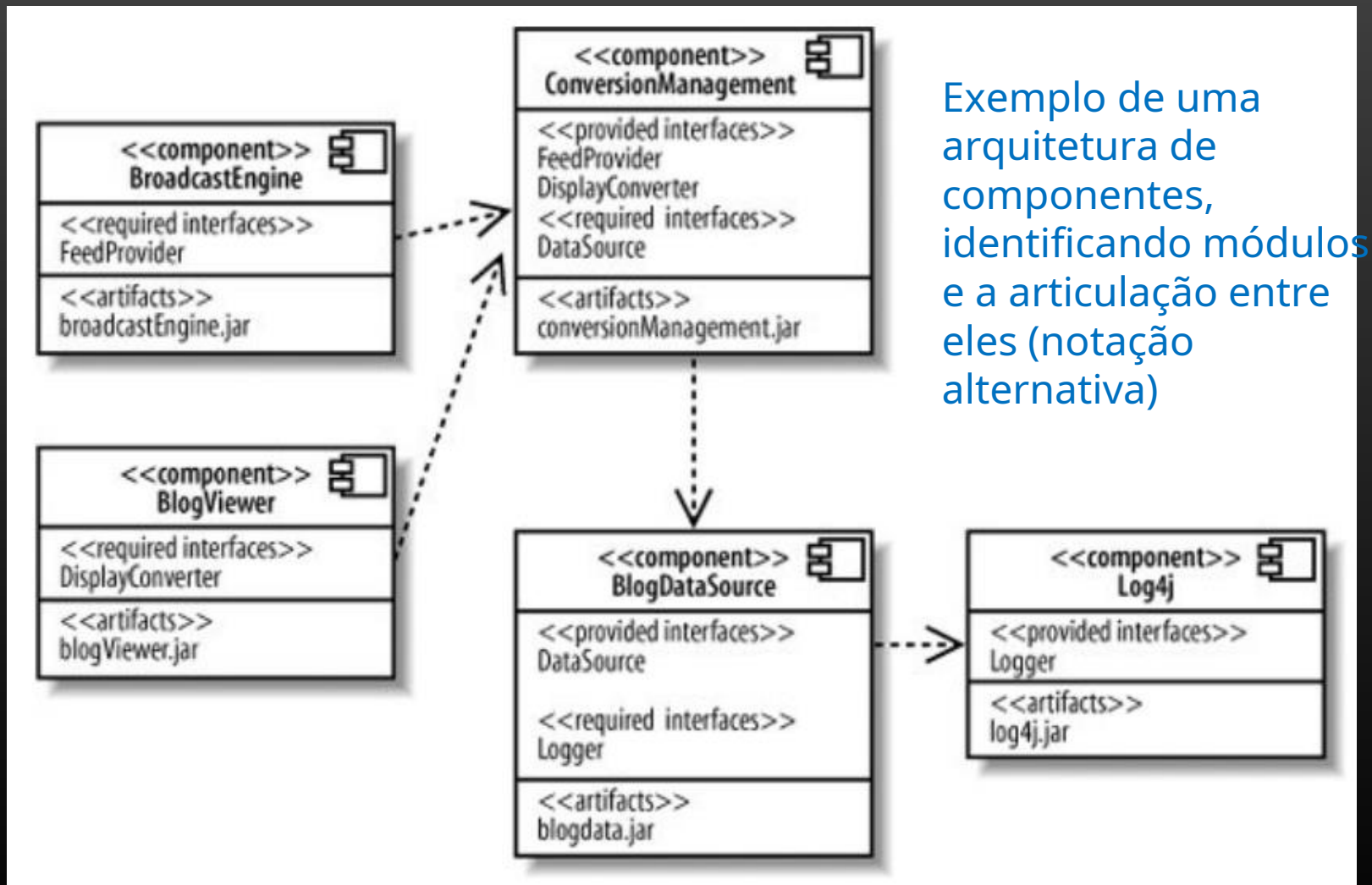


O mesmo modelo, notação ligeiramente diferente 1

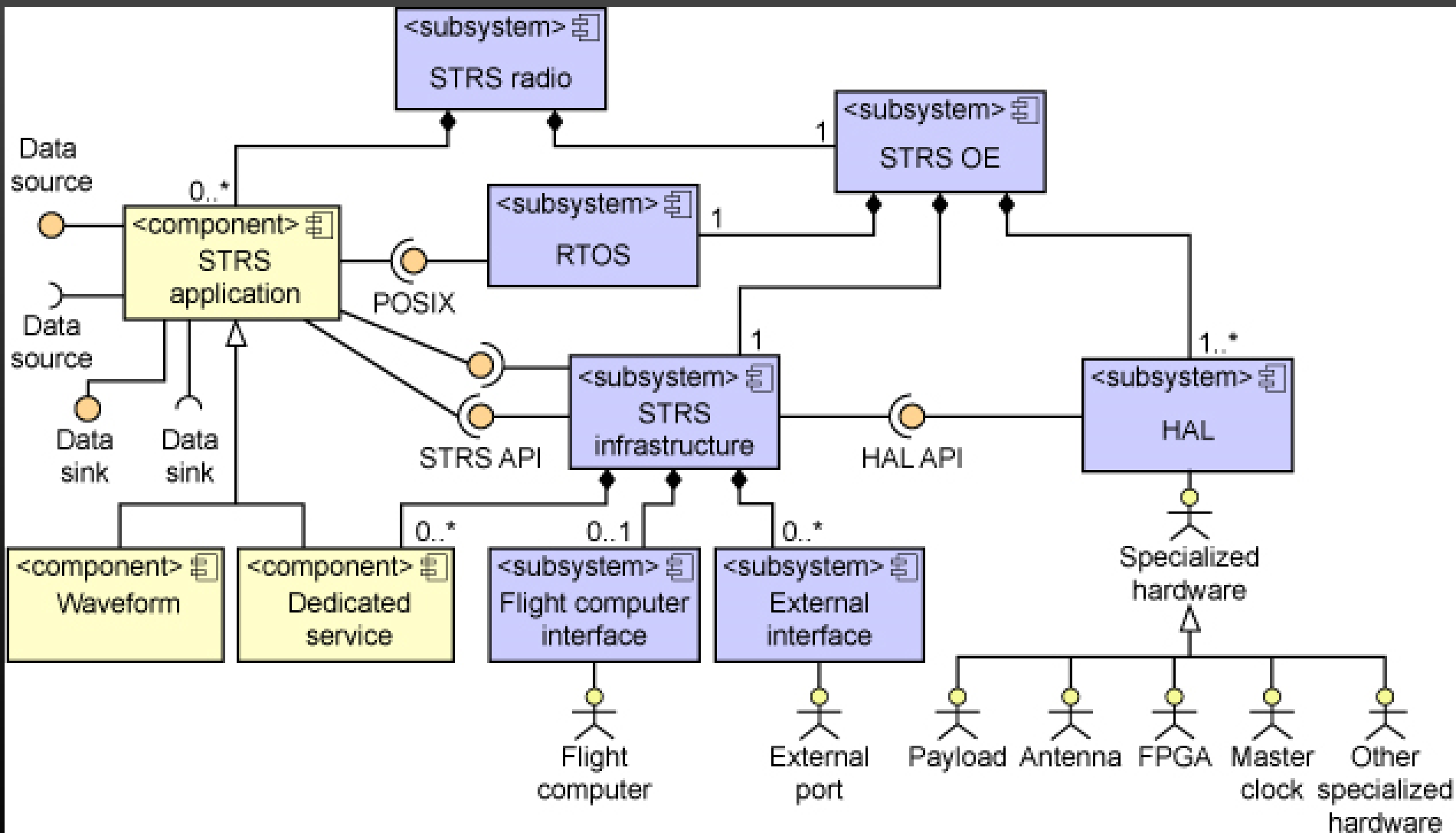


Exemplo de uma arquitetura de componentes, identificando módulos e a articulação entre eles

O mesmo modelo, notação ligeiramente diferente 2

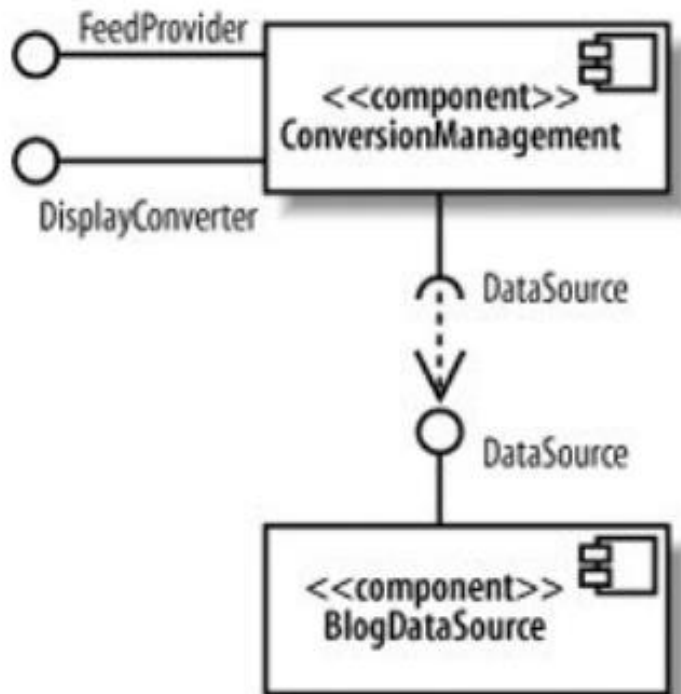


An example from NASA: Radio System

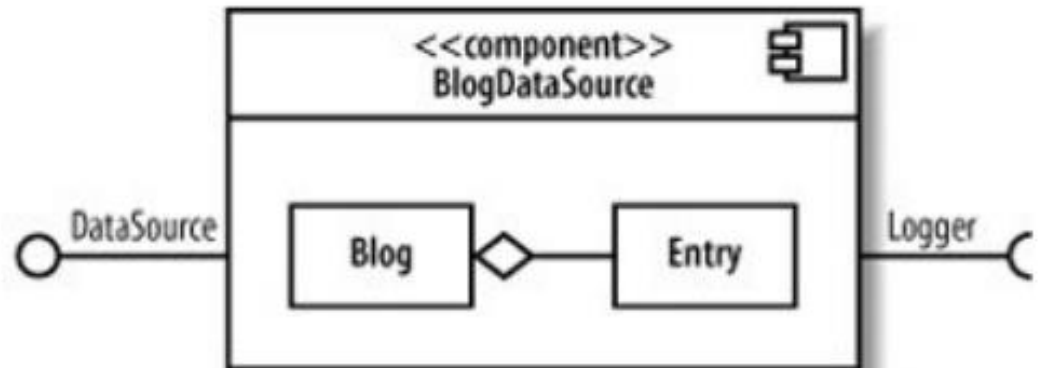


Notação “caixa fechada” / “caixa aberta”

Example Black-Box Component View



Example White-Box Component View



A arquitetura do sistema aborda diferentes perspectivas de análise

❶ Arquitetura lógica do software

Organização geral dos blocos de software

Independente da tecnologia de implementação

❷ Arquitetura 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 aplicativo


Artefactos (*artifact*)

Ficheiros concretos que são executados ou utilizados pela solução

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

A node:

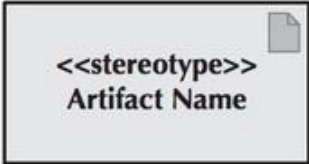
- Is a computational resource, e.g., a client computer, server, separate network, or individual network device.
- Is labeled by its name.
- May contain a stereotype to specifically label the type of node being represented, e.g., device, client workstation, application server, mobile device, etc.

A light gray rectangular box with a 3D effect, representing a node. It contains the text "<<stereotype>>" on the top line and "Node Name" on the bottom line.

<<stereotype>>
Node Name

An artifact:

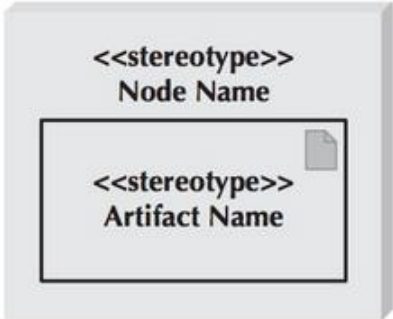
- Is a specification of a piece of software or database, e.g., a database or a table or view of a database, a software component or layer.
- Is labeled by its name.
- May contain a stereotype to specifically label the type of artifact, e.g., source file, database table, executable file, etc.

A light gray rectangular box with a 3D effect, representing an artifact. It contains the text "<<stereotype>>" on the top line and "Artifact Name" on the bottom line. A small document icon is in the top right corner.

<<stereotype>>
Artifact Name

A node with a deployed artifact:

- Portrays an artifact being placed on a physical node.

A light gray rectangular box with a 3D effect, representing a node. It contains the text "<<stereotype>>" on the top line and "Node Name" on the bottom line. Inside the box is a smaller light gray rectangular box with a 3D effect, representing an artifact. It contains the text "<<stereotype>>" on the top line and "Artifact Name" on the bottom line. A small document icon is in the top right corner of the artifact box.

<<stereotype>>
Node Name

<<stereotype>>
Artifact Name

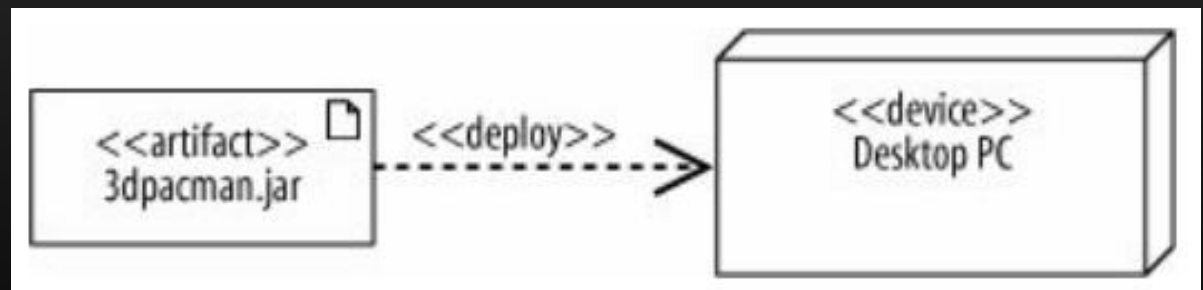
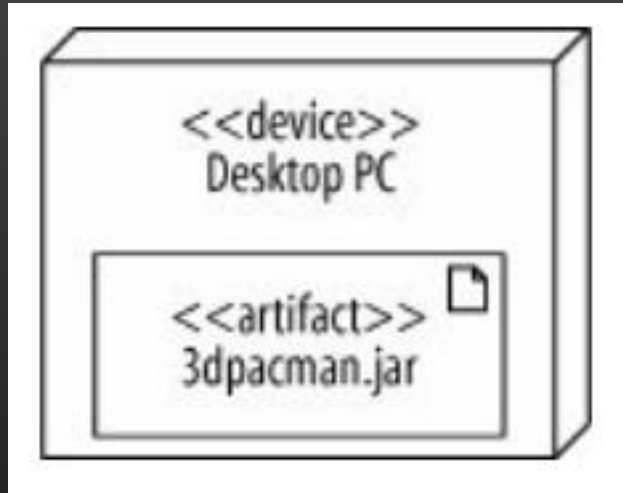
A communication path:

- 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).

<<stereotype>>

Os artefactos são executados em nós

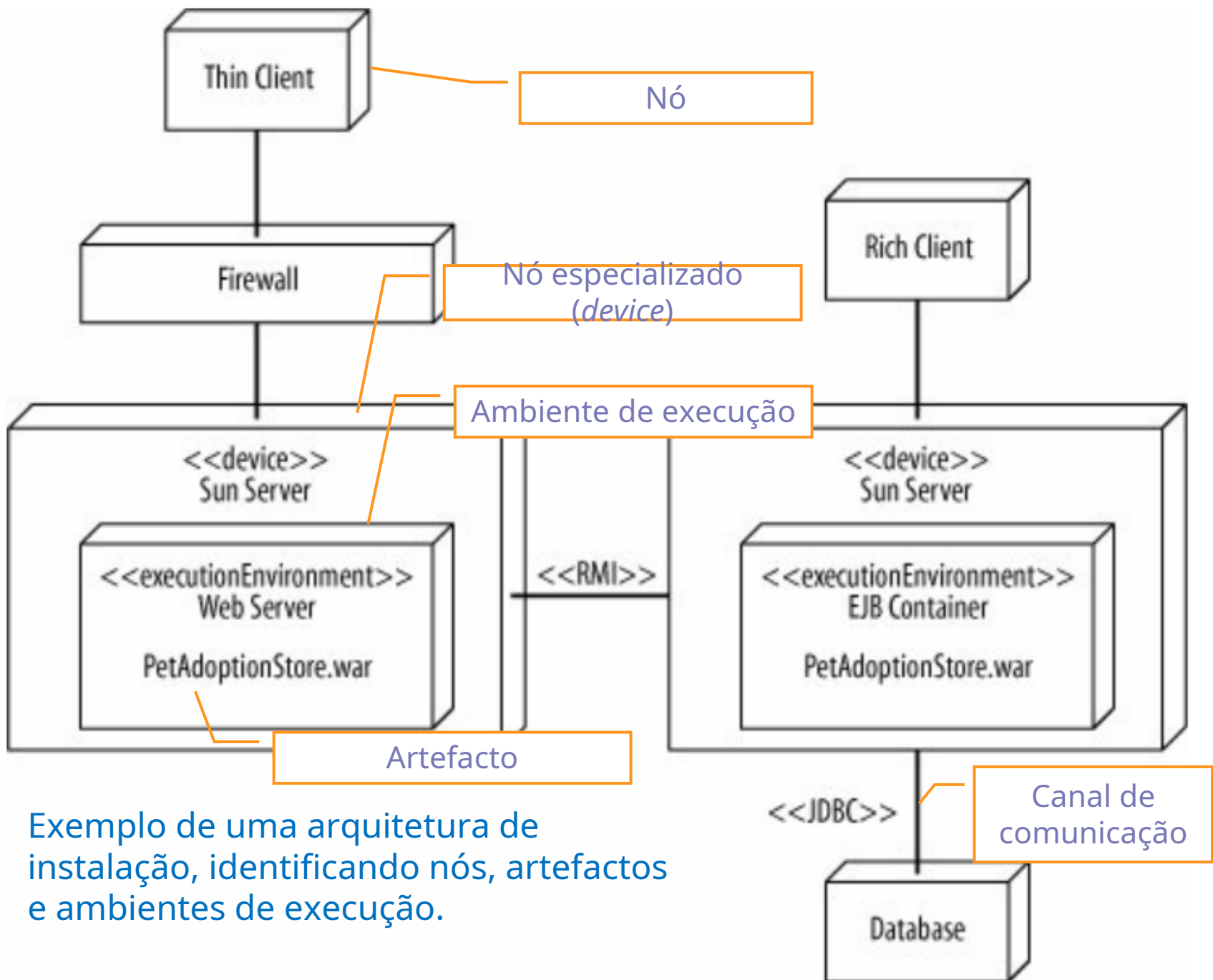
Notações alternativas.



Rastreabilidade até aos componentes

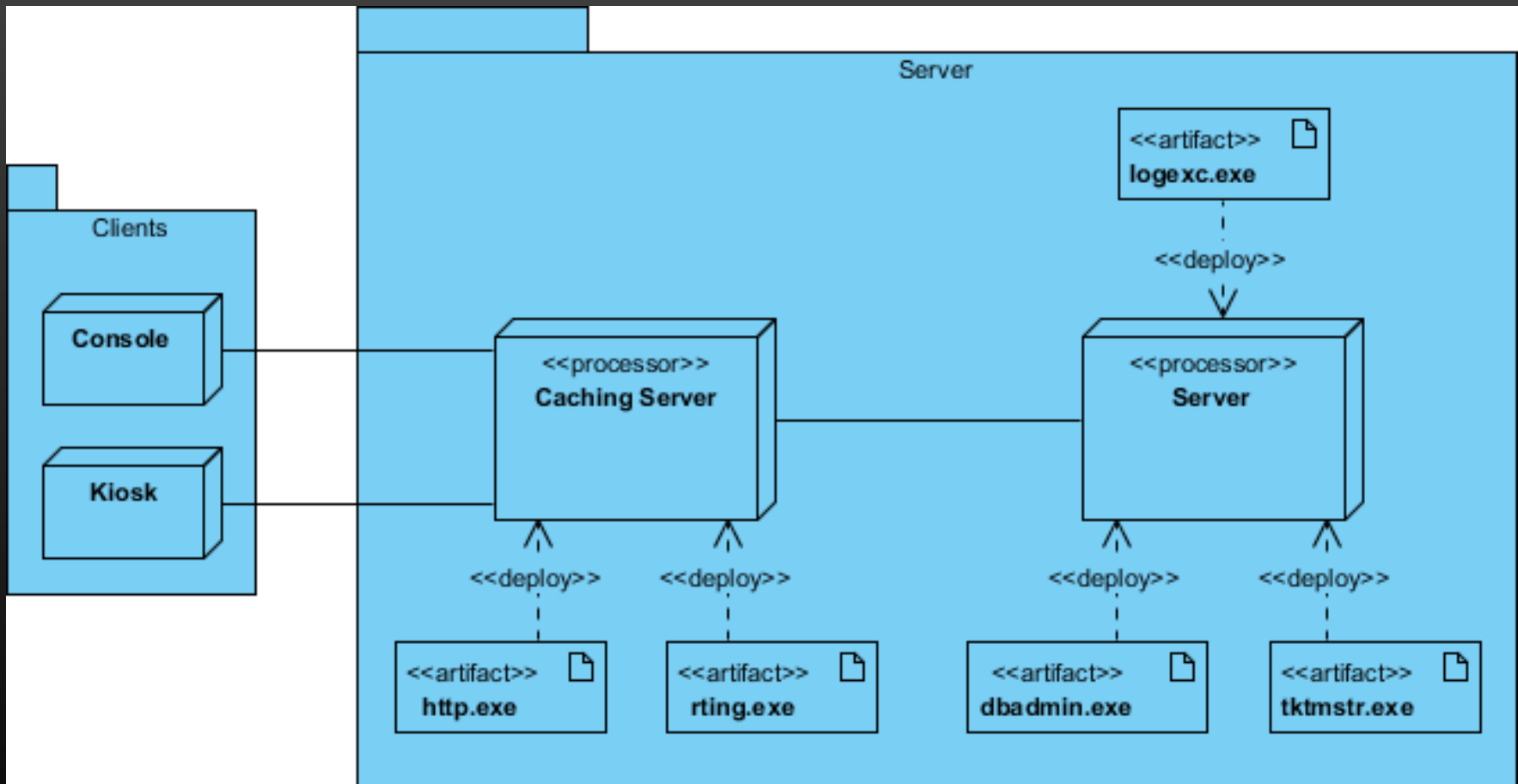


A relação “manifest” permite associar componentes a artefactos.

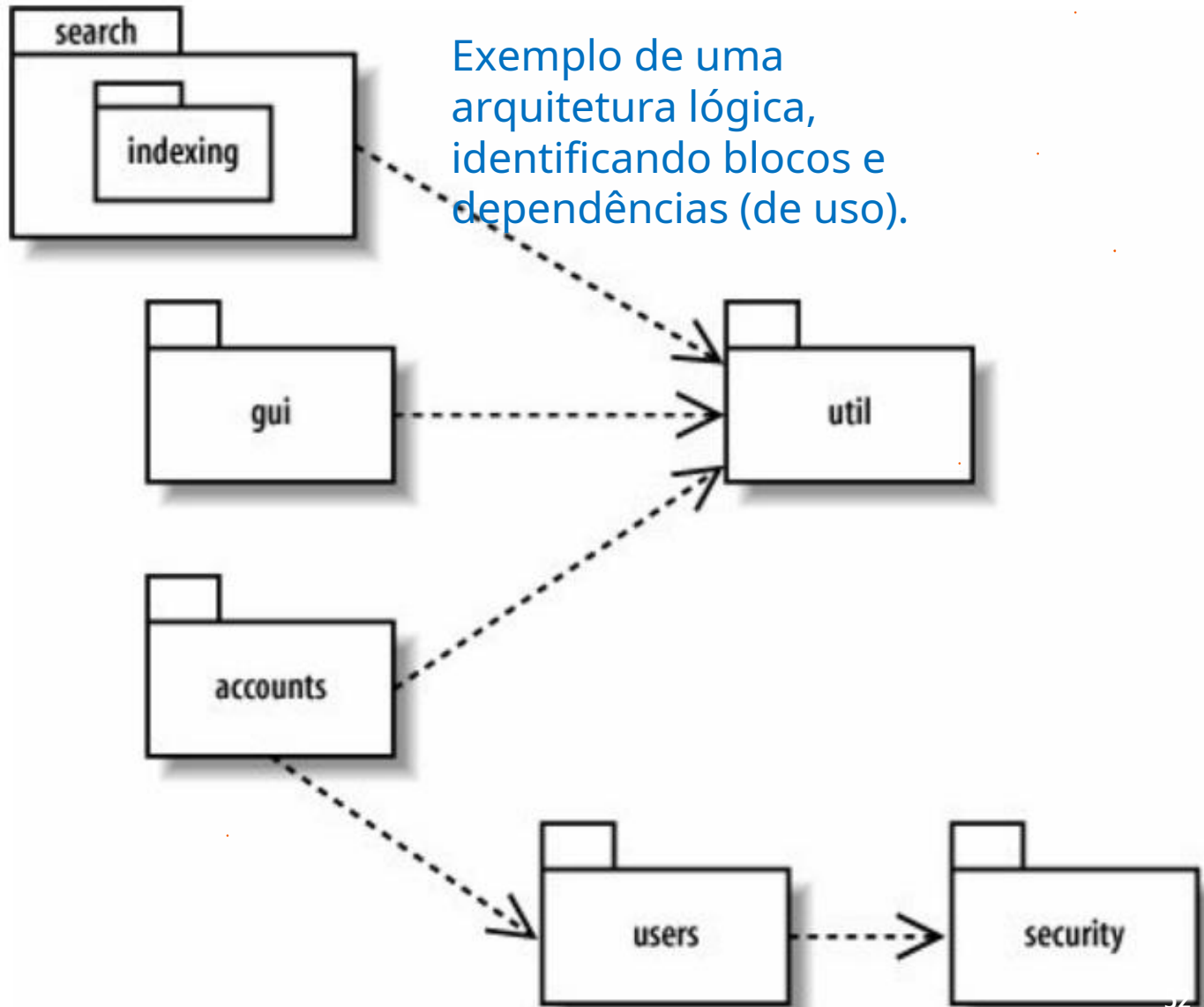


Exemplo de uma arquitetura de instalação, identificando nós, artefactos e ambientes de execução.

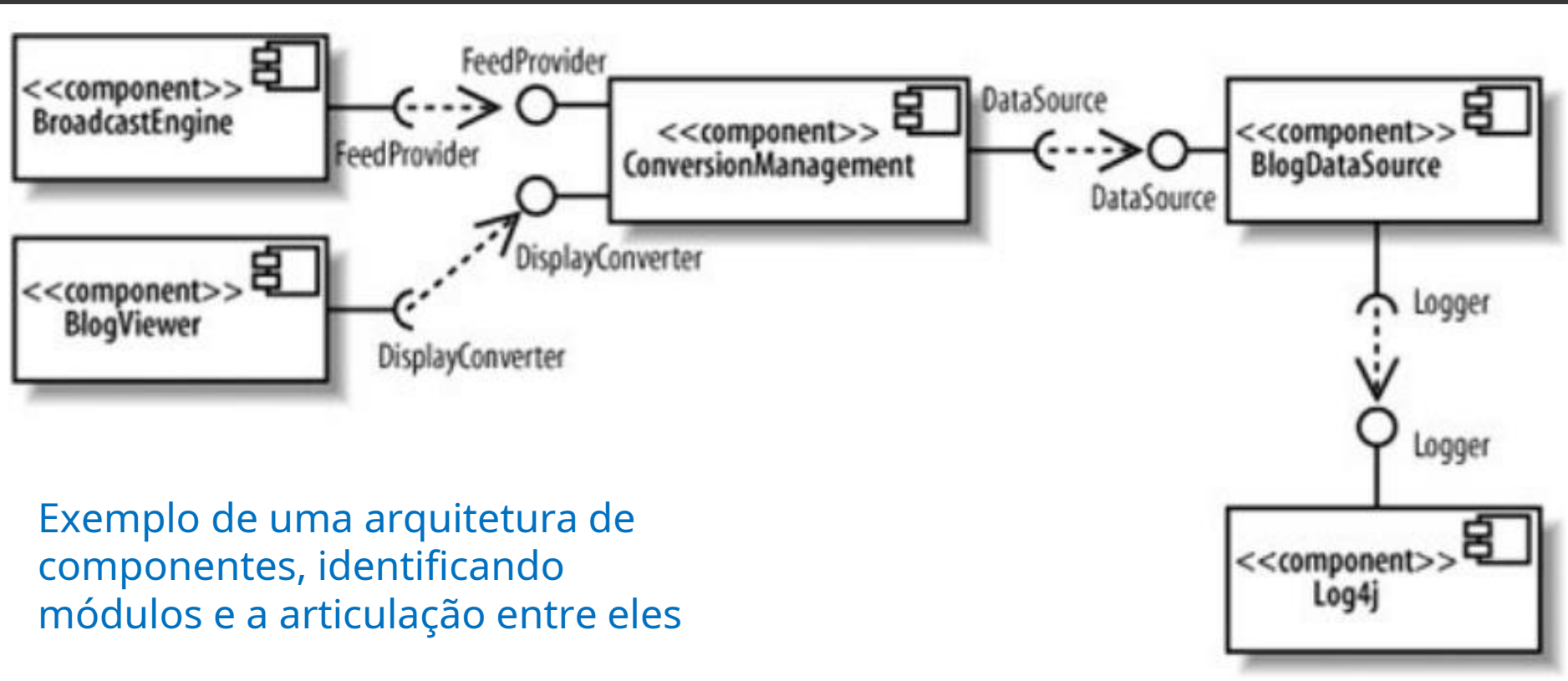
Exemplos (diagrama de instalação)



Vista de módulos → na UML: d. de pacotes

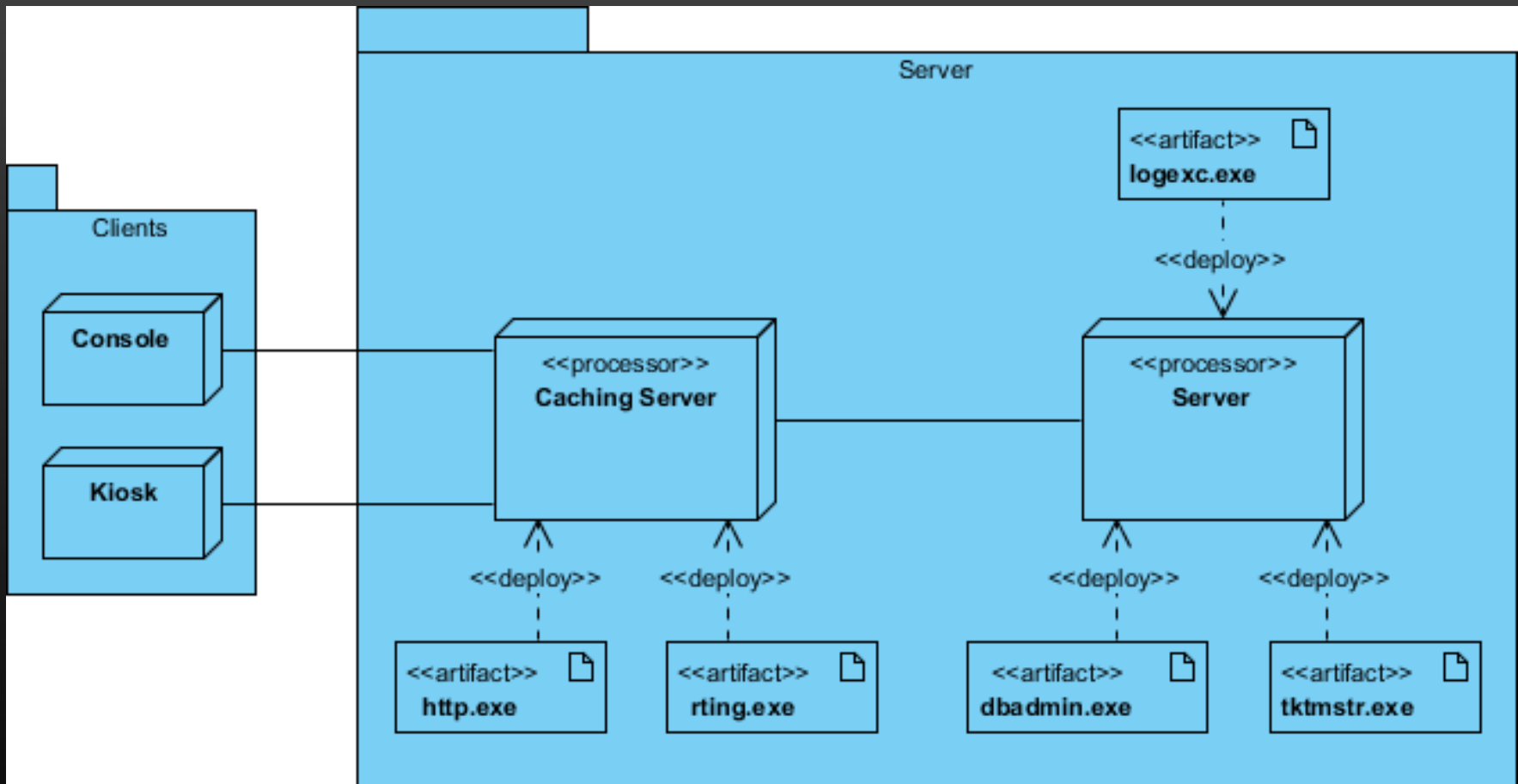


Vista C&C → na UML: d. componentes

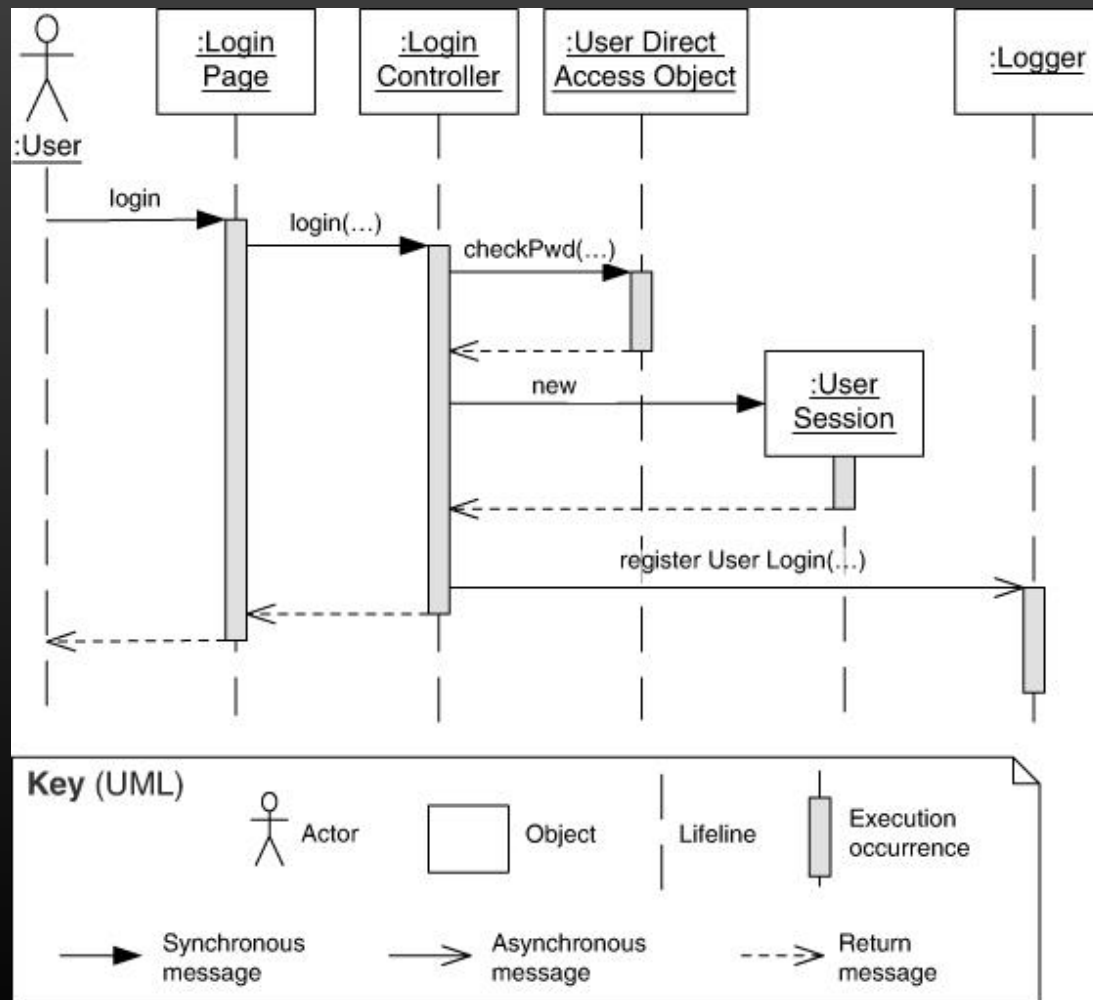


Exemplo de uma arquitetura de componentes, identificando módulos e a articulação entre eles

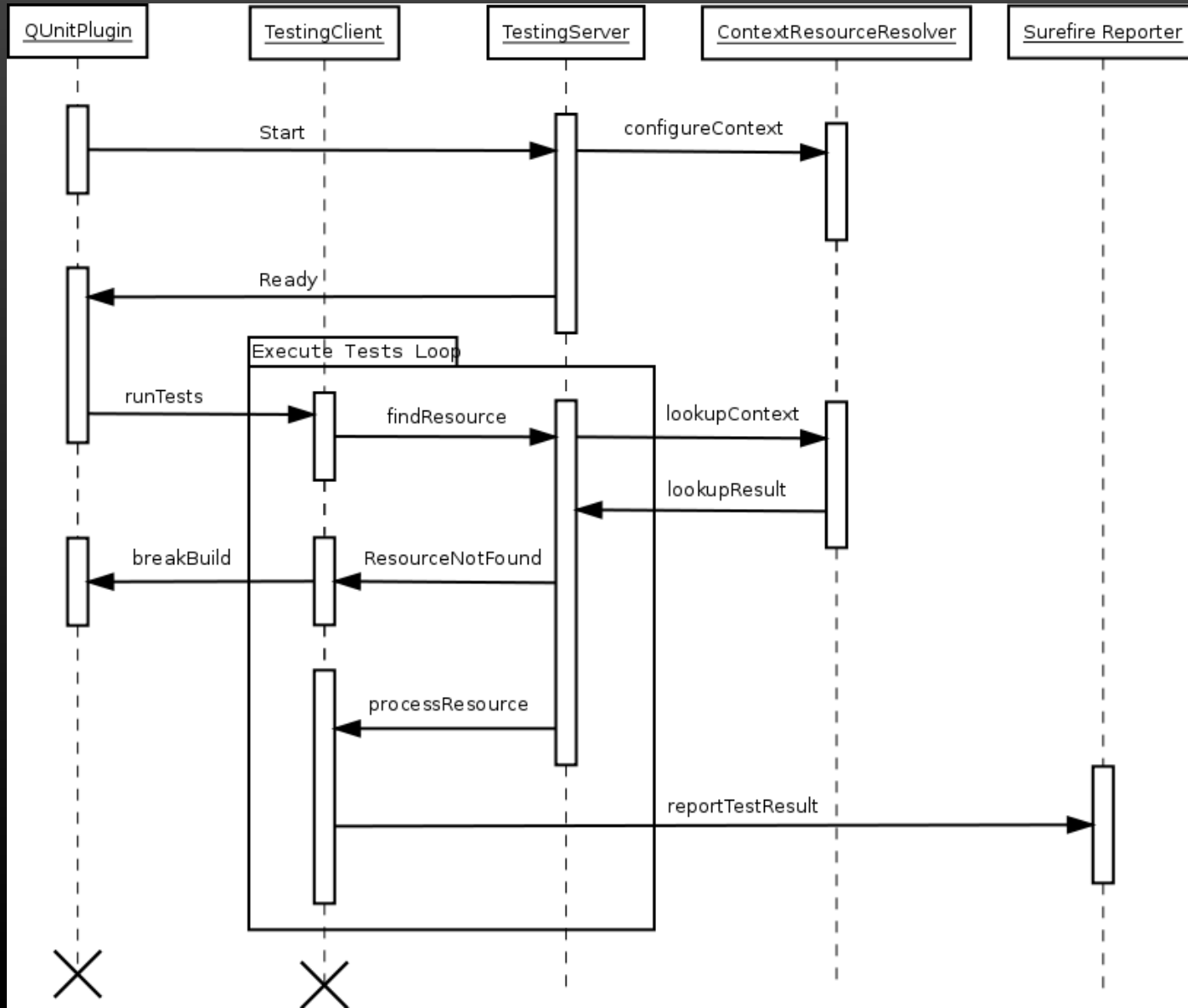
Vista de alocação → na UML: d. instalação/*deployment*



Vista comportamental → na UML: d. sequência



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.

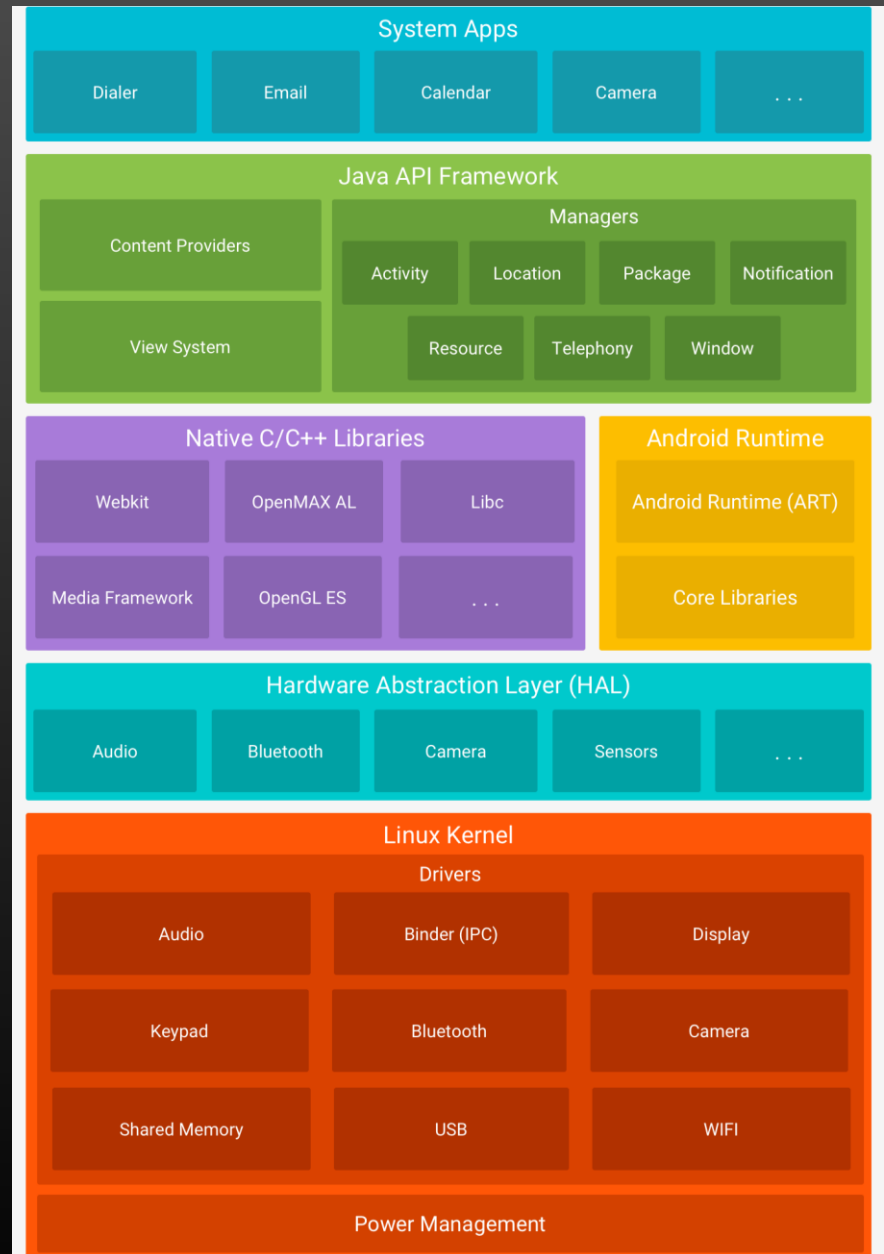
Exemplos:

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

Qual é o “padrão”?

Uma arquitetura por camadas

- Camadas “debaixo” fornecem serviços às camadas “de cima”
- Cada camada comunica apenas com as camadas adjacentes (consome serviços da de baixo, oferece serviços à de cima)
- Há geralmente um crescendo no nível de abstração: camadas abaixo mais próximas do hardware/armazenamento; camada superior voltada para o utilizador.
- Cada camada pode ser organizada em módulos, mostrando componentes no mesmo nível de abstração

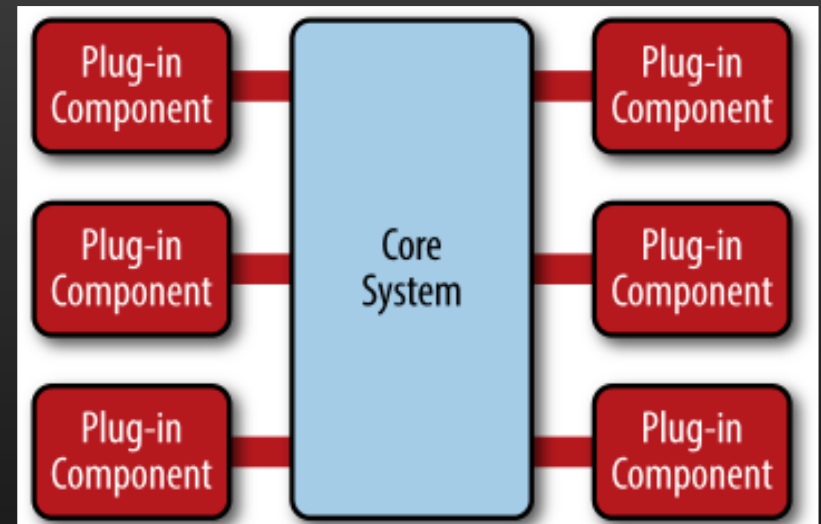


Arquitetura *Microkernel*

**Núcleo da aplicação (estável) +
plugins (dinâmico)**

Novas funcionalidades da aplicação
são adicionadas como plug-ins

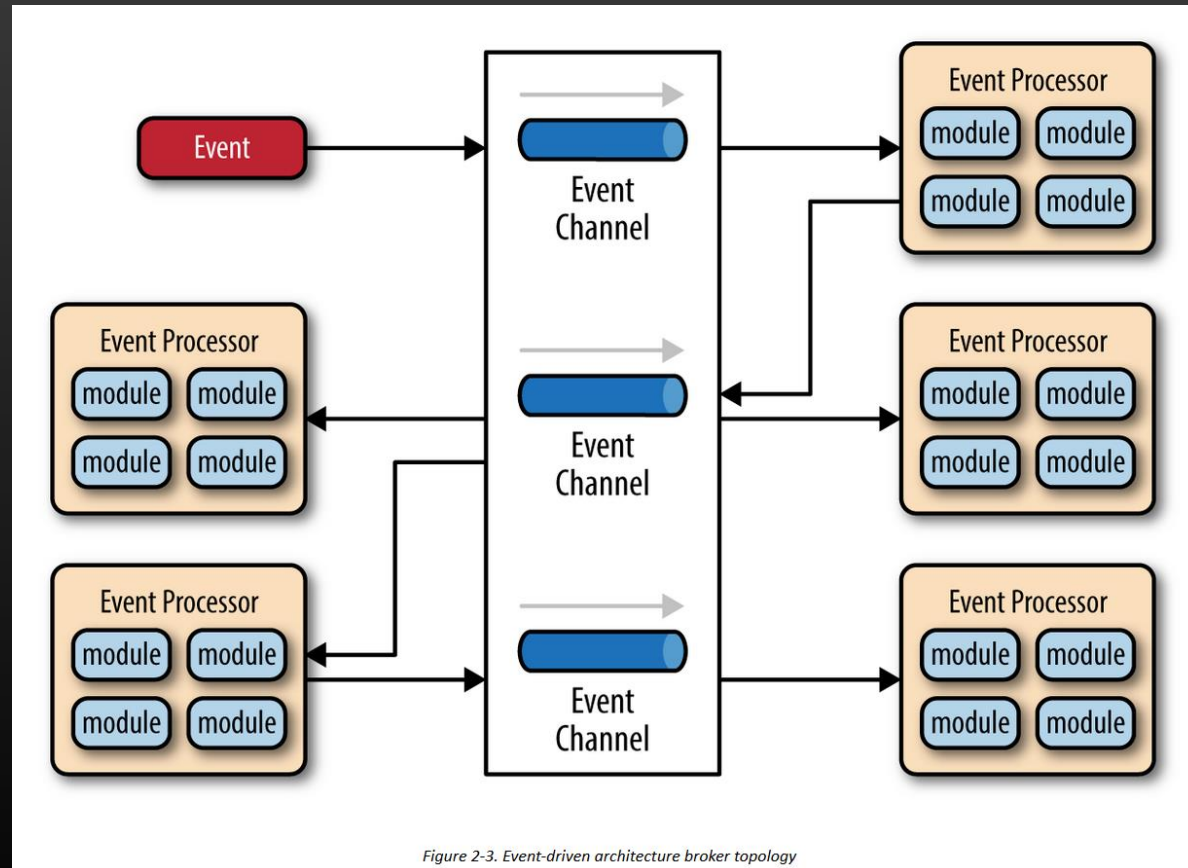
Vantagens: extensibilidade, separação
de assuntos e isolamento.



Processamento de eventos (*event-driven*)

O padrão de “arquitetura por eventos” é um estilo de arquitetura assíncrono e distribuído, usado em aplicações que precisam de ser muito escaláveis (e.g.: processar milhares de leituras de sensores por segundo)

A arquitetura por eventos é composta por componentes de processamento de eventos altamente dissociados (*decoupled*) e especializados (dedicados a um tipo de processamento) que recebem e processam os eventos.



Um exemplo da indústria: uma solução de M2M (*machine-to-machine*)

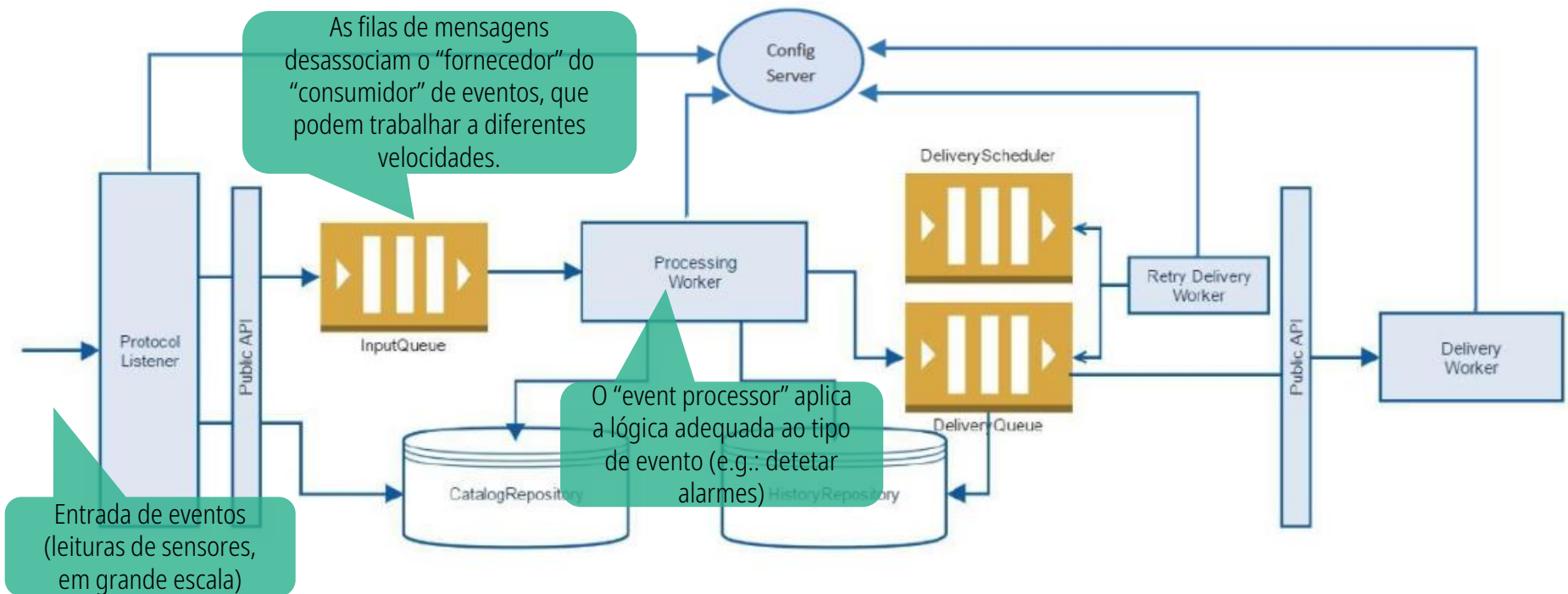


Figura 5.2: Arquitetura e componentes da SmartIOT.

Mais info: <https://repositorio-aberto.up.pt/handle/10216/109650>

Arquitetura por camadas

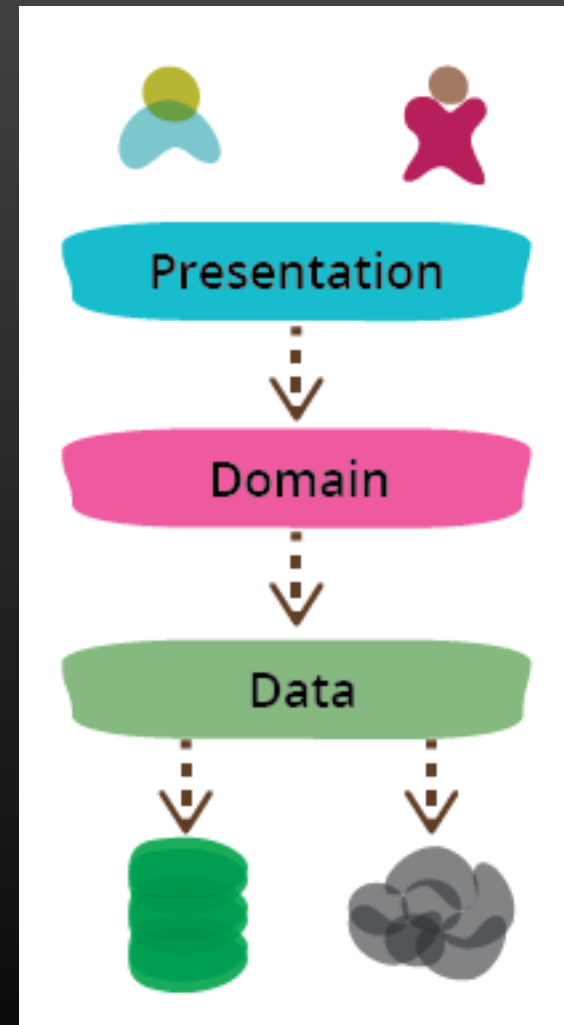
Divisão modular da solução de software em camadas/níveis de abstração.

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>

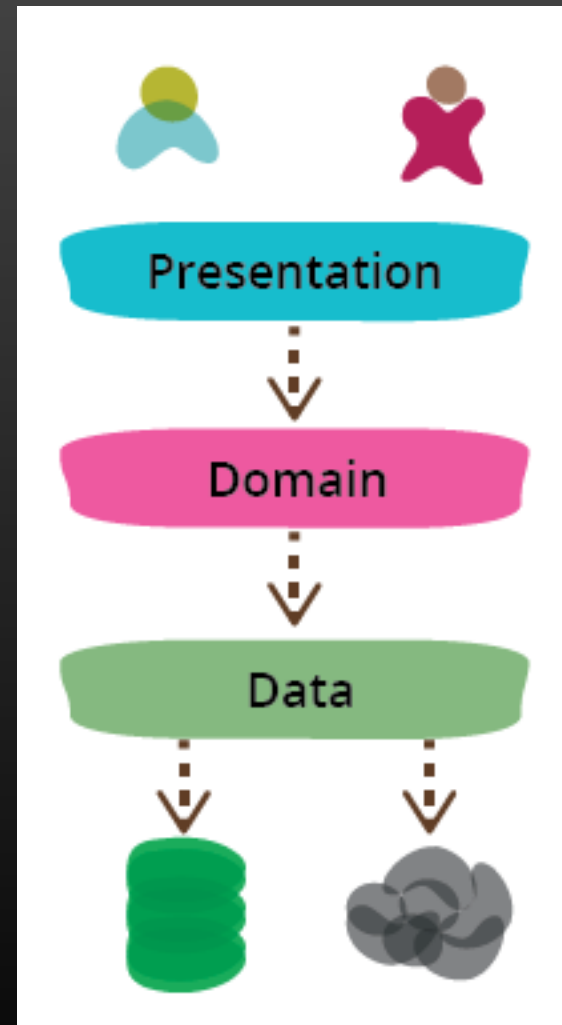
A arquitetura de 3 camadas

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)

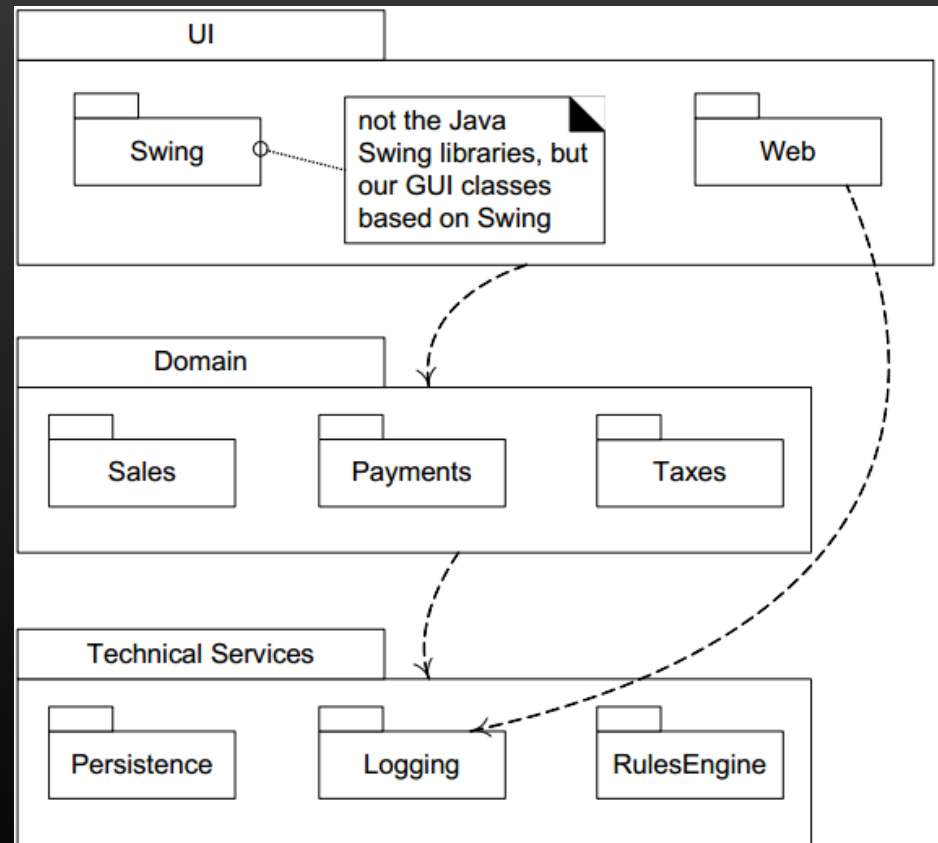
Camadas verticais:

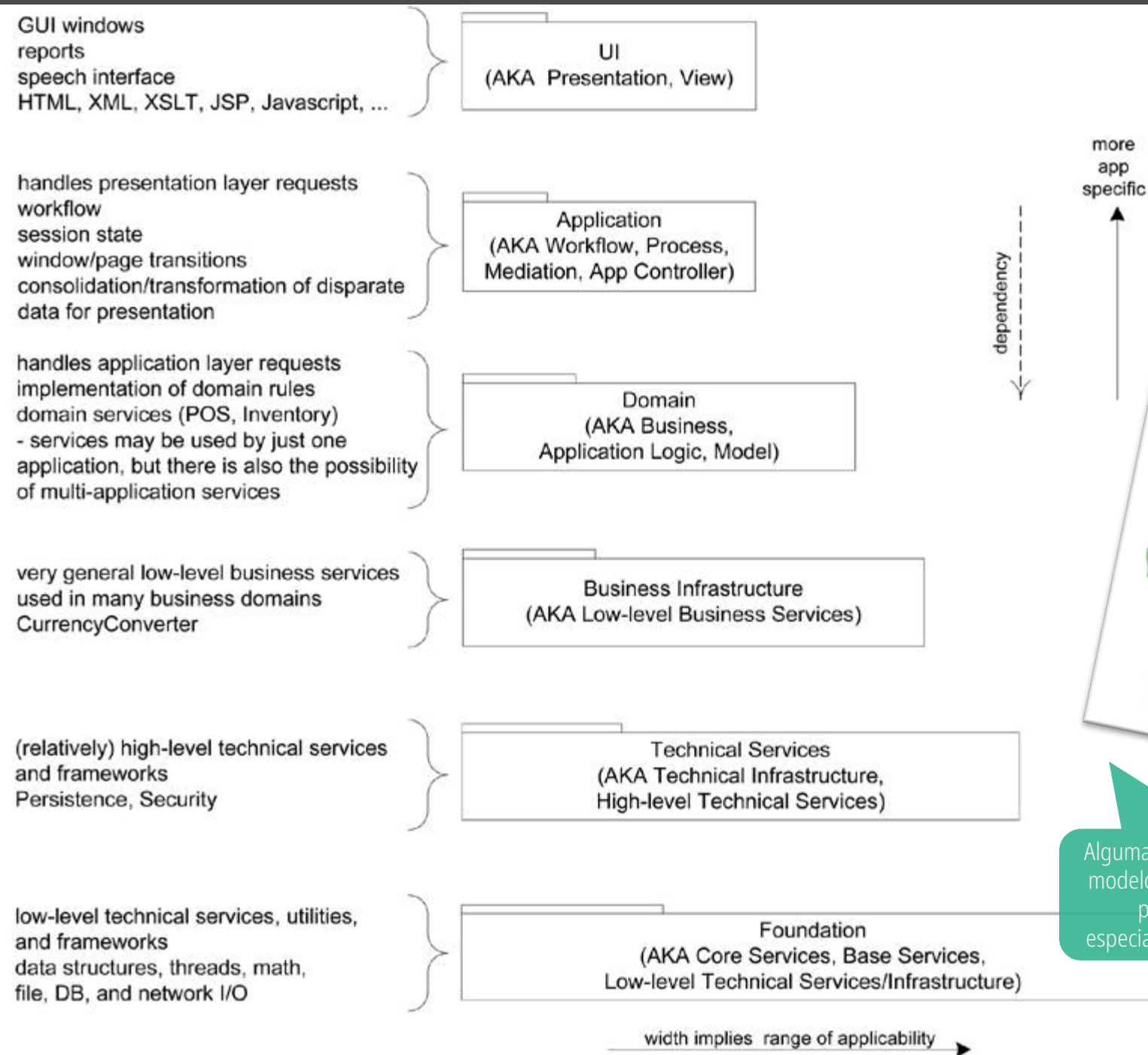
- divisão por níveis de abstração

Partições horizontais:

- Módulos dentro de uma camada

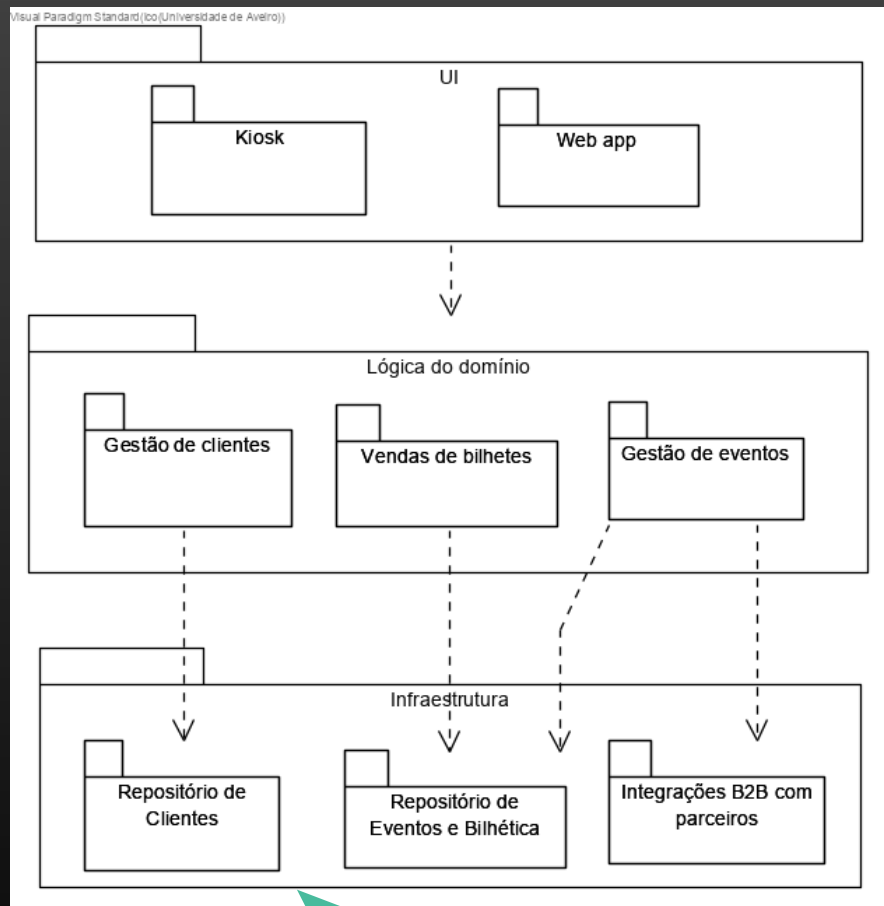
E.g.: a lógica do domínio está dividida em grandes módulos funcionais especializados, agrupando as programação relativa às Vendas, aos Pagamentos e à Fiscalidade.





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

Considerar o uso de arquiteturas lógicas nos projetos de grupo.



Opção I: vista puramente lógica, sem elementos de implementação. Destaca a modularização esperada do sistema.

I Oliveira

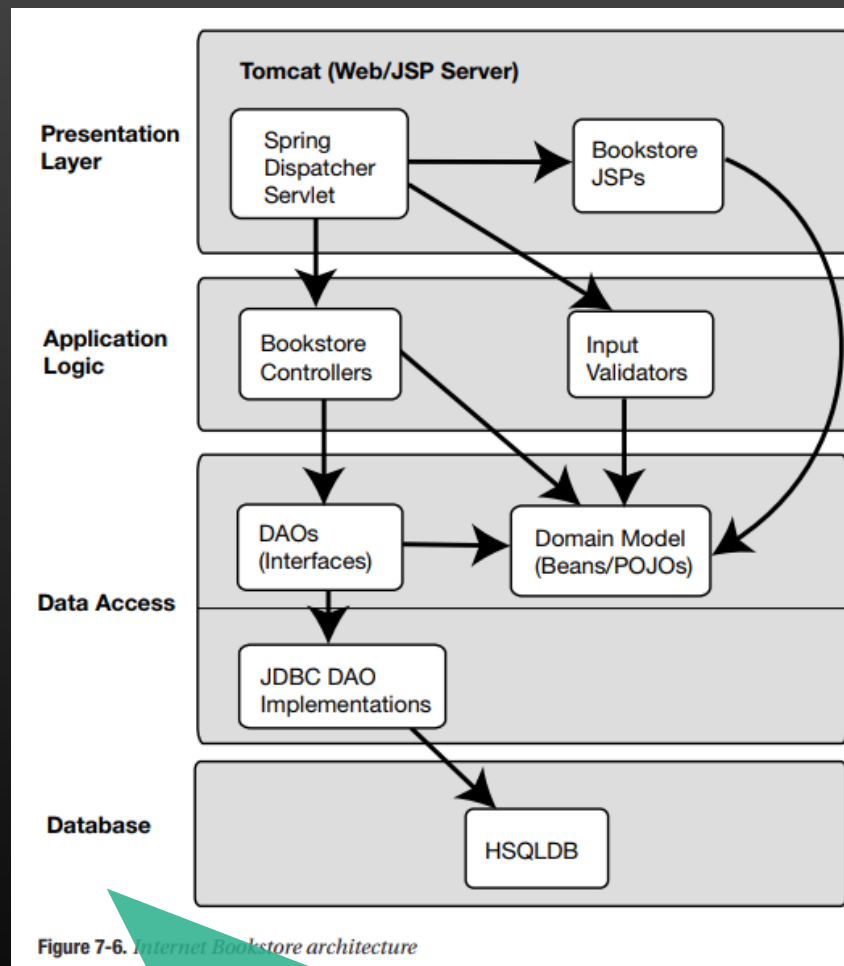


Figure 7-6. Internet Bookstore architecture

Opção II: *stack* de módulos, mostra os componentes da aplicação tendo em conta os elementos da implementação. Explica a forma como as tecnologias/ambientes/bibliotecas serão usadas na construção da "full stack".

Voltando ao início: arquitetura no openUP

Elaboration: saber como construir, construindo uma parte

Importante mitigar riscos técnicos

Definir a arquitetura do Sistema,
implementar uma parte (→ arquitetura executável)

Validar a arquitetura

Construir “esqueletos” para os componentes principais e começar a sua integração.

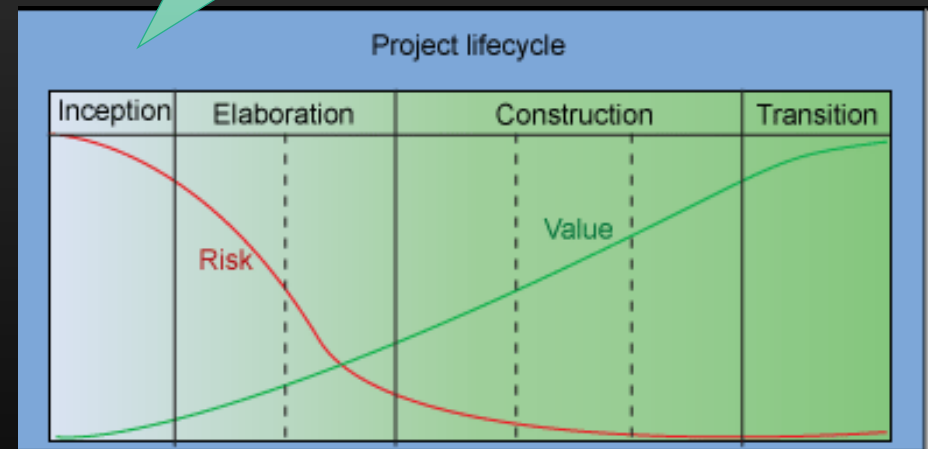
Identificar dependências de sistemas e componentes externos e especificar como serão integrados.

~10% do código será implementado

Basear a arquitetura nos casos de utilização nucleares

20% dos casos de uso determinam 80% da arquitetura

Controlar os riscos técnicos, aprofundando os requisitos e desenvolvendo a arquitetura / plano técnico. Comprovar a arquitetura implementando uma parte.



Credit: Per Kroll (IBM)

OpenUP practice: evolutionary architecture

Practices > Technical Practices > Evolutionary Architecture

Practice: Evolutionary Architecture



Analyze the major technical concerns that affect the system, and capture the architectural decisions to ensure that those decisions are assessed and communicated.

Analisar as principais preocupações técnicas; recolher decisões de arquitetura; avaliar, documentar e comunicar decisões.

Relationships

Content References

- How to adopt the Evolutionary Architecture
- Key Concepts
 - Architectural Mechanism
 - Architectural Views and Viewpoints
 - **Software Architecture**
- Architecture Notebook
- Envision the Architecture
- Refine the Architecture
- Guidance
 - Guidelines
 - Abstract Away Complexity
 - Modeling the Architecture
 - Software Reuse

Inputs

- [\[Technical Design\]](#)

Porque é que é “evolutiva”?

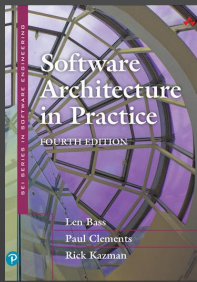
No OpenUP:

- Analisar as principais questões técnicas que afetam a solução
- Documentar decisões de arquitetura para garantir que foram avaliadas e comunicadas
- Implementar e testar capacidades-chave como forma de lidar com os desafios de arquitetura
- Evoluir ao longo do tempo, a par com o trabalho de implementação “normal”

Ideia de fundo:

a escolha da arquitetura comporta riscos que devem ser controlados cedo, experimentando as capacidades-chave.

A arquitectura de software é importante! Razões técnicas e não técnicas



Uma arquitectura irá limitar ou potenciar os atributos de qualidade de um sistema.

As decisões tomadas numa arquitectura permitem pensar e gerir a mudança à medida que o sistema evolui.

A análise de uma arquitectura permite uma visão antecipada das qualidades de um sistema.

Uma arquitectura documentada promove a comunicação entre as pessoas envolvidas (stakeholders).

A arquitectura é precursora das primeiras (e mais fundamentais) decisões de projecto, e as mais difíceis de alterar.

Uma arquitectura pode fornecer a base para o desenvolvimento incremental.

Uma arquitectura é o artefacto chave que permite ao arquitecto e ao gestor do projecto argumentarem sobre custos e prazos.

Uma arquitectura pode ser criada como um modelo transferível e reutilizável que forma o cerne de uma linha de produtos.

O desenvolvimento baseado na arquitectura centra a atenção na conjugação de componentes, em vez de simplesmente na sua criação.

Ao restringir alternativas de desenho, a arquitectura canaliza a criatividade dos programadores de forma produtiva.

Uma arquitectura pode servir de base para a formação de um novo membro da equipa.

Referências

Core readings	Suggested readings
<ul style="list-style-type: none">• [Larman04] – Chap. 13• [Dennis15] – Chap. 7 & 11	<ul style="list-style-type: none">• Bass, Clements, Kazman, "Software Architecture in Practice", 4th ed.• Fowler, "Software Architecture Guide"