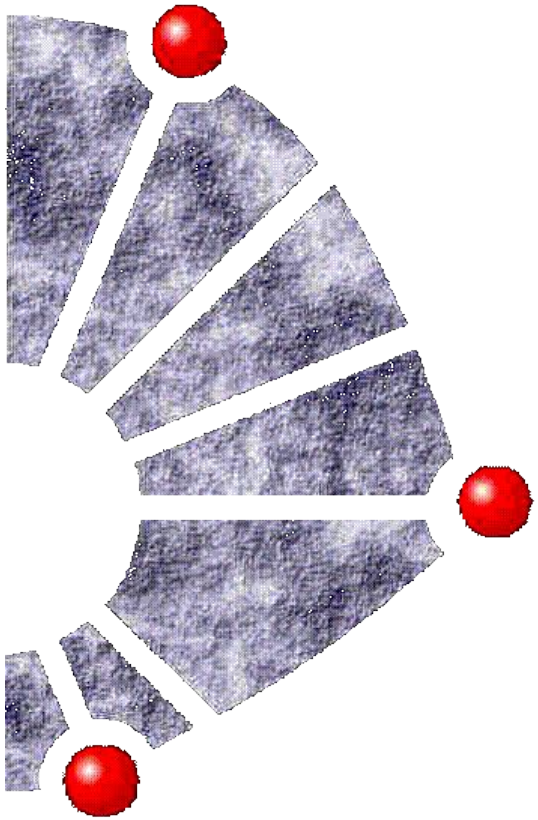
A Linguagem UML



FEEC-UNICAMP

Ricardo Gudwin

A Linguagem UML

- A Linguagem UML (Unified Modeling Language)
  - Linguagem de Modelagem
    - Modelos de Processos do Mundo Real
    - Modelos de Processos em Arquiteturas de Software
  - utilizada tanto para a análise dos elementos ontológicos participantes de um processo como do comportamento destes elementos no processo
  - provê elementos para modelar todas as etapas do processo de desenvolvimento de um software
  - Convergência de diversas outras linguagens de modelagem utilizadas em diferentes processos de desenvolvimento de software
  - Linguagem Visual, baseada em diferentes tipos de diagramas

Características do UML

- Mecanismos de Extensão
  - estereótipos, tagged values e restrições
- Threads e Processos
- Distribuição e Concorrência
- Patterns e Colaborações
- Diagramas de Atividades
  - para a modelagem de processos no mundo real
- Refinamento
  - para descrever relacionamentos entre diferentes níveis de abstração
- Interfaces e Componentes
- Linguagem de Restrição

História do UML

- Linguagens de Modelagem
  - Começaram a aparecer no final dos anos 70
    - experimentos em diferentes abordagens orientadas a objeto
- Diversas Técnicas influenciaram estas linguagens
  - Modelos Entidade/Relacionamento
  - SDL (Specification & Description Language)
- Número de linguagens identificadas
  - passou de pouco mais de 10 para mais de 50 até 1994
  - Guerra dos Métodos
    - Booch, OMT, Fusion, OOSE, OMT-2
- Desenvolvimento do UML
  - começou no final de 1994, quando Booch e Rumbaugh passaram a trabalhar em conjunto

História do UML

- Versão Preliminar do UML (versão 0.8)
  - outubro de 1995, quando Jacobson se une ao grupo
- A partir dos esforços de Booch, Rumbaugh e Jacobson
  - versão 0.91 (outubro de 1996), liberada para a comunidade
- Uma RFP (Request for Proposal) foi realizada pela OMG
  - buscando contribuições da comunidade para o estabelecimento de uma linguagem unificada
  - diversas contribuições lançaram o UML 1.0
    - Digital Equipment Corp., HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI e Unisys.
  - Em janeiro 1997, novas contribuições lançaram o UML 1.1
    - IBM & ObjecTime, Platinum Technology, Ptech, Taskon & Reich Technologies e Softeam

História do UML

- A Partir da Versão 1.1
  - comunidade de desenvolvimento de software faz uma aderência maciça ao UML
- Em novembro 1997
  - O UML 1.1 foi adotado como norma pela OMG
  - OMG estabeleceu um RTF (Revision Task Force) para aperfeiçoar pequenos detalhes na linguagem
- Em Junho 1999
  - O RTF libera a versão UML 1.3
- UML 1.4
  - na iminência de ser liberado (previsão: novembro 2000)
- UML 2.0
  - RTF da OMG já está trabalhando para coletar propostas

Elementos de Modelagem

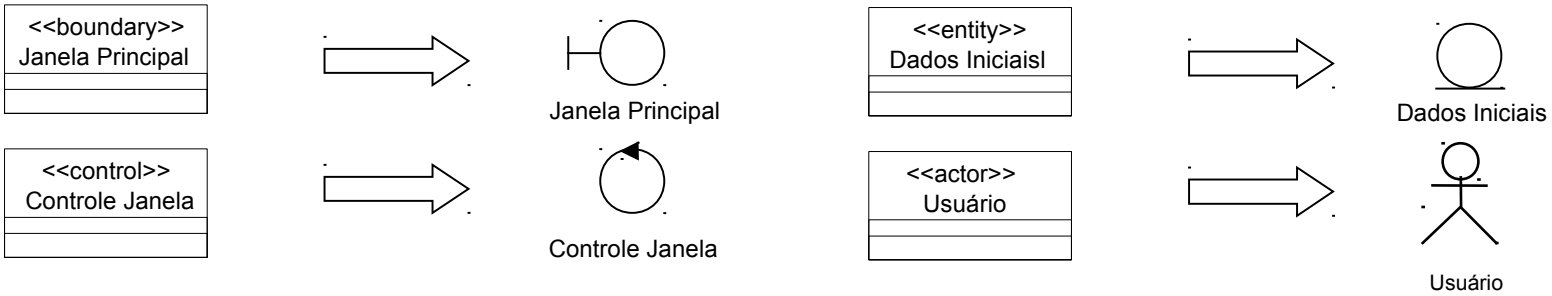
- Diagramas
  - Grafos contendo nós, caminhos entre os nós e textos
  - **Nós** podem ser símbolos gráficos (figuras 2D) complexos e estruturados
  - **Caminhos** podem ser linhas cheias ou pontilhadas com possíveis decorações especiais nas pontas
  - **Textos** em diferentes posições têm diferentes significados
- Relacionamento Visual entre Elementos
  - modela algum tipo de relacionamento entre os elementos envolvidos
    - **conexão** (usualmente linhas entre figuras 2D)
    - **inclusão** (de símbolos ou figuras 2D dentro de figuras 2D)
    - **proximidade visual** (um símbolo estando perto de outro símbolo ou figura 2D dentro de um diagrama)

Tipos de Elementos

- Ícones
  - figura gráfica de tamanho e formato fixo que não pode ser expandido para ter algum tipo de conteúdo. Pode aparecer como participante de um **símbolo 2D**, como terminador de **caminhos** ou simplesmente isoladamente
- Símbolos 2D
  - figuras bi-dimensionais que podem ter tamanho variável e que pode ser expandido de modo a conter outros elementos, tais como listas de **Strings** ou outros símbolos. Podem ser divididos em compartimentos de tipo similar ou diferente
- Caminhos
  - sequências de linhas cujas extremidades estão conectadas a outros elementos. Podem ter terminadores.
- Strings
  - apresentam diversas informações cujo significado depende de onde aparecem.

Estereótipos

- Estereótipos
  - tipos especiais de Strings que servem para modificar a semântica de elementos de um diagrama
  - definem um novo elemento de modelagem em termos de um elemento já conhecido
  - podem ser aplicados a qualquer elemento de modelagem
  - notação é na forma «estereótipo»
  - alternativamente, podem ser utilizados símbolos gráficos diferenciados para representar esses elementos modificados

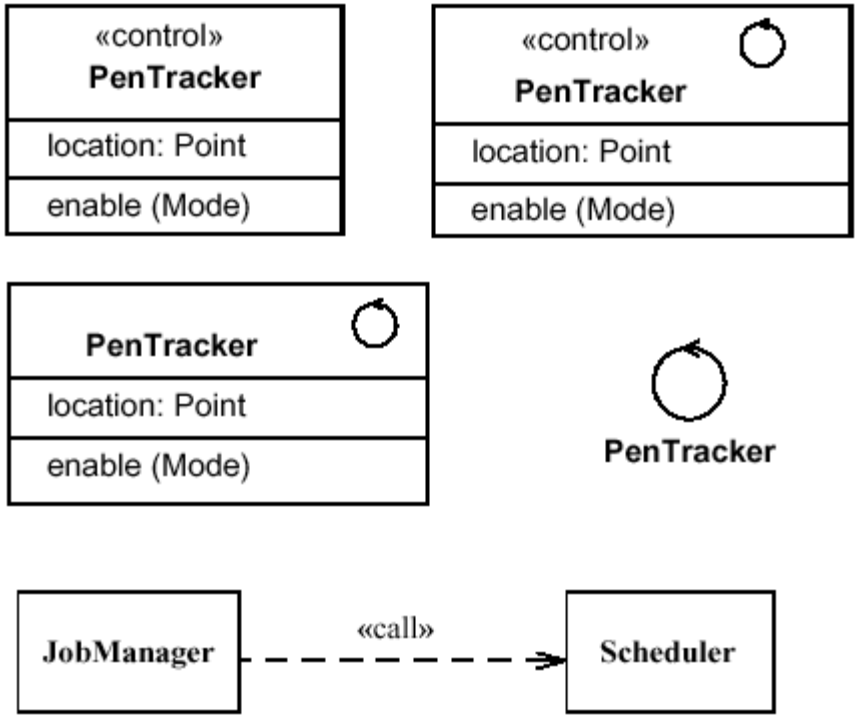


Estereótipos

- Estereótipos
  - podem ser aplicados a qualquer elemento de modelagem: classes, relacionamentos, componentes, etc
  - cada elemento UML pode ter no máximo um único estereótipo
  - alguns tipos de elementos contém um conjunto de estereótipos previamente definidos
    - e.g. <<actor>>, <<extends>>, <<includes>>, etc...
- Usos de Estereótipos
  - modificar o modo de geração de código
  - usar um ícone diferente para modelar tipos de entidades em etapas específicas do processo de modelagem
  - em qualquer caso em que uma extensão é necessária para modelar alguma característica em particular

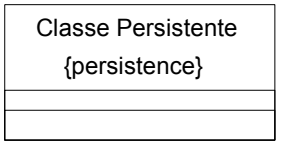
Estereótipos

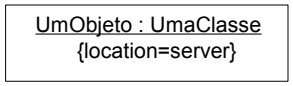
- Exemplos de Diferentes Visualizações de Estereótipos



Propriedades

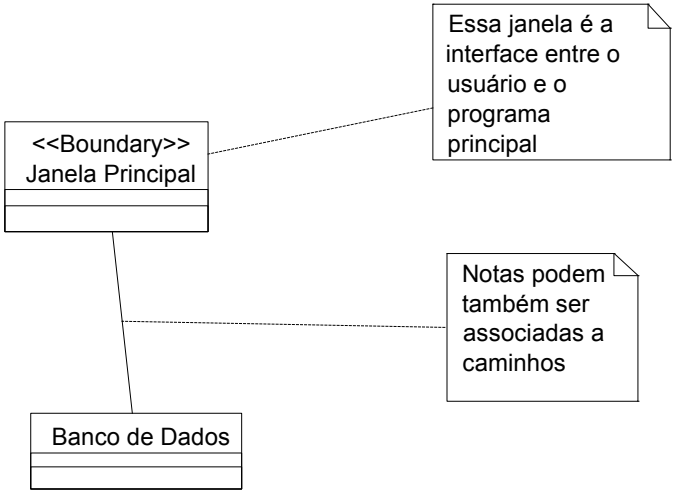
- Propriedade
  - atributo específico de um elemento UML
  - também chamada de *tagged value*
  - podem ser criadas para qualquer propósito
- Algumas propriedades
  - definidas previamente pelo UML
    - persistence, location (e.g. cliente, servidor, etc..)
- Persistence
  - pode ser aplicada a atributos, classes e objetos
  - denotam se o estado do elemento deve ser preservado quando a instância é destruída
- Location
  - pode ser aplicada a classes e componentes





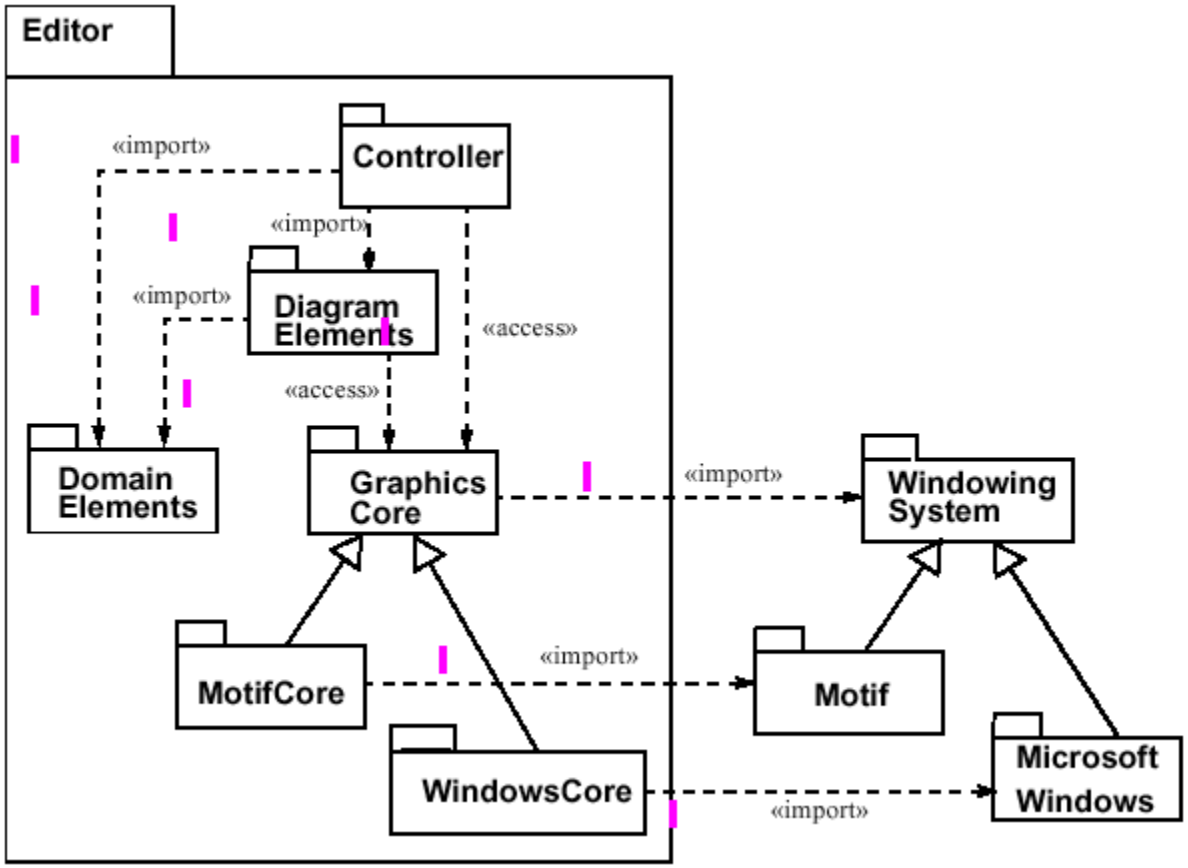
Notas

- Notas
  - descrições de texto que complementam a informação contida em algum diagrama
  - devem estar ancoradas a um elemento por meio de uma linha pontilhada



Pacotes

- Pacotes
- agrupamento de elementos de modelagem
- podem ser aninhados em outros pacotes recursivamente
- qualquer elemento UML pode ser agrupado em um pacote
- podem se relacionar a outros pacotes

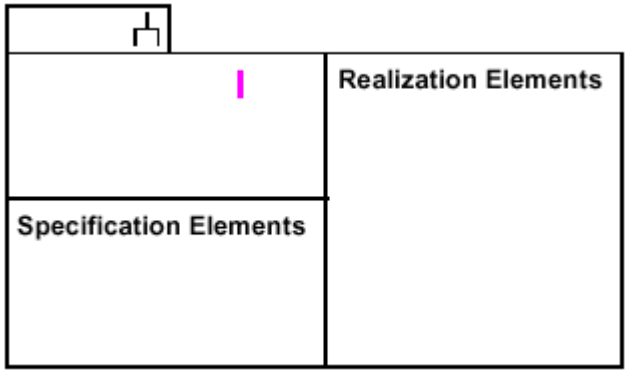


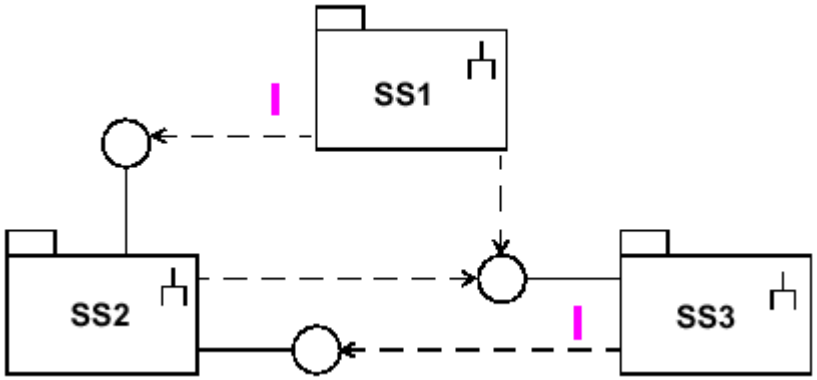
Sub-Sistema

- Sub-sistema
  - tipo de pacote específico denotado pelo estereótipo <<subsystem>>
  - representa uma unidade comportamental em um sistema físico
  - pode oferecer interfaces e ter operações
  - seu conteúdo pode ser particionado em elementos de especificação e realização
- Especificações de um Sub-sistema
  - consiste de operações sobre o sub-sistema, jnto com elementos de especificação tais como casos de uso ou máquinas de estado
- Podem ou não ser instanciáveis
  - sub-sistemas não instanciáveis servem meramente como unidades de especificação para o comportamento dos elementos nele contidos

Sub-Sistemas

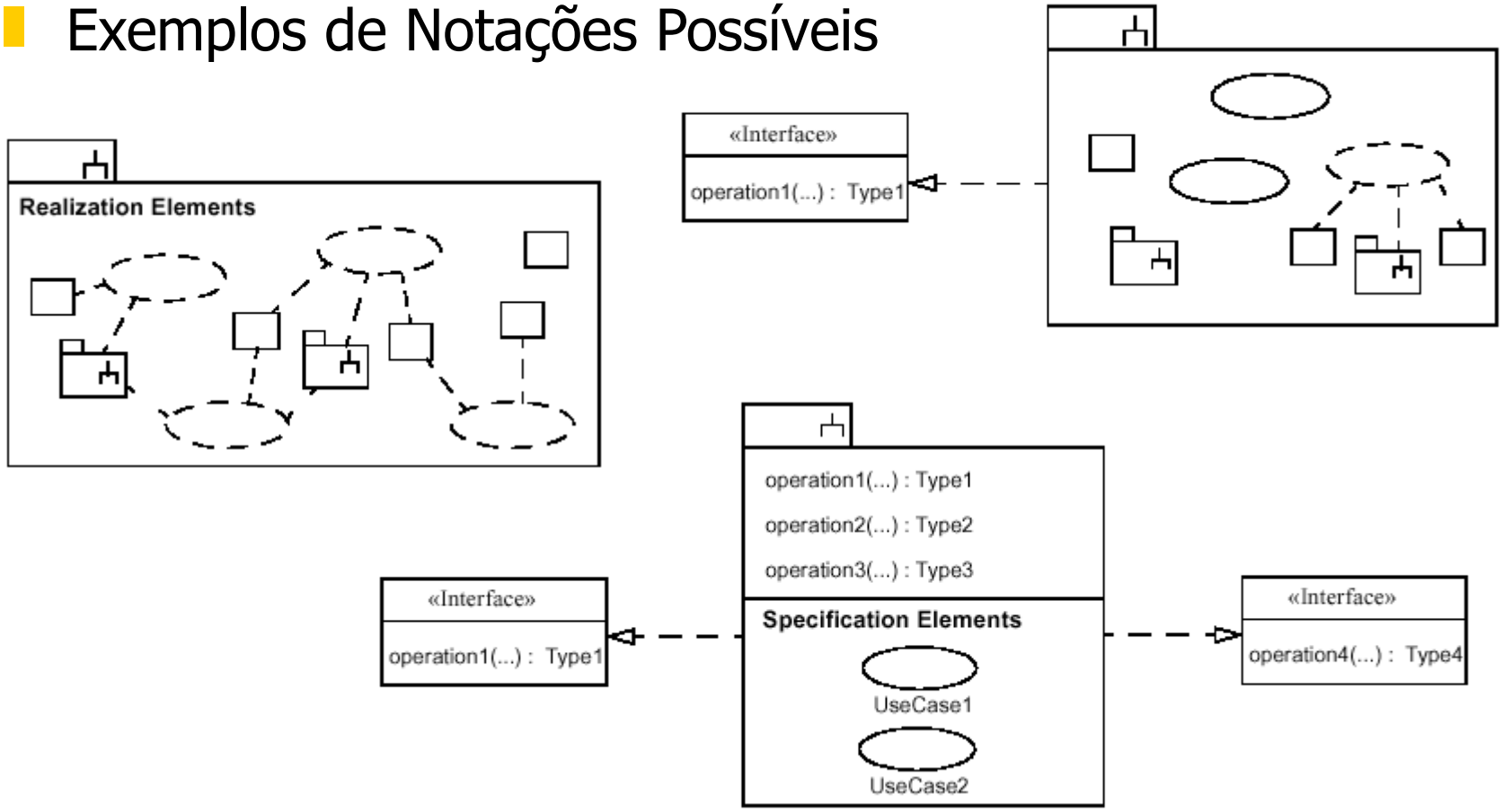
- Compartimentos
  - permitem a distribuição dos elementos em espaços reservados
- Interfaces
  - permitem especificar um conjunto de operações para o acesso ao sub-sistema





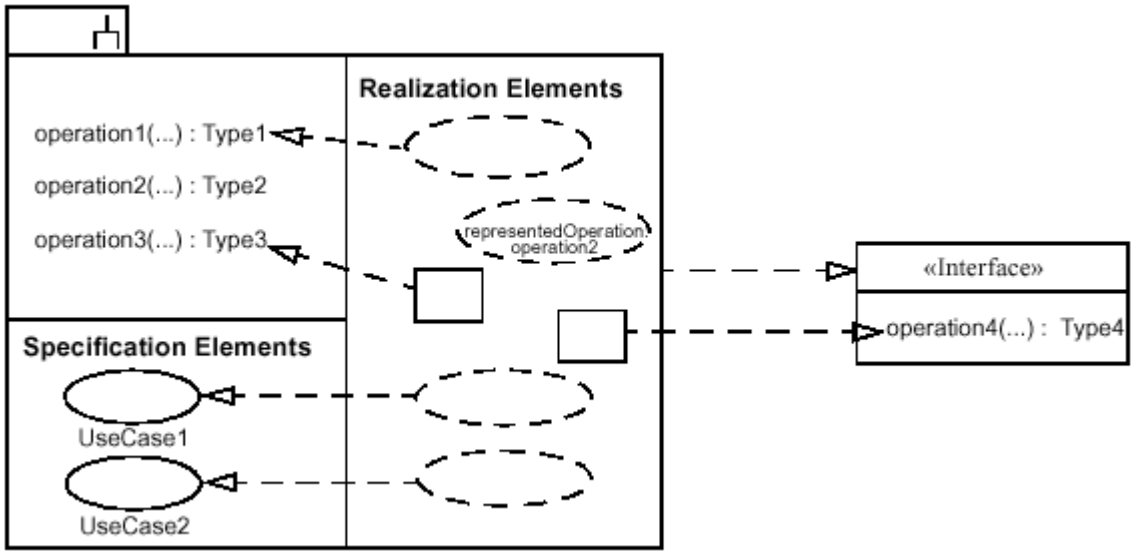
Sub-Sistemas

- Exemplos de Notações Possíveis



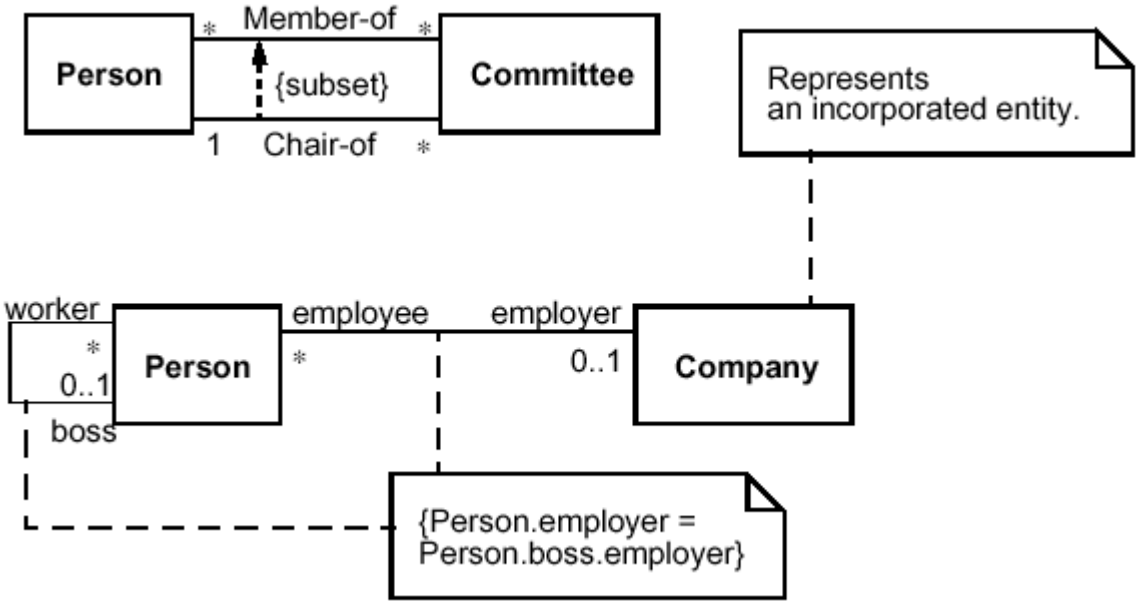
Sub-Sistemas

- Mapeamento
  - entre as partes de especificação e realização usando os três compartimentos
  - somente os elementos de realização com relevância
  - diferentes maneiras de expressar o mapeamento



Restrições

- Restrições
  - relacionamento semântico entre elementos do modelo que especificam condições e proposições que necessitam ser mantidas
  - algumas são pré-definidas, mas podem ser definidas pelo usuário



Diagramas Estruturais Estáticos

- Diagramas Estruturais Estáticos
  - denotam a estrutura estática de um modelo em particular
    - coisas que existem (tais como classes, tipos e objetos)
    - estrutura interna dessas coisas
    - relacionamento entre essas coisas
  - não mostram informações temporais, somente estruturais
- Diagramas de Classes
  - mostra um conjunto de classes e tipos relacionados entre si
  - *templates* e classes instanciadas
  - relacionamentos (associações e generalizações)
  - conteúdo de classes (atributos e operações)
- Diagramas de Objetos
  - mostra instâncias compatíveis com algum diagrama de classes em particular e o relacionamento estrutural entre elas

Diagrama de Classes

- Diagramas de Classes
  - grafos de elementos do tipo **Classifier** conectados por diversos tipos de relacionamentos estáticos
  - pode conter pacotes e outros tipos de elementos gerais
  - um diagrama representa uma **visão** do modelo estrutural estático, que pode ser entendido como a união de todos os diagramas de classe e de objetos
- Classifier
  - Classes, Interfaces e DataTypes
- Classe
  - descritor para um conjunto de objetos com estrutura, comportamento e relacionamentos similares
  - seu modelo diz respeito à intensão de uma classe, ou seja, as regras que a definem enquanto classe

Exemplos de Representações de Classes

Window

Window

size: Area
visibility: Boolean

display ()
hide ()

«controller»

PenTracker

{ leaf, author="Mary Jones"}

Window
{abstract,
author=Joe,
status=tested}

+size: Area = (100,100)
#visibility: Boolean = invisible
+default-size: Rectangle
#maximum-size: Rectangle
-xptr: XWindow*

+display ()
+hide ()
+create ()
-attachXWindow(xwin:Xwindow*)

Reservation

operations

guarantee()
cancel ()
change (newDate: Date)

responsibilities

bill no-shows
match to available rooms

exceptions

invalid credit card

Rectangle

p1:Point
p2:Point

«constructor»
Rectangle(p1:Point, p2:Point)
«query»
area (): Real
aspect (): Real
. . .
«update»
move (delta: Point)
scale (ratio: Real)
. . .

Símbolo Gráfico para Classe

- Símbolo Gráfico para Classe
  - Caixa, possivelmente dividida em compartimentos
- Compartimentos
  - utilizados em diferentes situações, dependendo se a classe pertence a um modelo de análise, design ou implementação
  - podem ser nomeados
- Compartimento do Nome
  - contém o nome da classe
  - opcionalmente, pode conter um estereótipo, um conjunto de propriedades (tagged-values) e um ícone referente ao estereótipo
- Compartimentos de Listas
  - atributos, operações ou outros

Compartimento de Atributos

- Compartimento de Atributos
  - usado para mostrar os atributos de uma classe
- Sintaxe Padrão
  - *visibility name [multiplicity] : type-expression = initial-value { property-string }*
- Visibilidade
  - + público
  - # protegido
  - - privado
- Multiplicidade
  - usado para representar arrays - e.g. [3] ou [2..*] ou [0..7]
- Atributos de Classe (Atributos Estáticos)
  - devem ser sublinhados

Compartimento de Operações

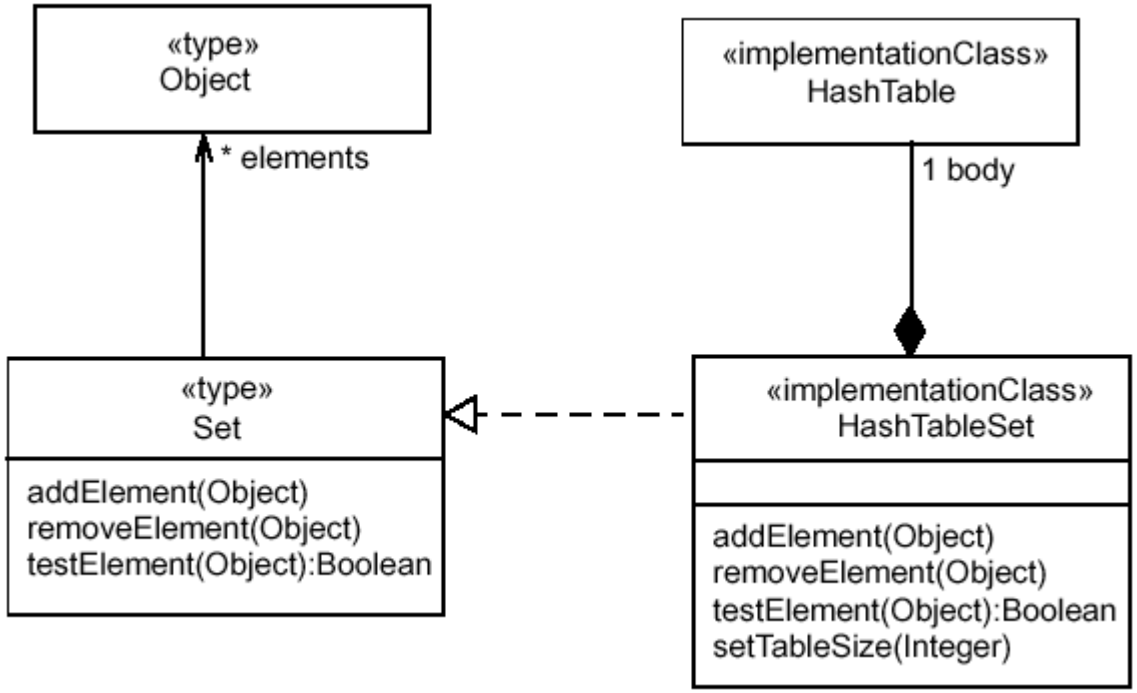
- Compartimento de Operações
  - mostram as operações definidas para uma classe e/ou os métodos supridos por uma classe
- Sintaxe Padrão
  - *visibility name (parameter-list) : return-type-expression { property-string }*
  - cada elemento de `parameter-list` tem a seguinte sintaxe
  - `kind name : type-expression = default-value`
  - onde `kind` *deve ser* `in`, `out`, ou `inout`
- Algumas Propriedades
  - {query} a operação não modifica o estado do sistema
  - {concurrency = name}, onde name deve ser um dos nomes: sequential, guarded, concurrent
  - {abstract} especifica operações não dotadas de implementação

Tipos de Classes

- Classes Conceituais e Classes de Implementação
  - estereótipos associados a uma classe
- Classe Conceitual
  - pode não incluir métodos, mas prover especificações comportamentais para sua operação
  - pode ter atributos e associações
- Classe de Implementação
  - define uma estrutura de dados física (para atributos e associações) e métodos de um objeto como implementados em linguagens tradicionais (C++, Java, etc..)
  - realiza uma classe conceitual se provê todas as operações especificadas em uma classe conceitual
  - uma classe de implementação pode realizar diversas classes conceituais

Classes Conceituais e Classes de Implementação

- Exemplo

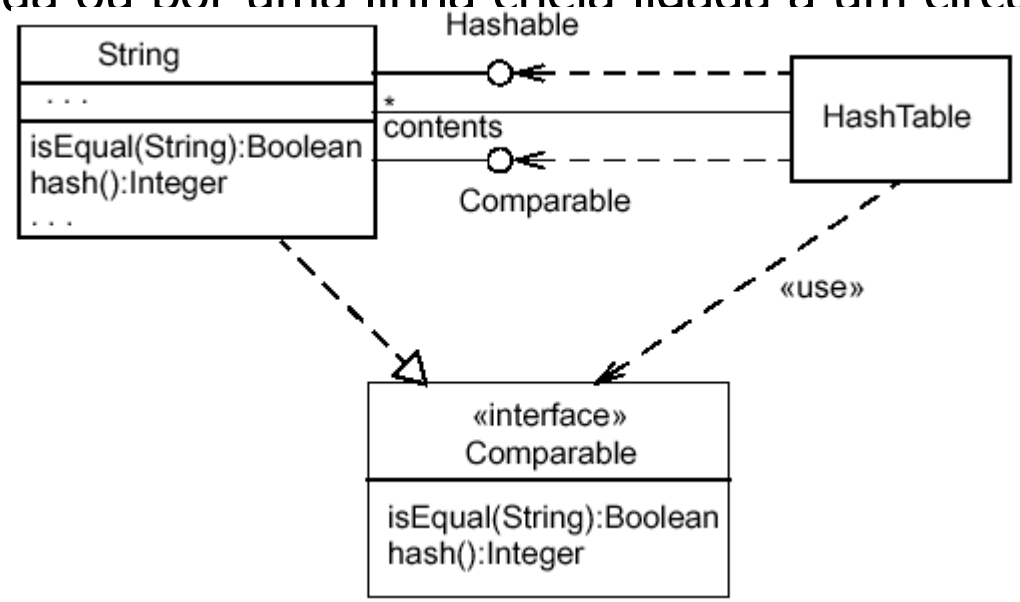


Interfaces

- Interface
  - é um especificador para um conjunto de operações externamente visíveis de uma classe, componente ou outro tipo de *classifier* (incluindo um sub-sistema) sem a especificação de sua estrutura interna
  - cada interface especifica somente uma parte limitada do comportamento de uma classe
  - interfaces não possuem implementação
  - não possuem atributos, estados ou associações, mas somente operações
  - podem ter relacionamentos do tipo generalização
  - formalmente equivalente a uma classe abstrata sem atributos e sem métodos, com somente operações abstratas

Interfaces

- Notação
  - classe sem compartimento de atributos, com o estereótipo <<interface>> ou simplesmente um círculo
  - uma dependência abstrata entre uma classe e uma interface pode ser representada por uma linha do tipo generalização tracejada ou por uma linha cheia ligada a um círculo

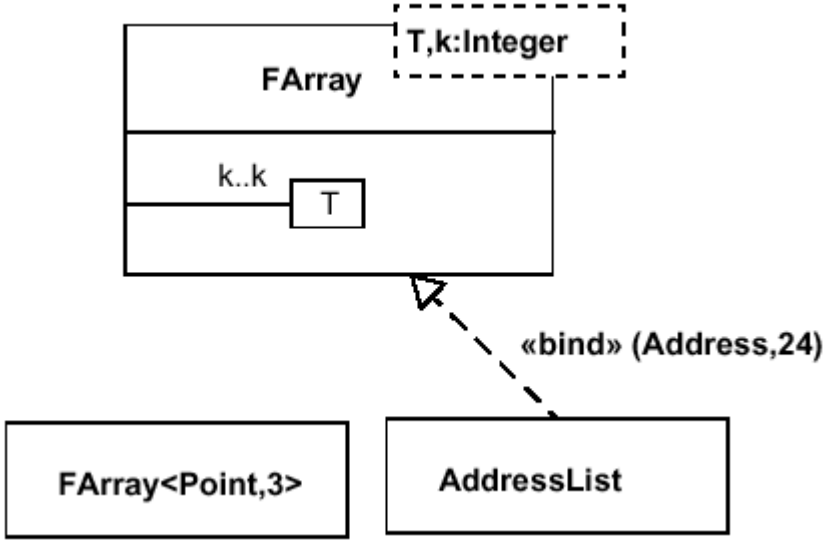


Classes Parametrizadas (Templates)

- Template
  - é um descritor para uma classe com um ou mais parâmetros formalmente livres
  - define uma família de classes, onde cada classe deve instanciar um parâmetro a um dos parâmetros livres
  - tipicamente esses parâmetros livres correspondem a atributos, mas também podem ser operações
  - atributos e operações dentro de um template são definidas em termos de parâmetros formais que são instanciados quanto o template por si próprio é instanciado a um valor definitivo
  - não corresponde a uma classe diretamente usável, pois possui parâmetros não definidos
  - não podem participar de associações, a não ser no sentido de serem instanciadas em alguma classe

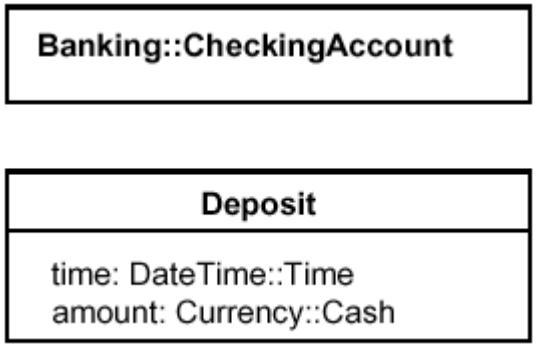
Classes Parametrizadas (Templates)

- Notação
  - um retângulo tracejado é superimposto no canto direito superior da classe, contendo uma lista dos parâmetros formais a serem substituídos



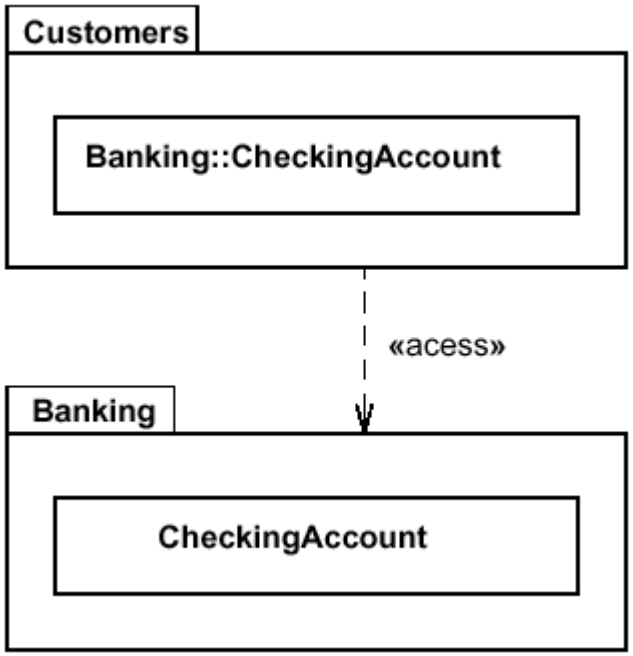
Classes e Pacotes

- Cada Diagrama de Classes
  - corresponde a um Pacote
- Mais de um Pacote
  - podem aparecer no mesmo diagrama
- Em algumas situações
  - pode ser necessário referenciar classes em outros pacotes não disponíveis em um diagrama
- Neste caso
  - classe deve ser referenciada da seguinte forma:
  - *package-name* :: *class-name*



Acessando e Importando Pacotes

- Elemento
  - referenciar elementos em outros pacotes
- Nível de Pacotes
  - dependência do tipo «access» indica que o conteúdo de um pacote faz referência a elementos do outro pacote
  - visibilidade deve ser condizente
  - dependência do tipo «import» libera o acesso e automaticamente carrega os nomes com a visibilidade apropriada no *namespace* do pacote fazendo a importação, o que evita o uso de *pathnames* para referenciar as classes



Diagramas de Objetos

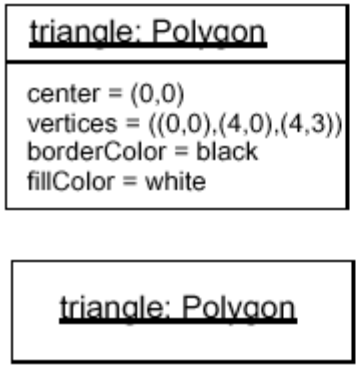
- Diagrama de Objetos
  - grafo de instâncias de classes, incluindo objetos e variáveis
  - instância de um diagrama de classes mostrando detalhadamente o estado do sistema em um determinado instante de tempo
  - formato do diagrama de objetos é semelhante ao do diagrama de classes, sendo diferenciado somente por seu uso
- Uso de Diagramas de Objetos
  - normalmente é limitado, sendo utilizado para mostrar exemplos de estruturas de dados em pontos estratégicos do sistema
- Objeto
  - representa uma instância particular de uma classe
  - tem uma identidade e um conjunto de valores de atributos próprio

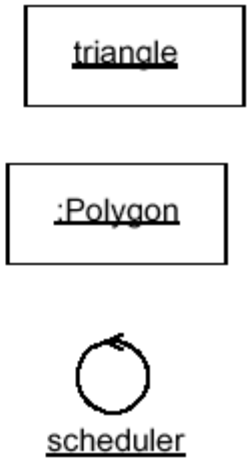
Objetos

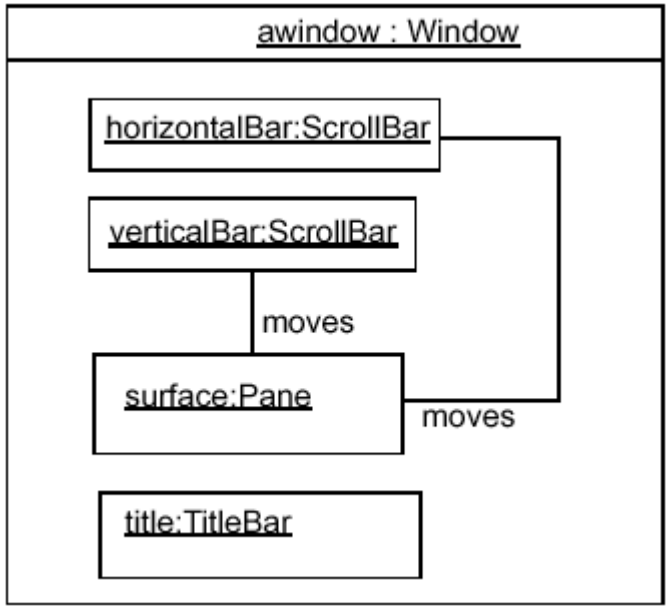
- Notação
  - derivada da notação de classe, sublinhando-se o nome do objeto/classe
  - pode ter dois compartimentos
    - primeiro mostra o nome do objeto, seu estereótipo e propriedades
      - *objectname : classname*
    - classname pode incluir o caminho completo do pacote onde se encontra a classe que o objeto referencia
      - e.g. `display_window: WindowingSystem::GraphicWindows::Window`
    - caso haja herança múltipla, as classes devem ser separadas por vírgulas
    - segundo mostra os valores dos atributos do objeto, na forma de uma lista, onde cada linha deve ser como segue:
      - `attributename : type = value`

Objetos

- Objetos Compostos
  - representa um objeto de alto nível que contém outros objetos como partes
- Exemplos de Objetos





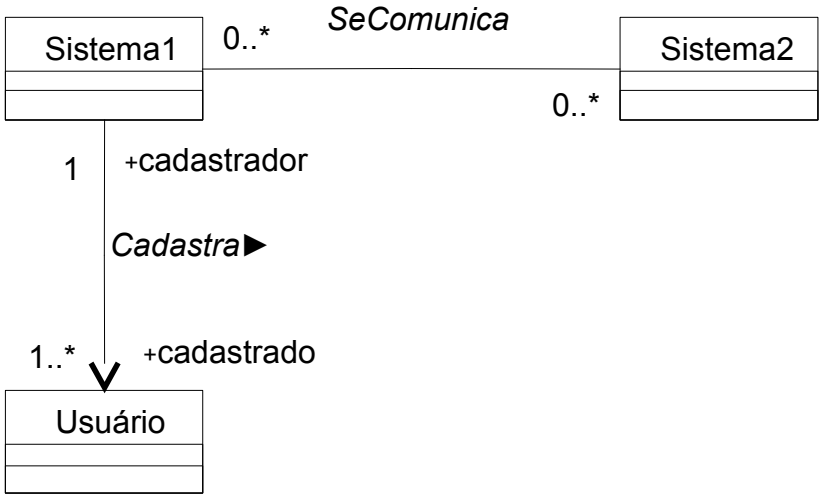


Relacionamentos entre Classes e entre Objetos

- Relacionamentos
  - Associações
    - Associações Simples
    - Agregações
    - Composições
  - Generalizações
  - Dependências
- Relacionamentos Binários
  - mostrados como linhas conectando dois símbolos de *classifiers*
  - linhas podem ter uma variedade de decorações para diferenciar suas propriedades
- Relacionamentos Ternários ou de Ordem Superior
  - exibidos como diamantes conectados por linhas a símbolos de *classifiers*

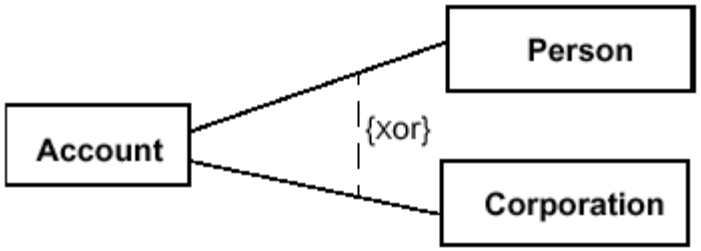
Associações

- Associações Simples
  - representam que existe alguma conexão entre dois elementos do tipo *classifier*, de tal forma que um deve manter alguma referência ao outro
  - representadas na forma de uma linha cheia conectando os dois *classifiers*
  - pode possuir
    - nome
    - duas extremidades
- Extremidades
  - podem possuir
    - navegabilidade
    - multiplicidade
    - papel



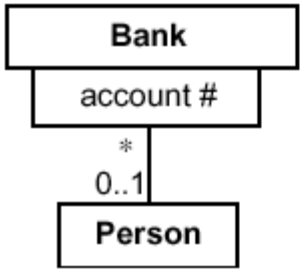
Associações

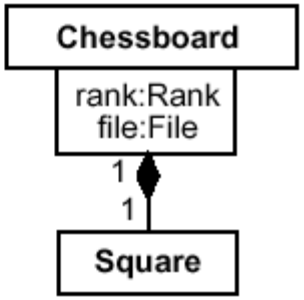
- Associação XOR
  - indica uma situação onde somente uma dentre diversas potenciais associações podem ser instanciadas em um determinado instante, para uma dada instância
  - qualquer instância do classificador poderá participar de somente de uma das associações indicadas
  - este é apenas um exemplo da aplicação de uma restrição a uma associação



Qualificadores

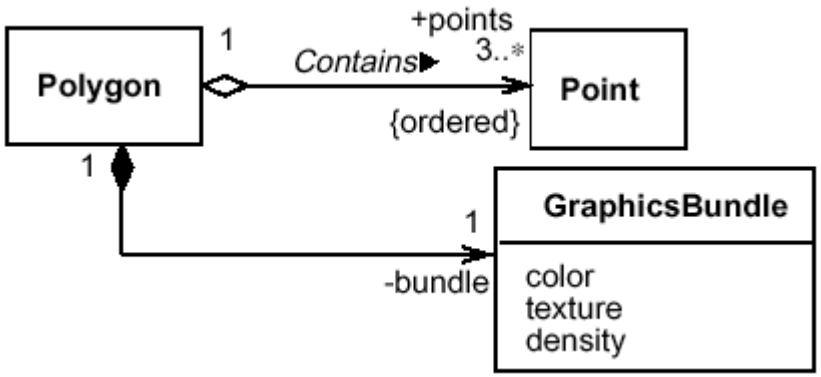
- Qualificadores
  - são atributos ou listas de atributos cujos valores servem para particionar o conjunto de instâncias associadas a uma instância através de uma associação
  - qualificadores são atributos da associação





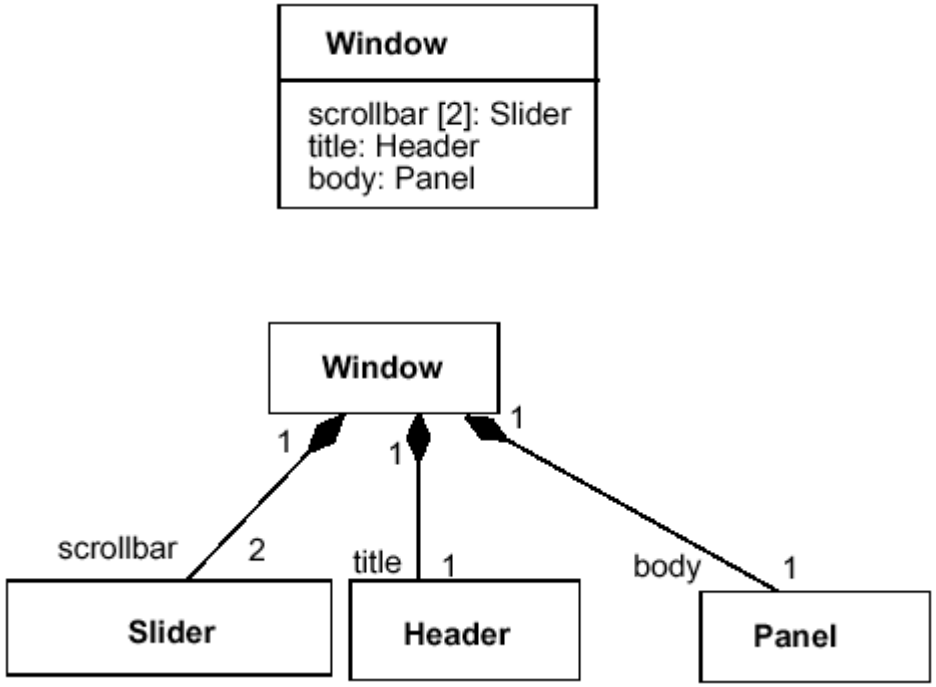
Agregações e Composições

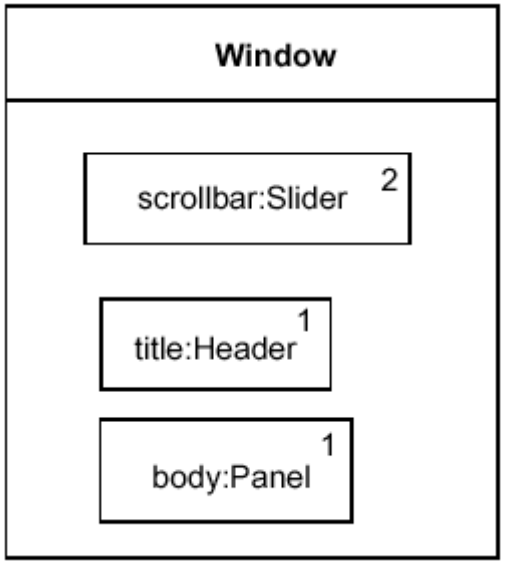
- Agregação
  - tipo especial de associação, onde o elemento associado corresponde a uma parte do elemento principal
- Composição
  - tipo especial de agregação, onde necessariamente a parte indicada deve existir



Composição

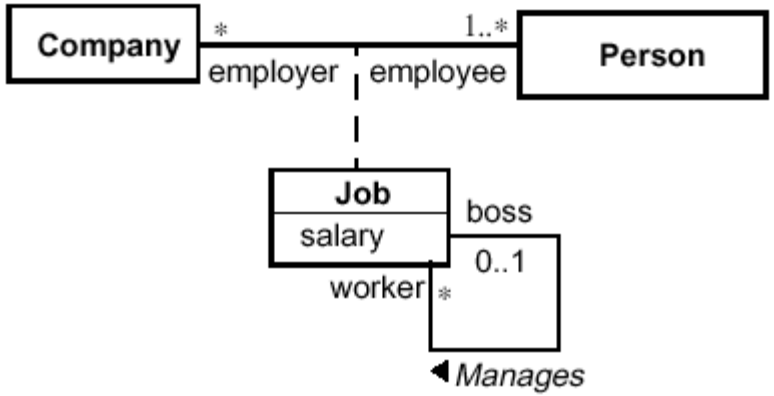
- Diversas Maneiras de Representar uma Composição





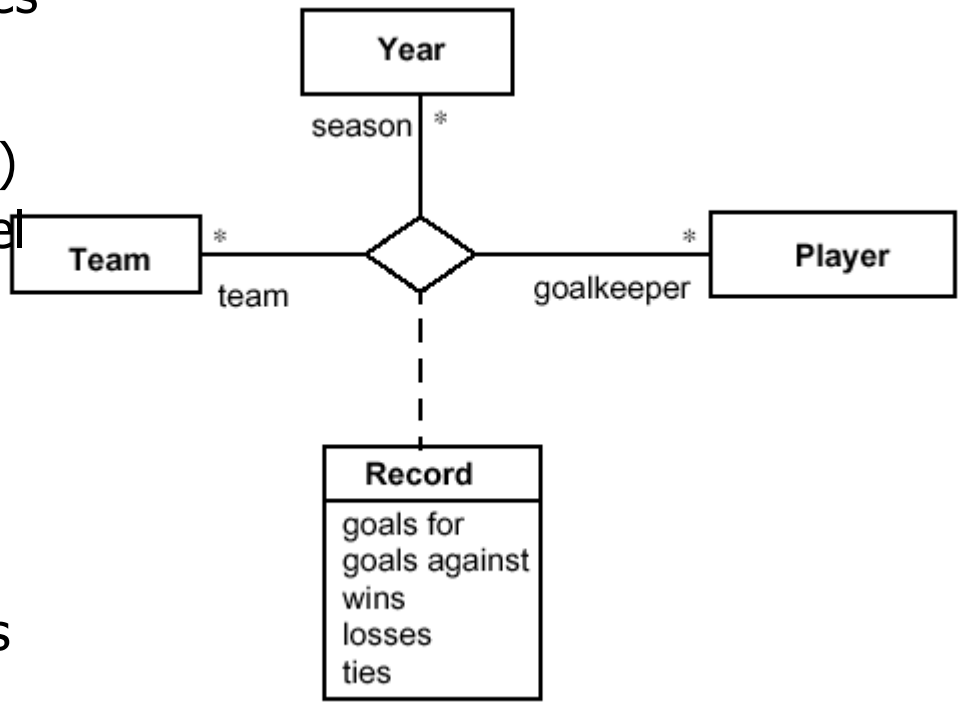
Classe de Associação

- Classe de Associação
  - é utilizada quando a associação em si necessitar uma representação diferenciada, por exemplo, tendo atributos ou operações associadas
  - uma classe de associação é uma associação que ao mesmo tempo possui propriedades de uma classe (ou uma classe que tem propriedades de uma associação
  - corresponde a um único elemento, apesar de seu aspecto dual



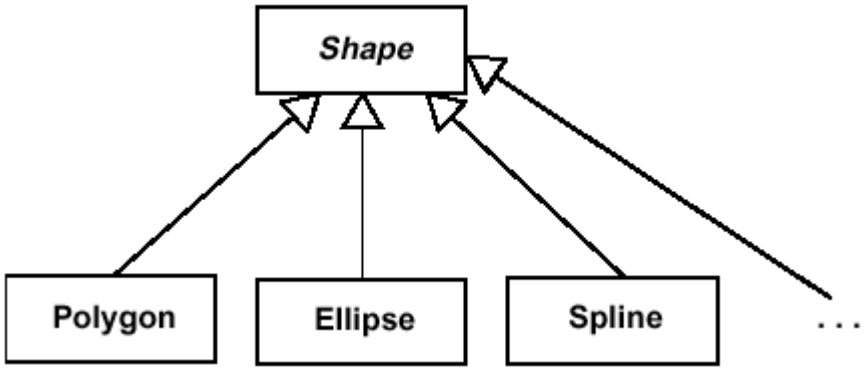
Associação N-ária

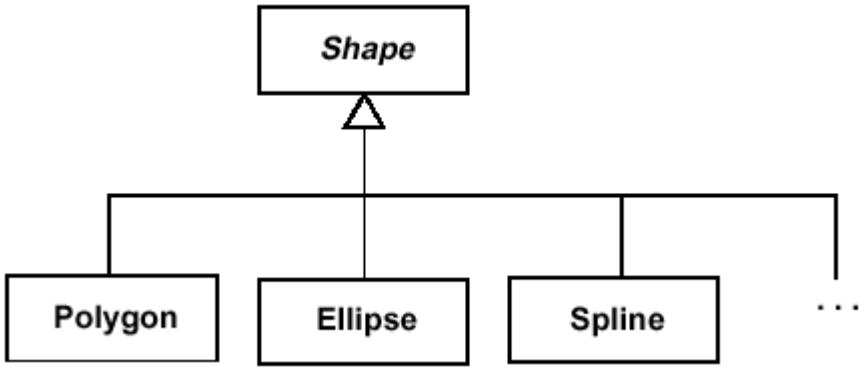
- Associação N-ária
  - é uma associação entre três ou mais classifiers (onde um mesmo classifier pode aparecer mais de uma vez)
  - multiplicidade em um papel representa o número potencial de instâncias de uma tupla na associação quando os outros N-1 valores são definidos
  - não pode conter marcadores de agregações



Generalização

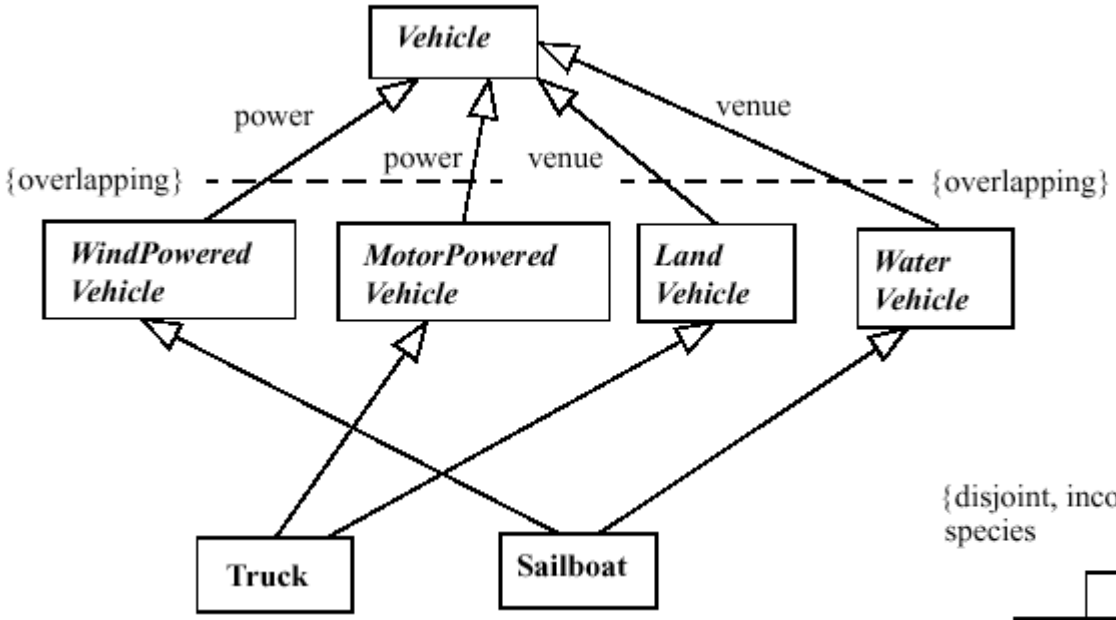
- Generalização
  - é um relacionamento taxonômico entre um elemento mais geral (o pai) e um elemento mais específico (o filho) que deve ser consistente com o primeiro elemento e que adiciona informações adicionais
  - pode ser usada para classes, pacotes, casos de uso e outros elementos

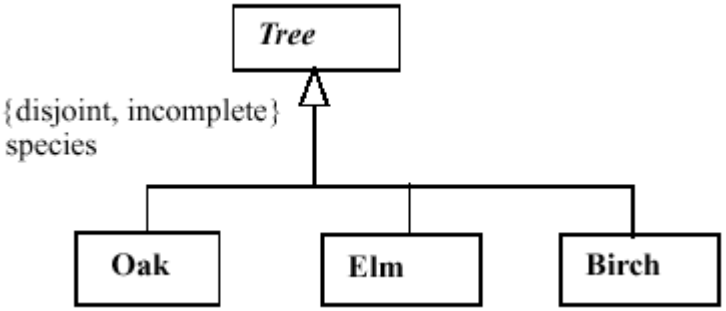




Generalização com Restrições ou Descrições

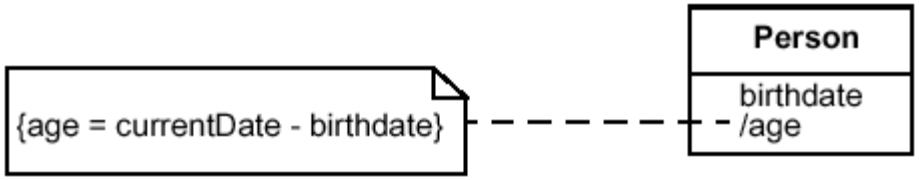
- Outros Exemplos de Generalização





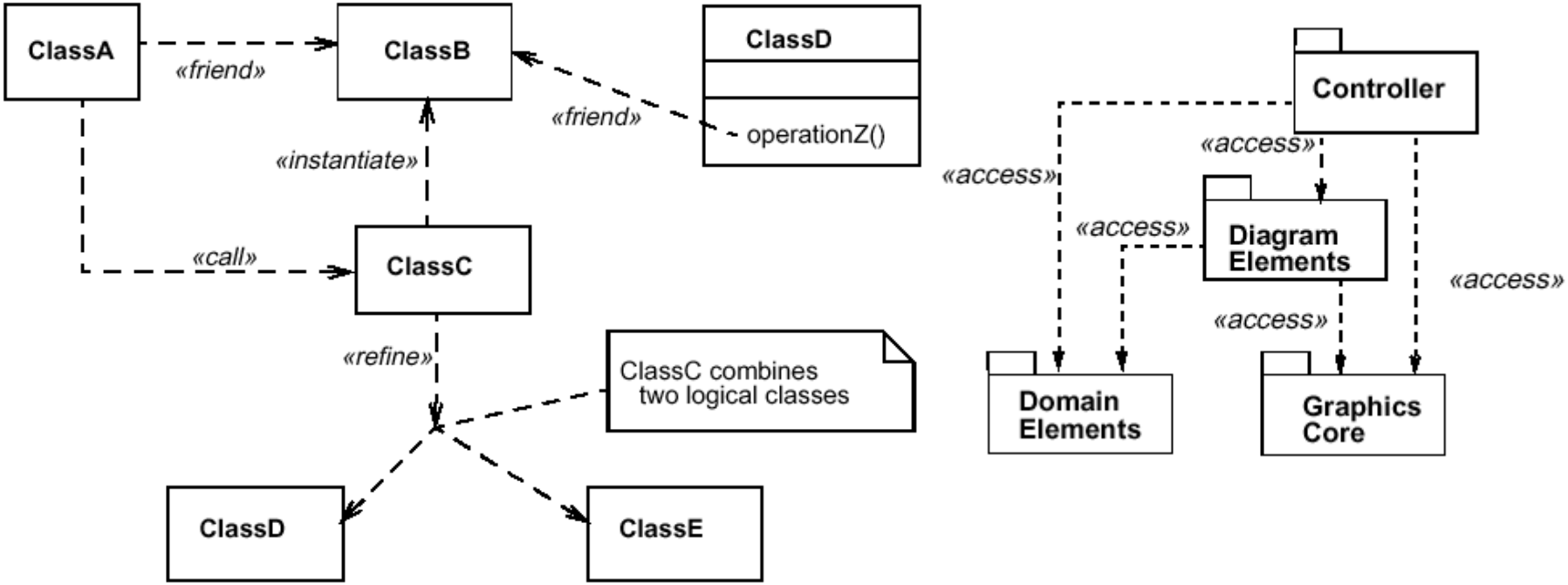
Dependências

- Dependências
  - indicam um relacionamento semântico entre os dois elementos de modelagem (ou conjuntos de elementos de modelagem)
  - relacionam os elementos de modelagem por si só, não demandando um conjunto de instâncias para seu significado
  - indicam situações em que uma mudança em um dos elementos pode demandar uma mudança no elemento que dele depende
- Estereótipos Padrões Para Dependências
  - access, bind, derive, import, refine, trace, use
- Elementos Derivados
  - podem ser indicados por meio de dependências



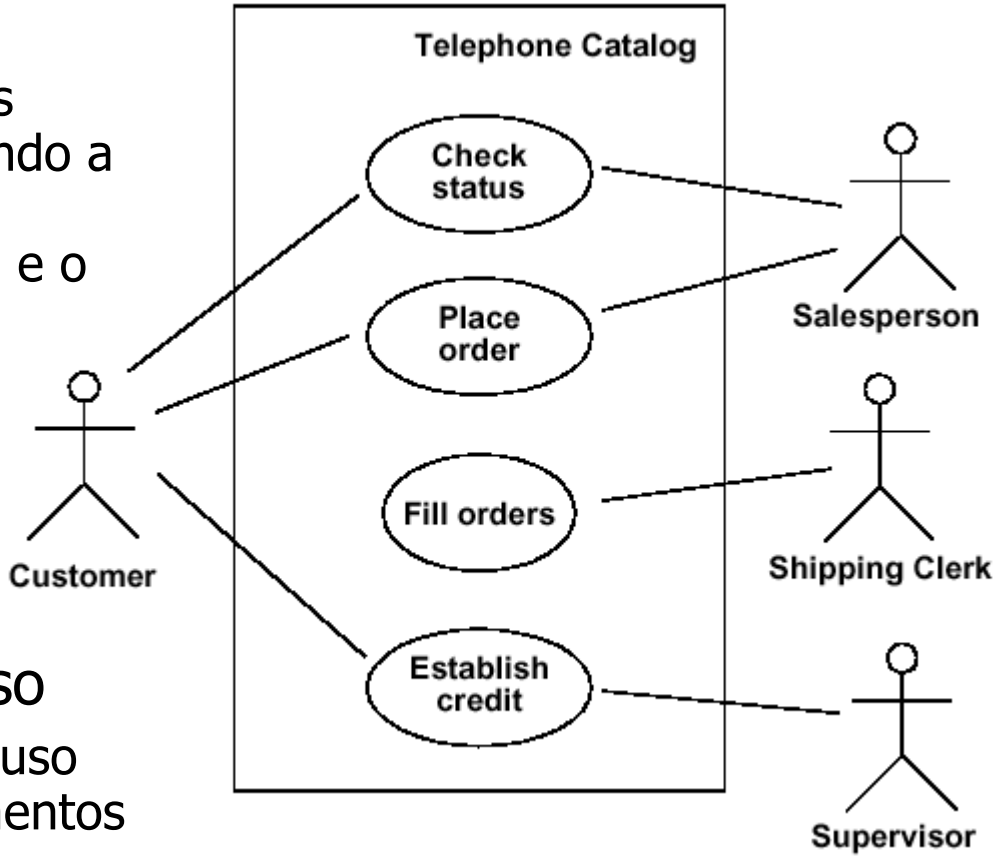
Dependências

- Exemplos de Dependências



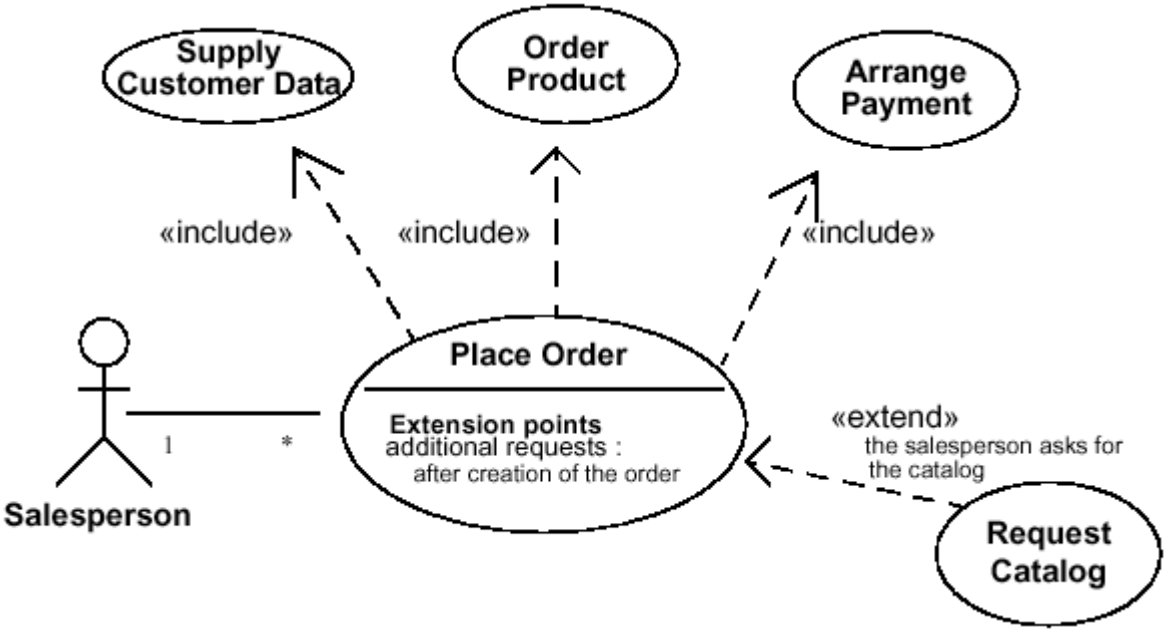
Diagramas de Casos de Uso

- Casos de Uso
  - são abstrações de pequenas histórias narrativas envolvendo a interação entre um ou mais usuários (chamados atores) e o sistema
  - representam, por meio dessas pequenas histórias, as funcionalidades de um sistema
- Diagramas de Casos de Uso
  - mostram atores e casos de uso juntos com seus relacionamentos



Diagramas de Casos de Uso

- Pontos de Extensão
  - são referências a uma localização dentro de um caso de uso onde sequências de ações de outros casos de uso podem ser inseridas
  - cada ponto de extensão tem um único nome dentro de um caso de uso

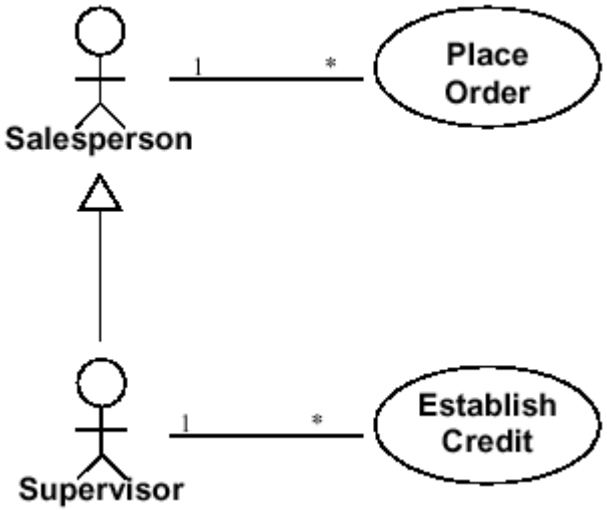


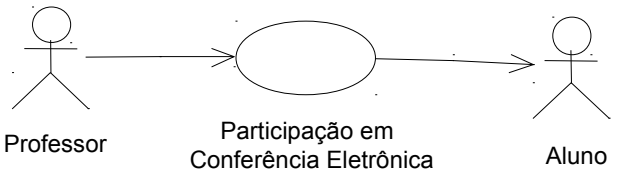
Diagramas de Caso de Uso

- Relacionamentos em Diagramas de Caso de Uso
  - **Associações**: denotam a participação de um ator em um caso de uso. É o único tipo de relacionamento entre um ator e um caso de uso
  - **Extend**: uma relação entre um caso de uso A para um caso de uso B indica que uma instância do caso de uso B pode ser aumentada (sujeita a condições específicas da extensão) pelo comportamento especificado por a . Esse comportamento é inserido conforme especificado por um ponto de extensão em B.
  - **Include**: uma relação entre um caso de uso A para um caso de uso B que indica que uma instância do caso de uso A contém o comportamento especificado por B. Esse comportamento é incluído na localização indicada em A por um ponto de extensão
  - **Generalização**: uma generalização de um caso de uso A para um caso de uso B indica que A é uma especialização de B

Diagramas de Caso de Uso

- Atores
  - podem também manter relacionamento do tipo generalização com outros atores
- Associação
  - entre um ator e um caso de uso pode conter multiplicidades e navegabilidade
- Navegabilidade
  - indica quem inicia o caso de uso
  - caso seja do ator para o caso de uso, é o ator quem inicia a interação
  - caso seja do caso de uso para o ator, é o sistema quem inicia a interação





Diagramas de Interação

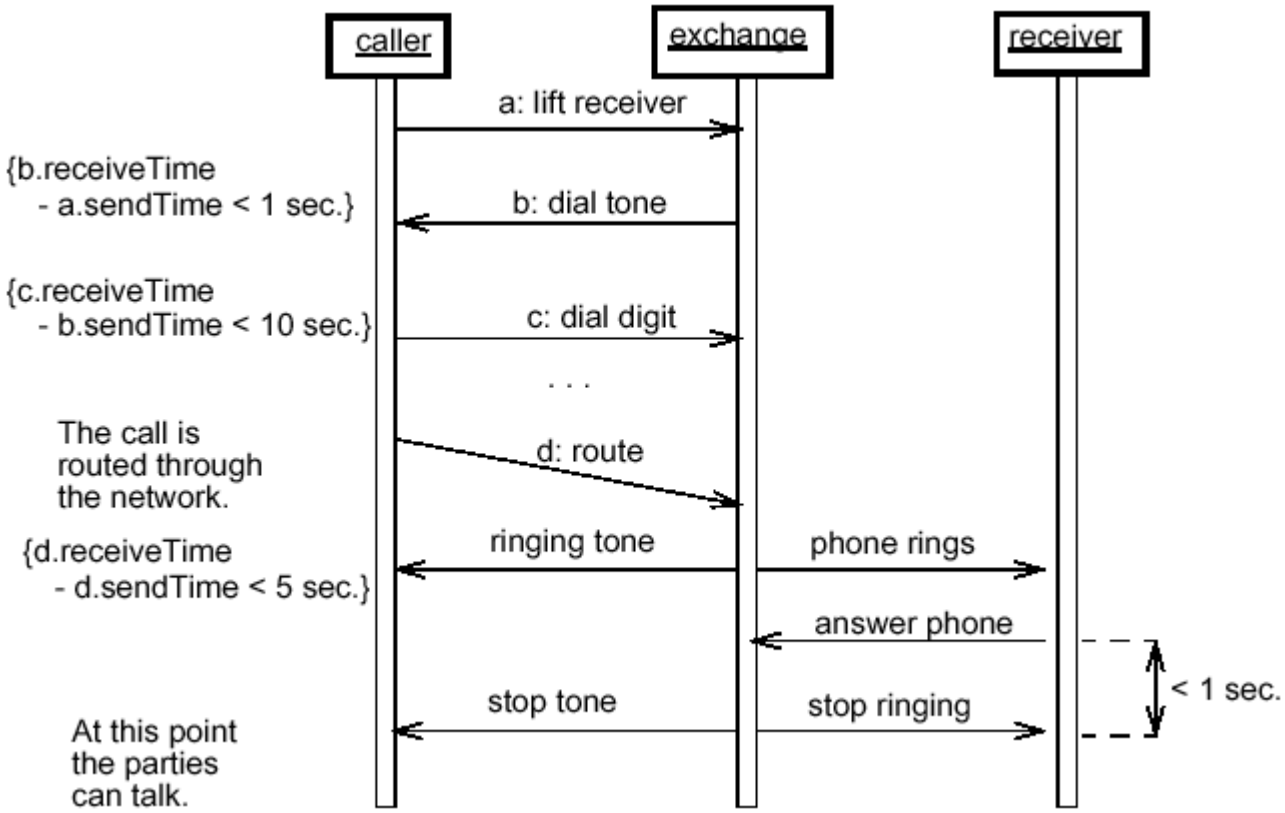
- Diagramas de Interação
  - mostram padrões de interação entre instâncias
  - podem aparecer em duas formas diferentes
    - diagramas de sequência e diagramas de colaboração
  - as informações em ambos diagramas é equivalente, mas cada tipo de diagrama enfatiza um aspecto particular da interação
- Diagramas de Sequência
  - mostram a sequência explícita de estímulos entre objetos e são melhores para especificações de tempo real e para cenários complexos
- Diagramas de Colaboração
  - mostram o relacionamento entre as instâncias e são melhores para o entendimento de todos efeitos sobre uma determinada instância, bem como para um design procedural

Diagramas de Sequência

- Diagrama de Sequências
  - mostra uma interação na forma de uma sequência temporal de mensagens sendo enviadas entre instâncias
  - em particular, mostra as instâncias participando de uma interação por meio de "lifelines" e o estímulo que trocam entre si arranjados na forma de uma sequência temporal
  - não mostra associações entre objetos
- Lifeline
  - denota um objeto executando um papel específico
  - setas entre lifelines denotam uma comunicação entre objetos
  - existência e duração do objeto em um papel é mostrada, mas o relacionamento entre os objetos não é mostrado
  - lifeline pode se dividir em duas ou mais lifelines concorrentes para expressar condicionalidade

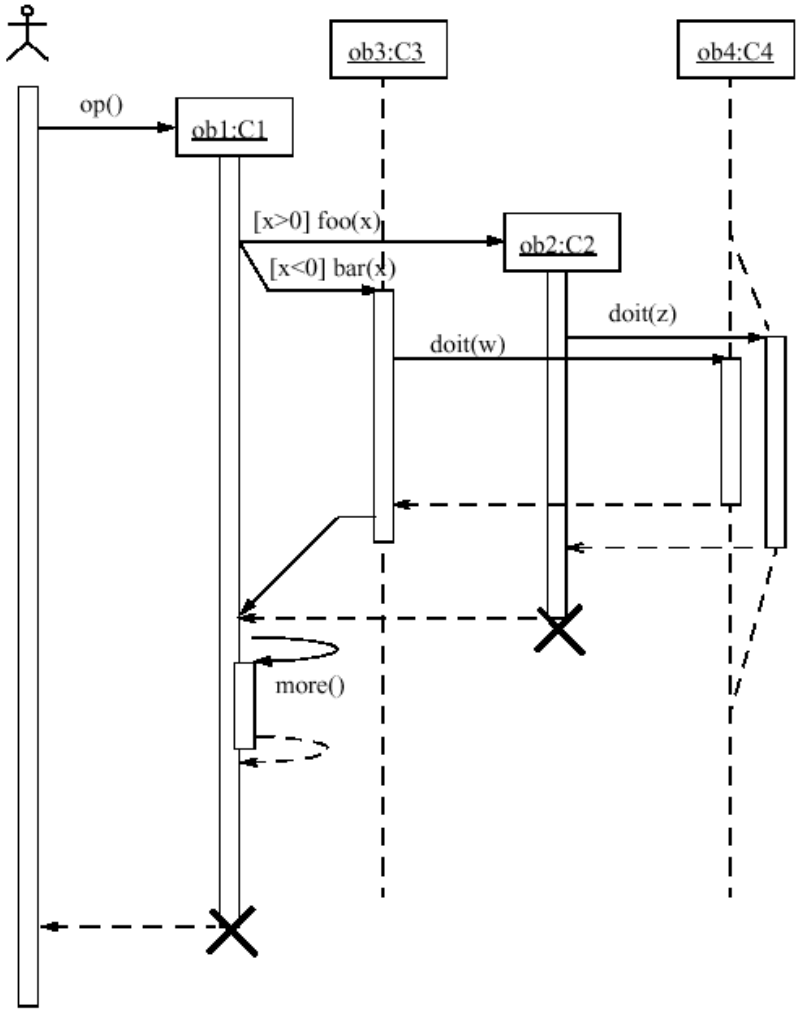
Diagrama de Sequências

- Exemplo de Diagrama de Sequências



Diagramas de Sequências

- Foco de Controle
  - ou ativação, mostra o período no qual um objeto está executando uma ação
- Mensagem Condicional
  - condição de guarda
- Recursão
  - mensagem mandada para o próprio objeto
- Criação
  - objeto é deslocado
- Destruição
  - marcado com X



Diagramas de Sequência

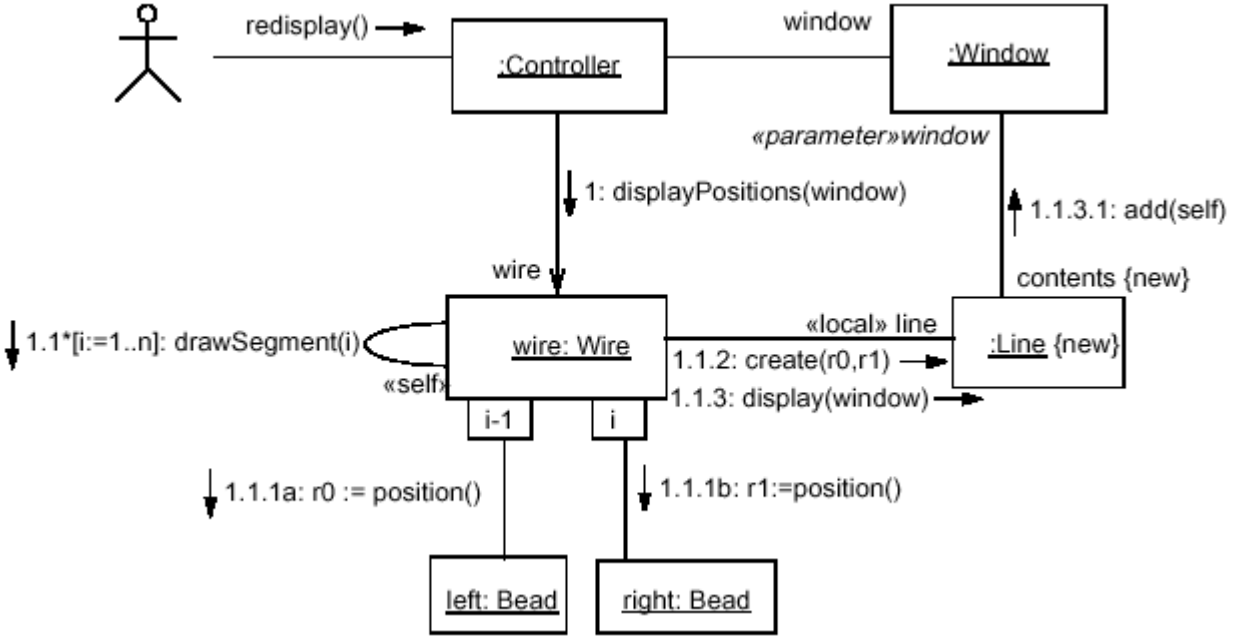
- Tipos de Comunicação
  - **Chamada de Procedimento ou outro tipo de fluxo de controle.** Toda uma sequência aninhada é completada antes que a sequência mais externa termine. Pode ser usada em chamadas de procedimento ordinárias. Pode também ser usada em objetos concorrentes quando um deles manda um sinal e espera uma sequência de comportamentos ser completada
  - **Fluxo de Controle Fraco.** Cada seta mostra a progressão do próximo passo na sequência. Normalmente todas as mensagens são assíncronas
  - **Estímulo Assíncrono.** Usado no lugar do anterior quando se quer mostrar explicitamente uma comunicação assíncrona entre dois objetos
  - **Retorno de uma Chamada de Procedimento**

Diagramas de Colaboração

- Diagrama de Colaboração
  - mostra a interação entre objetos organizada de acordo com os papéis de cada objeto na interação, e sua ligação entre si
  - Ao contrário de um diagrama de sequência, mostra o relacionamento entre objetos executando os diversos papéis
  - Da mesma maneira, não mostra o tempo como uma dimensão separada. Assim, a sequência de mensagens é mostrada na forma de números
  - pode ser desenvolvido em duas diferentes formas
    - nível de especificação (Papéis de Classifier, Papéis de Associação e Mensagens)
    - nível de instância (Objetos, Links e Estímulos)
  - primeiro caso modela os papéis a a estrutura de colaboração entre esses papéis, sendo que no segundo caso, mostra-se os objetos que assumem esses papéis

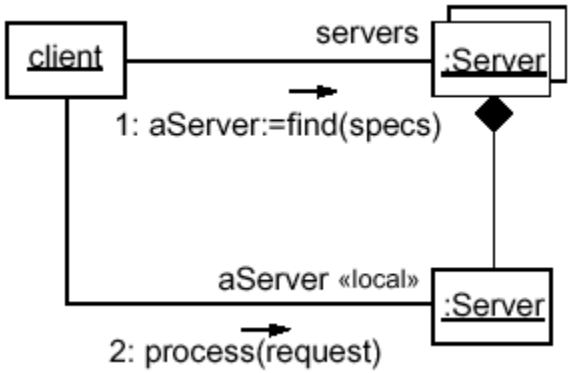
Diagramas de Colaboração

- Exemplo de Diagrama de Colaboração



Diagramas de Colaboração

- Multi-Objetos
  - representa um conjunto de objetos à extremidade de uma associação que contenha multiplicidade maior que 1
  - é utilizado para representar mensagens que são enviadas à coleção inteira de objetos ao invés de um único objeto dentro da coleção

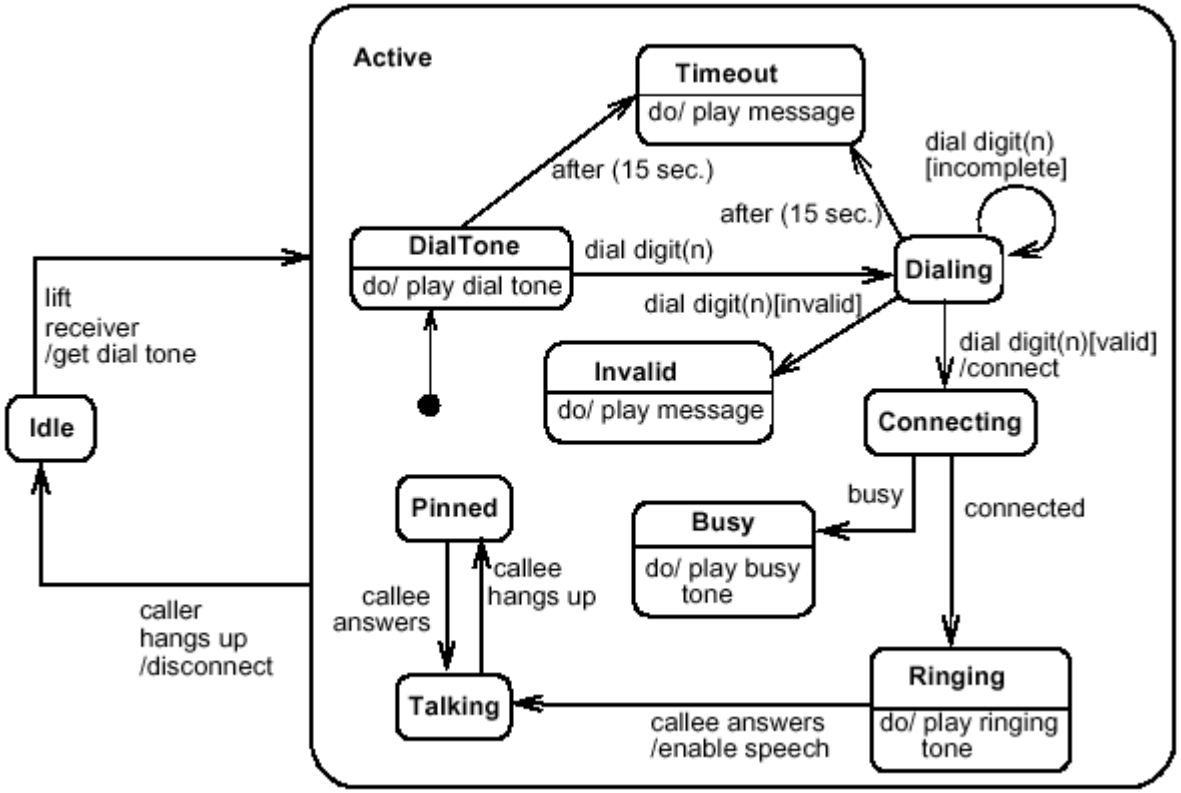


Diagramas de Estado

- Diagramas de Estado
  - descrevem os estados e transições de estados possíveis em objetos participantes de interações
  - especificamente, descreve possíveis sequências de estados e ações pelas quais os elementos podem passar durante seu ciclo de vida, como um resultado de uma reação a eventos discretos (e.g. sinais, invocações de operações, etc...)
- Semântica dos Diagramas de Estado
  - derivada dos statecharts de David Harel, com algumas modificações para torná-los orientados a objeto
  - significam um substancial avanço sobre máquinas de estado simples
  - implementam aspectos tanto das máquinas de Moore como das máquinas de Mealy (modelos tradicionais de máquinas de estado)

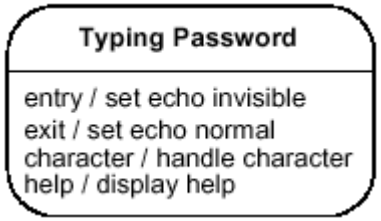
Diagramas de Estados

- Exemplo de Diagrama de Estados



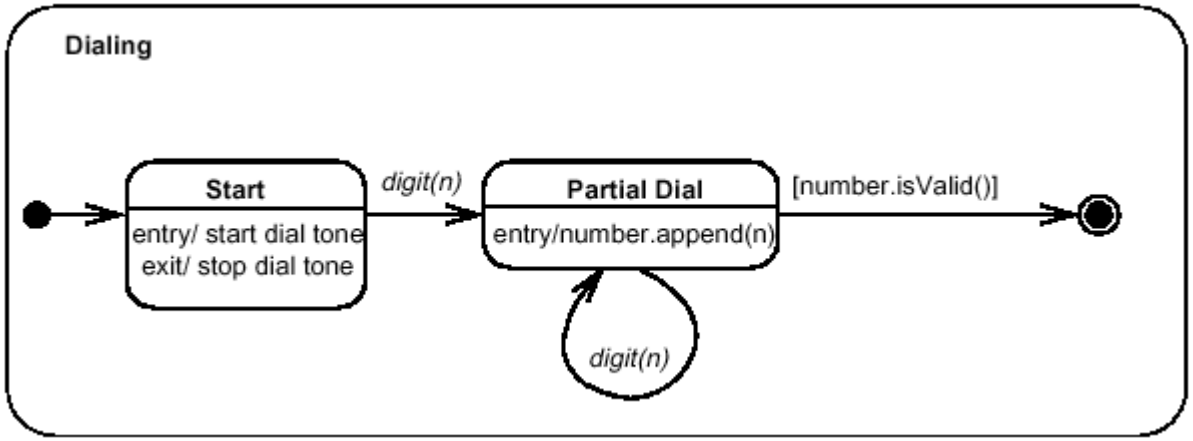
Diagramas de Estados

- Estado
  - condição, durante o ciclo de vida de um objeto ou durante uma interação, na qual este satisfaz alguma condição, executa alguma ação ou aguarda por algum evento
  - representação utiliza possivelmente dois compartimentos
    - compartimento do nome
    - compartimento de ações executadas
- Compartimento de Ações Executadas
  - *action-label* '/' *action-expression*
- *Action-Label*
  - *entry* - ação executada ao adentrar o estado
  - *exit* - ação executada ao sair do estado
  - *do* - ação executada durante a permanência no estado
  - *include* -identifica uma invocação de sub-máquina



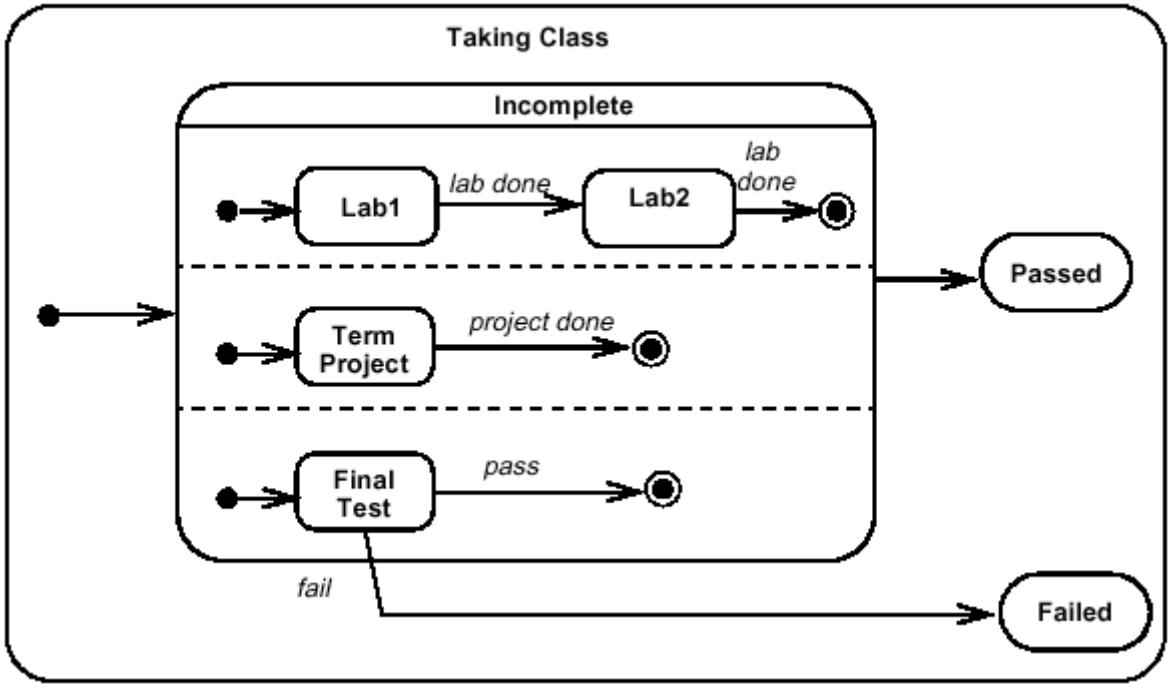
Diagramas de Estado

- Estado Composto
  - decomposto em dois ou mais sub-estados concorrentes (chamados de regiões), ou sub-estados mutuamente e exclusivamente disjuntos
  - cada sub-estado de um estado composto pode também ser um estado composto
  - pseudo-estados determinam o início e o fim de um sub-estado interior



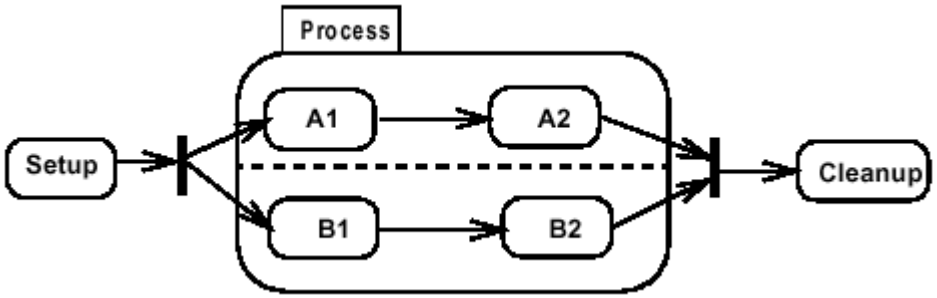
Diagramas de Estados

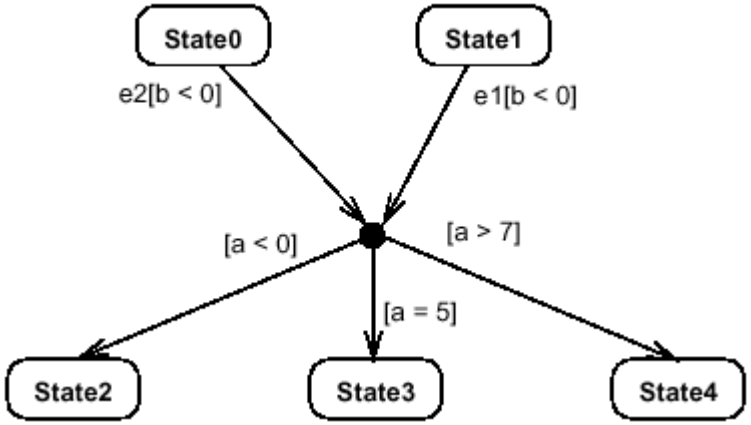
- Sub-Estados Concorrentes

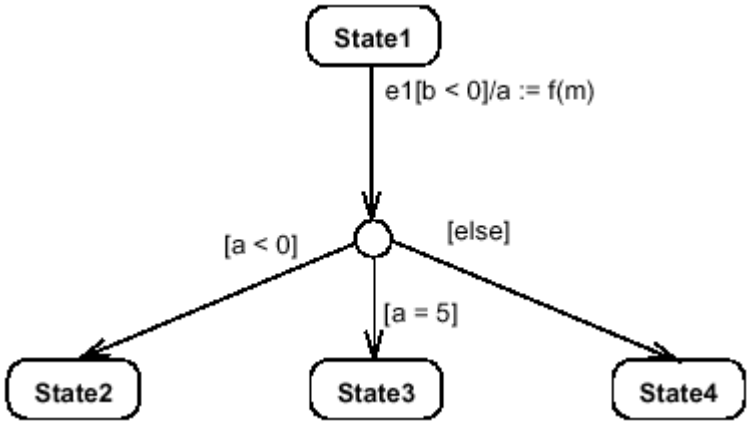


Diagramas de Estados

- Transições, Junções e Choices

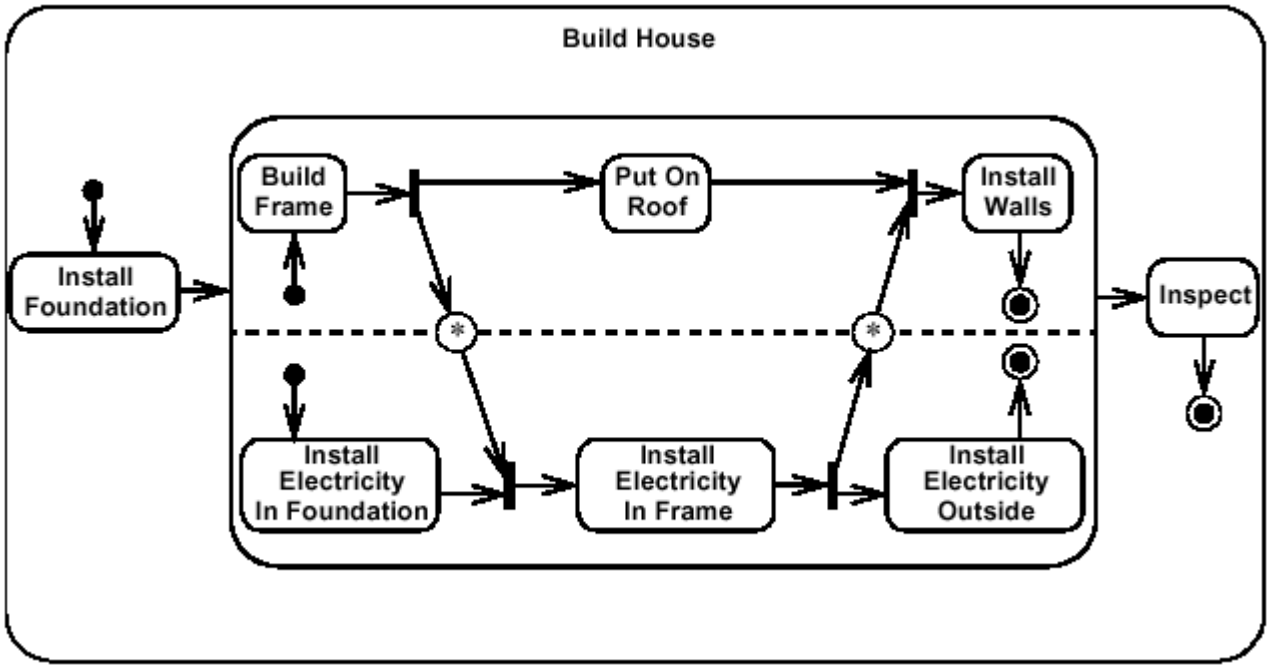






Diagramas de Estados

- Estados de Sincronização

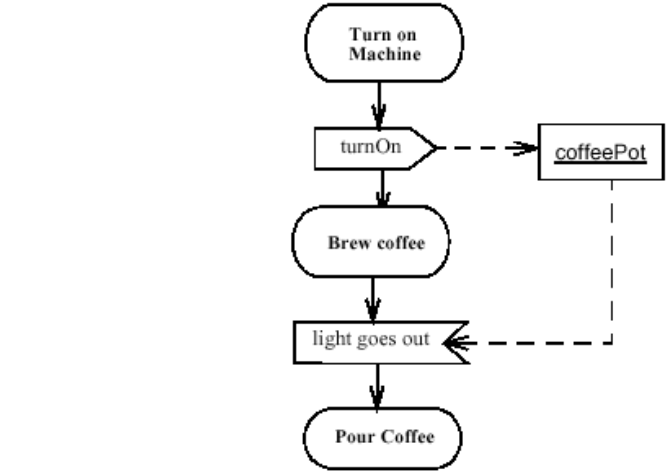


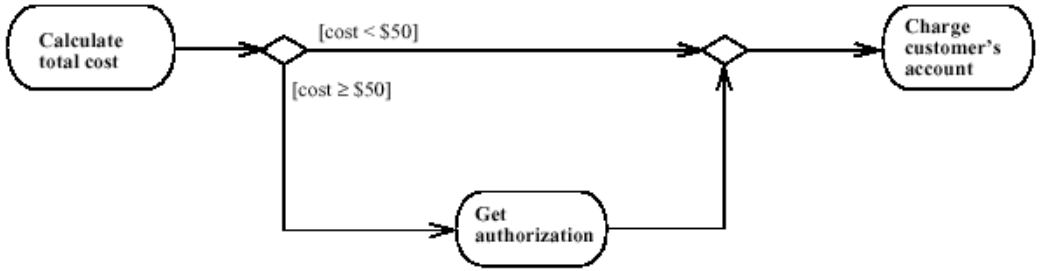
Diagramas de Atividades

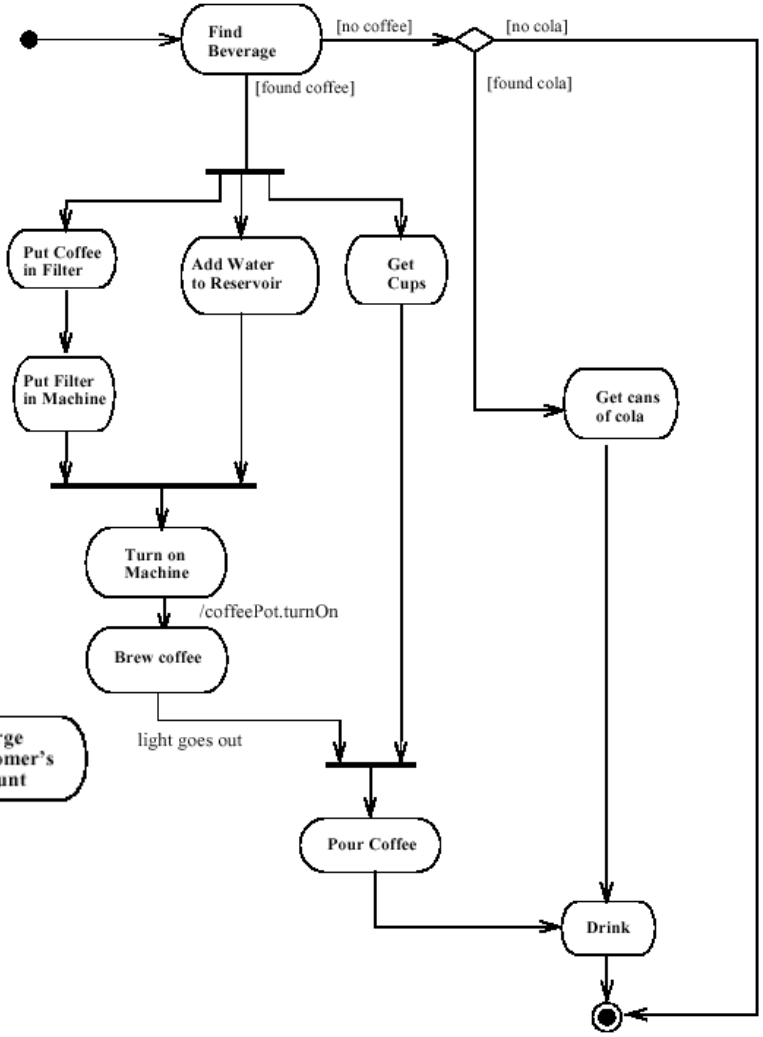
- Diagrama de Atividades
  - Variação de uma máquina de estados onde os estados representam o desenvolvimento de ações ou sub-atividades e as transições são disparadas pelo fato de se completar as ações ou sub-atividades
  - todo o diagrama de atividades está associado (pelo modelo) a uma classe, tal como um caso de uso ou um pacote ou uma implementação de uma operação
  - propósito deste diagrama é focalizar no fluxo dirigido por processos internos ao contrário de eventos externos
  - devem ser utilizados em situações onde todos ou a maioria dos eventos representam a conclusão de ações geradas internamente (fluxo de controle procedural)
  - quando eventos assíncronos puderem ocorrer, é melhor utilizar diagramas de estados

Diagramas de Atividades

- Exemplos de Diagramas de Atividades

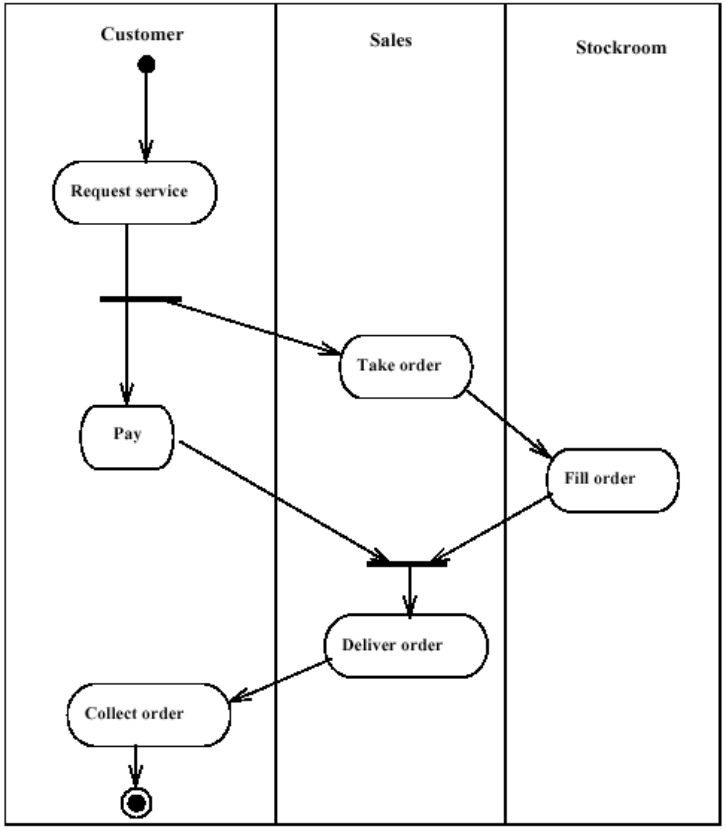


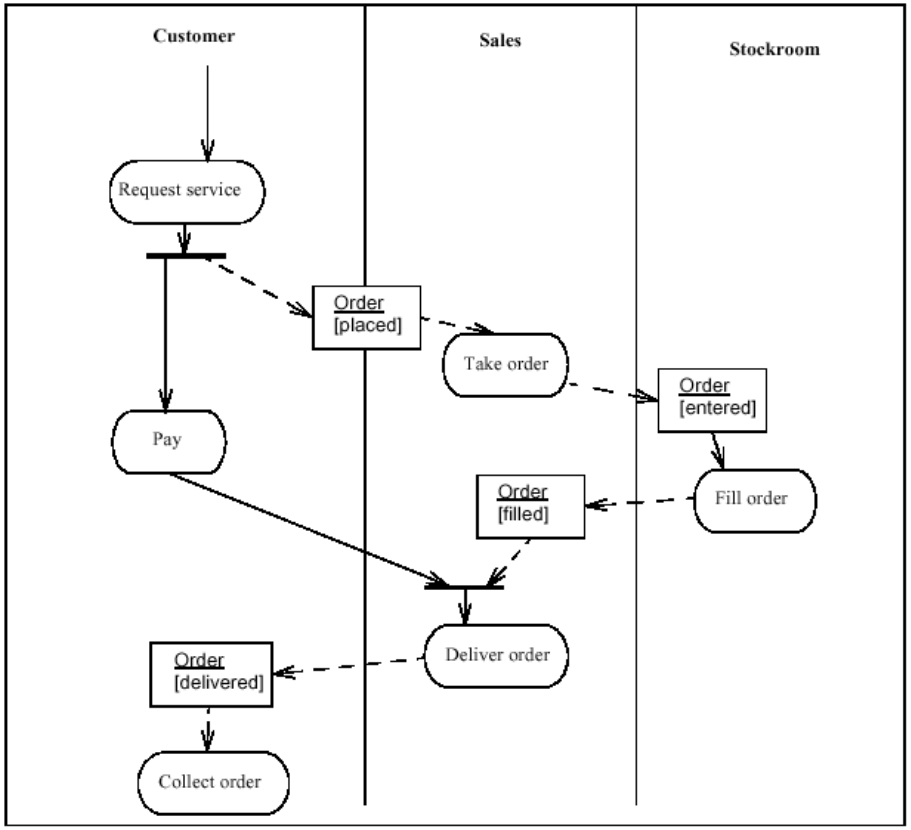




Diagramas de Atividades

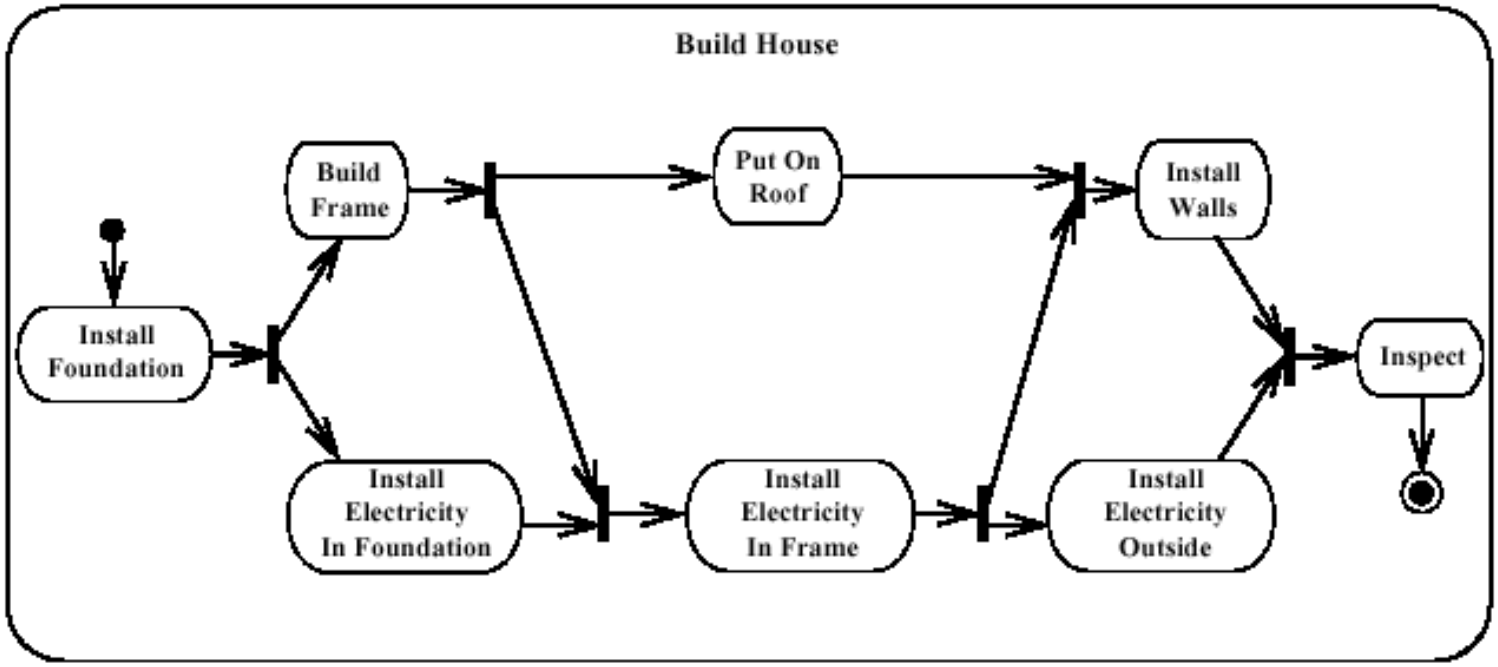
- Diagramas com Swimlanes





Diagramas de Atividades

- Diagramas com Sincronização entre Atividades Paralelas



Diagramas de Implementação

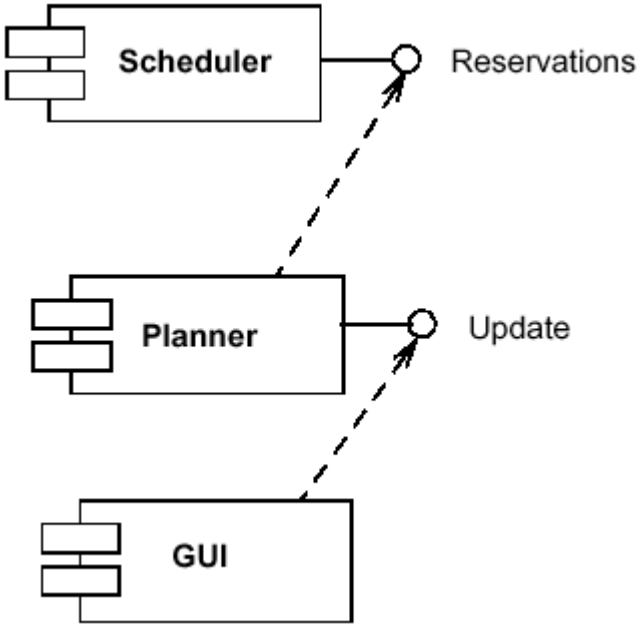
- Diagramas de Implementação
  - mostram aspectos de implementação, incluindo a estrutura do código fonte, bem como a estrutura de implementação em tempo de execução (componentes executáveis)
  - aparecem na forma de dois tipos de diagramas
    - diagramas de componentes - mostram a estrutura do código fonte
    - diagramas de distribuição - mostram a estrutura do código executável
  - ambos podem ser aplicados em um senso mais amplo a modelos de processos do mundo real onde *código fonte* seriam os procedimentos organizacionais e documentos e o *código executável* seriam as unidades organizacionais e recursos (humanos e outros) de uma organização

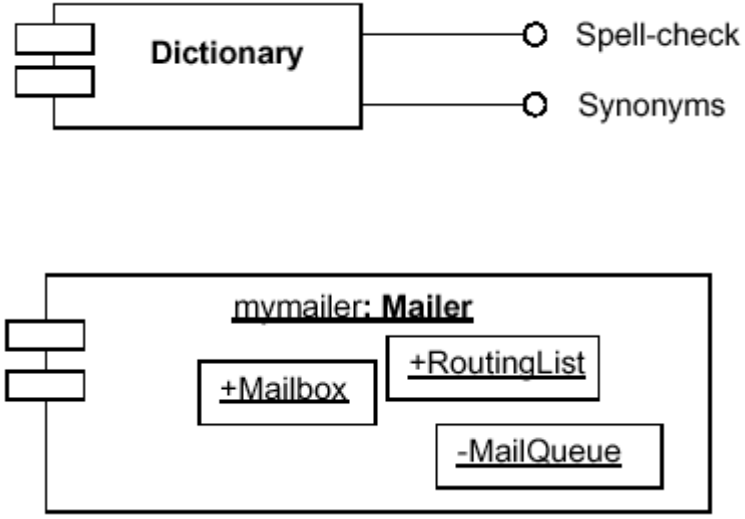
Diagrama de Componentes

- Diagrama de Componentes
  - mostra as dependências entre componentes de software, incluindo-se o código fonte, componentes de código binário e componentes executáveis
  - para um processo organizacional, “componentes de software” são abstraídos de modo a incluir procedimentos organizacionais e documentos
  - um módulo de software pode ser representado por meio de estereótipo do tipo componente
  - alguns componentes existem em tempo de compilação, outros existem em tempo de linkagem, outros existem em tempo de execução e outros existem em mais de um tempo
  - diagrama de componentes representam somente “tipos”, não “instâncias”

Diagramas de Componentes

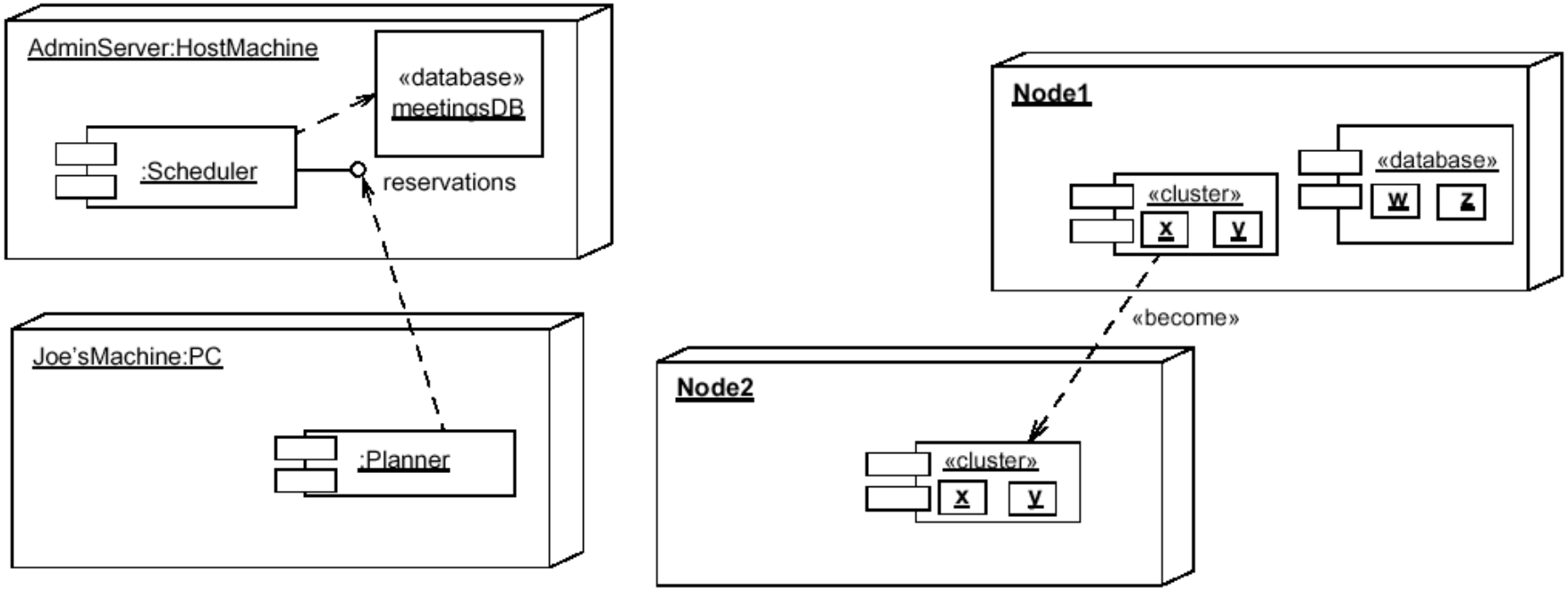
- Exemplos de Diagramas de Componentes





Diagramas de Distribuição

- Diagramas de Distribuição
  - mostram a configuração de elementos capazes de processar software (computadores, dispositivos periféricos, etc...) e como os componentes de software, processos e objetos se encontram distribuídos dentre eles
  - Componentes de software representam manifestações de tempo real de unidades de código - componentes que não existem como entidades de tempo real (e.g. por que são compilados durante a execução) não aparecem nestes diagramas, devendo ser mostrados em diagramas de componentes
  - para a modelagem de processos organizacionais, os "elementos capazes de processar software" podem ser abstraídos na forma de trabalhadores e unidades organizacionais, ao passo que os componentes de software representam os procedimentos e documentos utilizados pelos trabalhadores e unidades organizacionais

Diagramas de Distribuição

- Exemplo de Diagrama de Distribuição