MC302A
Modelagem de Sistemas com UML

Prof. Fernando Vanini
vanini@ic.unicamp.br

Modelamento de Sistemas e Orientação a Objetos

- O paradigma de Orientação a Objetos oferece um conjunto de características que se bem utilizadas apresentam inúmeras vantagens no projeto de um sistama.
- Tirar proveito dos recursos oferecidos pela Orientação a Objetos exige que se considere o sistema sendo projetado à luz dessas características.

UML

- UML (Unified Modeling Language) é uma notação *gráfica* que tem por objetivo o modelamento de sistemas baseados em objetos.
- Definida por “los tres amigos”: Booch, Rambaugh e Jacobson.
- Padronizada pelo OMG (Object Management Group).

UML

- UML é apenas uma notação. Para que o seu uso num projeto seja efetivo é necessário definir um *processo* ou *metodologia de projeto*.

Modelo

Um *modelo* é uma *abstração* da realidade.

- Exemplo: mapa de uma cidade
  - modelo que representa as ruas e como elas se conectam.
  - Através de um mapa é possível determinar o caminho para se chegar a um determinado ponto.
- O mesmo mapa, no entanto omite detalhes irrelevantes para o uso ao qual se destina.
- Exemplos:
  - tipo da pavimentação
  - localização e altura dos edifícios
  - inclinação do terreno.

Uso de Modelos em Projetos

- Modelos tem sido usados há muito tempo como ferramenta de projeto.
- Exemplo: Construção civil
  - maquetes
  - planta baixa
  - planta elétrica
  - planta hidráulica

Cada modelo enfatiza os aspectos importantes para os seus objetivos.

Uso de Modelos em Projetos

- Dependendo do projeto, vários tipos de modelo podem ser necessários, cada um enfatizando um conjunto de aspectos do projeto.
- O uso de vários modelos é necessário para que se possa descrever toda a complexidade do projeto.

Modelos e Visões

- Ao enfatizar um conjunto de aspectos do sistema, um modelo dá uma *visão* do sistema.
- A *descrição completa* do sistema pode exigir *várias visões* - daí a necessidade de vários tipos de modelos.

As visões em UML

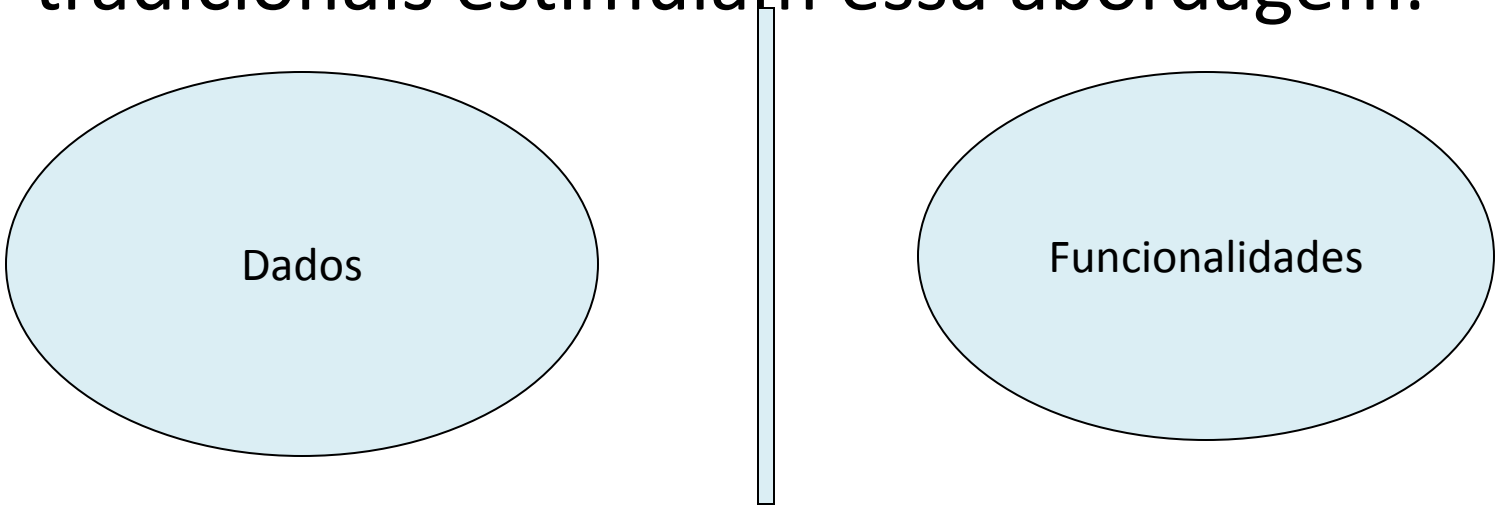
- As *visões* são usadas em UML para descrever os diferentes aspectos do sistema sendo modelado.
- Uma visão é uma abstração do sistema, formada por um conjunto de diagramas.
- A partir de um conjunto de visões se chega a uma descrição completa do sistema a ser construído

As visões em UML

- Visão de Casos de Uso - mostra a funcionalidade do sistema do ponto de vista externo.
- Visão Lógica - descreve a organização do sistema, seus módulos principais, como eles se relacionam e suas funcionalidades
- Visão de Componentes - descreve a arquitetura física do sistema, em termos de componentes de software
- Visão de Implantação – mostra a alocação dos componentes de software no equipamento alvo.

Metodologias Tradicionais

- Dados e funcionalidades são tratados como elementos independentes.
- As linguagens e ambientes de programação tradicionais estimulam essa abordagem.



Metodologias Tradicionais

- Um exemplo

Funcionário

nome
endereço
registro
cargo
salário
situação
data de admissão

Dados descrevem características estáticas do funcionário. Características dinâmicas (relativas ao negócio) são expressas pelos programas que implementam as funcionalidades

Diferentes aplicações que utilizem os mesmos dados devem implementar as 'características dinâmicas' da mesma maneira, para que não ocorram inconsistências.

Objetos

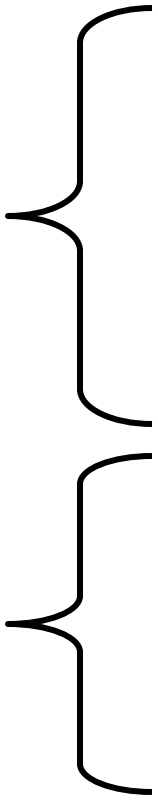
- Objetos reunem num único elemento as *características estáticas* e *comportamentais*.
- Características estáticas são representadas como *dados* associados ao *objeto*. Elas descrevem o *estado* atual do objeto.
- Características dinâmicas são descritas através de operações ou *métodos* executados pelo objeto.
- *Estado* e *comportamento* são partes integrantes do objeto.

Objetos

- Um exemplo

Estado: dados (ou atributos) que descrevem o 'estado atual' do objeto

Comportamento: métodos que descrevem os aspectos dinâmicos do objeto.



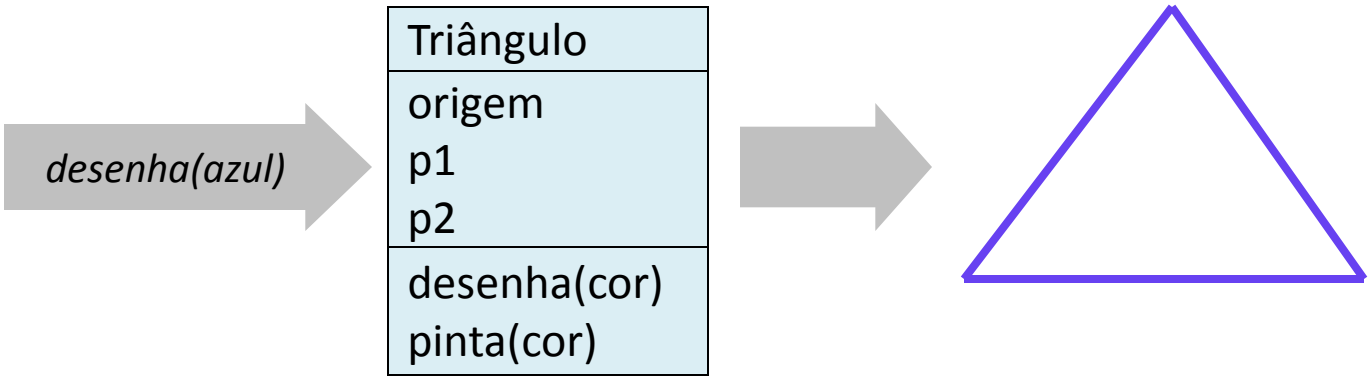
Funcionário
nome endereço registro cargo salario_bruto situação data de admissão
calcular_salário_líquido() atualizar_situação() alterar_cargo() calcular_FGTS() verificar_saldo_férias()

Encapsulamento

- Um objeto *encapsula* numa única entidade o seu estado e os métodos que definem o seu comportamento.
- Consequência
  - acoplamento mais fraco entre as diversas partes do sistema
  - menor probabilidade de interferências espúrias entre as partes do sistema

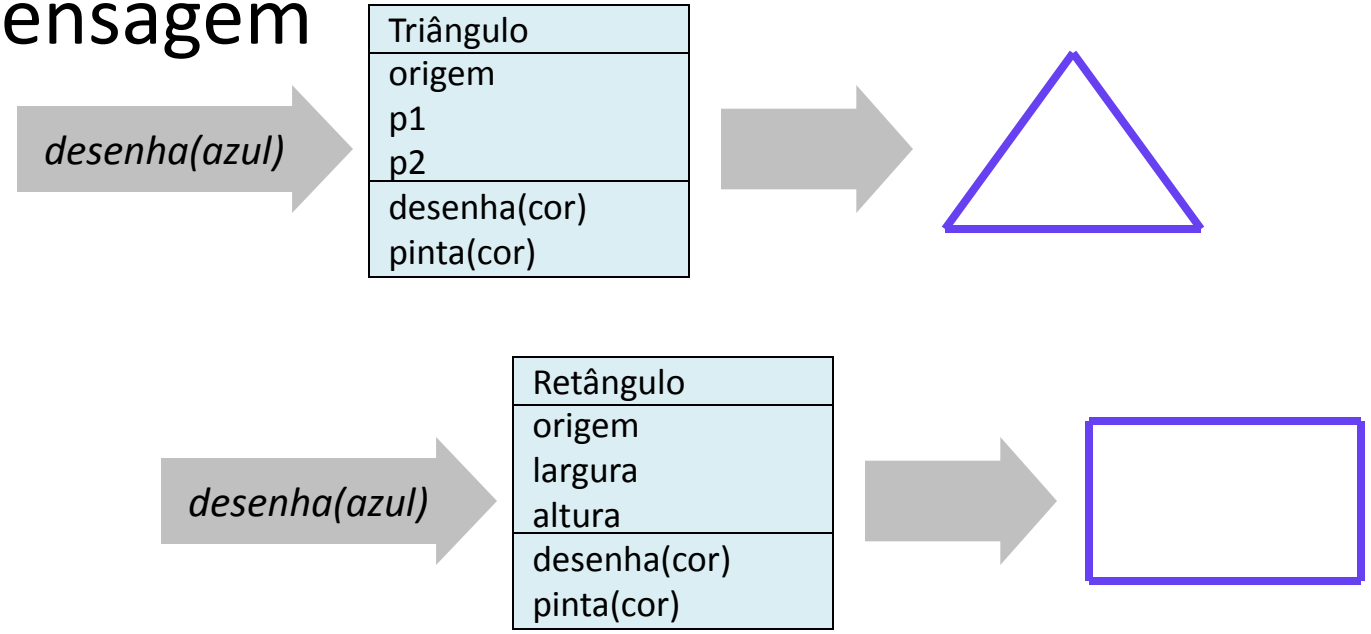
Métodos e mensagens

- A execução de um *método* é disparada pelo envio de uma *mensagem* ao objeto.
- A execução do método é a *interpretação* que o objeto dá à mensagem.



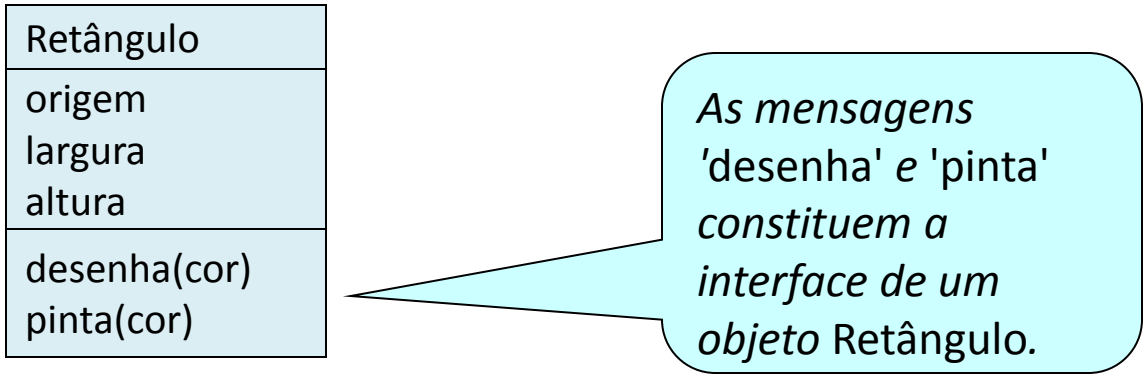
Métodos e mensagens

- Objetos diferentes podem dar interpretações diferentes a uma mesma mensagem



Interface

- A *interface* de um objeto é formada pelo conjunto de mensagens tratadas pelo objeto.

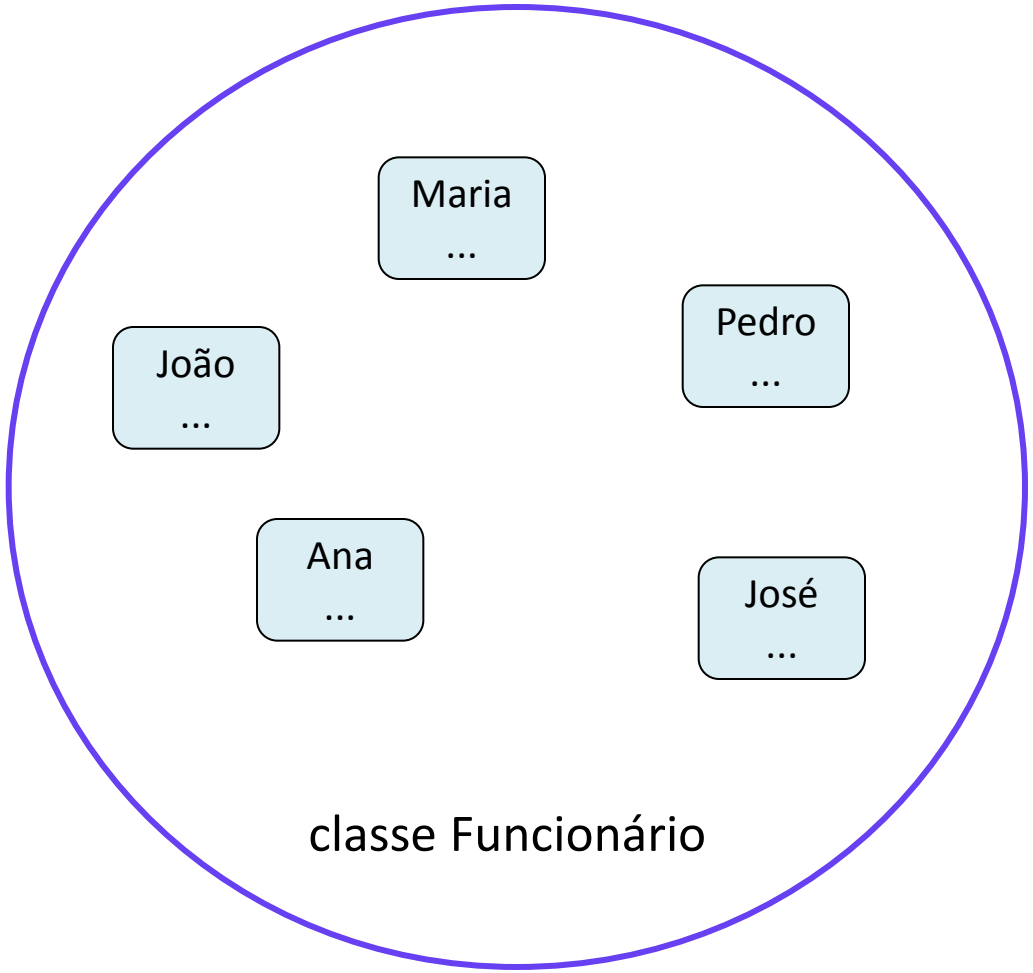


Objetos e Classes

- Um*objeto* é um *conceito*, *abstração* ou algo que tenha um significado bem definido para o problema em questão
- uma *classe* descreve um *grupo de objetos* com o mesmo *conjunto de propriedades* (atributos), os mesmos *relacionamentos* com outros objetos e a mesma *semântica* (operações ou métodos)

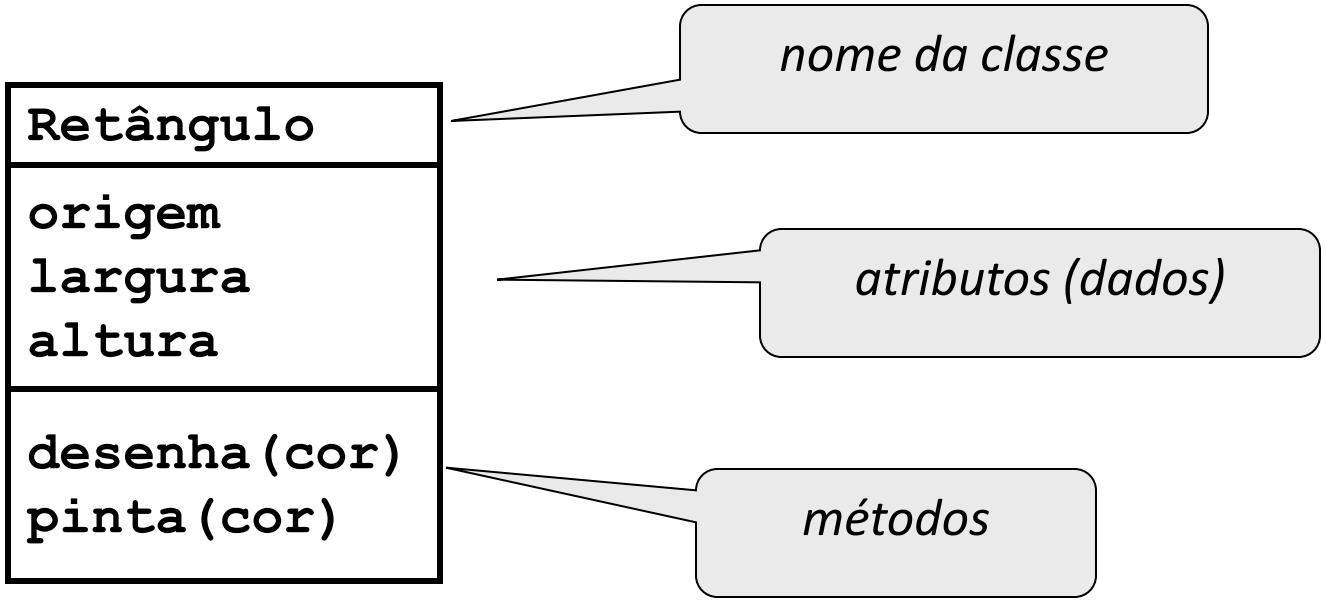
Um objeto é uma *instância* (ou um exemplar) de uma classe.

Objetos e Classes



- Todos os objetos da classe funcionário oferecem a mesma interface e mantêm o mesmo conjunto de atributos.
- Cada instância no entanto tem os seus próprios valores para os cada um dos atributos

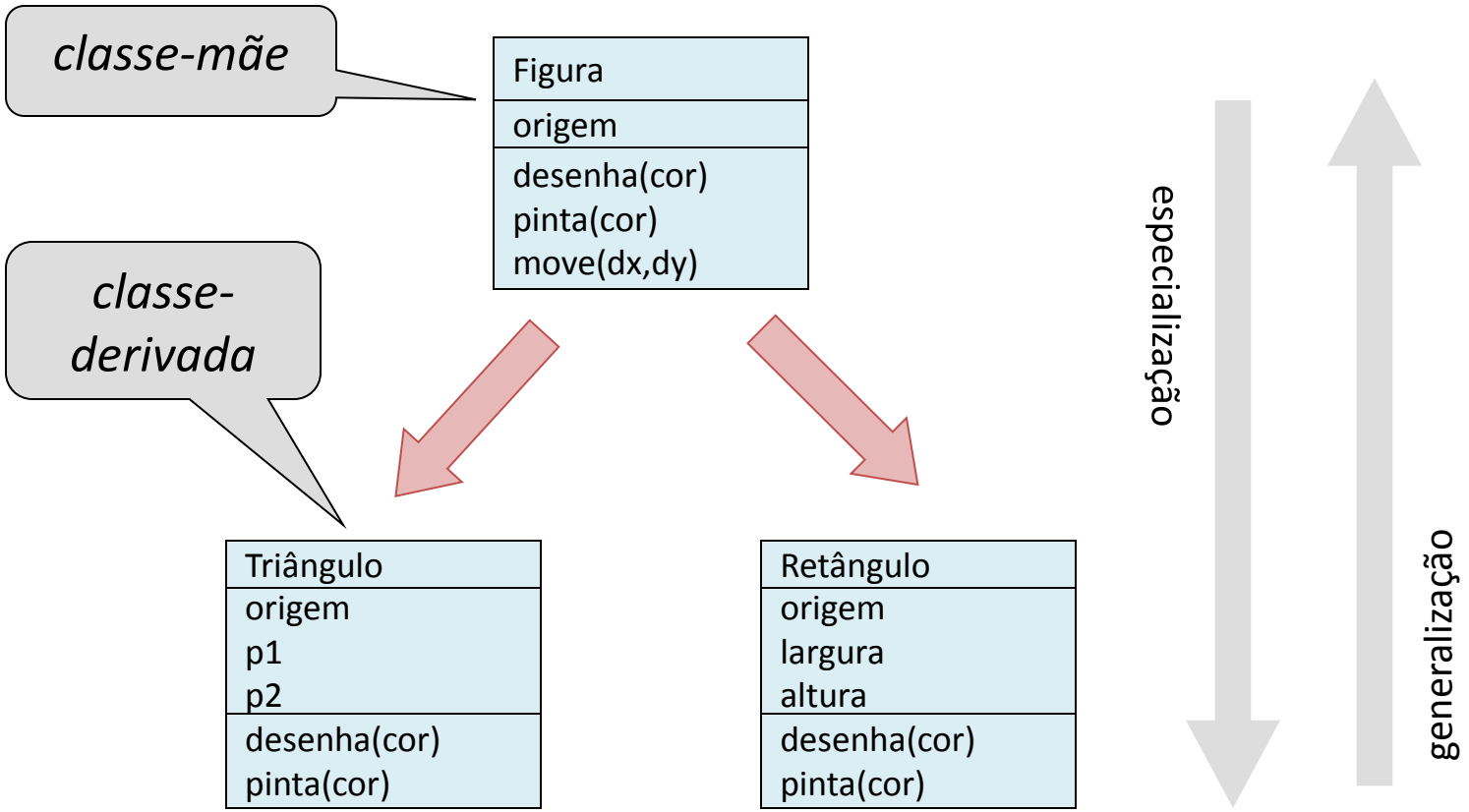
Classes em UML



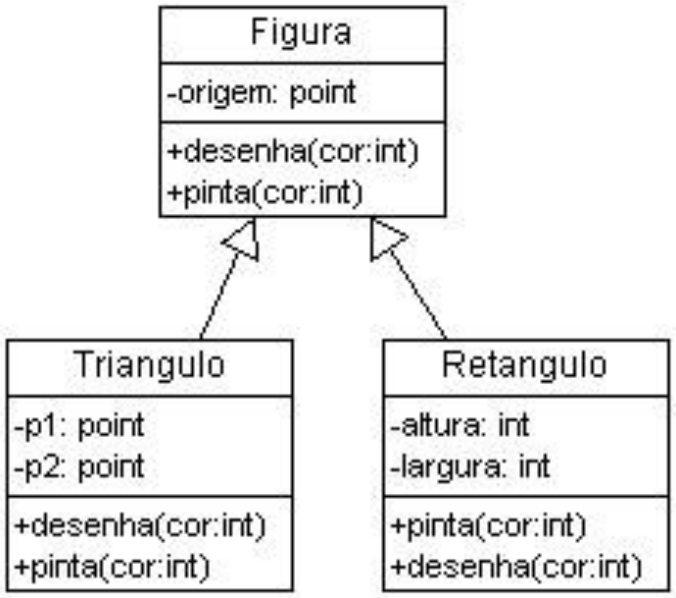
Herança

- Uma classe pode dar origem a outras classes através de *herança* (ou *especialização*).
- Os objetos da *classe derivada* herdam da classe origem (ou *classe-mãe*) todos os atributos e métodos.
- A classe derivada pode estender a classe-mãe agregando novos atributos e métodos. Ela pode também redefinir métodos da classe-mãe.

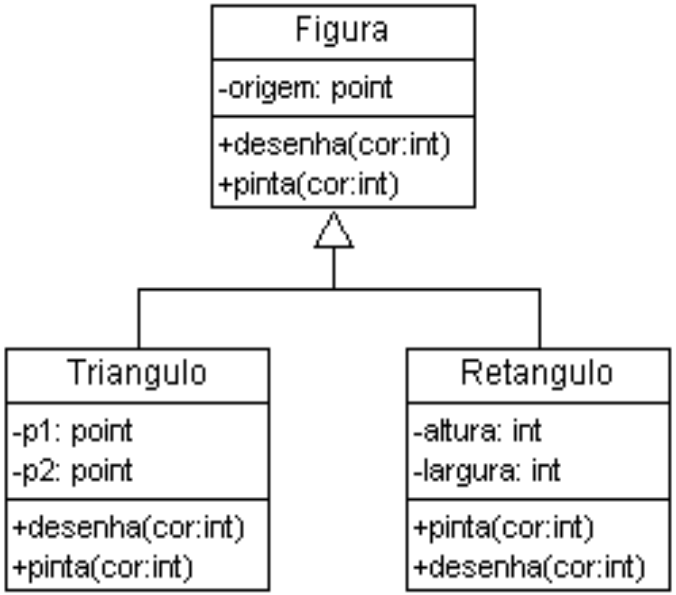
Herança


classe-mãe
Figura
origem
desenha(cor)
pinta(cor)
move(dx,dy)
classe-derivada
Triângulo
origem
p1
p2
desenha(cor)
pinta(cor)
Retângulo
origem
largura
altura
desenha(cor)
pinta(cor)
especialização
generalização

Herança em UML

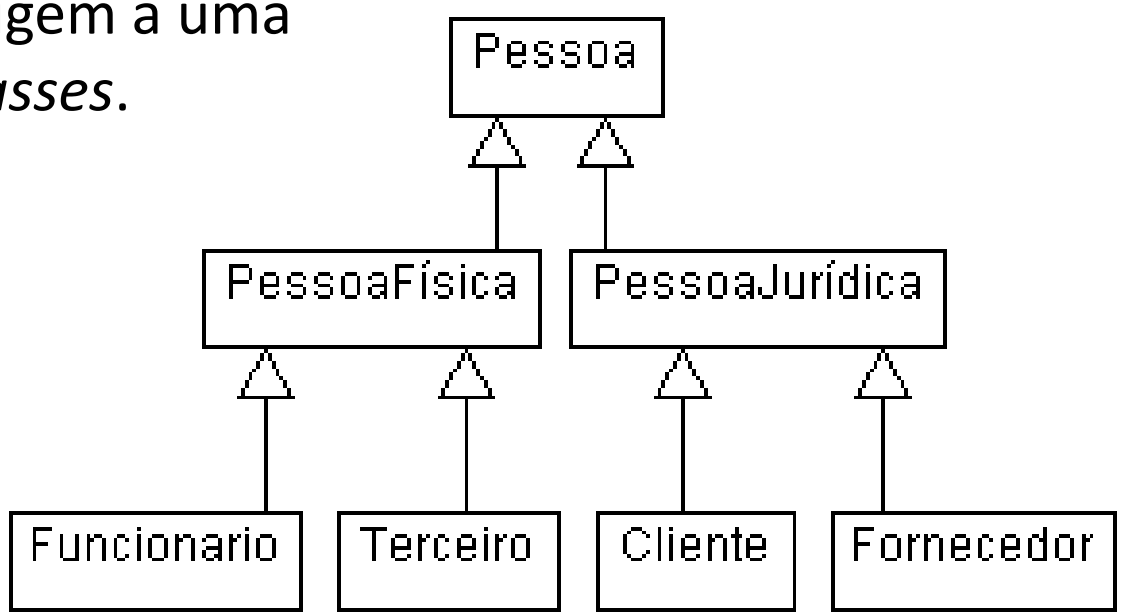


ou



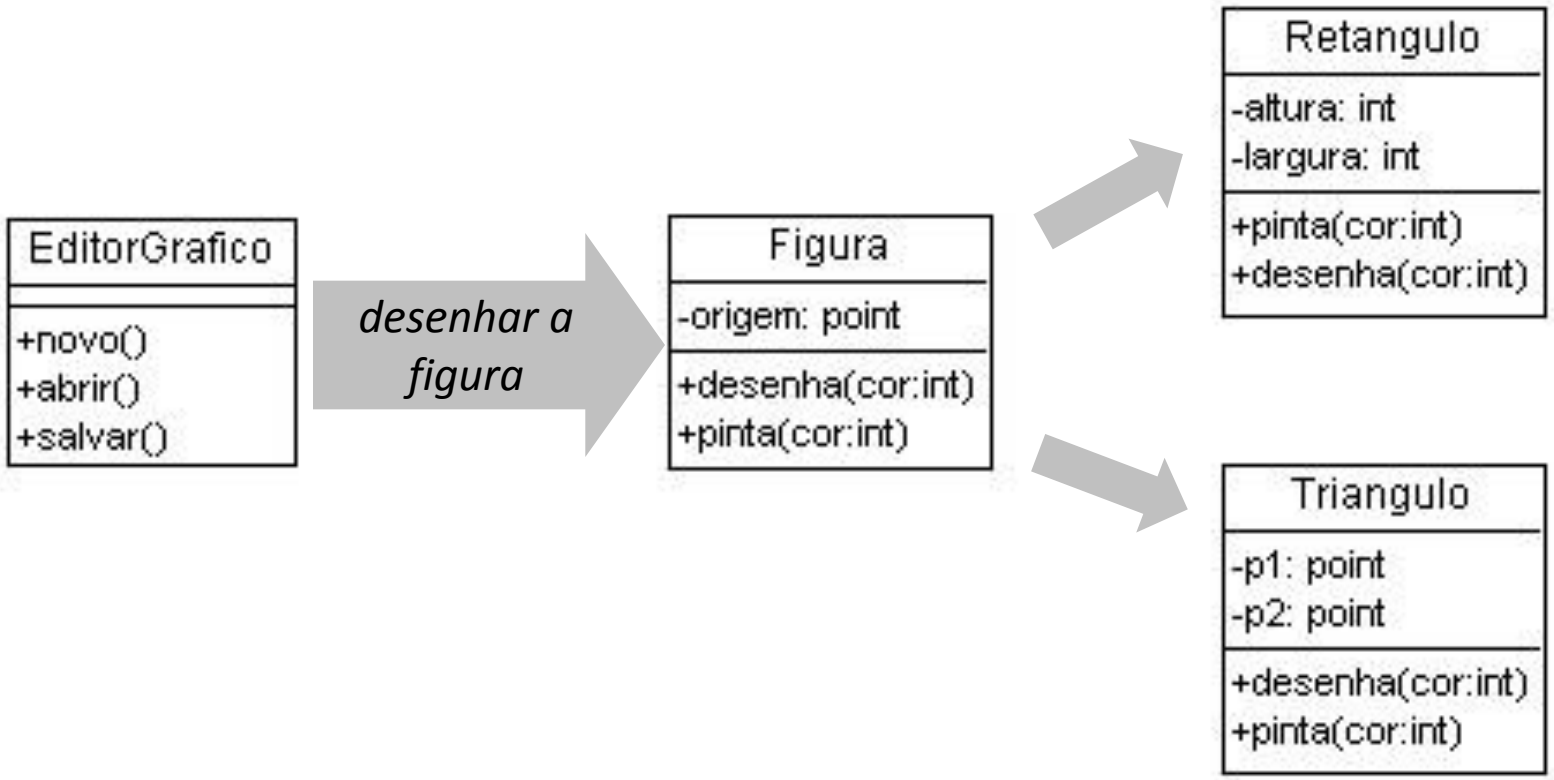
Hierarquia de Classes

- Herança também se aplica a classes derivadas de outras
- isso pode dar origem a uma *hierarquia de classes*.



Polimorfismo

- Numa situação onde se espera um objeto de uma determinada classe C é sempre possível utilizar um objeto de uma classe derivada de C.



Os pilares da OO

- Encapsulamento, herança e polimorfismo constituem os pilares da orientação a objetos.
- Encapsulamento
  - melhora a modularidade do sistema
  - disciplina as interfaces
  - reduz interferências entre as partes
- Herança e polimorfismo
  - fatoração de elementos comuns
  - favorece o *reuso*

Objetos e o 'mundo real'

- Objetos têm se mostrado convenientes para descrever elementos e situações do *mundo real*.
- Por essa razão, quando se usa objetos para modelar um sistema, é possível partir de conceitos pertencentes ao *domínio do problema*.
- Os detalhes no *domínio da solução* podem ser postergados até o momento em que se tenha uma idéia clara do que o sistema deve fazer.

Domínio do Problema

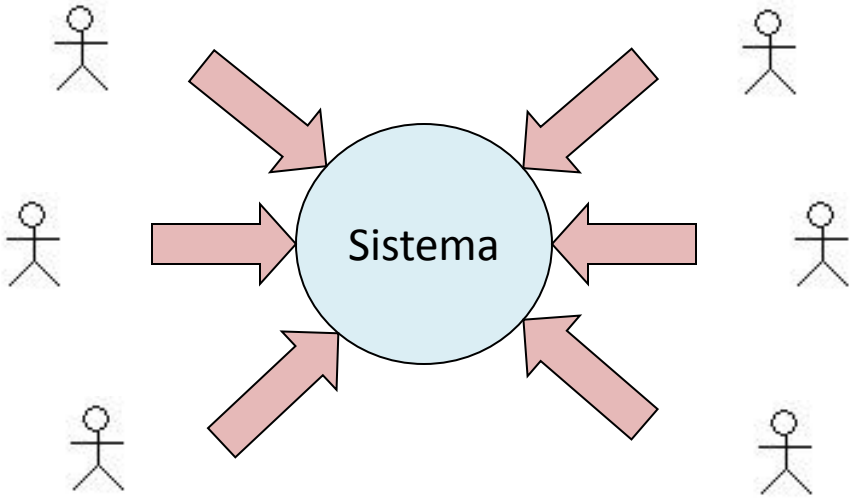
- Os conceitos no domínio do problema tendem a se manter estáveis ao longo da vida do sistema.
- Exemplo: num sistema de administração escolar
  - aluno, sala, turma, professor, disciplina, curso
- Um sistema cuja arquitetura é modelada com base nesses conceitos tende a suportar melhor a evolução (necessária em qualquer sistema).

As visões em UML

- Visão de Casos de Uso - mostra a funcionalidade do sistema do ponto de vista externo.
- Visão Lógica - descreve a organização do sistema, seus módulos principais, como eles se relacionam e suas funcionalidades
- Visão de Componentes - descreve a arquitetura física do sistema, em termos de componentes de software
- Visão de Implantação – mostra a alocação dos componentes de software no equipamento alvo.

Visão de Casos de Uso

- Descreve a funcionalidade que o sistema deve oferecer, do ponto de vista do *mundo externo*.
- O *mundo externo* é representado por um conjunto de *atores* ou *atores externos* que interagem com o sistema.



Visão de Casos de Uso - atores

- Exemplo: num sistema de automação comercial, os *atores* seriam
  - *clientes*
  - *operadores de caixa*
  - *operador de crédito*
  - *computadores* do sistema bancário e administradoras de cartão de crédito

  - ator em UML:

Visão de Casos de Uso

- Os casos de uso são levantados a partir da idéia de necessidade do sistema, antes mesmo de se pensar numa arquitetura para o mesmo.
- Eles são a base do processo uma vez que dirigem o desenvolvimento das demais visões.

Visão de Casos de Uso

- O objetivo final do sistema é oferecer a funcionalidade descrita pelos casos de uso.
- Sendo assim, a visão dos casos de uso é importante também na validação do sistema.

Visão de Casos de Uso

- Casos de uso podem ser descritos através de uma descrição textual, e/ou visualmente através dos Diagramas de Casos de Uso.

Descrição do caso de uso

- Para cada caso de uso deve conter:
  - Atores que participam do caso de uso
  - O que caracteriza o início e o fim do caso de uso
  - A interação 'do sistema' com os atores
  - Quais os dados necessários à sua realização
  - Sequência normal de eventos
  - Sequências alternativas (exceções)

Em geral a metodologia de desenvolvimento especifica a estrutura da descrição do caso de uso.

Diagrama de Casos de Uso

Os diagramas de caso de uso têm por objetivo a descrição dos mesmos de forma visual.

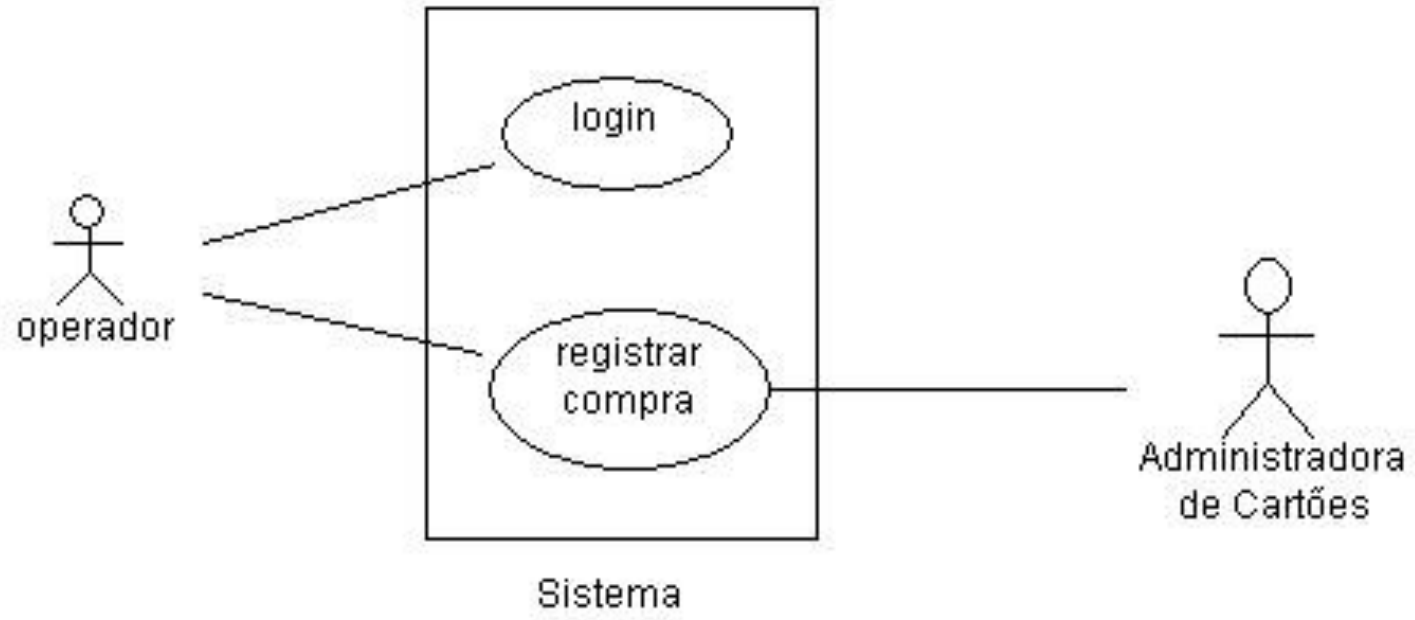
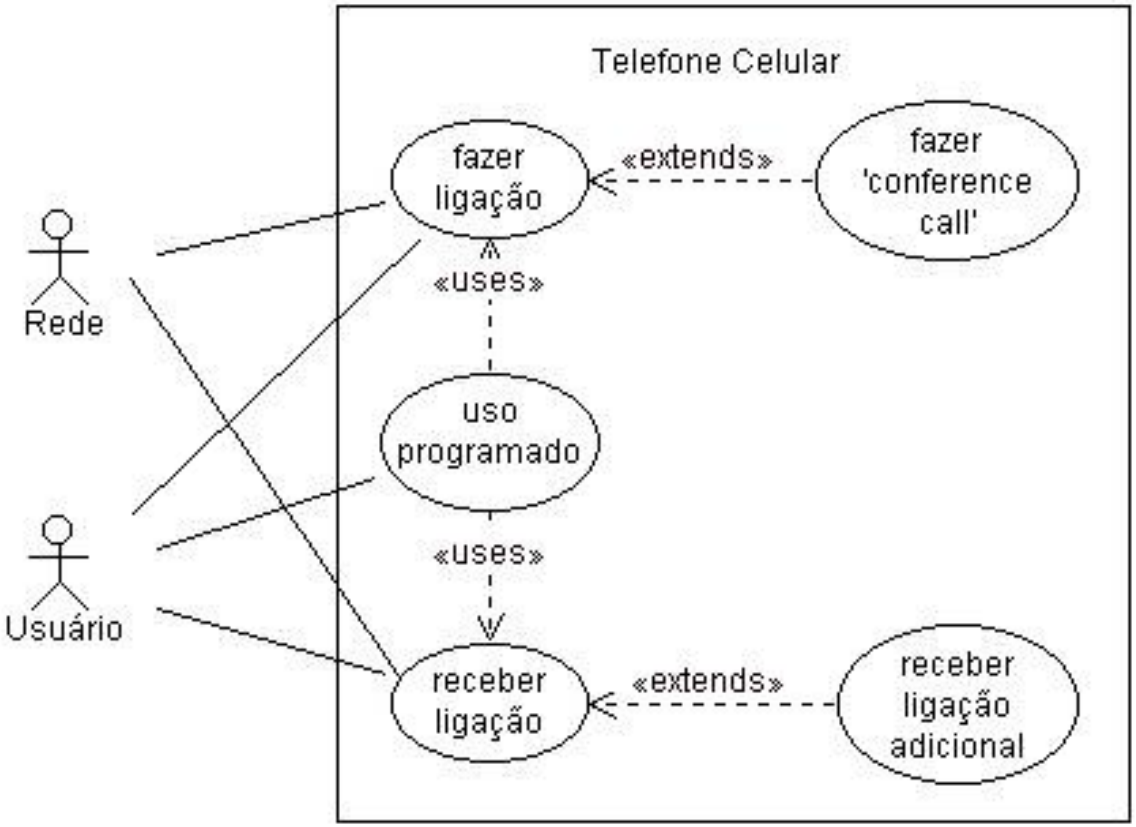
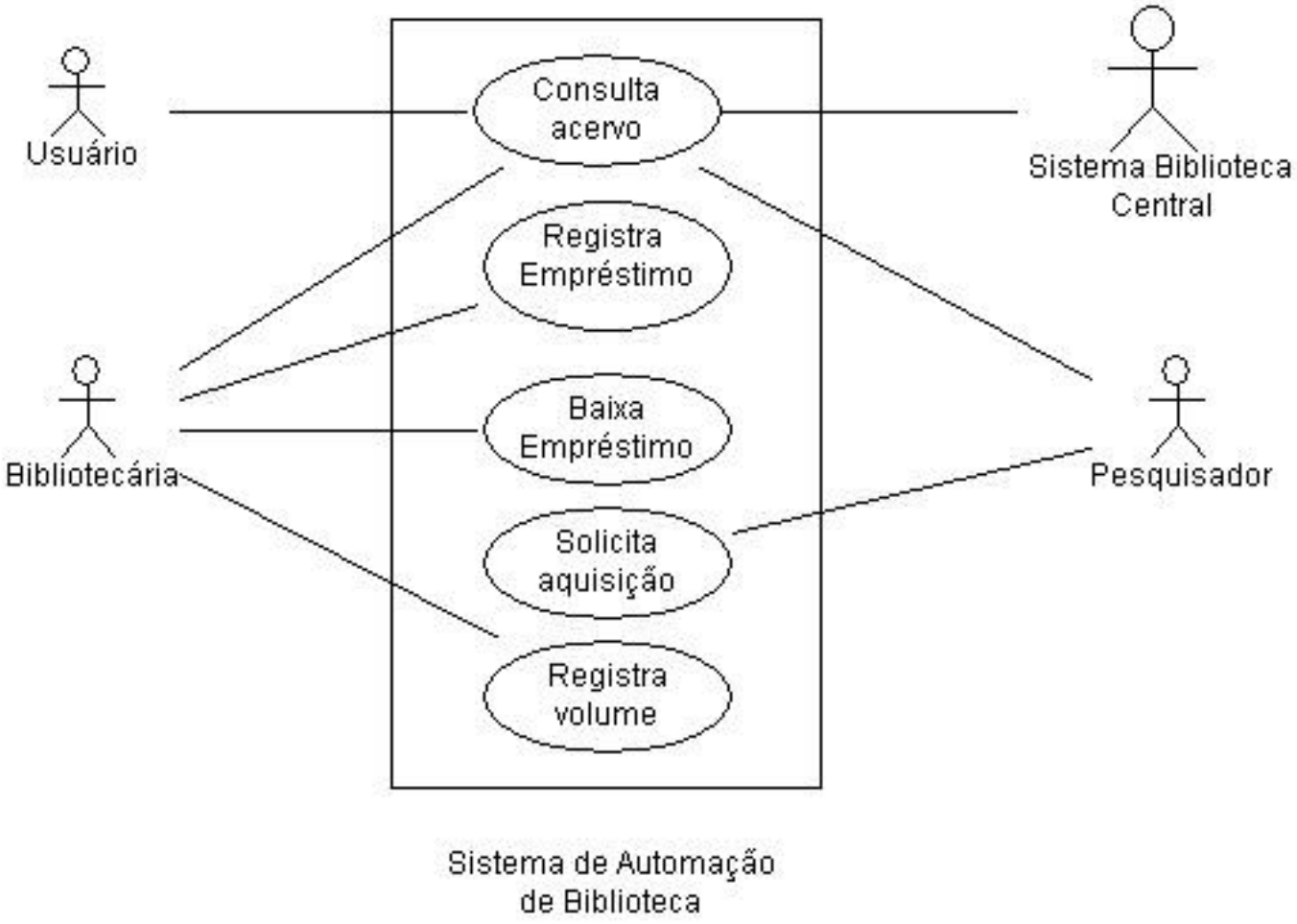


Diagrama de Casos de Uso

- Relacionamento entre casos de uso



Casos de Uso - exemplo



Casos de Uso

- Os diagramas de caso de uso têm por objetivo descrever graficamente as situações de uso do sistema.
- Eles podem não ser suficientes para descrever todos os detalhes da situação de uso necessários às fases subseqüentes do projeto.
- A descrição textual dos casos de uso em geral é necessária para descrever esses detalhes.

Visão Lógica

- A visão de casos de uso descreve o sistema do ponto de vista do mundo externo a ele.
- A *visão lógica* tem por objetivo descrever *como* a funcionalidade do sistema será obtida.
- A visão lógica descreve o sistema do ponto de vista *interno*.

Visão Lógica

- A visão lógica envolve
  - *estrutura estática* do sistema (módulos, classes, objetos e relacionamentos)
  - *aspectos dinâmicos* (manifestados através de mensagens entre objetos, eventos externos, etc).

Visão Lógica

- A estrutura estática é descrita através de *diagramas de classes* e *diagramas de componentes*.

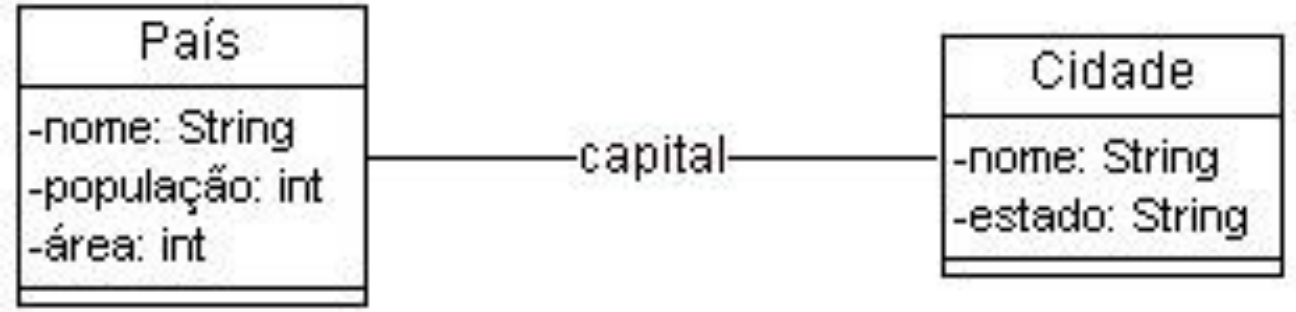
- O modelamento dinâmico é feito através de *diagramas de sequência*, *diagramas de colaboração* e *diagramas de estado*.

Relacionamentos entre classes

- Num sistema, as classes podem ter diversos tipos de relacionamentos entre si.
- *Herança* é um dos tipos possíveis de relacionamento entre classes.
- Uma *associação* é o tipo mais comum desses relacionamentos.
- Uma *associação* é uma "conexão" bidirecional entre duas classes.

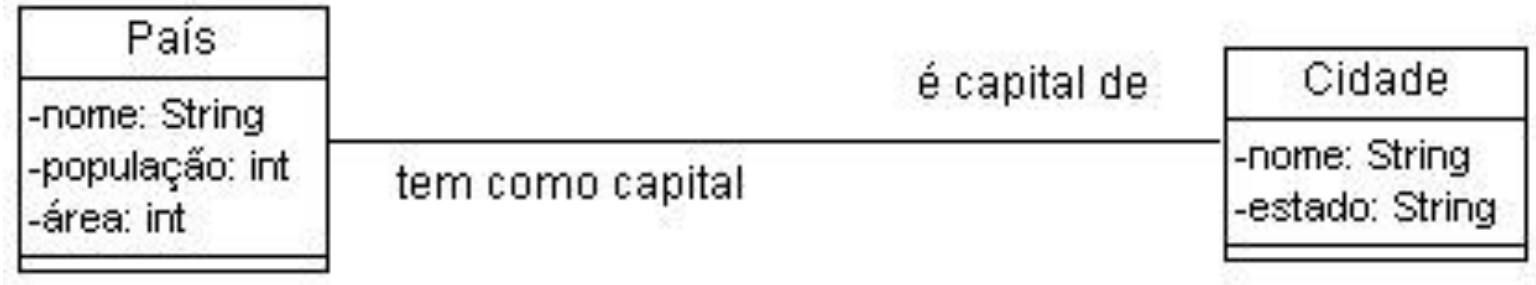
Associação

- Um exemplo: supondo que *país* e *cidade* são classes de um sistema, um país está associado a uma cidade que é a sua capital.



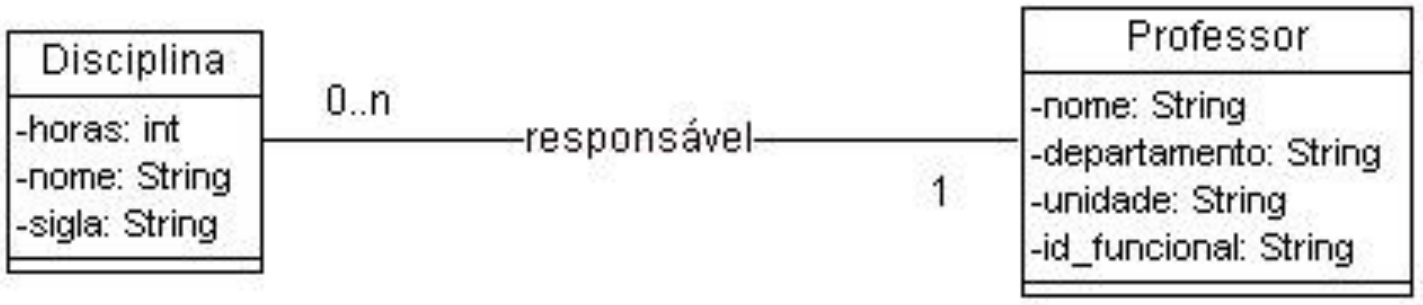
Relacionamentos e “papéis”

- No *relacionamento* entre duas classes, cada uma desempenha um papel específico.
- Esse papel, sempre que possível, deve ser explicitado no modelo.



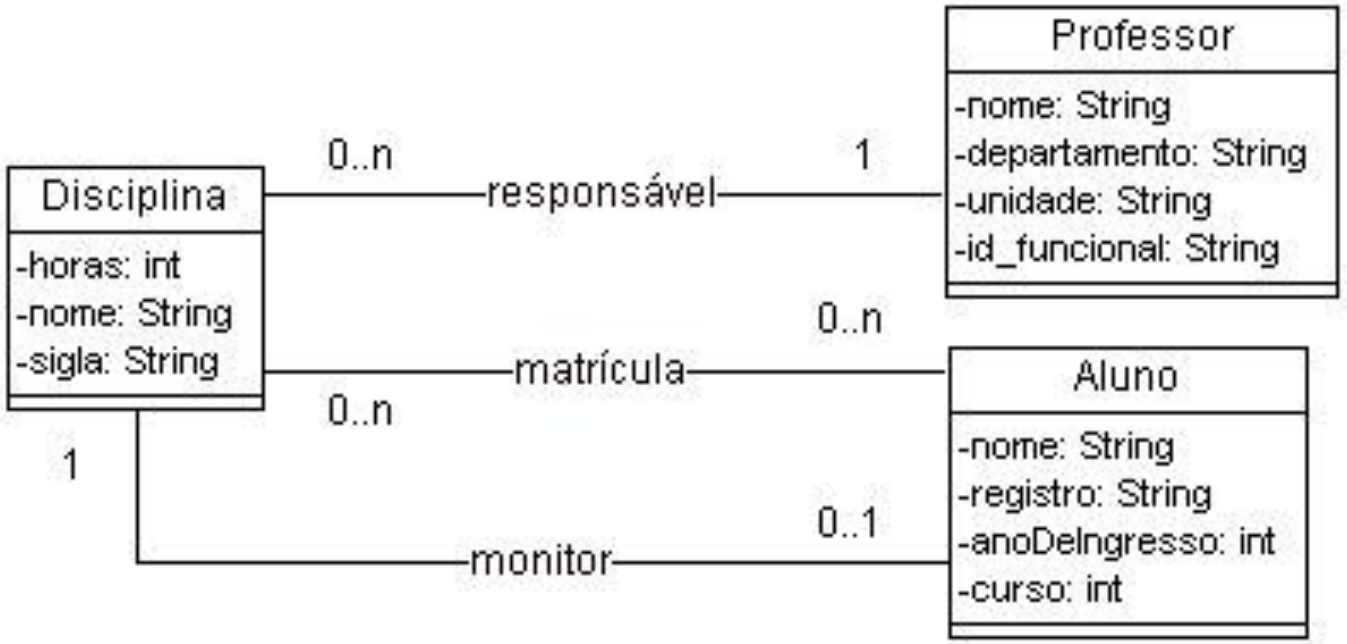
Multiplicidade (cardinalidade)

- Numa associação, a participação de cada classe pode ser associada a uma indicação de multiplicidade.
  - Exemplo: uma disciplina tem um único professor responsável e este por sua vez pode ser responsável por várias disciplinas.



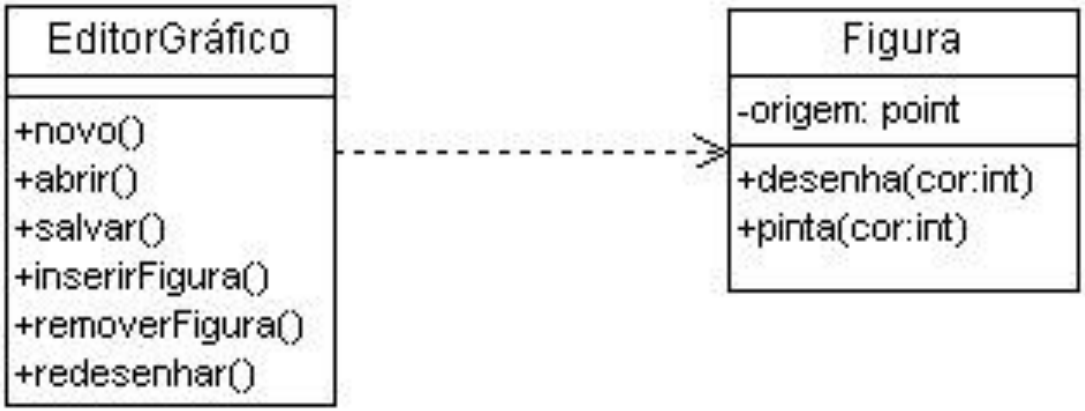
Associação

- Uma classe pode estar envolvida em mais de uma associação.



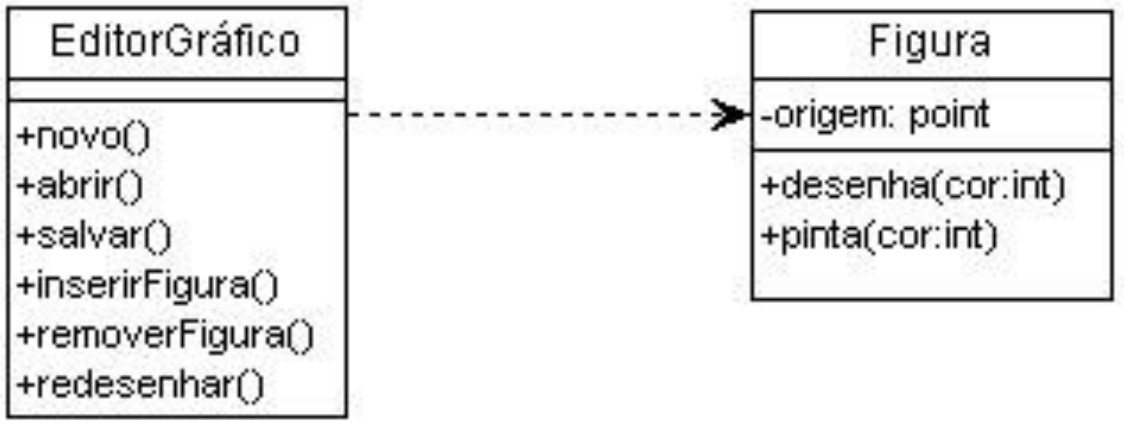
Dependência

- Uma classe pode fazer uso de outras classes
  - como tipo de um de seus atributos
  - como tipo de um dos parâmetros de seus métodos
  - na implementação de seus métodos



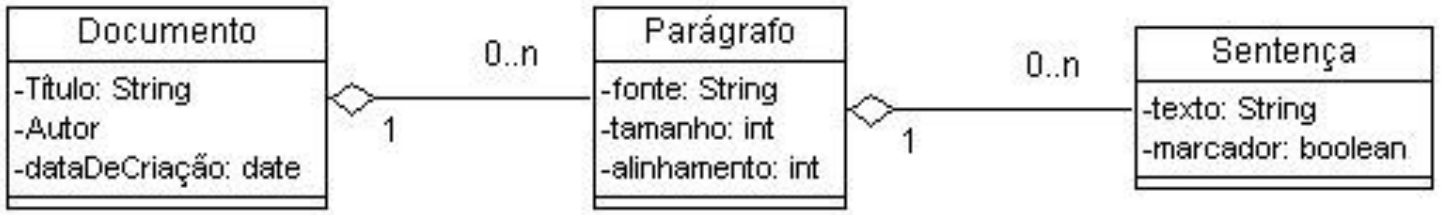
Instanciação

- Um objeto de uma classe, ao executar seus métodos pode criar instâncias de objetos de outras classes.



Agregação

- Descreve o relacionamento da forma "todo-parte" entre classes.



  - Um parágrafo, no exemplo acima, só faz sentido se estiver agregado a um documento (não existem 'parágrafos soltos'). O mesmo vale para sentença, com relação a parágrafo.

Agregação bi-direcional

- A agregação usada nos exemplos anteriores pressupõe que a sua implementação é bi-direcional
  - No exemplo, um documento consegue identificar os seus parágrafos assim como um parágrafo consegue identificar o documento ao qual pertence.

Agregação uni-direcional

- Em muitos casos, durante a modelagem se deseja deixar explícito que a agregação é uni-direcional.

  - Neste exemplo, um documento consegue identificar os seus parágrafos mas um parágrafo não é capaz de identificar o documento do qual é parte.

Agregação - outro exemplo

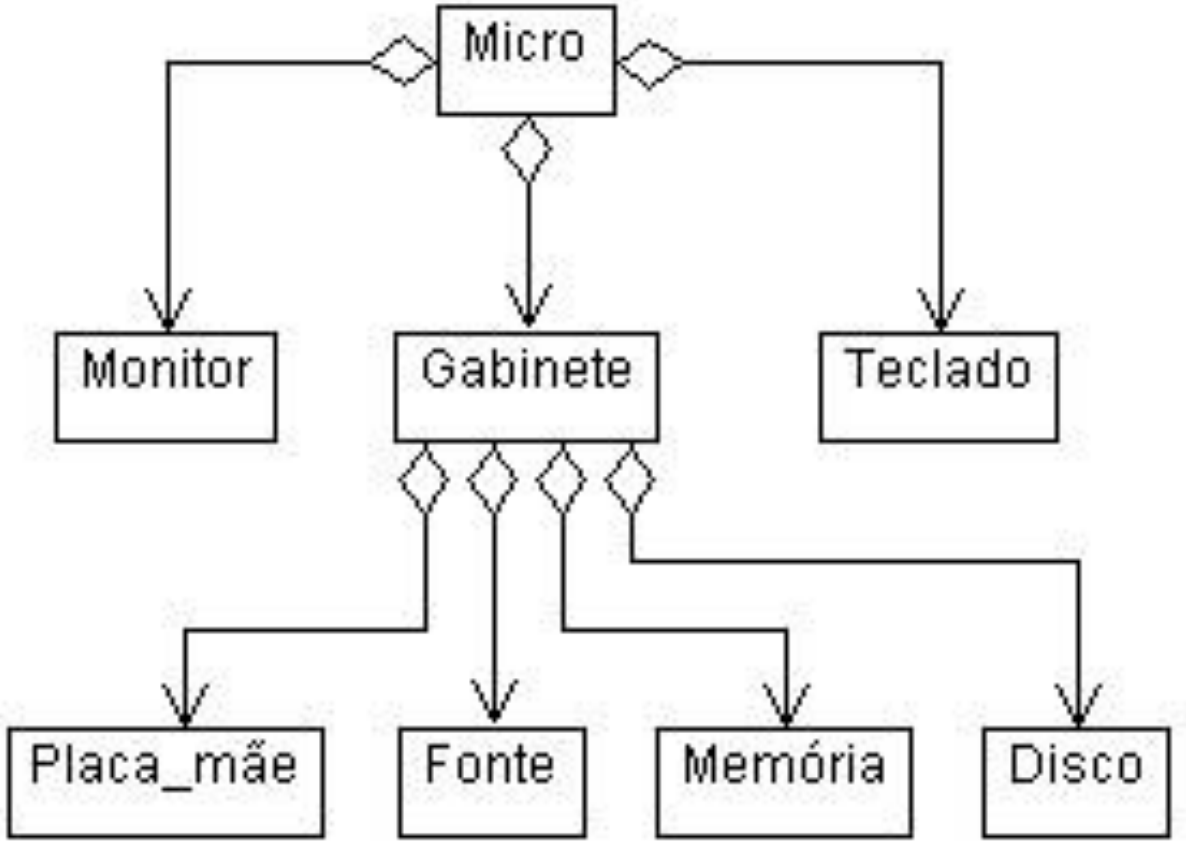


Diagrama de Classes

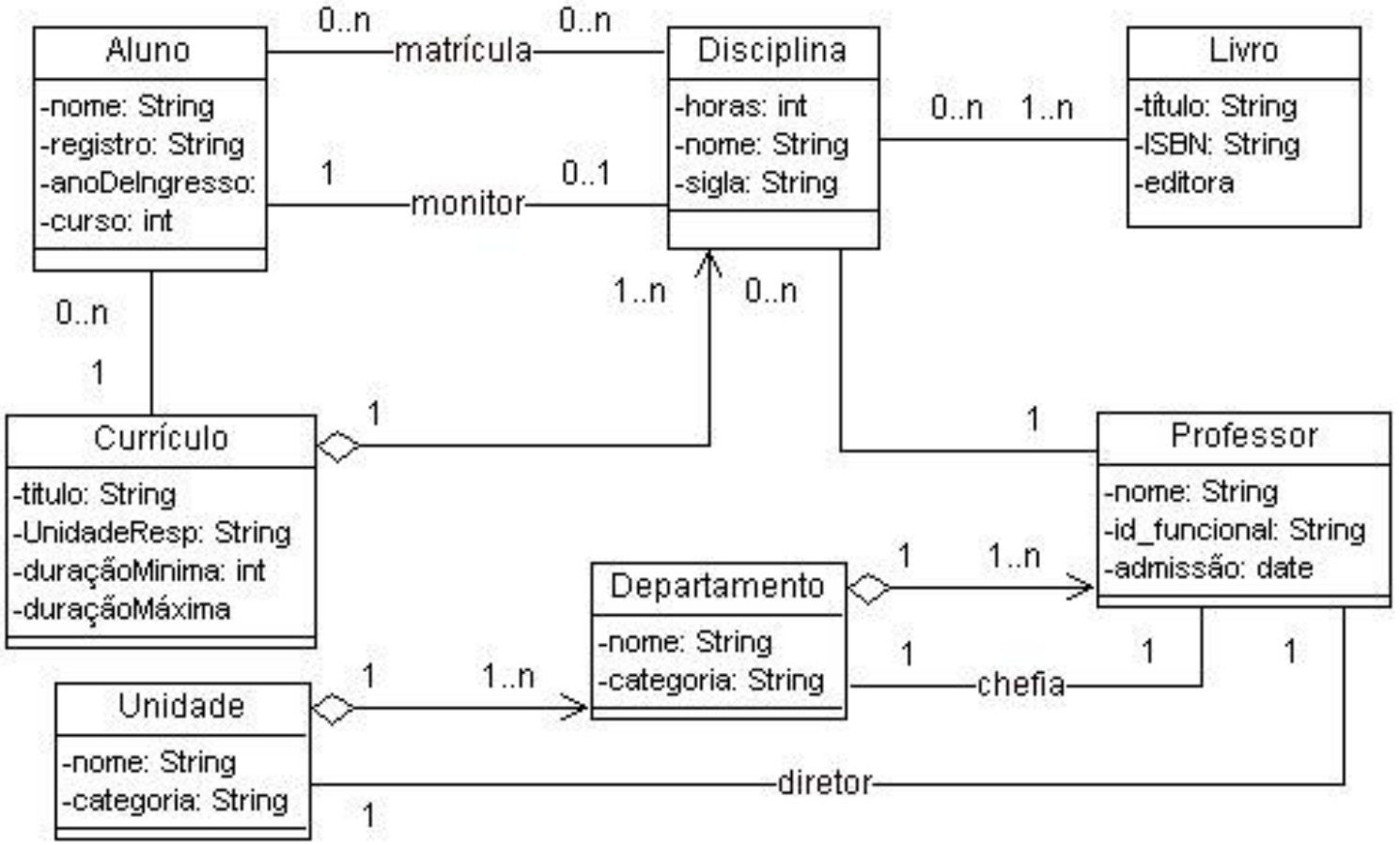


Diagrama de Classes - Exemplo

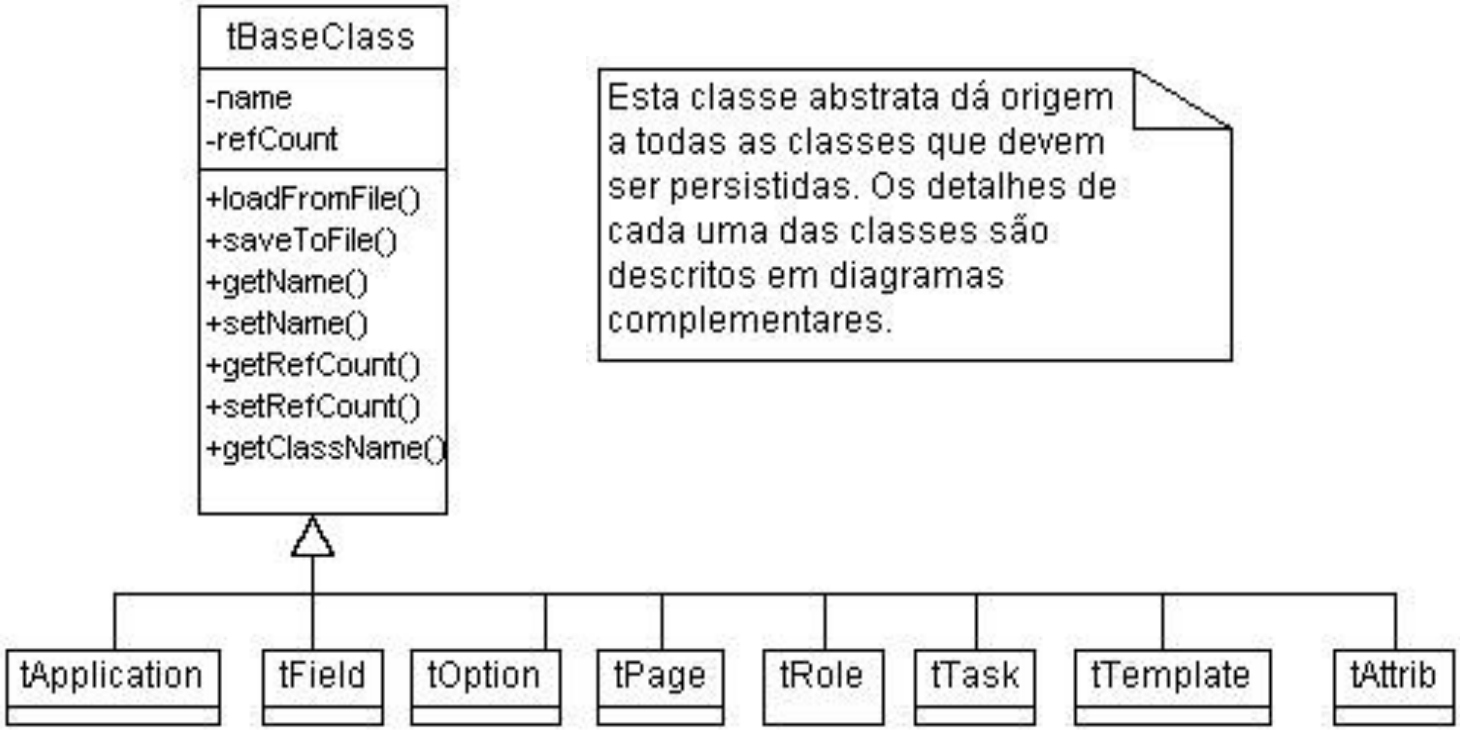


Diagrama de Classes - Exemplo

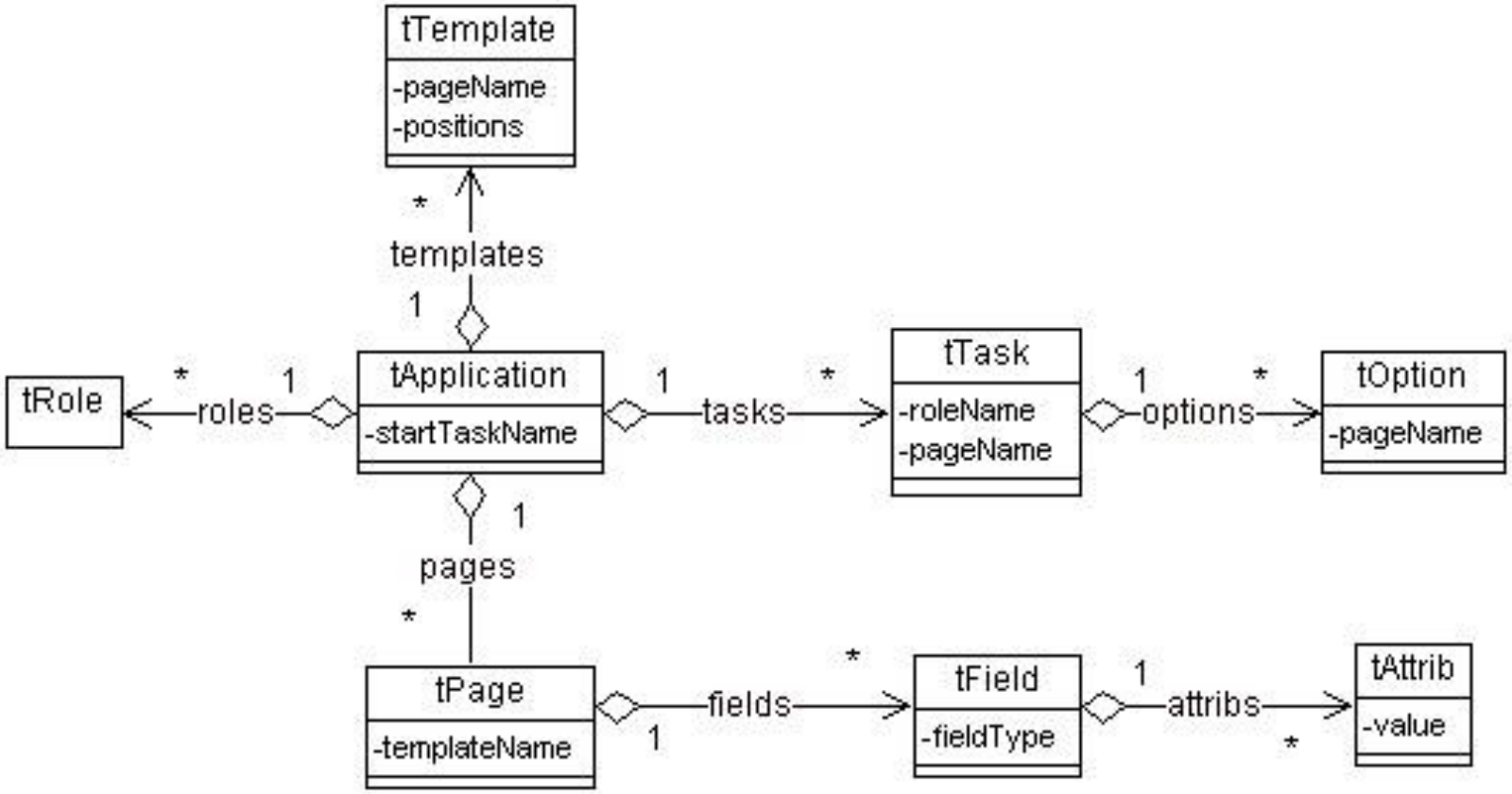


Diagrama de Classes - Exemplo

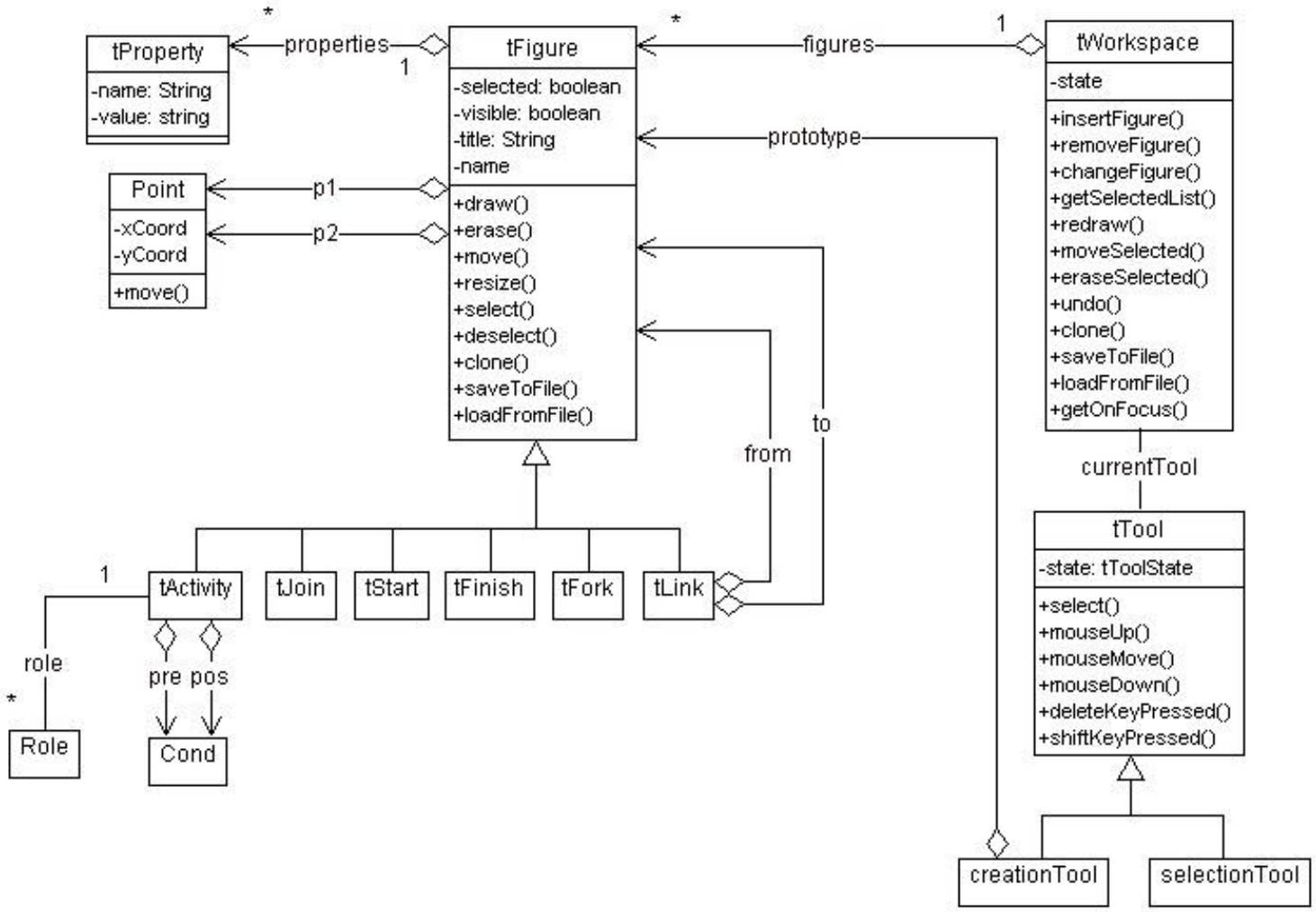
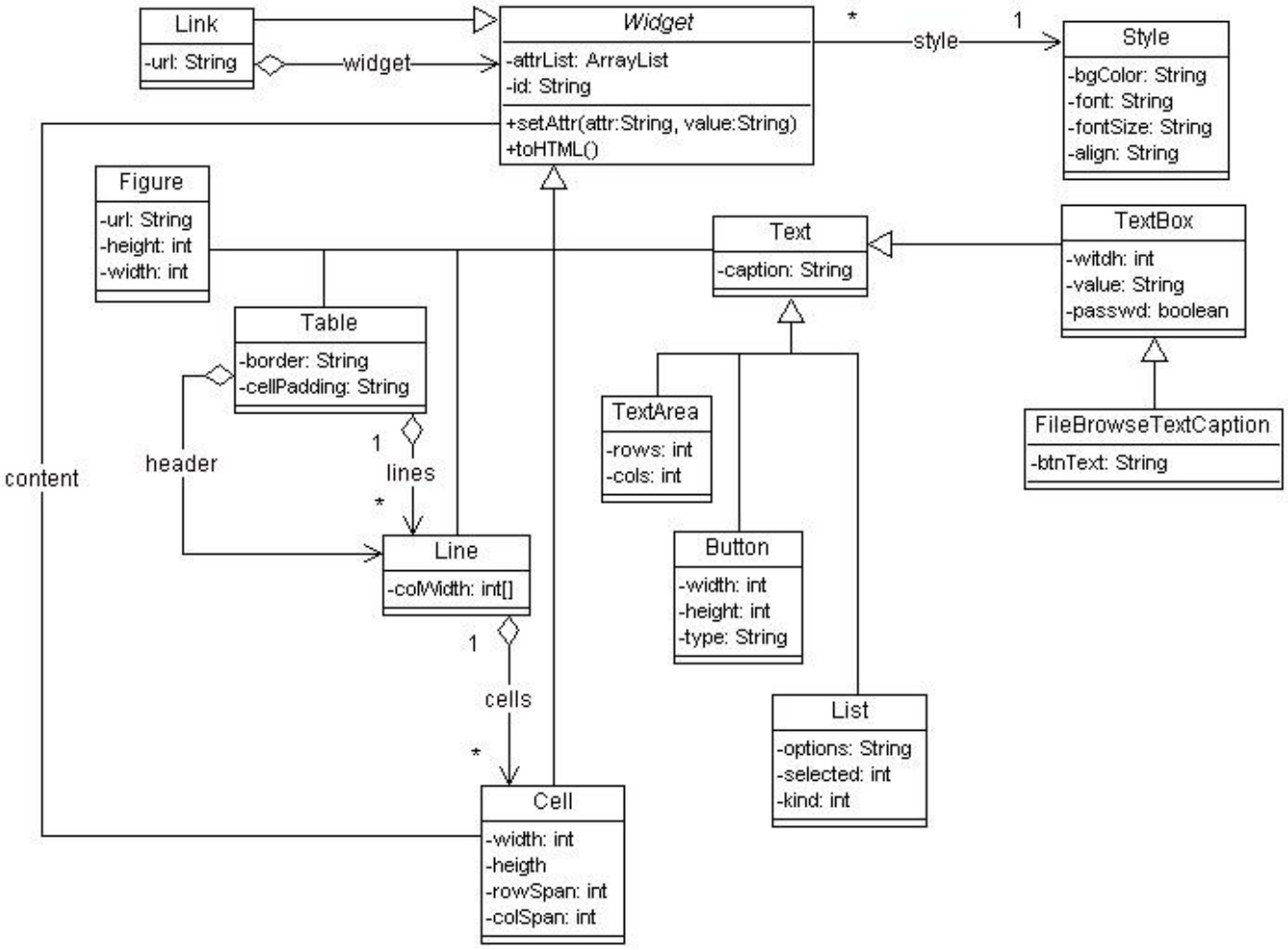


Diagrama de Classes - Exemplo



Estereótipos de Classes

- UML permite extensões pelo usuário através da criação de *estereótipos*.
- na maioria dos sistemas algumas classes se encaixam em tres categorias, que podem ser descritas como "*estereótipos padrão*":
  - *Classes de Entidade*
  - *Classes de Controle*
  - *Classes de Interface*

Classes de Entidade

- Normalmente são identificadas com os principais elementos no domínio do problema.
  - Exemplo, num sistema de administração escolar
    - *estudante*
    - *curso*
    - *currículo*
    - *professor*
    - *disciplina*
- *Classes de Entidade*, na maioria dos casos, se referem a informações que devem ser mantidas em *banco de dados*.

Classes de Controle

- As Classes de Controle normalmente são responsáveis pela coordenação das atividades especificadas nos casos de uso e aparecem como participantes nos Diagramas de Interação.
- São classes dependentes da aplicação, responsáveis pelas “regras de negócio” envolvidas.

Classes de Interface

- As *Classes de Interface* descrevem as interações entre o sistema sendo descrito e o *mundo exterior*.
- Tipicamente correspondem às classes responsáveis pela *interface com o usuário* e pela *comunicação* com outros sistemas.

Diagramas de Interação

- Diagramas de Interação mostram, para cada *cenário* descrito num *caso de uso*, as interações entre os atores e as classes envolvidas
- Em UML existem dois tipos de Diagramas de Interação: *Diagramas de Sequência* e *Diagramas de Colaboração*.

Diagramas de Sequência

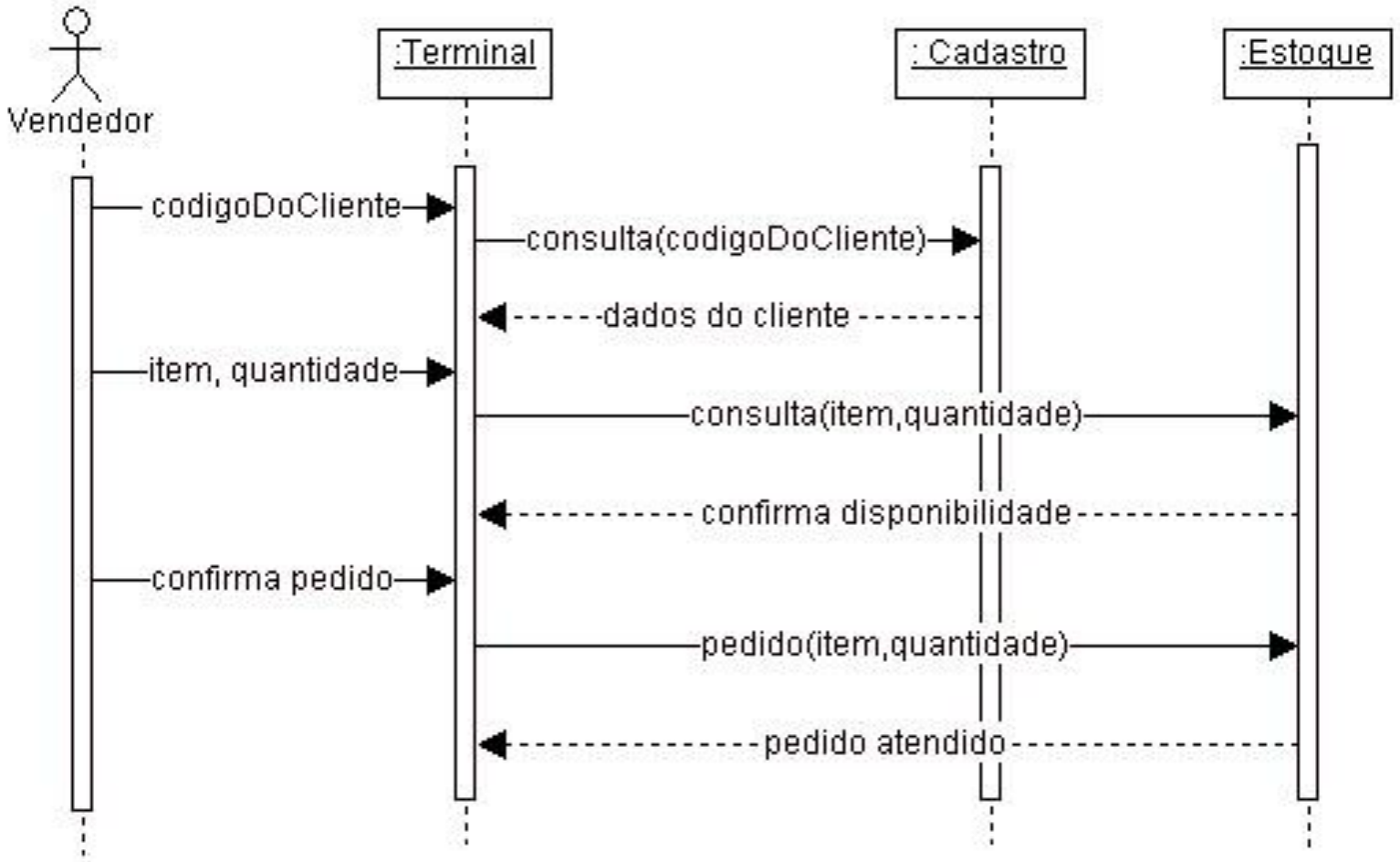


Diagrama de Sequência

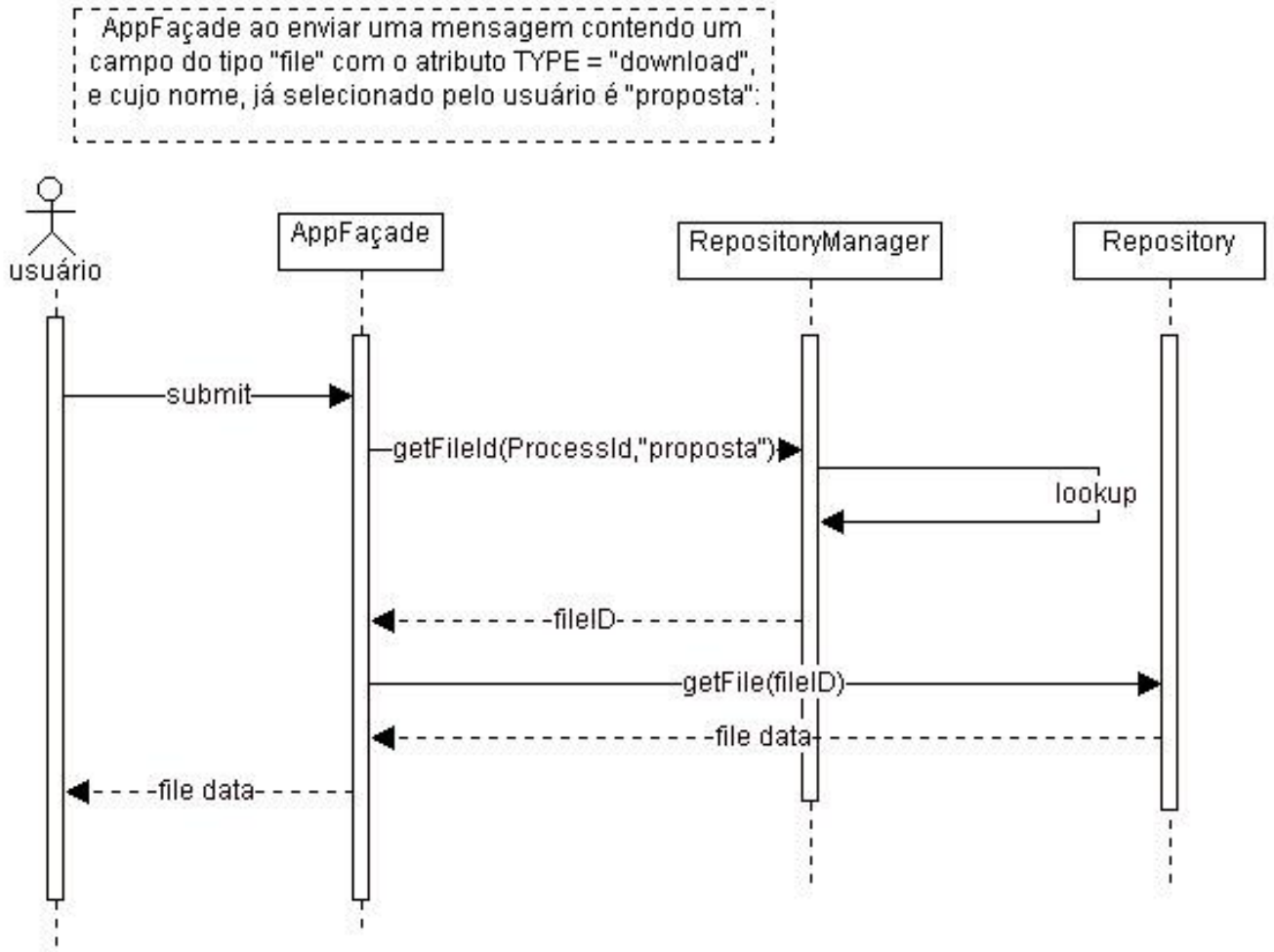
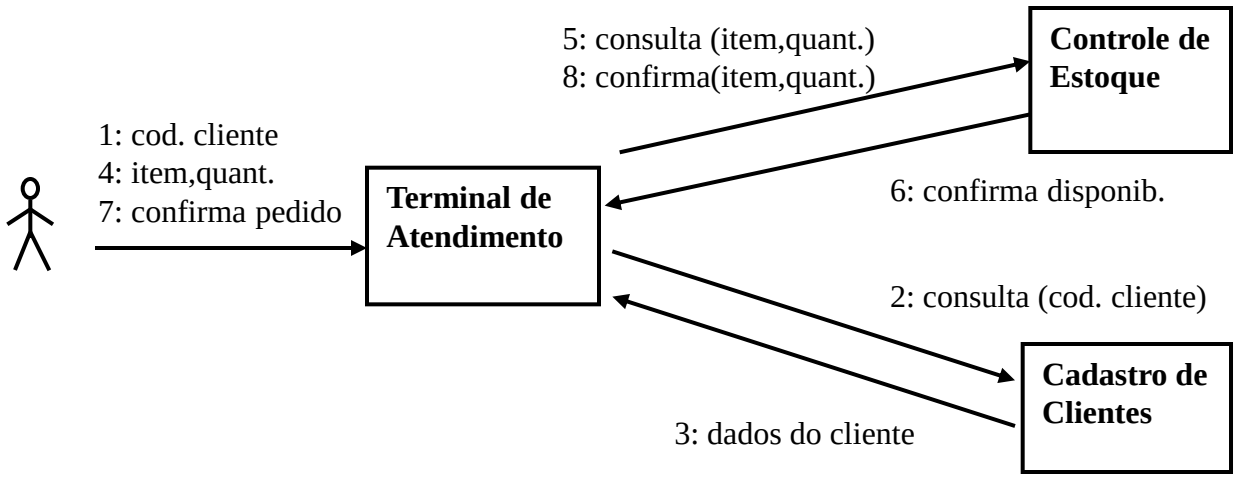


Diagrama de Colaboração



Diagramas de Estado

- Os cenários nos *Casos de Uso* e os *Diagramas de Interação* descrevem o comportamento do sistema.
- Em alguns casos é necessário descrever o *comportamento interno* de um *objeto*. Isso é feito através dos *Diagramas de Estado*.

Diagramas de Estado

- Os principais elementos de um diagrama de estado são
  - estados (possíveis)
  - transições (indicam mudanças de estado do objeto)
  - eventos externos (provocam a mudança de estado no objeto)
  - estado inicial
  - estado final

Diagrama de Estados

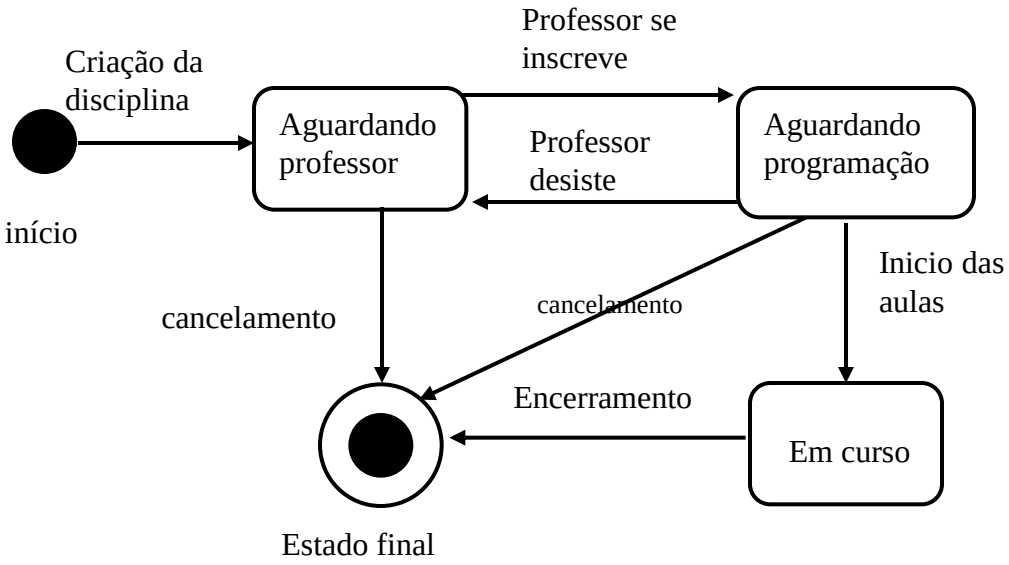


Diagrama de Estados

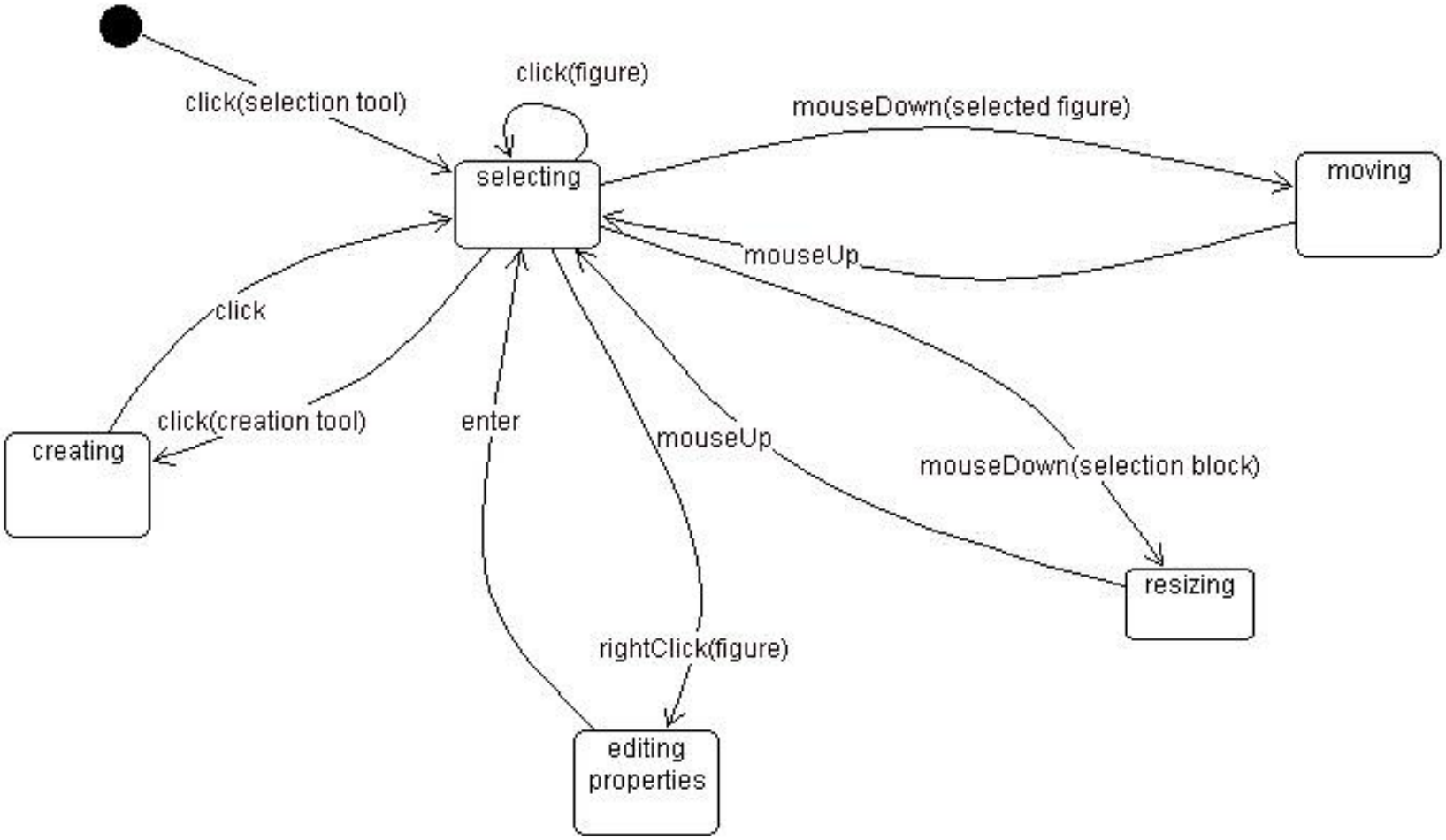


Diagrama de Estados

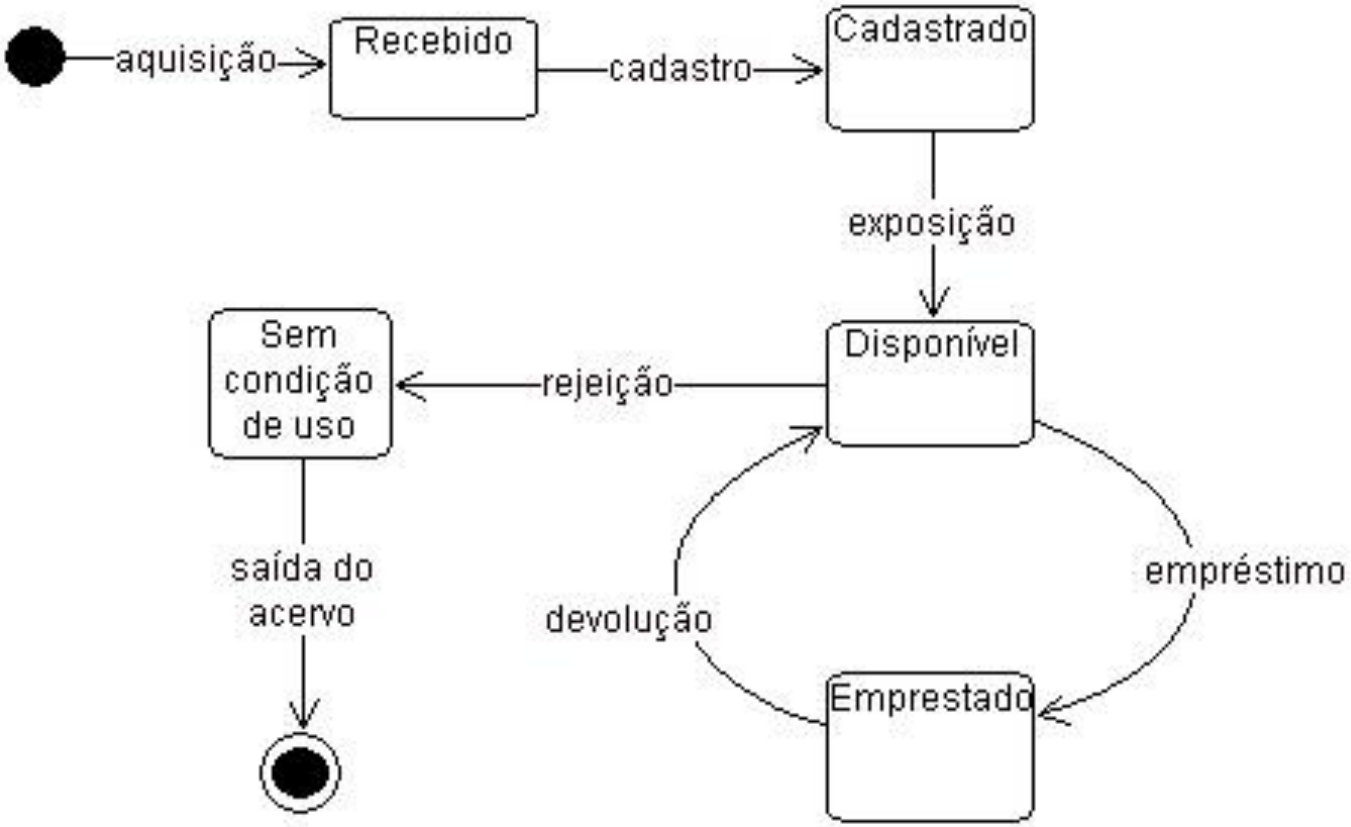


Diagrama de Atividades

- Um diagrama de estados descreve as mudanças de estado de um objeto, em função de eventos externos.
- Em muitas situações, os objetos interagem entre si e as mudanças de estado de um objeto dependem de ações executadas por outro objeto.
- O diagrama de atividades permite descrever as operações associadas a um objeto e também as interações com outros objetos (concorrência).

Diagrama de Atividades

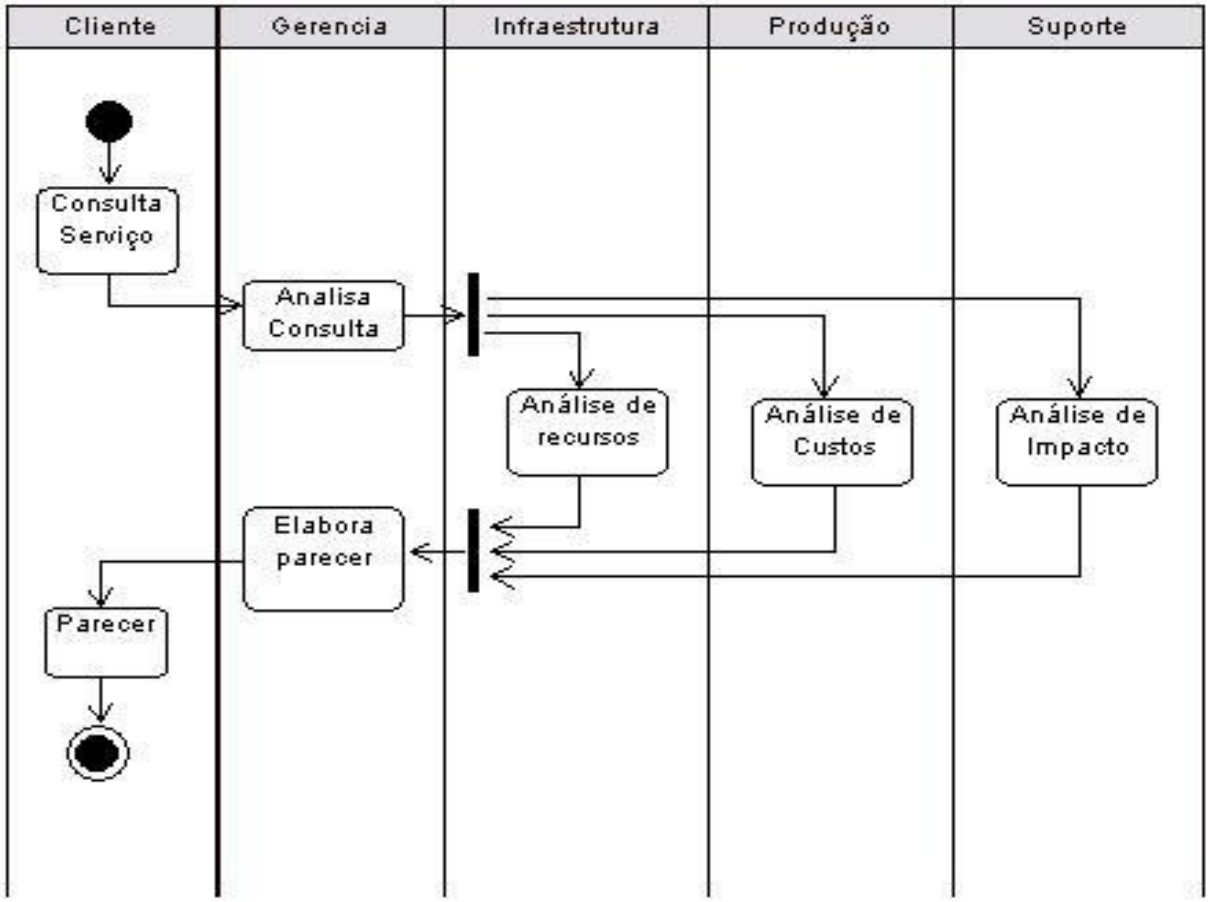


Diagrama de Componentes

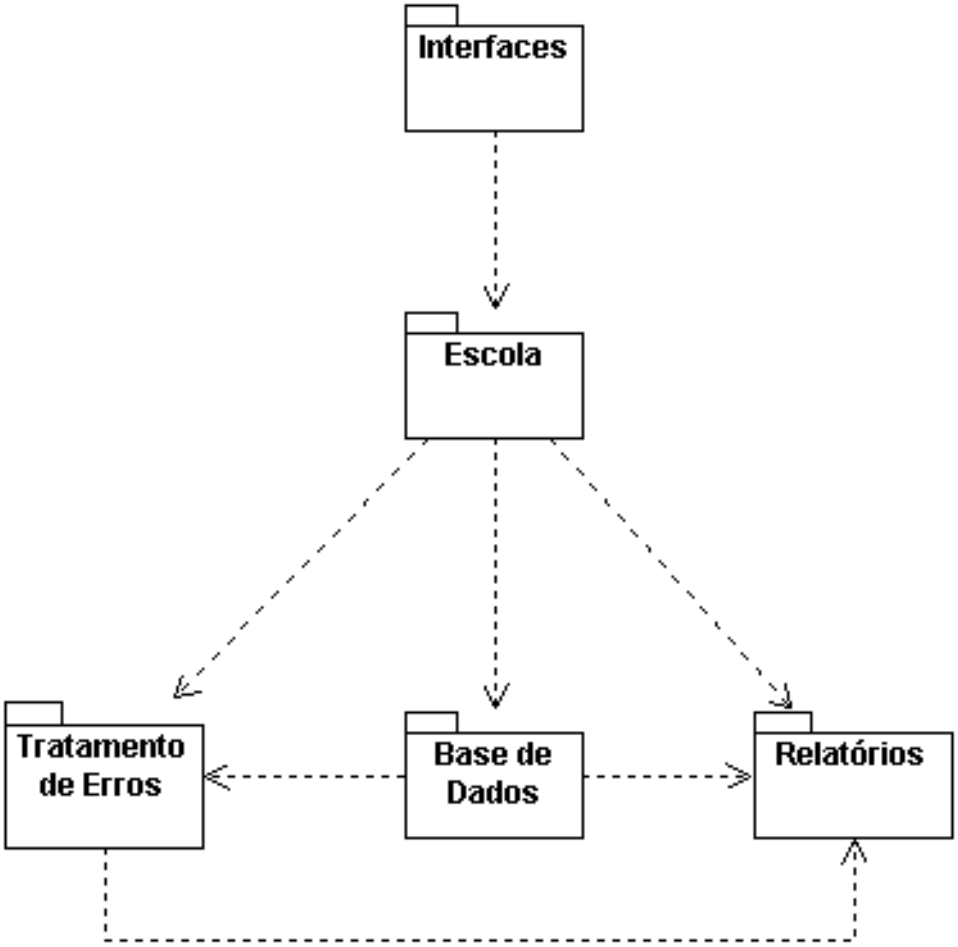


Diagrama de Componentes

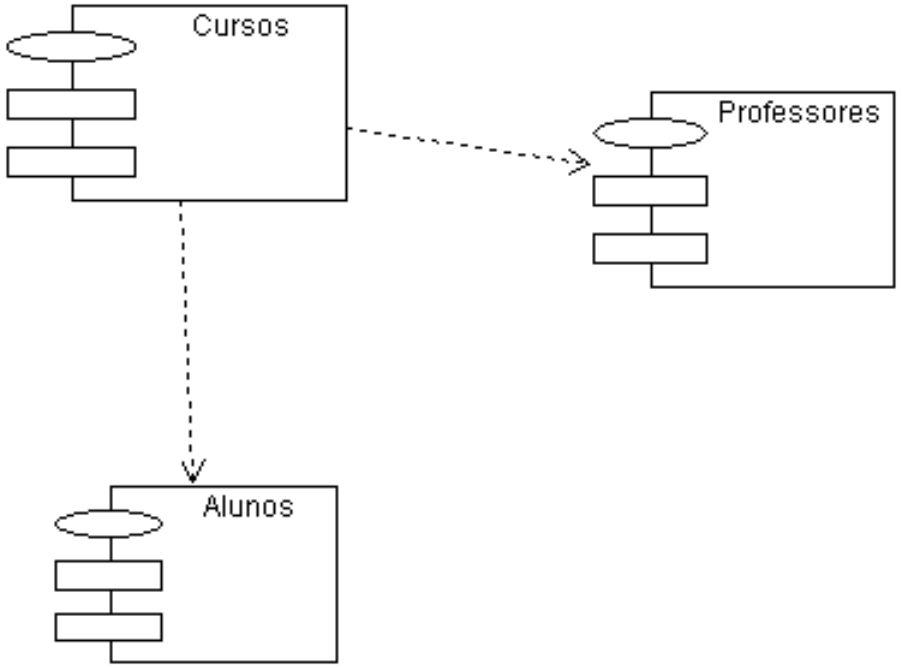
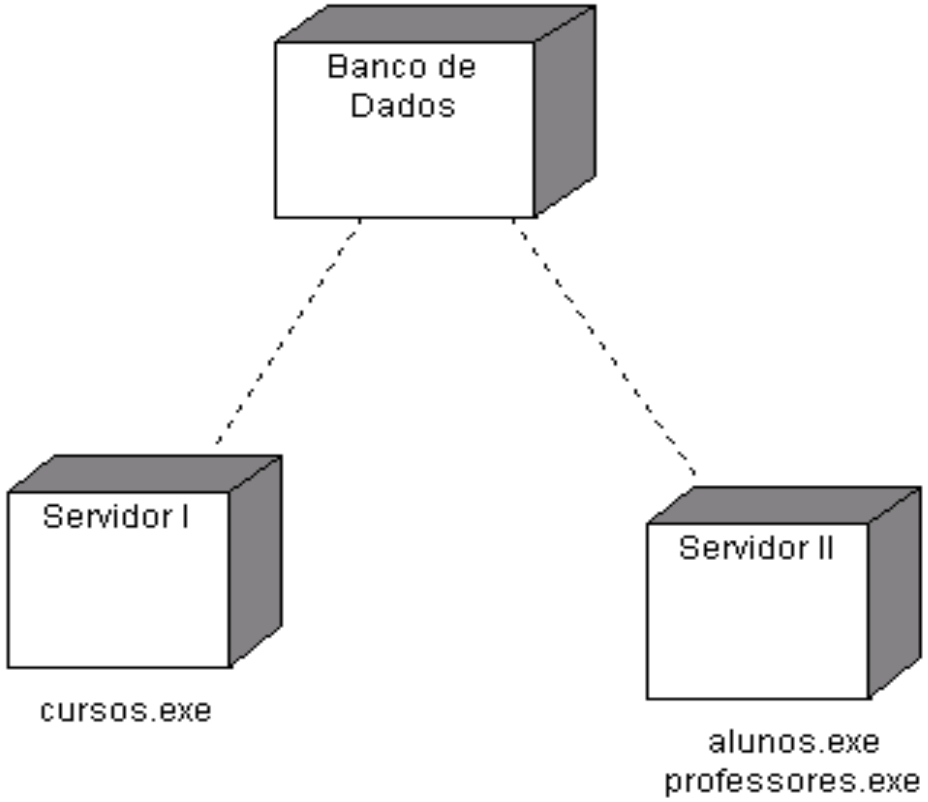


Diagrama de Implantação



Na prática ...

- UML é apenas uma *notação* - para que seja eficaz num projeto é necessário que o seu uso ocorra sob um *processo* bem definido.
- O objetivo final do projeto é (em geral) o *software rodando* adequadamente na *casa do cliente*.

Na prática ...

- Qualquer que seja o processo de desenvolvimento, o ponto de partida natural para um projeto que use UML são os diagramas de caso de uso.
- Não existe, no entanto uma *receita mágica* para, partindo dos casos de uso, se chegar aos demais diagramas.

Na prática ...

- Nem todos os recursos da notação UML são necessários num projeto:
  - diagramas de estado ou de atividades nem sempre são usados.
  - diagramas de classe e diagramas de seqüência quase sempre são necessários.
  - alguns tipos de relacionamentos podem não aparecer num diagrama de classe.
  - a metodologia utilizada pode enfatizar o uso de determinados recursos ou se basear em estereótipos pré-definidos.

Na prática ...

- Os modelos em UML podem ser tão detalhados quanto se queira:
  - À medida que se desce em detalhes, aumenta o comprometimento da implementação com o modelo.
  - Para que os modelos façam sentido para a equipe de programação, o projetista deve conhecer os recursos oferecidos pelo ambiente de desenvolvimento até o nível de detalhe que ele vai usar nos modelos.

Uma abordagem iterativa

1. Levantamento dos casos de uso e elaboração dos respectivos diagramas.
2. A partir dos casos de uso, procurar identificar as principais partes (módulos, pacotes ou componentes do sistema).
3. Com base no particionamento e nos casos de uso, elaborar os diagramas de seqüência correspondentes.
4. Com base nos diagramas de seqüência, em geral é possível refinar o particionamento.

Uma abordagem iterativa

5. Tendo refinado o particionamento, os diagramas de seqüência podem ser refinados. Neste ponto, já se tem idéia se serão necessários novos diagramas de seqüência, de estado ou de atividades.
6. Os dois passos anteriores devem ser repetidos até que os detalhes levantados sejam suficientes para se iniciar a fase de construção do sistema.
7. Os diagramas de componentes e implantação podem ser elaborados a partir da finalização do passo anterior.

Uma abordagem iterativa

- A abordagem sugerida é subjetiva e depende da *experiência* e *bom senso* do projetista. Sua aplicabilidade depende do *processo de desenvolvimento* sendo utilizado.

Referências

- The Unified Modeling Language User Guide, G. Booch, J. Rambaugh, Ivar Jacobson, Addison-Wesley, 1999
- Applying Use Case Modeling with UML, D. Rosenberg, K. Scott, Addison-Wesley, 2001
- UML in a Nutshell, S.S.Alhir, O'Reilly, 1998
- Visual Modeling with Rational Rose and UML, T. Quatrani, Addison-Wesley, 2000

Referências na web:

- UML Tutorial, http://www.sparxsystems.com.au/UML_Tutorial.htm
- Unified Modeling Language Tutorial, http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/

Impacto no Ciclo de Vida do Software

- Pelo fato de encapsular estado e comportamento, objetos são adequados a representar conceitos ou elementos do mundo real.
- Como os requisitos evoluem ao longo do processo de desenvolvimento, a realimentação por parte do cliente final nesse período é importante e em geral influencia o sistema sendo desenvolvido.
- Metodologias que prevêem esse tipo de realimentação, baseadas em variantes da espiral evolutiva acabam sendo mais adequadas para o desenvolvimento de sistemas baseados em objetos.

Impacto no Ciclo de Vida do Software

- As características impostas à arquitetura, principalmente baixo acoplamento e reuso influenciam a fase de verificação e validação do sistema.
- Pelas mesmas razões, a evolução do sistema também é afetada.

Impacto na Arquitetura do Software

- baixo acoplamento: objetos, pelas sua característica de encapsulamento, permitem que se modele a arquitetura do sistema a partir de elementos com *baixo acoplamento* entre si. As partes do sistema que usam um dado objeto, não precisam saber como ele funciona, precisam saber apenas quais os serviços que ele oferece (expresso pelos métodos que ele exporta).
- reuso: herança e polimorfismo estimulam a reutilização do código que implementa as classes. O baixo acoplamento entre as parte por sua vez facilita o reuso.

Impacto na Arquitetura do Software

- foco *no domínio do problema* ao invés de *domínio da solução*.
- *componentes de software*: objetos têm sido usados para a criação de componentes de software, que além de oferecerem métodos para acesso aos seus serviços, oferecem métodos adicionais que permitem o seu registro, identificação e caracterização dos seus serviços.

Impacto nas Ferramentas de Desenvolvimento

- ambientes de desenvolvimento: novos ambientes disponibilizam ao projetista um conjunto de componentes que podem ser facilmente integrados às aplicações.
- estensibilidade: novos componentes podem, desenvolvidos por terceiros ou pela própria equipe, podem ser integrados ao ambiente de desenvolvimento, abrindo um leque de infinitas possibilidades.

Impacto nas Ferramentas de Desenvolvimento

- Integração IDE-CASE: atualmente, a maior parte dos IDE's (*integrated development environment*) está se integrando com ferramentas CASE (*computer aided software engineering*), de forma que modelamento e desenvolvimento andam juntos, de forma integrada.

Impacto na Equipe de Desenvolvimento

- a partir do domínio de bibliotecas e ferramentas de desenvolvimento, o programador passa a ser uma peça importante no processo de desenvolvimento.
- A definição da arquitetura, por ser influenciada pelo uso dos objetos, pelo reuso e pela farta disponibilidade de componentes, na maioria dos casos, necessita da participação do ‘programador-arquiteto’.

Impacto na Equipe de Desenvolvimento

- o programador por sua vez, necessita cada vez mais dominar todo o contexto formado pelos recursos de programação oferecidos pela linguagem e ambiente sendo utilizados, assim como bibliotecas de classes, componentes e módulos de software candidatos ao reuso.

Componentes

- *componente de software*: é um objeto que além de oferecer métodos para acesso aos seus serviços, oferece métodos adicionais que permitem o seu registro, identificação e caracterização dos seus serviços.
- um componente para operar como tal, necessita de um suporte do *ambiente operacional* que permita o seu registro, sua identificação e identificação de seus serviços.
- o *ambiente operacional* que dá suporte aos componentes pode ser o próprio sistema operacional, ou mesmo uma biblioteca.

Toolkits (bibliotecas de classes)

- Conjunto de classes reutilizáveis projetadas para oferecer um funcionalidades voltadas para uma área de aplicação específica.
- Toolkits enfatizam o reuso de código sem impor nenhuma característica estrutural à arquitetura.

Frameworks

- Um framework é um conjunto de classes cooperantes voltado ao desenvolvimento de aplicações numa determinada área.
- A construção de uma aplicação se dá a partir da construção de classes abstratas definidas no framework.
- O framework estabelece a arquitetura da aplicação.

Método Estruturado e Processos baseados em UML/OO

Estruturado	UML/OO	Obs.
Diagrama de Contexto	Casos de Uso Diagramas de Sequência	
DFD	Diagrama de Sequência	DFD: nenhuma idéia de tempo e sequenciamento.
Depósito de Dados	Classes de entidade	
Processo	Métodos/mensagens	
Diagrama Estrutural	Diagrama de Classes	Diagrama estrutural: não indica os relacionamentos.
Dicionário de Dados	??	
Descrição dos processos	??	UML: indica apenas as interações entre objetos
DTE	Diagrama de Transição	
???	Diagrama de Atividades	