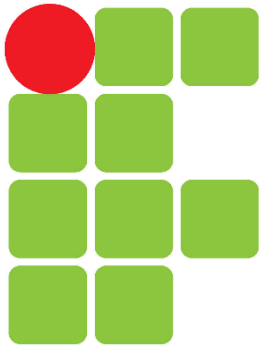
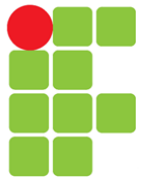


INSTITUTO FEDERAL
SÃO PAULO



Introdução a componentes de software

Desenvolvimento de Componentes (BRADECO)

Prof. Luiz Gustavo Diniz de Oliveira Véras

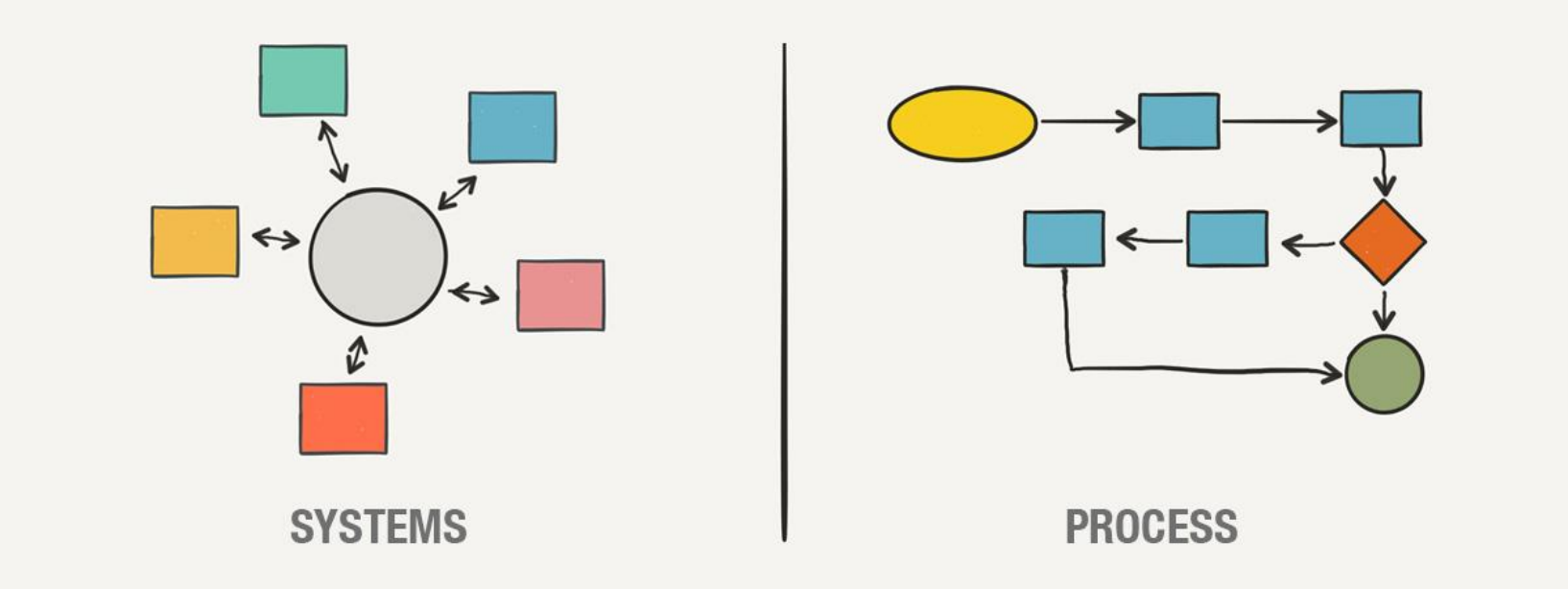
E-mail: gustavo_veras@ifsp.edu.br

Objetivos

- ✓Sistemas Distribuídos
- ✓Engenharia orientada à reuso
- ✓Definição de componentes
- ✓Notações de especificação de componentes
- ✓Exercícios

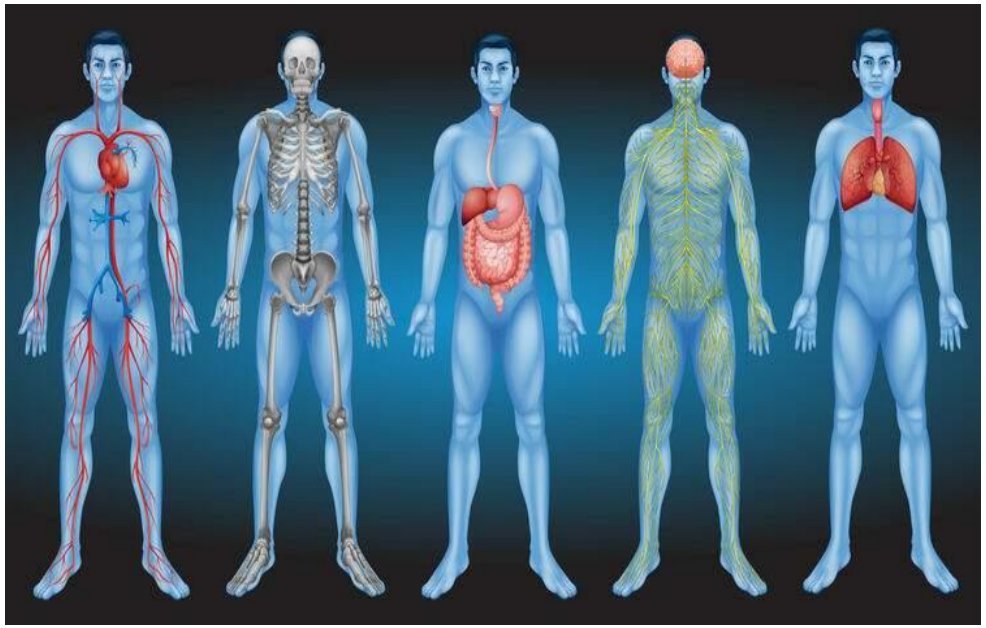
Sistemas Distribuídos

Um sistema é uma coleção intencional de ***componentes*** *inter-relacionados, de diferentes tipos, que funcionam em conjunto para atingir um objetivo* (SOMMERVILE, 2011).



Sistemas Distribuídos

Analogia*:* O corpo humano – Dividido em subsistemas ou processos que realizam tarefas especializas e estão integrados à outros subsistemas.



Fonte: https://www.educamaisbrasil.com.br/enem/biologia/sistemas-do-corpo-humano

Subsistema (Processo)	Papel
Sistema Nervoso	Controla e coordena funções corporais, processa informações e responde a estímulos.
Sistema Circulatório	Transporta oxigênio, nutrientes e hormônios pelo corpo, além de remover resíduos metabólicos.
Sistema Respiratório	Permite a troca de gases (oxigênio e dióxido de carbono) entre o corpo e o ambiente.
Sistema Digestivo	Processa os alimentos, absorve nutrientes e elimina resíduos sólidos.
Sistema Endócrino	Regula funções corporais por meio de hormônios produzidos por glândulas.
Sistema Muscular	Permite movimentos, mantém a postura e gera calor.
Sistema Esquelético	Dá suporte estrutural, protege órgãos e produz células sanguíneas.
Sistema Urinário	Filtra o sangue, remove resíduos e regula o equilíbrio hídrico e eletrolítico.
Sistema Imunológico	Defende o corpo contra patógenos e substâncias estranhas.
Sistema Reprodutivo	Permite a reprodução e a continuidade da espécie.
Sistema Tegumentar	Protege o corpo, regula a temperatura e atua como barreira contra patógenos.

Sistemas Distribuídos

O termo ‘sistema’ é usado universalmente. Falamos de:

- sistemas de computador,
- sistemas operacionais,
- sistemas de pagamento,
- sistema de ensino,
- Sistemas sociais,
- sistema de governo,
- Sistemas termodinâmico,
- Sistema elétricos,
- Etc.

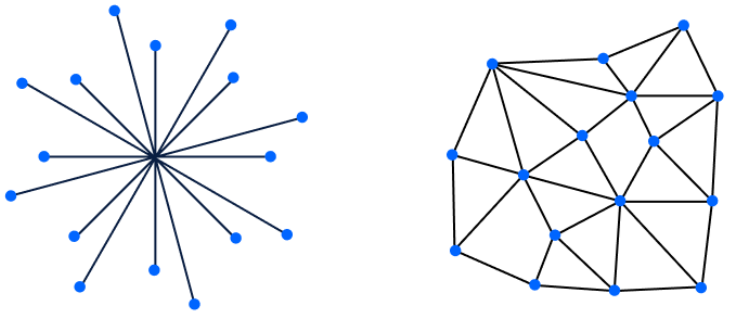
Sistemas Distribuídos

Praticamente todos os grandes sistemas computacionais agora são sistemas distribuídos. Um **sistema distribuído** é um sistema que **envolve vários computadores**, em contraste com sistemas centralizados, em que todos os **componentes** do sistema executam **em um único computador**. Tanenbaum e Van Steen (2007) definem um sistema distribuído como:

... uma coleção de computadores independentes que aparece para o usuário como um único sistema coerente.

Sistemas Distribuídos

Nos sistemas distribuídos outras questões de engenharia de software surgem porque os componentes do sistema podem ser executados em computadores gerenciados de forma independente e porque eles se comunicam por meio de uma rede.



Fonte: https://www.atlassian.com/br/microservices/microservices-architecture/distributed-architecture

Sistemas Distríbuidos

Algumas vantagens da Abordagem Distribuída de Desenvolvimento de Sistemas segundo Coulouris et al. (2005)

1. **Compartilhamento de Recursos**: Permite compartilhar hardware e software, como discos, impressoras, arquivos e compiladores em uma rede.
2. **Abertura**: Projetado para protocolos-padrão (interoperabilidade), permitindo a combinação de equipamentos e softwares de diferentes fornecedores.
3. **Concorrência**: Vários processos podem operar simultaneamente em computadores separados na rede e realizar comunicação entre si.
4. **Escalabilidade**: Recursos do sistema podem ser ampliados com a adição de novos componentes.
5. **Tolerância a Defeitos**: Redução do impacto de falhas por meio da redundância de informação ou recursos. Perda total do serviço ocorre apenas em caso de falha de rede completa.

Sistemas Distríbuidos

Os sistemas distribuídos são, inerentemente, **mais complexos que os sistemas centralizados**, o que traz alguns desafios (SOMMERVILLE, 2011) :

- São mais difíceis para projetar, implementar e testar.
- É mais difícil compreender suas propriedades emergentes por causa da complexidade das interações entre os componentes do sistema e sua infraestrutura.
- Depende de muitos fatores, como largura da banda de rede, da carga de rede e da velocidade de todos os computadores que fazem parte do sistema.
- Mover os recursos de uma parte do sistema para outra pode afetar significativamente o desempenho do sistema.

Sistemas Distríbuidos

Os sistem **e os sistemas**

- São m
- É mais causa da compl a e sua infrae
- Deper a carga de rede e te do sistem
- Move fetar signifi

O mais importante desenvolvimento que tem afetado os sistemas de software distribuído, nos últimos anos, é a **abordagem orientada a serviços** e os **microserviços**.

Veremos mais sobre eles na segunda parte desta disciplina.

Eng. de Software orientada à reuso

A engenharia de software baseada em reúso é uma estratégia da engenharia de software em que o processo de desenvolvimento é orientado para o reúso de softwares existentes.

Motivação:

O reúso tem sido promovido para aumentar o retorno sobre os investimentos em software.

Eng. de Software orientada à reuso

A disponibilidade de softwares reusáveis tem aumentado significativamente.

- O movimento *open source* significa que existe uma enorme base de código reusável disponível a baixos custos. Isso pode dar-se na forma de bibliotecas de programas ou aplicações inteiras.
- Existem muitos sistemas de aplicação de domínios específicos disponíveis, os quais podem ser customizados e adaptados às necessidades de uma empresa específica.
- Algumas grandes empresas fornecem uma variedade de componentes reusáveis para seus clientes. Padrões, como os de web service, tornaram mais fácil o desenvolvimento de serviços gerais e reúso destes em uma variedade de aplicações.

Eng. de Software orientada à reuso

As unidades de software reusadas podem ser de tamanhos radicalmente diferentes.

- **Reúso de sistema de aplicação.** A totalidade de um sistema de aplicação pode ser reusada sem alterações em outros sistemas ou pela configuração da aplicação para diferentes clientes. Como alternativa, podem ser desenvolvidas famílias de aplicações com uma arquitetura comum, mas adaptadas para clientes específicos.
- **Reúso de componentes.** Os componentes de uma aplicação, variando em tamanho desde subsistemas até objetos únicos, podem ser reusados. Por exemplo, um sistema de identificação de padrões desenvolvido como parte de um sistema de processamento de textos pode ser reusado em um sistema de gerenciamento de banco de dados.
- **Reúso de objetos e funções.** Componentes de software que implementam uma única função, como uma função matemática ou uma classe de objeto, podem ser reusados. Essa forma de reúso, baseada em bibliotecas-padrão, tem sido comum nos últimos 40 anos. Muitas bibliotecas de funções e classes estão disponíveis gratuitamente. Você reusa as classes e funções nessas bibliotecas, ligando-as com o código da aplicação recém-desenvolvido. Essa é uma abordagem particularmente eficaz em áreas como algoritmos e gráficos matemáticos, em que o conheci mento especializado é necessário para o desenvolvimento de funções e objetos eficientes.

Eng. de Software orientada à reuso

Uma forma complementar de reúso é o ‘reúso de conceito’, em que, em vez de reusar um componente de software, você reusa uma ideia, uma forma, um trabalho ou um algoritmo.

O conceito reusado é representado em uma notação abstrata (por exemplo, um modelo de sistema), que inclui detalhes de implementação.

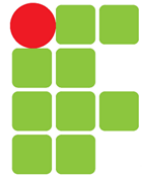
Pode, portanto, ser configurado e adaptado para uma série de situações.

Conceitos de reuso podem ser:

- padrões de projeto
- Padrões e estilos arquiteturais
- geradores de programa.

Quando conceitos são reusados, o processo de reúso inclui atividades nas quais os conceitos abstratos são instanciados para criar componentes reusáveis executáveis.

Eng. de Software orientada à reuso



Vantagens do reuso de software (SOMMERVILLE, 2011)

Benefício	Explicação
Confiança aumentada	Os softwares reusados, experimentados e testados em sistemas em funcionamento provavelmente são mais confiáveis do que um novo software pois seus defeitos já foram corrigidos.
Risco de processo reduzido	O custo do software existente já é conhecido, considerando que os custos de desenvolvimento são sempre uma questão que envolve julgamento. Esse é um importante fator para o gerenciamento de projeto, pois reduz a margem de erro de estimativa de custos de projeto, o que é particularmente verdadeiro quando componentes de software relativamente grandes, como subsistemas, são reusados.
Uso eficaz de especialistas	Em vez de repetir o mesmo trabalho, especialistas em aplicações podem desenvolver softwares reusáveis que encapsulem seu conhecimento.
Conformidade com padrões	Alguns padrões, como os de interface de usuário, podem ser implementados como um conjunto de componentes reusáveis. Exemplos: os menus em uma interface de usuário.
Desenvolvimento acelerado	Muitas vezes, os custos gerais de desenvolvimento não são tão importantes quanto entregar um sistema ao mercado o mais rápido possível. O reúso de um software pode acelerar a produção do sistema, pois pode reduzir o tempo de desenvolvimento e validação.

Eng. de Software orientada à reuso

Dificuldades com reuso de software (SOMMERVILLE, 2011)

Problema	Explicação
Maiores custos de manutenção	Se o código-fonte de um sistema ou componentes de software reusáveis não estiverem disponíveis, os custos de manutenção podem ser maiores, pois os elementos reusados do sistema podem tornar-se cada vez mais incompatíveis com as alterações do sistema.
Falta de ferramentas de suporte	Algumas ferramentas de software não suportam o desenvolvimento com reúso. Pode ser difícil ou impossível integrar essas ferramentas com um sistema de biblioteca de componentes.
Síndrome de 'não-inventado-aqui'	Alguns engenheiros de software preferem reescrever componentes, pois acreditam poder melhorá-los. Isso tem a ver, parcialmente, com aumentar a confiança e, parcialmente, com o fato de que escrever softwares originais é considerado mais desafiador do que reusar softwares de outras pessoas.
Criação, manutenção e uso de uma biblioteca de componentes	Preencher uma biblioteca de componentes reusáveis e garantir que desenvolvedores de software possam utilizar essa biblioteca podem ser ações caras. Processos de desenvolvimento precisam ser adaptados para garantir que a biblioteca seja usada.
Encontrar, compreender e adaptar os componentes reusáveis	Componentes de software precisam ser compreendidos em uma biblioteca, e, às vezes, adaptados para trabalhar em um novo ambiente. Os engenheiros precisam estar confiantes de que encontrarão, na biblioteca, um componente, antes de incluírem a pesquisa de componente como parte de seu processo normal de desenvolvimento.

Eng. de Software orientada à reuso

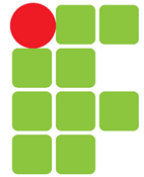
Frameworks

É uma das peças-chave na abordagem de desenvolvimento orientada à reuso**,** dão suporte ao reúso de projeto, bem como ao reúso de classes específicas de sistema, pois fornecem uma arquitetura de esqueleto para a aplicação.

Como o nome sugere, um *framework* é uma estrutura genérica estendida para se criar uma aplicação ou sub sistema mais específico. Schmidt et al. (2004) definem um framework como:

“... um conjunto integrado de artefatos de software (como classes, objetos e componentes) que colaboram para fornecer uma arquitetura reusável para uma família de aplicações relacionadas.”

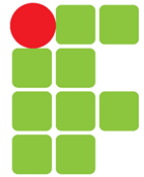
Eng. de Software orientada à reuso



Exemplos de frameworks

Framework	Enfoque
Spring (Java)	Desenvolvimento de aplicações empresariais com suporte a injeção de dependência, segurança, transações e integração com bancos de dados.
Django (Python)	Desenvolvimento web rápido e seguro, seguindo o padrão Model-View-Template (MVT).
Ruby on Rails (Ruby)	Desenvolvimento de aplicações web com convenções que facilitam a produtividade e boas práticas de código.
Angular (JavaScript/TypeScript)	Desenvolvimento de interfaces dinâmicas de aplicações web single-page (SPA).
Vue.js (JavaScript)	Criação de interfaces reativas e modulares para aplicações web.
React (JavaScript)	Construção de interfaces de usuário reutilizáveis e dinâmicas, especialmente para aplicações front-end.

Eng. de Software orientada à reuso



Exemplos de frameworks

Framework	Enfoque
Laravel (PHP)	Desenvolvimento de aplicações web robustas e escaláveis utilizando o padrão MVC.
Flask (Python)	Desenvolvimento web minimalista, ideal para APIs e microserviços.
.NET Framework / .NET Core (C#)	Desenvolvimento de aplicações empresariais, web, desktop e móveis com integração a serviços Microsoft.
Qt (C++)	Desenvolvimento de aplicações desktop e mobile com interface gráfica rica.
TensorFlow (Python, C++)	Desenvolvimento de aplicações de aprendizado de máquina e inteligência artificial.
Electron (JavaScript)	Desenvolvimento de aplicativos desktop usando tecnologias web.
Bootstrap (CSS, JavaScript)	Desenvolvimento rápido de interfaces responsivas para web.
Express.js (JavaScript)	Construção de APIs e servidores web com Node.js.
Flutter (Dart)	Desenvolvimento de aplicativos móveis multiplataforma com interface rica e otimizada.

Definição de componente

Muitos conceitos de componentes podem ser encontrados na literatura:

A definição mais amplamente adotada de um componente de software é a seguinte por Szyperski et al. (2002):

“Um componente de software é uma unidade de composição com interfaces contratualmente especificadas e dependências de contexto explícitas apenas. Um componente de software pode ser implantado independentemente e está sujeito à composição por terceiros.”

Definição de componente

Muitos conceitos de componentes podem ser encontrados na literatura:

Definição dada por Object Management Group apud Craig Larman no livro “Applying UML and Patterns”:

“Um componente representa uma parte modular, implantável e substituível de um sistema que encapsula a implementação e expõe um conjunto de interfaces”.

Definição de componente

Componentes representam algum elemento físico substituível.

Como ilustrado no caso do **ShoppingSession**, eles podem implementar uma interface.

Como ilustrado com **iostream.cpp**, eles podem ter dependências.
Esses exemplos ilustram estereótipos comuns e padronizados para componentes.

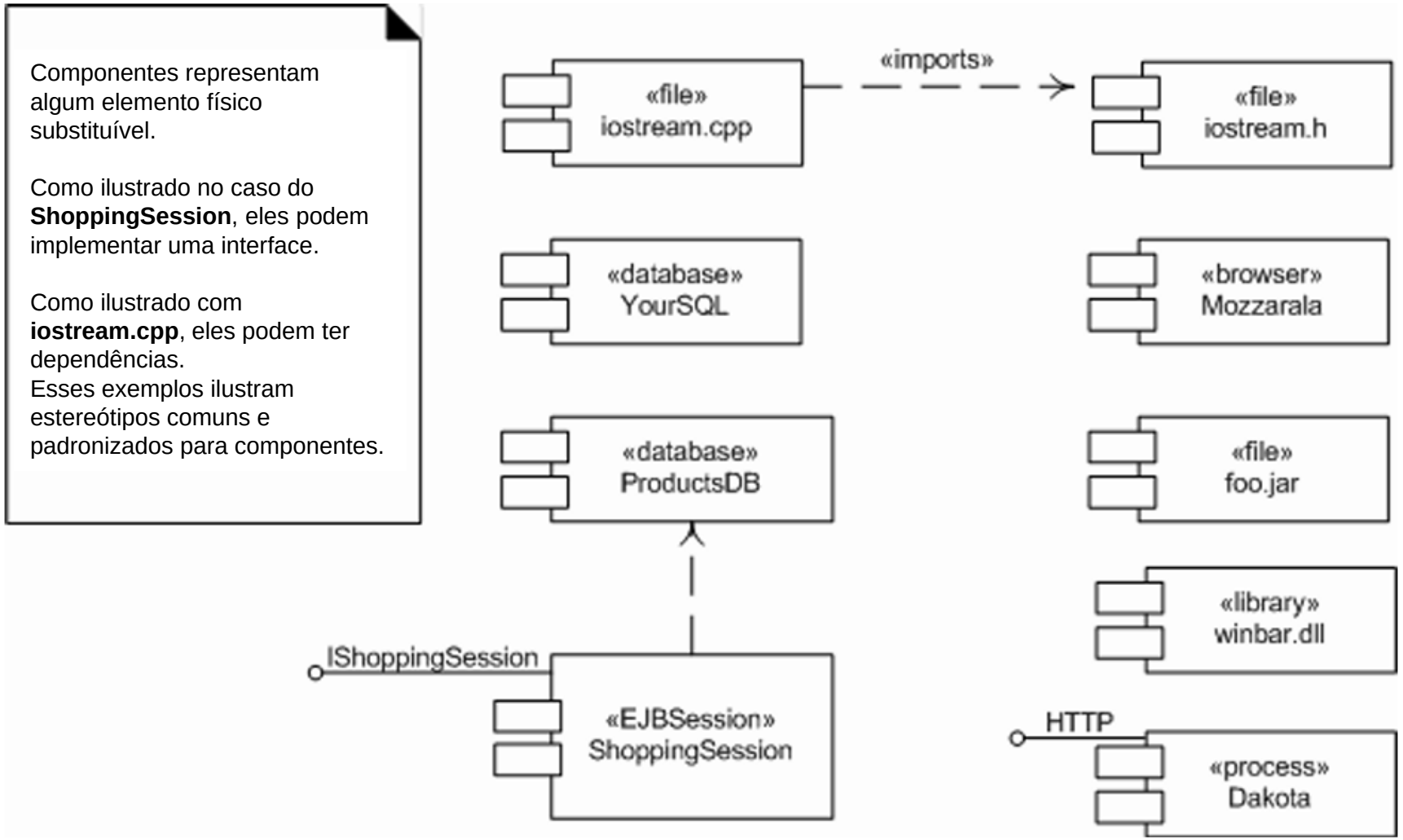


Figure 38.4 UML components.

Definição de componente

Muitos conceitos de componentes podem ser encontrados na literatura:

Definição dada por Robert C. Martin no livro “Arquitetura Limpa”

“Componentes são as unidades de implantação. Eles são as menores entidades que podem ser implantadas como parte de um sistema.”

Seu significado pode variar de acordo com cada tecnologia:

- Em Java, eles são arquivos **.jar**. Em Ruby, eles são arquivos **.gem**. No .NET, eles são **DLLs**.
- Em linguagens compiladas, eles são agregações de arquivos binários. Em linguagens interpretadas, eles são agregações de arquivos-fonte.
- Em todas as linguagens, eles são a menor unidade de implantação.
- Então o papel de componente pode ser assumido, por exemplo, um código-fonte, binário ou executável. Outrois Exemplos incluem executáveis como um navegador ou servidor HTTP, um banco de dados, uma DLL ou um arquivo JAR (como para um Enterprise Java Bean).

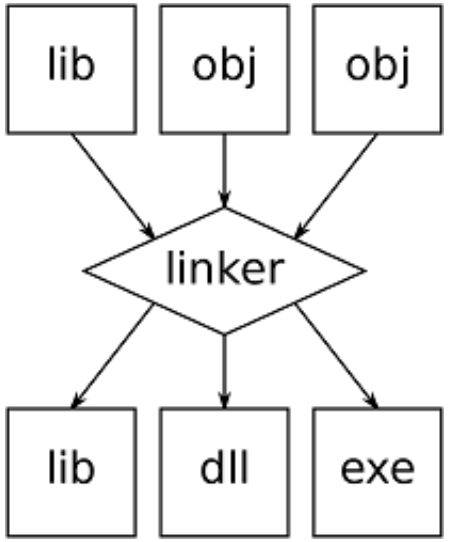
Definição de componente

Podemos construir aplicações componentizadas em dois momentos do processod de criação de um software

- Durante a sua compilação
  - Integração de componentes ao software durante a compilação, um processo que é conhecido como *linking*. É executado pelo *linker*.

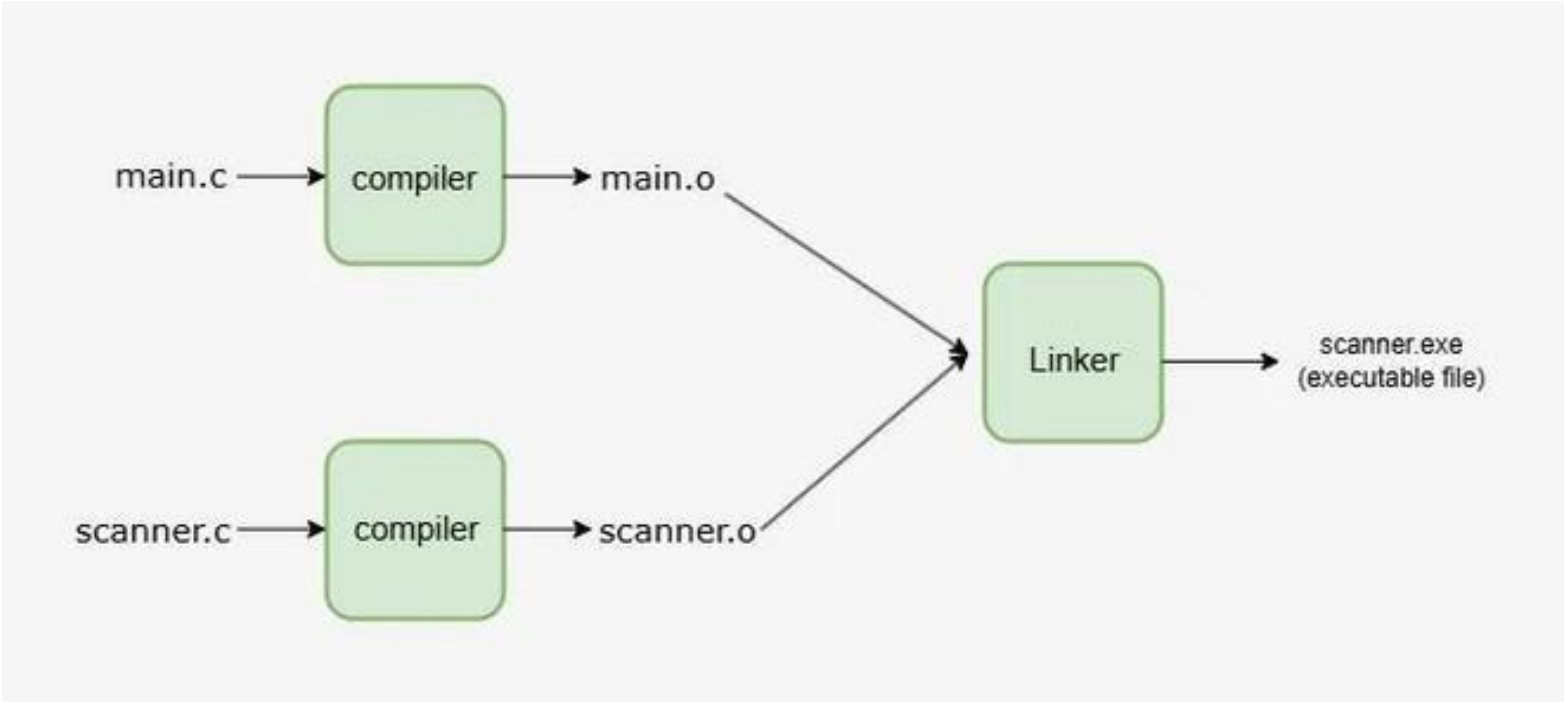
Por exemplo:

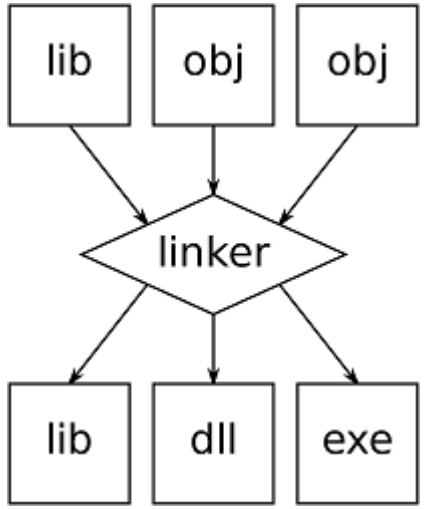
- Em C e C++, bibliotecas estáticas (.lib no Windows, .a no Linux) são vinculadas ao código-fonte durante a compilação, gerando um executável único.
- No caso de aplicações em Go, todos os pacotes importados podem ser compilados e vinculados a um binário final independente.
- No mundo do desenvolvimento de sistemas embarcados, diversos módulos de código compilado podem ser ligados para criar uma única firmware executável.



Definição de componente

Em C, os componentes são unidos por meio de um processo/ferramenta chamado *Linker.*





Definição de componente

Podemos construir aplicações componentizadas em dois momentos do processo de criação de um software

- Durante a sua execução;
  - Foi assim que nasceu a arquitetura de plugins baseada em componentes. Hoje, é comum distribuirmos **arquivos .jar**, **DLLs** ou **bibliotecas compartilhadas** como plugins para aplicações existentes.

Por exemplo:

- Se você quiser criar um *mod* para **Minecraft**, basta incluir seus arquivos **.jar** personalizados em uma pasta específica.
- Se quiser adicionar uma extensão ao **Visual Studio**, basta incluir as **DLLs** apropriadas.

Essa abordagem modular permite a extensão de funcionalidades sem modificar o código-fonte principal, tornando os sistemas mais flexíveis e personalizáveis.

Notações de especificação de componentes

O diagrama ao lado pode ser utilizado para representar componentes.

A função deste aplicativo não é importante para o propósito deste exemplo.

O que é importante é a estrutura de dependência dos componentes.

Observe que neste estrutura temos:
- um grafo direcionado
- os componentes são os nós,
- e os relacionamentos de dependência são as arestas direcionadas.

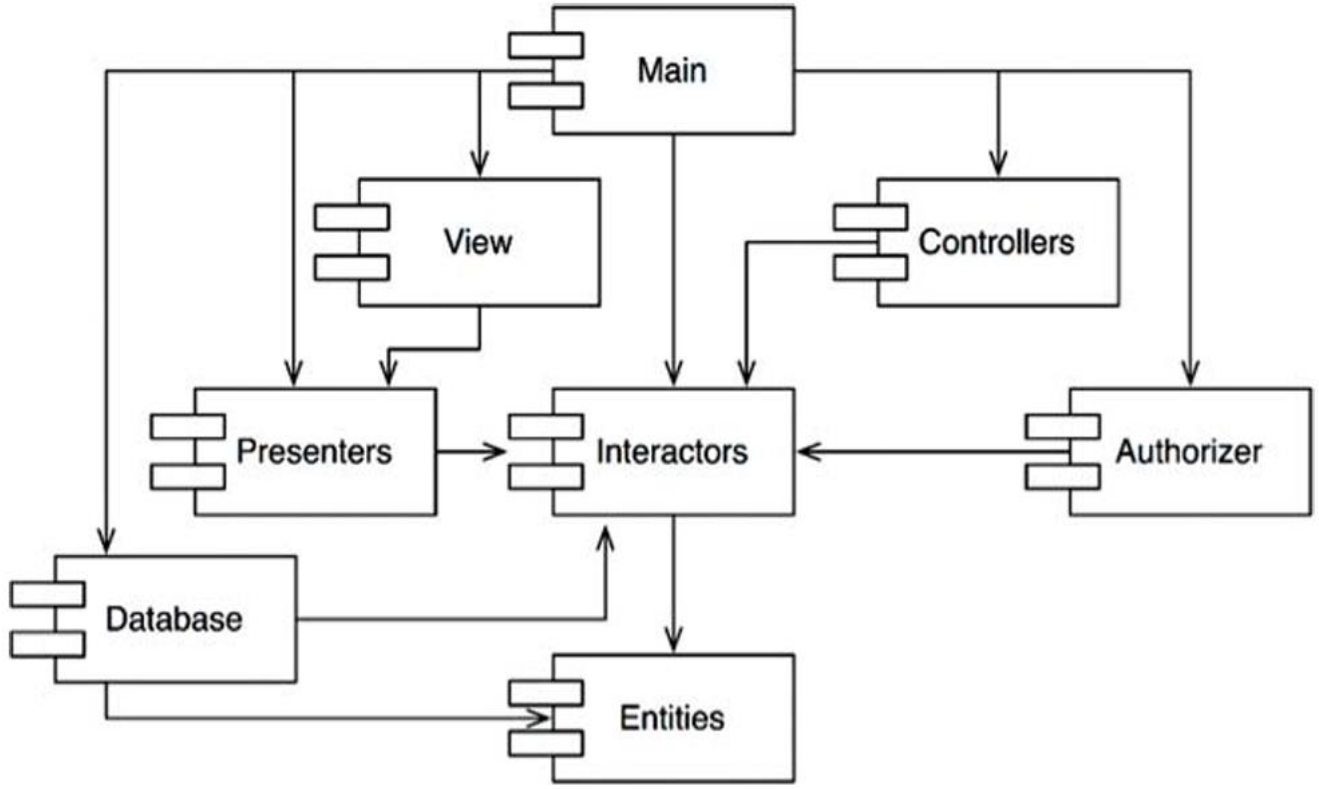


Figure 14.1 Typical component diagram

Fonte: Livro “Clean Achitecture”

Notações de especificação de componentes

Do diagrama podemos entender que:

Isolamento do Main: Quando Main é atualizado, isso não afeta nenhum outro componente, pois eles não dependem dele e não precisam se preocupar com suas mudanças. Isso reduz o impacto da liberação de Main.

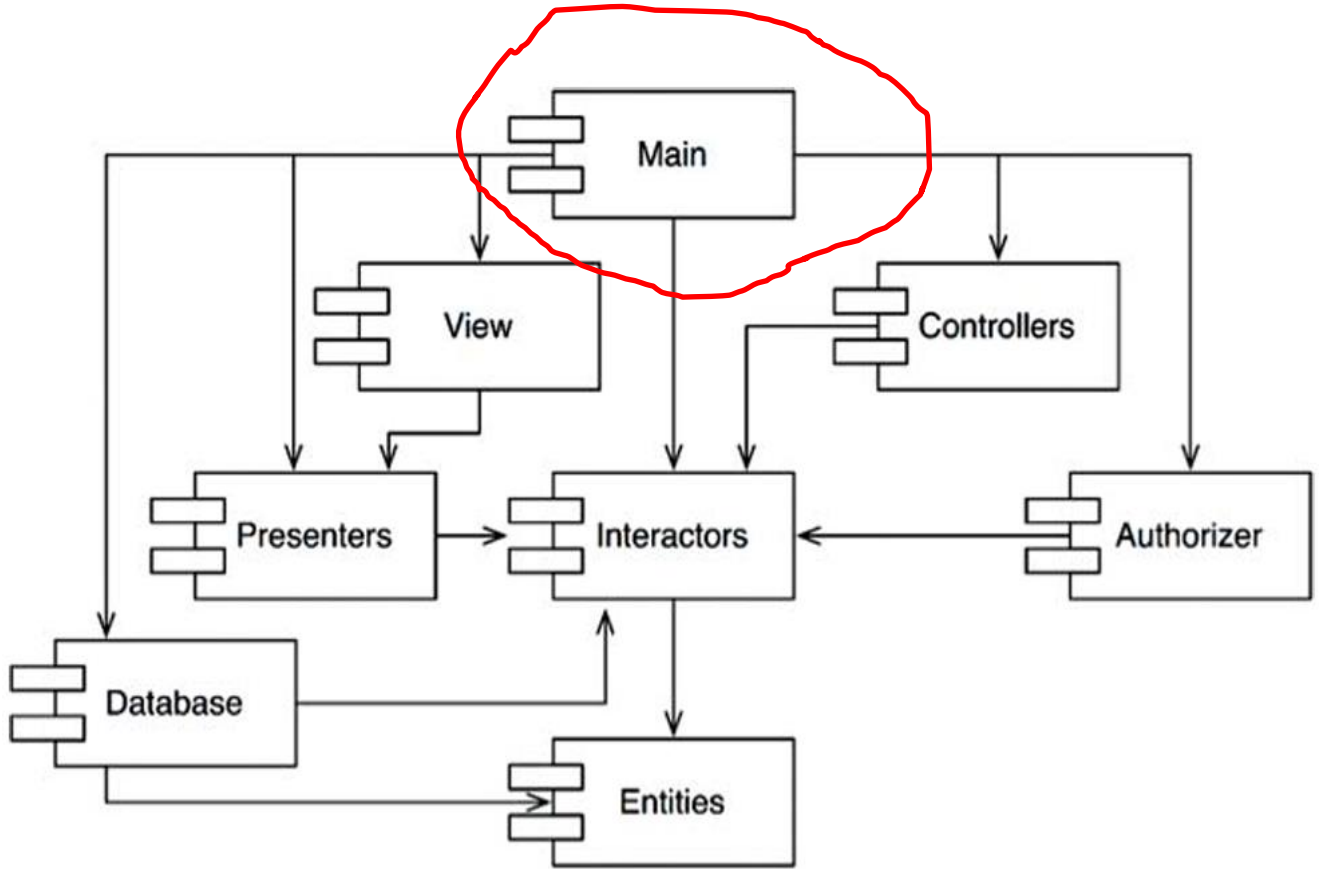


Figure 14.1 Typical component diagram

Fonte: Livro “Clean Achitecture”

Notações de especificação de componentes

Do diagrama podemos entender que:

Impacto da atualização de um componente: Quando a equipe responsável pelos **Presenters** lança uma nova versão, os componentes afetados podem ser identificados seguindo as setas de dependência para trás. No caso, **View** e **Main** serão impactados e precisarão decidir quando integrar suas versões com o novo lançamento de **Presenters**.

Os desenvolvedores de **Presenters** podem testar seu componente apenas compilando sua versão com as versões atuais de **Interactors** e **Entities**, sem envolver outros componentes. Isso simplifica os testes e reduz variáveis a serem consideradas

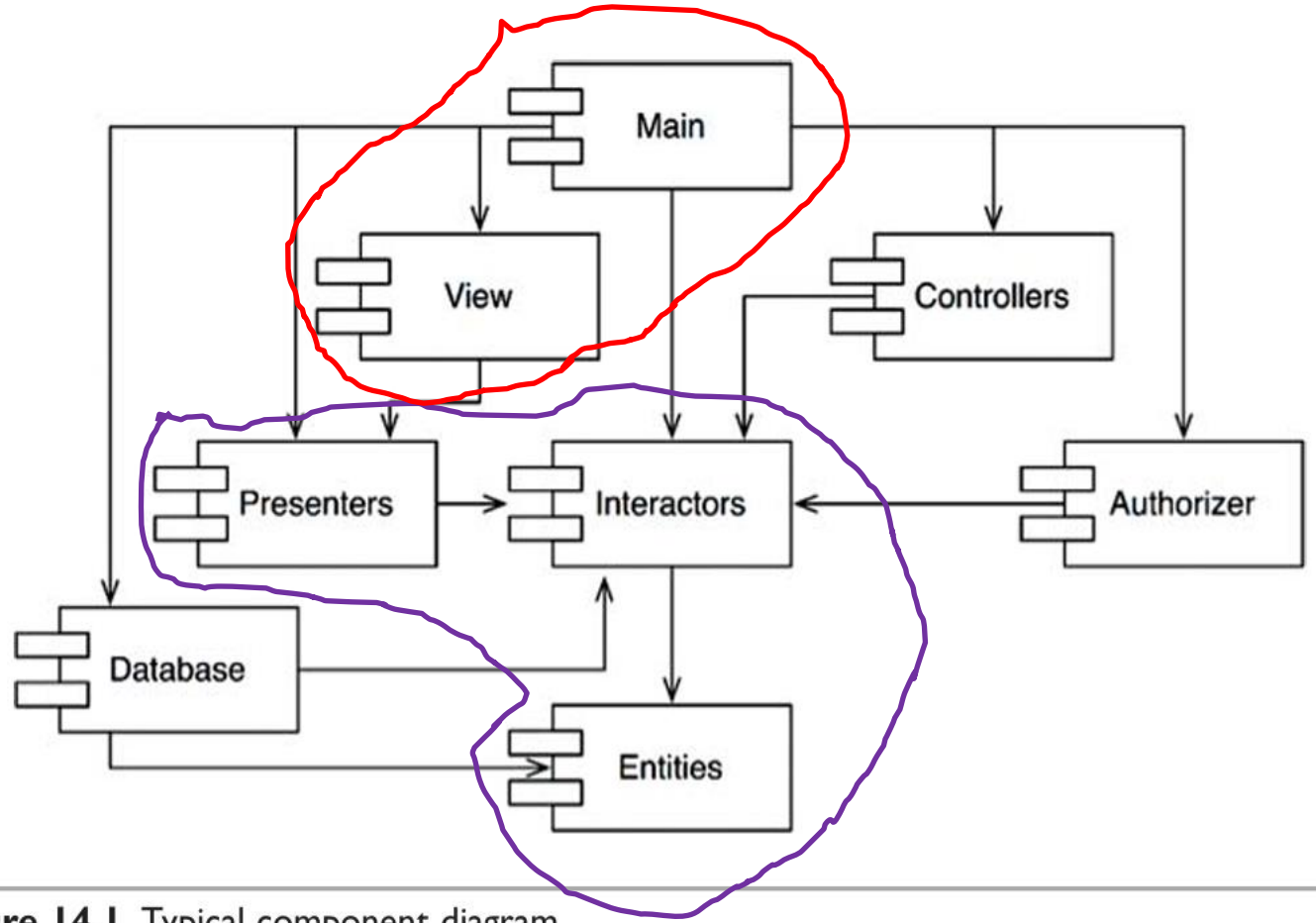


Figure 14.1 Typical component diagram

Fonte: Livro “Clean Achitecture”

Notações de especificação de componentes

Do diagrama podemos entender que:

Processo de liberação do sistema: O lançamento ocorre de forma **ascendente (do menos dependente para o mais dependente)**:

- Compilação, teste e liberação do **Entities**.
- Em seguida, **Interactors e Database**.
- Depois, **Presenters, View, Controllers e Authorizer**.
- **Main** é o último a ser lançado.

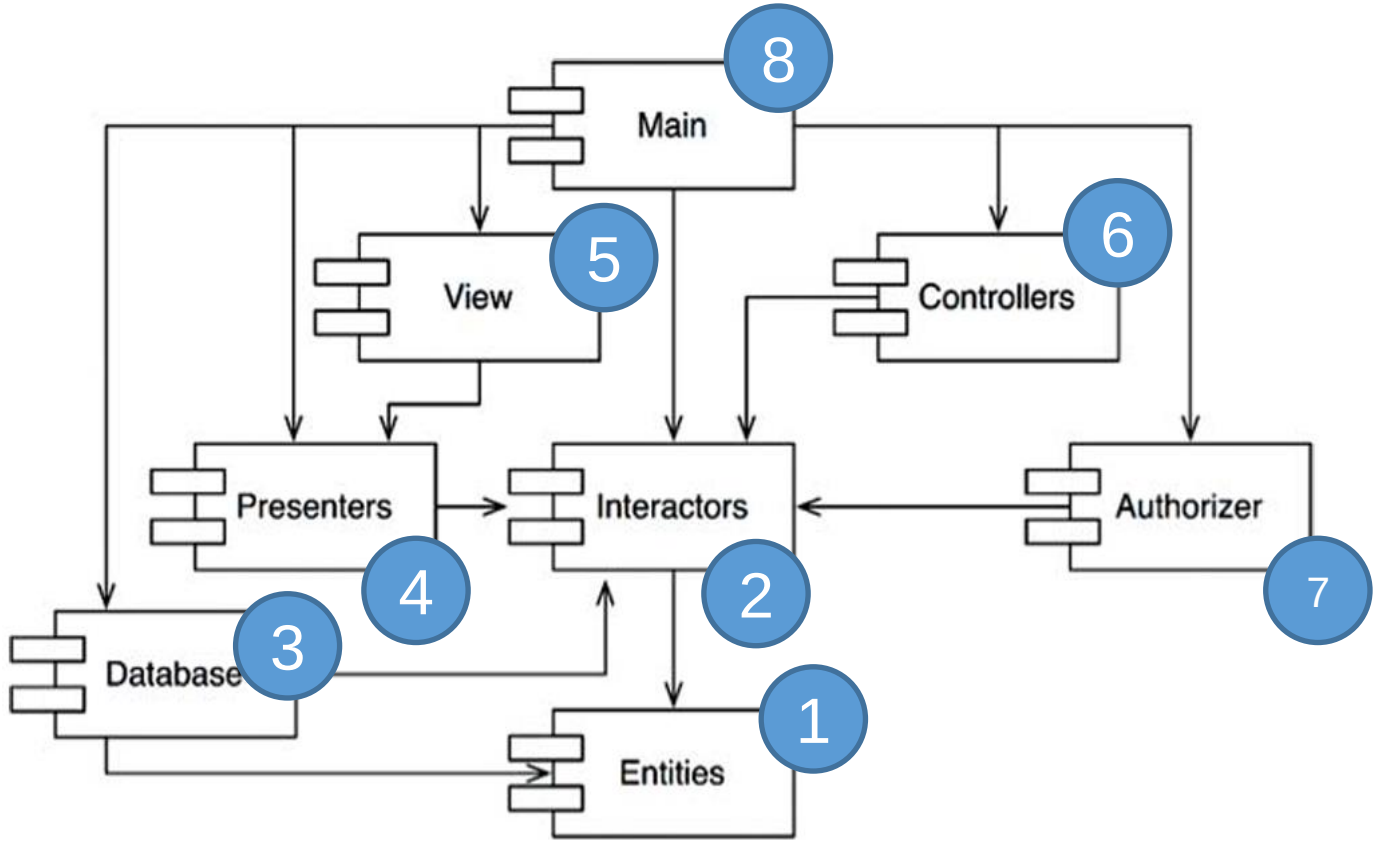


Figure 14.1 Typical component diagram

Fonte: Livro “Clean Achitecture”

Notações de especificação de componentes

Resultados!

Claridade na construção do sistema: O processo é bem definido porque as dependências entre os componentes são compreendidas, tornando o desenvolvimento mais previsível e organizado.

Estrutura sem ciclos: Independentemente de qual componente seja iniciado, não é possível seguir as relações de dependência e voltar ao mesmo componente. Isso forma um **grafo acíclico direcionado (DAG). Isso é o que se deseja alcançar na modelagem entre os componentes!**

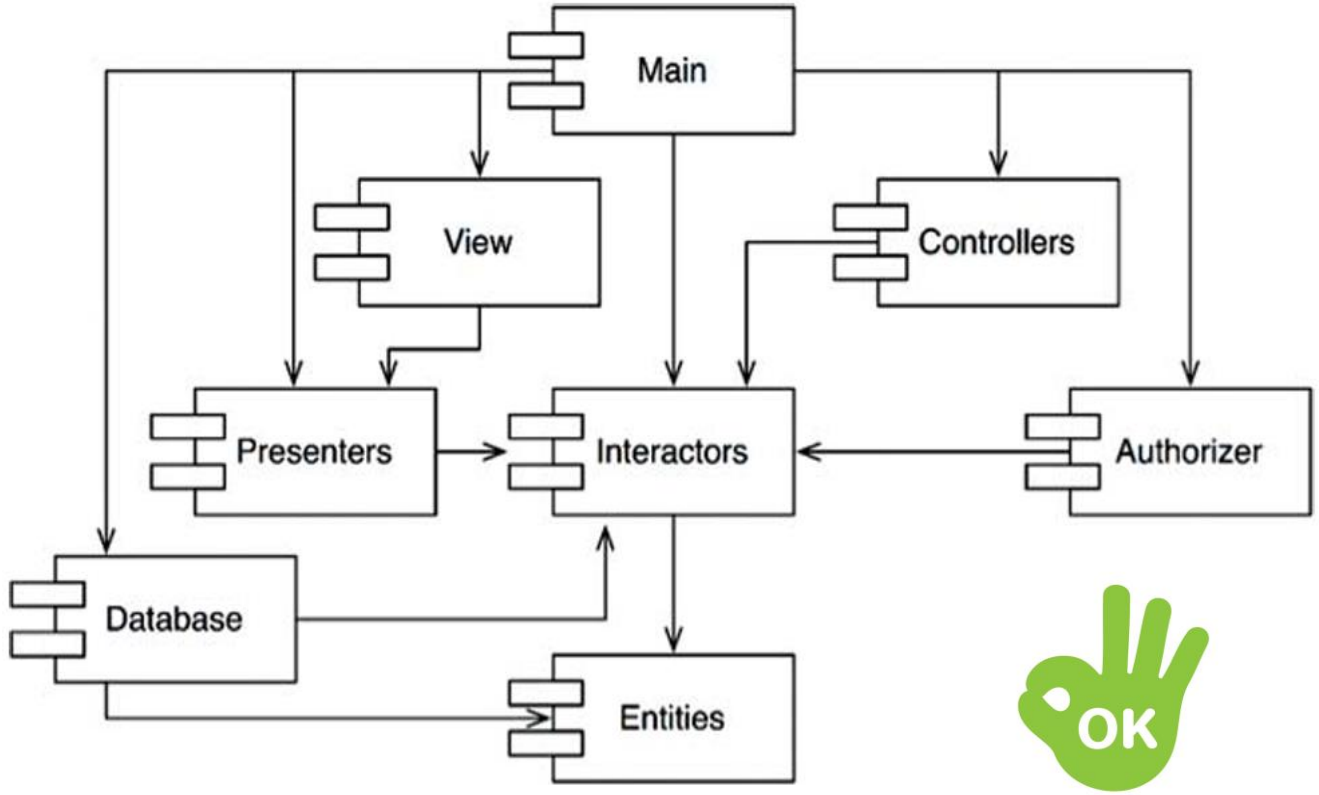


Figure 14.1 Typical component diagram

Fonte: Livro “Clean Achitecture”

Notações de especificação de componentes

Para criarmos análises como as do diagrama anterior, precisamos inicialmente conhecer uma notação para especificação de componentes. Precisamos portanto de um modelo de componentes para descrever tanto os componentes quanto sua composição.

Um modelo de componentes de software é uma definição que inclui:

- **Semântica dos componentes**: O que os componentes representam.
- **Sintaxe dos componentes**: Como os componentes são definidos, construídos e representados.
- **Composição dos componentes**: Como os componentes são combinados ou montados.

Notações de especificação de componentes

Modelo de componentes: UML 2.0

um componente é uma unidade arquitetônica com portas de entrada que são serviços necessários e portas de saída que são serviços fornecidos

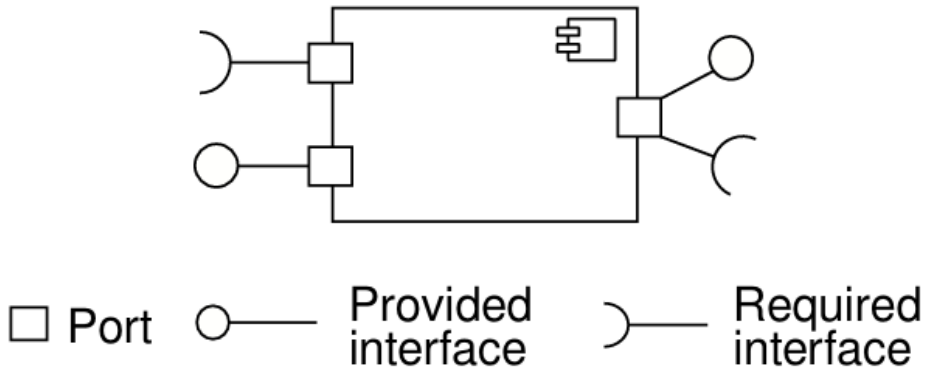


Fig. 4.5 UML2.0 component.

Notações de especificação de componentes

Modelo de componentes: UML 2.0

Conceitos:

Componente: Representa uma parte modular e autossuficiente de um sistema. Pode ser um software, um módulo ou uma biblioteca. Geralmente, é uma implementação que oferece serviços para outros componentes.

Portas: São pontos de comunicação entre o componente e seus ambientes externos, muitas vezes associadas a interfaces. Representadas como pequenos quadrados nas bordas do componente.

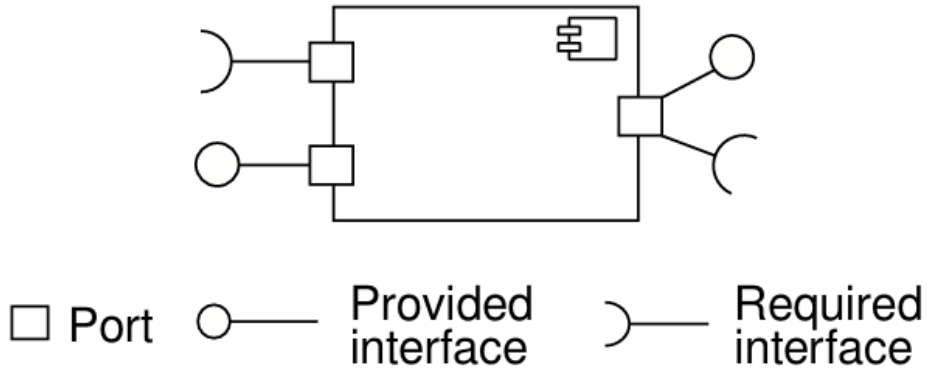


Fig. 4.5 UML2.0 component.

Notações de especificação de componentes

Modelo de componentes: UML 2.0

Conceitos:

Portas: São pontos de comunicação entre o componente e seus ambientes externos, muitas vezes associadas a interfaces. Representadas como pequenos quadrados nas bordas do componente.

Provided Interface: Uma interface que é oferecida por um componente para ser utilizada por outros componentes. Representado por um círculo vinculada a uma aresta ligada ao corpo do componente (formato de “pirulito”).

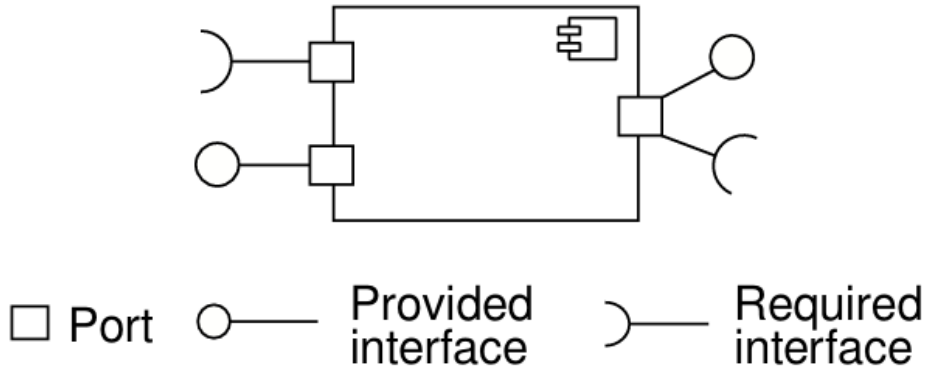


Fig. 4.5 UML2.0 component.

Notações de especificação de componentes

Modelo de componentes: UML 2.0

Conceitos:

Required Interface: Uma interface que um componente necessita e que será fornecida por outro componente. Representada por um semicírculo ligado ao corpo do componente por uma aresta.

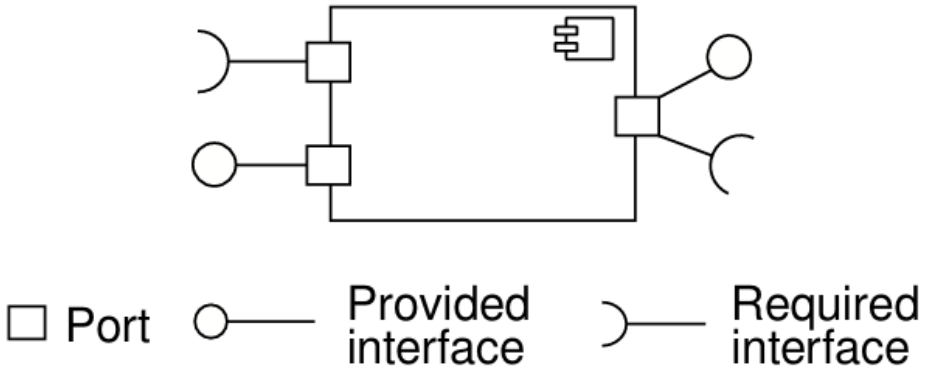
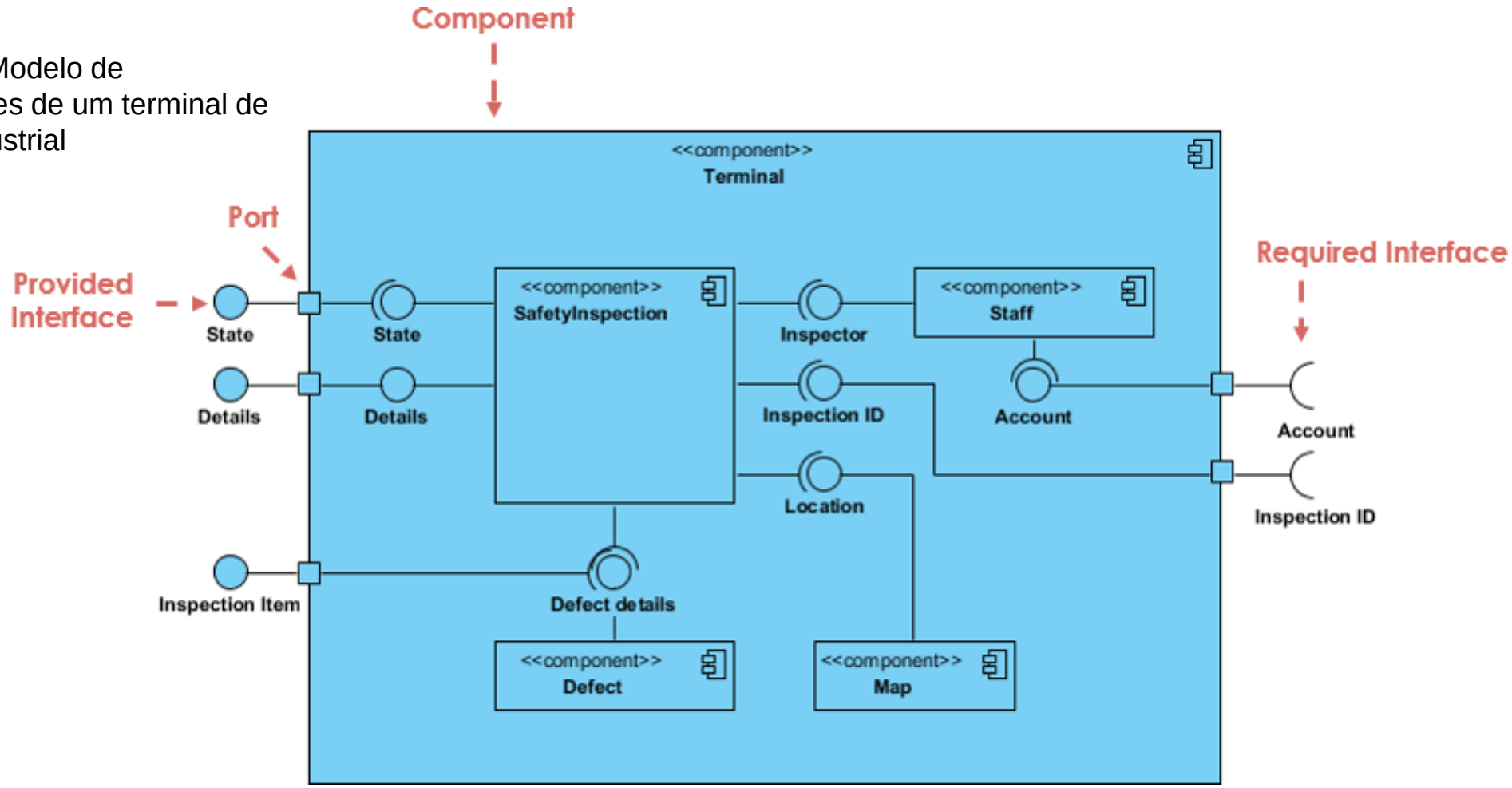


Fig. 4.5 UML2.0 component.

Notações de especificação de componentes

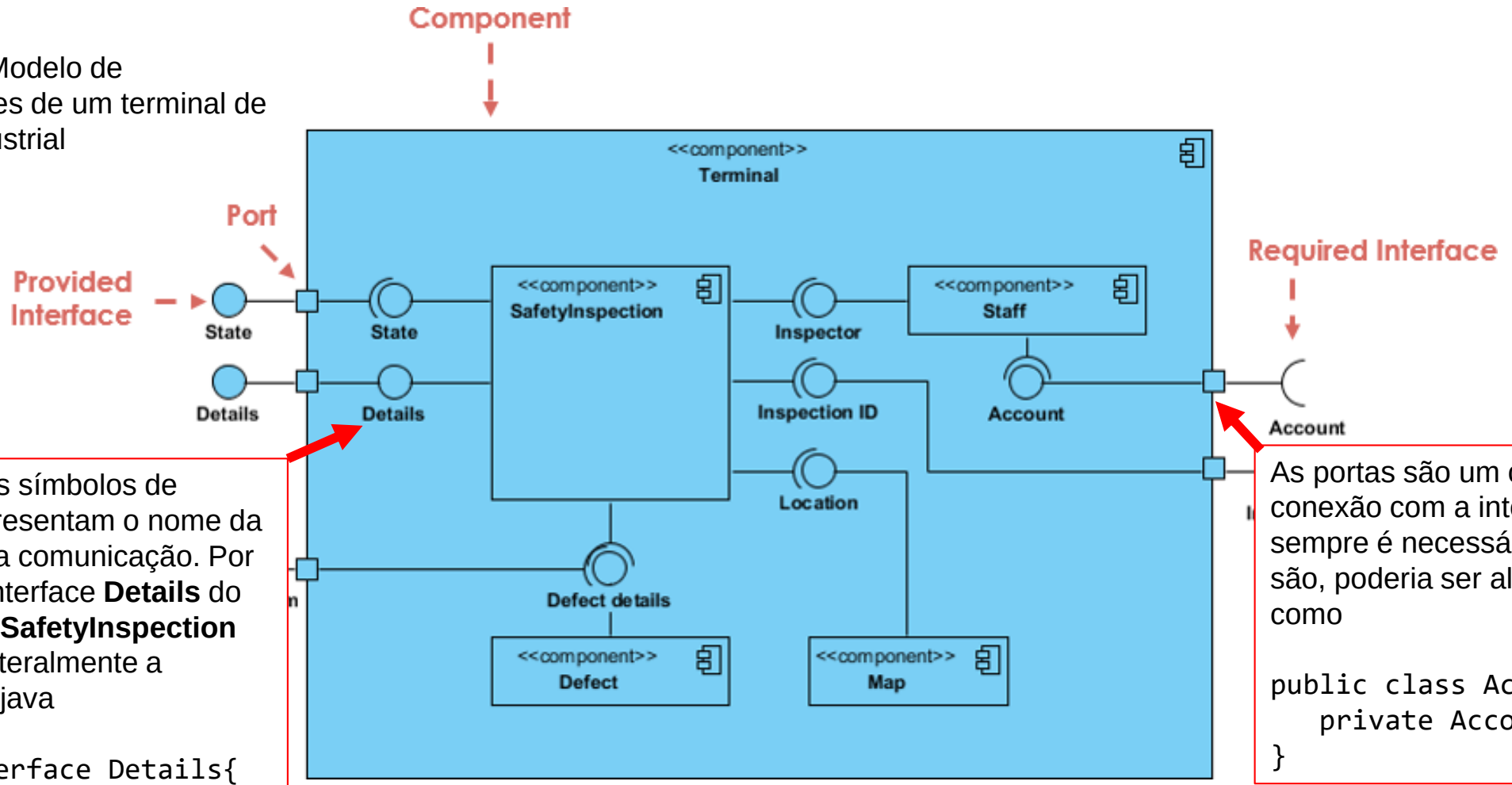
Exemplo: Modelo de componentes de um terminal de acesso industrial



Fonte: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/

Notações de especificação de componentes

Exemplo: Modelo de componentes de um terminal de acesso industrial



Os textos nos símbolos de conexão representam o nome da interface para comunicação. Por exemplo, a interface **Details** do componente **SafetyInspection** poderia ser literalmente a interface em java

```
public interface Details{
   // métodos abstratos
}
```

As portas são um código de conexão com a interface. Nem sempre é necessário. Mas se são, poderia ser algo em Java como

```
public class AccountPort{
   private Account account;
}
```

Fonte: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/

Exercícios

Atividade 1:

Um sistema de biblioteca online permite que usuários cadastrados realizem empréstimos de livros. O sistema é composto por um módulo de autenticação, que verifica o login dos usuários, e um módulo de catálogo, que exibe os livros disponíveis e registra os empréstimos. Há também um módulo de notificações, responsável por enviar lembretes aos usuários sobre a devolução dos livros. Modele esse sistema em UML, representando os principais componentes e suas interações.

Ferramentas: Lucidchart, Draw.io, Visual Paradigm, StarUML

Exercícios

Atividade 2:

Um sistema de pedidos para um restaurante online é formado por três partes principais: o módulo de cardápio, que lista os itens disponíveis para pedido; o módulo de pedidos, que processa as solicitações dos clientes e calcula o valor total; e o módulo de pagamento, que integra com serviços externos para confirmar a transação. Modele esse sistema em UML, destacando os componentes e suas conexões

Ferramentas: Lucidchart, Draw.io, Visual Paradigm, StarUML

Fontes

MARTIN, Robert C. Clean architecture [em linha]. 12 set. 2017.

SOMMERVILLE, Ian. Engenharia de Software, 9. edição. **Pearson, Addison Wesley**, v. 8, n. 9, p. 10, 2011.

ARRINGTON, C. T. **Enterprise Java with UML**. John Wiley & Sons, 2002.

LARMAN, Craig. **Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development**. Pearson Education India, 2012.

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/