Análise Orientada a Objetos

Prof. Vítor Souza
Análise e Projeto Orientado a Objetos

Departamento de Informática
Univ. Federal do Espírito Santo

11/04/2006

Licença para uso e distribuição

Este material está disponível para uso não-comercial e pode ser derivado e/ou distribuído, desde que utilizando uma licença equivalente.

Atribuição-Uso Não-Comercial-Compatilhamento pela mesma licença, versão 2.5

http://creativecommons.org/licenses/by-nc-sa/2.5/deed.pt

Você pode copiar, distribuir, exibir e executar a obra, além de criar obras derivadas, sob as seguintes condições: (a) você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante; (b) você não pode utilizar esta obra com finalidades comerciais; (c) Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.

Sobre o curso

- Estes slides foram criados no Departamento de Informática da Universidade Federal do Espírito Santo (UFES) e estão disponível no seguinte endereço:

  http://www.inf.ufes.br/~vsouza/

- O material usado como base foi cedido pelo professor Dr. Ricardo de Almeida Falbo, do DI/UFES:

  http://www.inf.ufes.br/~falbo/

- Este curso tem como objetivo apresentar os conceitos da análise e projeto orientado a objetos a profissionais e estudantes de Engenharia de Software.

Processo de desenvolvimento

- O que é um processo?

- Principais componentes de um processo:
  - Atividades;
  - Artefatos (insumos e produtos);
  - Recursos;
  - Procedimentos.

São necessários?

- Em pequenos projetos, podem não ser;
- Em grandes projetos, eles:
  - Formalizam a comunicação;
  - Guiam o desenvolvimento;
  - Auxiliam na garantia da qualidade.
- O que é qualidade?
  - Um sistema correto, rápido, confiável, fácil de usar, eficiente, etc.;
  - Um sistema fácil de entender, de modificar, reutilizável, portável, etc.

Como escolher um processo?

- Depende de vários fatores:
  - Produto a ser desenvolvido;
  - Time de desenvolvimento;
  - Cliente;
  - Rigor em relação à documentação;
  - Limites de prazo e custo;
  - Etc.

Definição do processo

- Definição das atividades-chave:
  - Planejamento, requisitos, análise, projeto, etc.
- Escolha de um paradigma:
  - Orientado a Objetos, Estruturado, etc.
- Escolha de um modelo de ciclo de vida:
  - Linear, incremental, evolutivo, etc.

Atividades-chave

- Planejamento;
- Levantamento de Requisitos;
- Análise;
- Projeto;
- Implementação;
- Testes;
- Implantação;
- Operação;
- Manutenção.

Planejamento

- Definição de estimativas de custo, tempo e utilização de recursos;
- Plano de projeto: proposta de desenvolvimento, definição do processo;
- Informações atualizadas regularmente (revisões a cada final de etapa).

Levantamento de requisitos

- Refinamento do escopo;
- Identificação dos requisitos;
- Compreensão do domínio do problema.

Análise

- Modelagem, avaliação e documentação dos requisitos;
- Foco no que o software deve fazer, e não em como ele fará.

Projeto

- Definição da plataforma de implementação;
- Incorporação de requisitos tecnológicos;
- Foco em como cada requisito será implementado;
- Refinamentos sucessivos até chegar ao nível de implementação.

Implementação

- Tradução do projeto para uma linguagem de programação;
- Objetiva ter um software executável.

Testes

- Validação da funcionalidade por parte do time de desenvolvimento;
- Diversos tipos:
  - Testes de unidade;
  - Testes de integração;
  - Testes de sistema;
  - Testes de carga/estresse;
  - Etc.

Implantação

- Colocação do sistema em produção;
- Envolve outros aspectos:
  - Treinamento de usuários;
  - Configuração do ambiente;
  - Conversão de tecnologias;
  - Etc.
- Validação das funcionalidades por parte do usuário.

Operação

- Software em utilização pelo cliente.

Manutenção

- Alterações no software:
  - Erros encontrados;
  - Adaptações a alterações do ambiente;
  - Funcionalidade adicional;
  - Ajuste fino;
  - Etc.
- Pode requerer a definição de um novo processo de software.

O paradigma

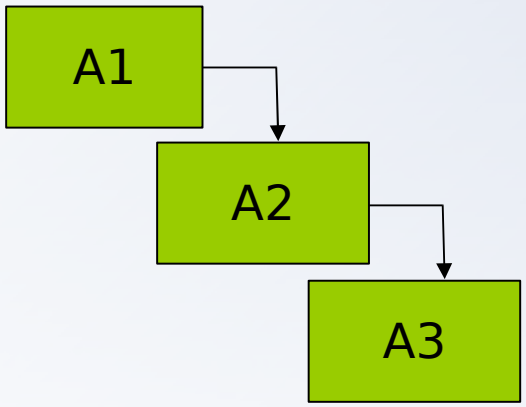
- Vantagens do paradigma OO:
  - Mesma notação durante todo o processo;
  - Redução do gap semântico;
  - Promove desenvolvimento incremental e evolutivo;
  - Reusabilidade;
  - Extensibilidade;
  - Modularidade;
  - Etc.

O modelo de ciclo de vida

- Estrutura as atividades do processo em fases e define como se relacionam;
- Muitos se encaixam no paradigma OO;
- Exemplos:
  - Seqüencial linear;
  - Incremental;
  - Evolutivos;
  - Ágeis.

Modelo seqüencial linear

- Mais antigo e mais fácil de gerenciar;
- Fases executadas uma vez, em seqüência;
- Várias fraquezas:
  - Projetos reais não seguem o fluxo seqüencial;
  - Não acomoda incertezas;
  - Demora para produzir algo concreto;
  - Não otimiza a alocação de recursos humanos.



Em Cascata

Modelo incremental

- Variação do seqüencial linear;
- Levantamento de requisitos e análise são feitos seqüencialmente;
- Projeto, implementação e testes são feitos em incrementos;
- Cada incremento gera uma parte operacional do sistema.

Modelos evolutivos

- Premissa de que os requisitos evoluem ao longo do tempo;
- Em geral são processos iterativos;
- Cada iteração constrói um incremento de software, que evolui com o tempo.

Evolutivo básico

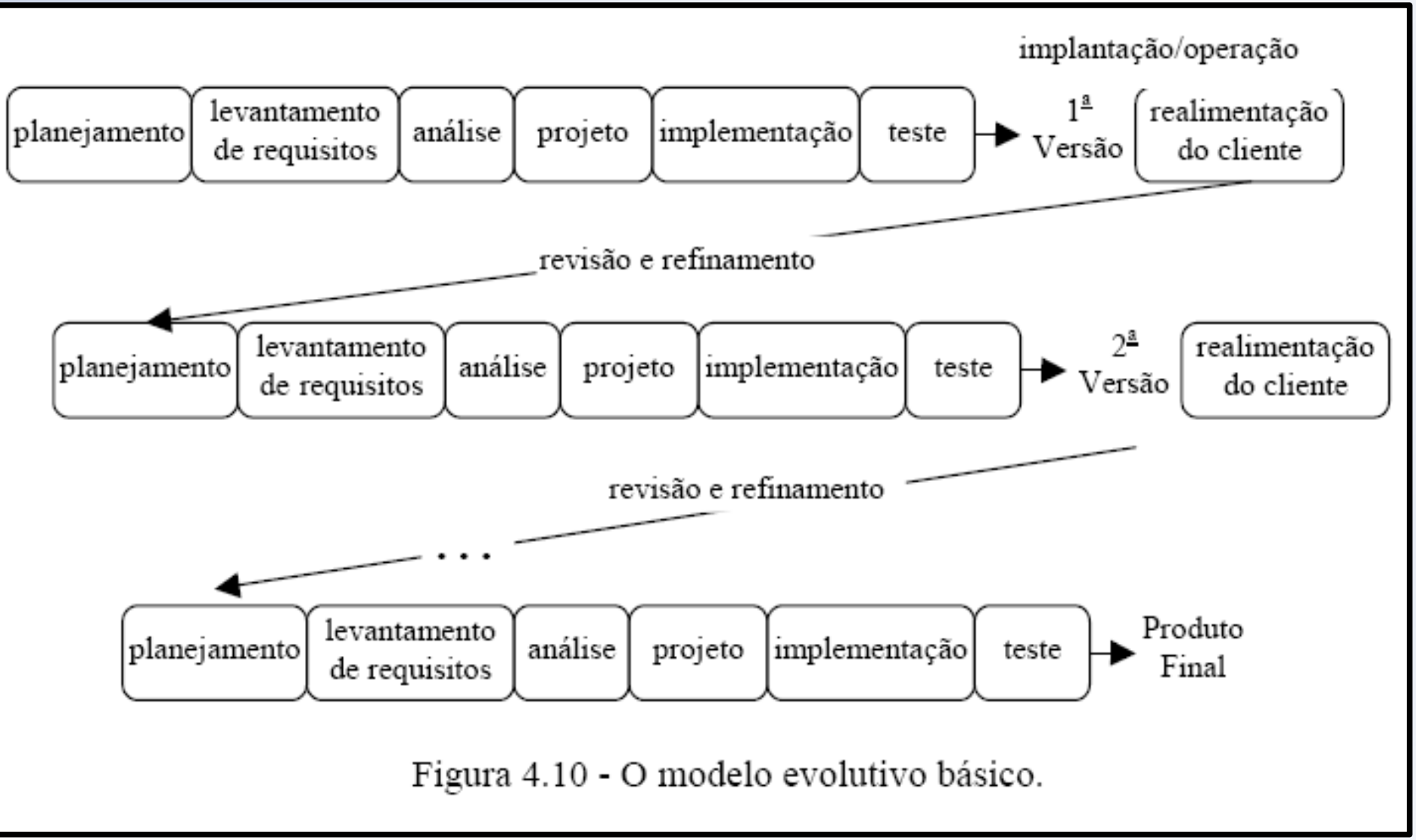


Figura 4.10 - O modelo evolutivo básico.

Modelo espiral

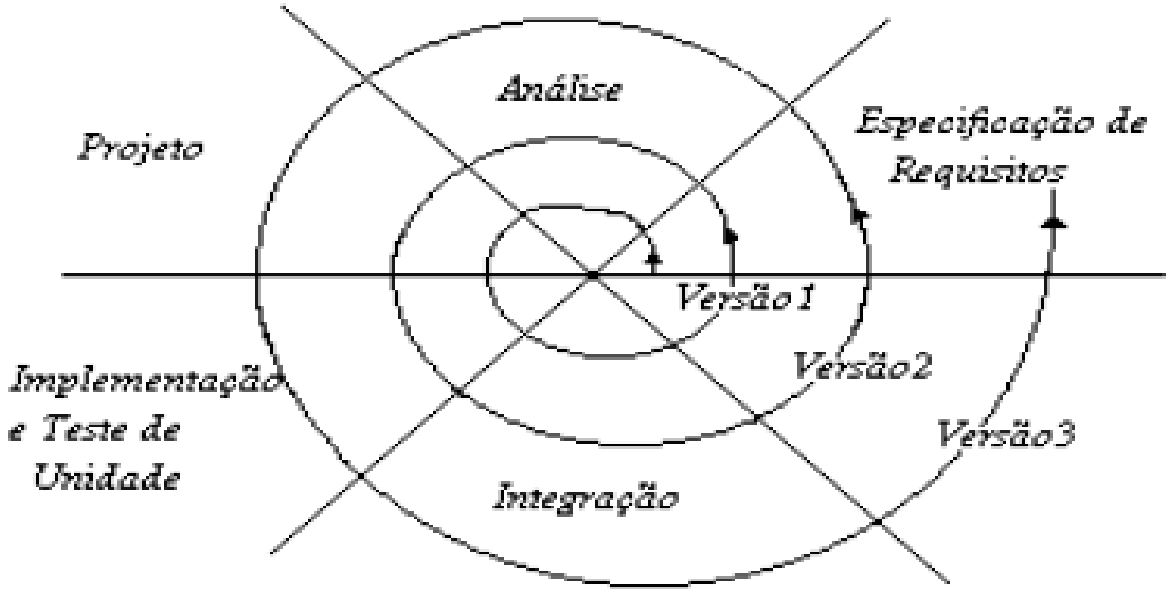
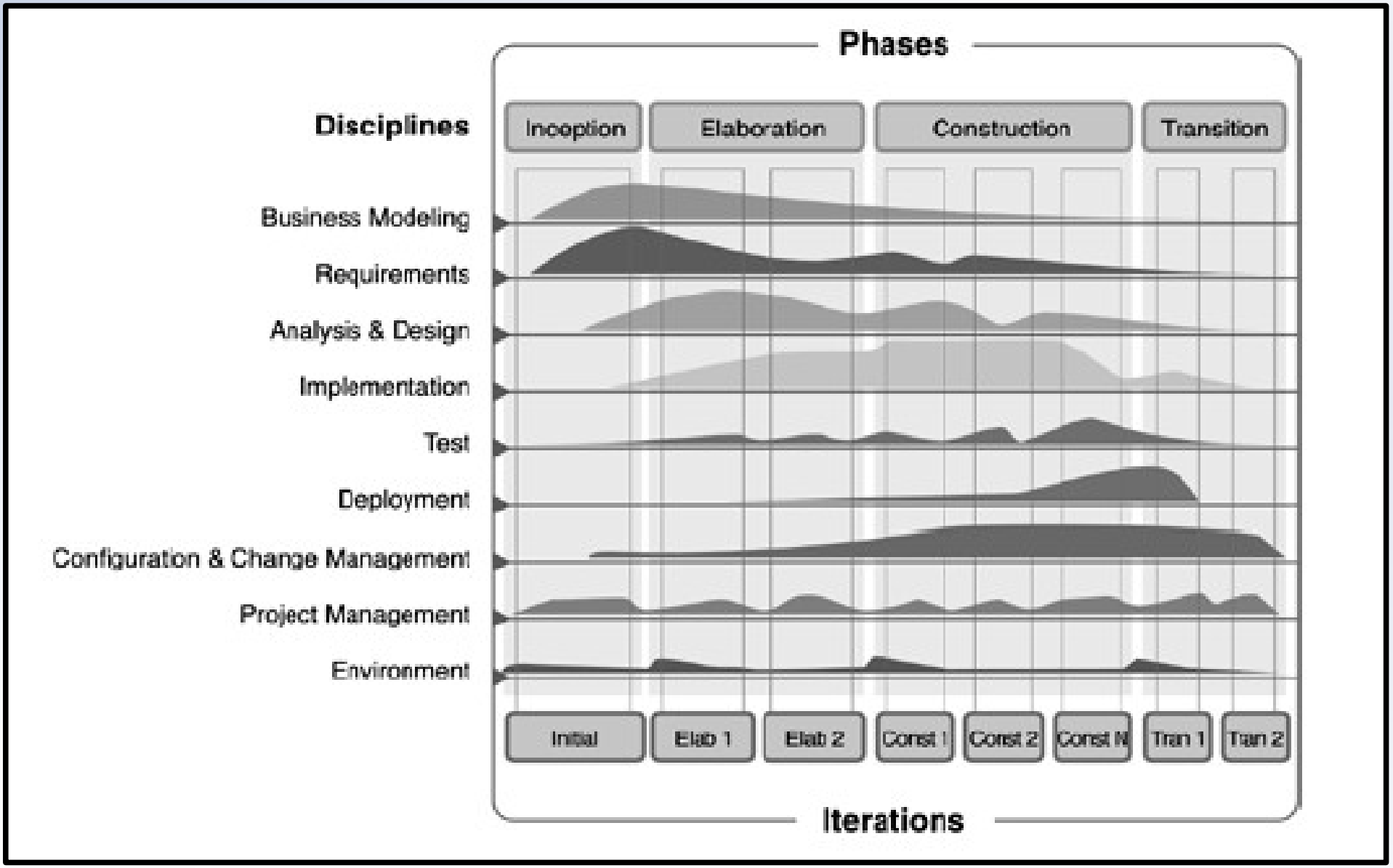


Figura 4.9 - Um modelo espiral [Jacobson92].

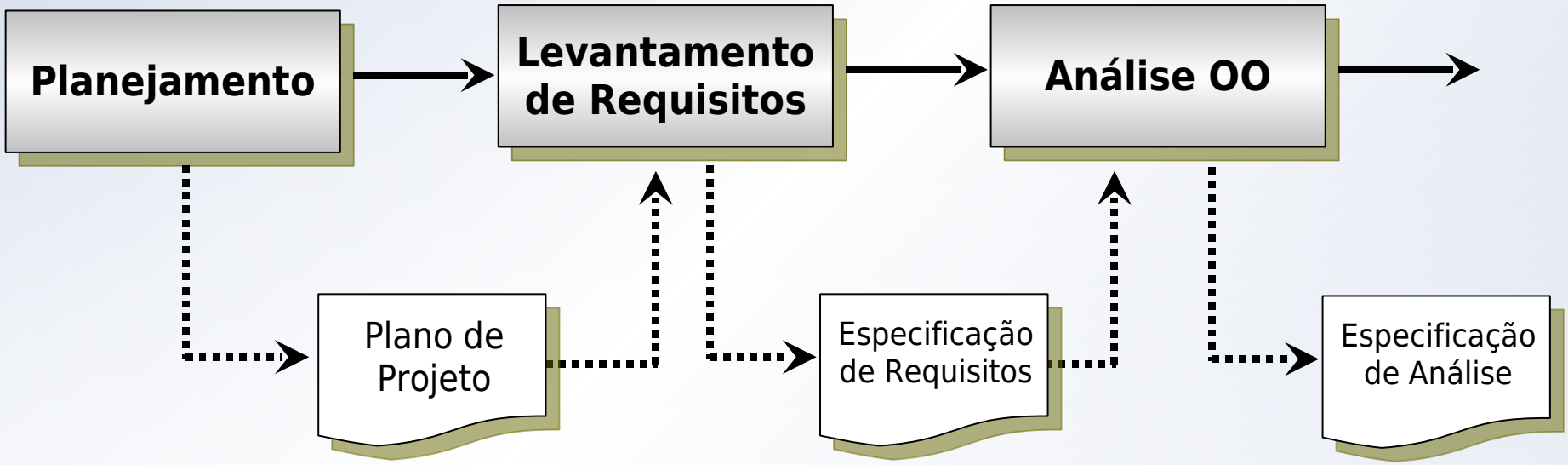
Rational Unified Process (RUP)



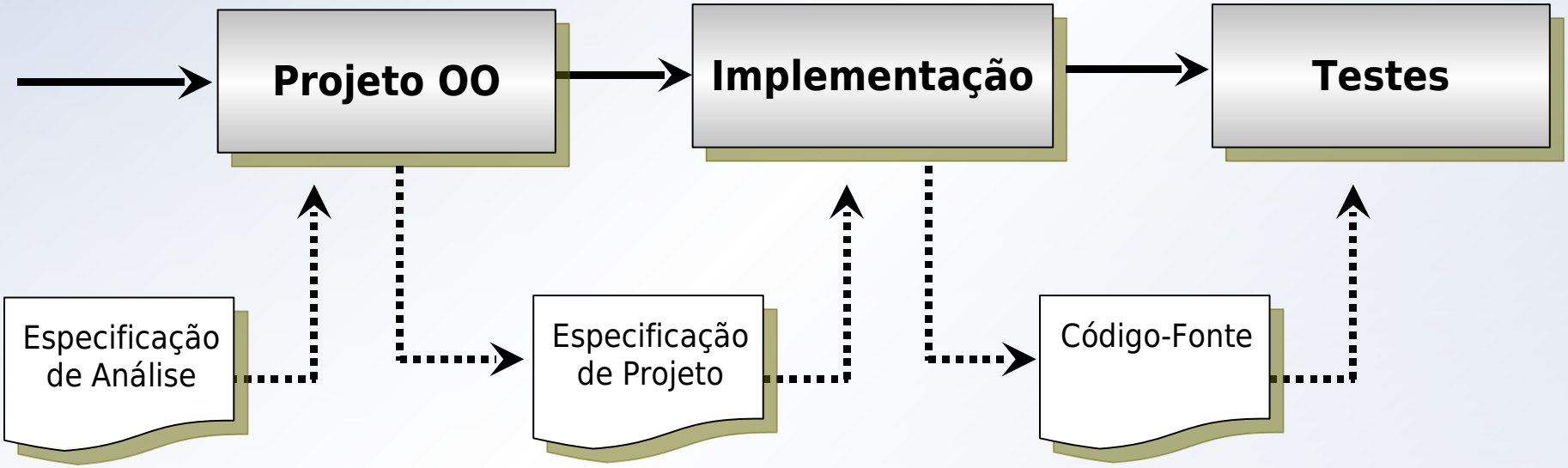
Modelos ágeis

- Nova filosofia que valoriza:
  - Indivíduos e interações ao invés de processos e ferramentas;
  - Software funcional ao invés de documentação;
  - Colaboração ao invés de negociação;
  - Responder às mudanças ao invés de seguir um plano.
- Exemplos:
  - Extreme Programming (XP);
  - Agile Modeling;
  - SCRUM;
  - Etc.

Exemplo de processo OO



Exemplo de processo OO



Neste curso...

- Já vimos Levantamento de Requisitos:
  - Resumo de Engenharia de Requisitos;
  - Técnicas de levantamento de requisitos;
  - Modelagem de casos de uso.
- Veremos Análise OO:
  - Identificação de classes, hierarquias, associações a atirbutos;
  - Identificação de subsistemas;
  - Identificação de comportamento e operações.

A Linguagem de Modelagem Unificada (UML)

Prof. Vítor Souza

Análise e Projeto Orientado a Objetos

Departamento de Informática
Univ. Federal do Espírito Santo

11/04/2006

Unified Modeling Language

- Padrão “de facto” para especificar, visualizar, documentar e construir artefatos de um sistema desenvolvido sob o paradigma Orientado a Objetos;
- Nasceu para o paradigma OO, mas não é exclusiva para ele;
- Teve origem em três outros métodos:
  - OMT (Rumbaugh *et al.*, 1994);
  - Método de Booch (Booch, 1994);
  - Método OOSE (Jacobson, 1992).

UML

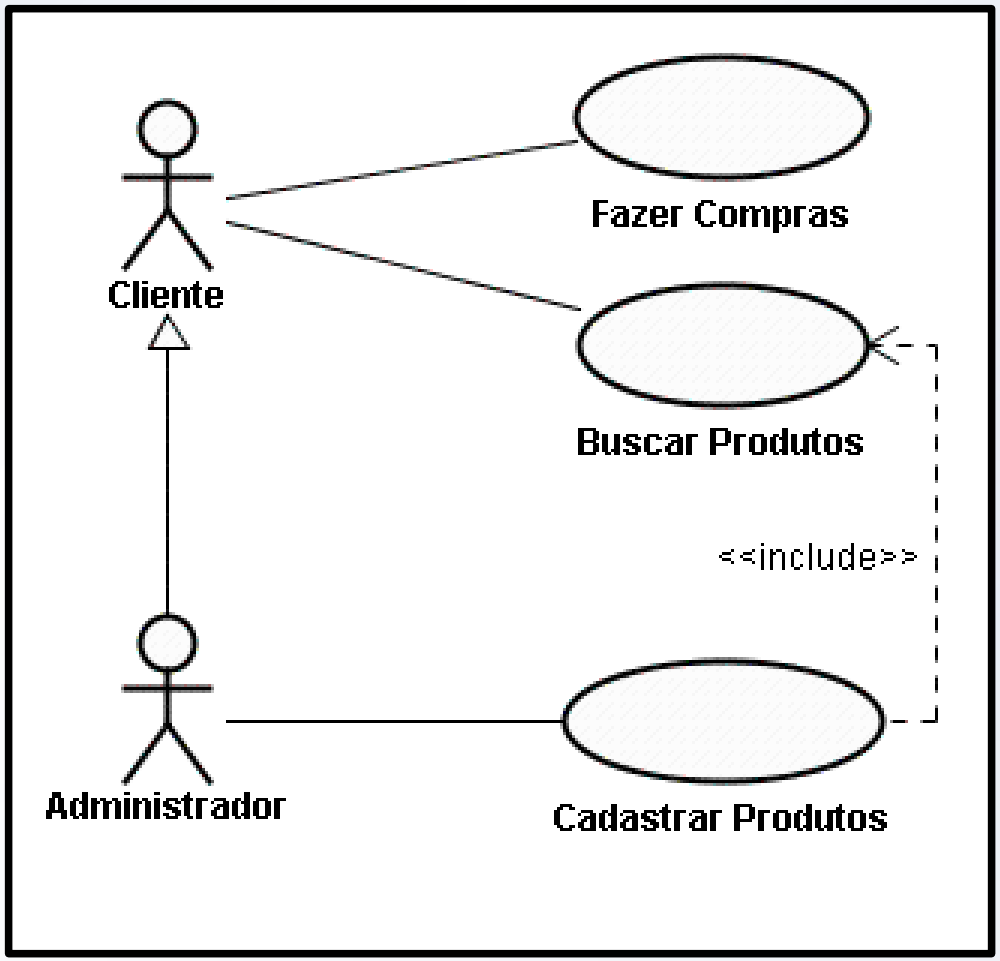
- Aprovada e homologada pela OMG (*Object Management Group*) em 1997. Sua versão atual é 2.0;
- A notação pode ser unificada, mas a decisão de qual artefato produzir depende do processo definido para o projeto;
- Pode ser utilizada em diferentes processos de desenvolvimento orientados a objetos, em todas as etapas do ciclo de desenvolvimento.

Diagramas da UML

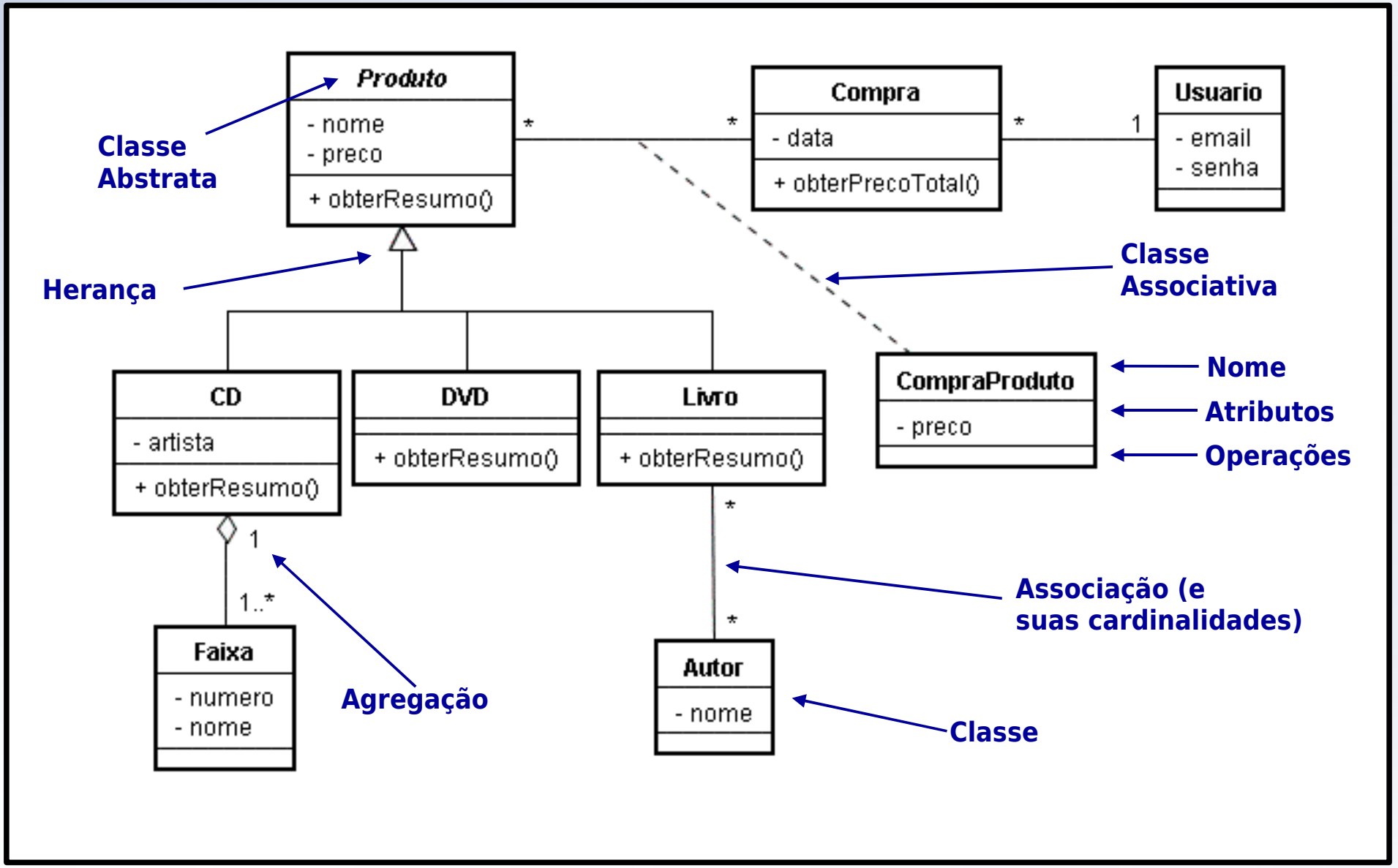
- de Casos de Uso;
- de Classes;
- de Objetos;
- de Estrutura Composta;
- de Seqüência;
- de Comunicação;
- de Estados;
- de Atividades;
- de Componentes;
- de Implantação;
- de Pacotes;
- de Interface Geral;
- de Tempo.

Diagramas de casos de uso

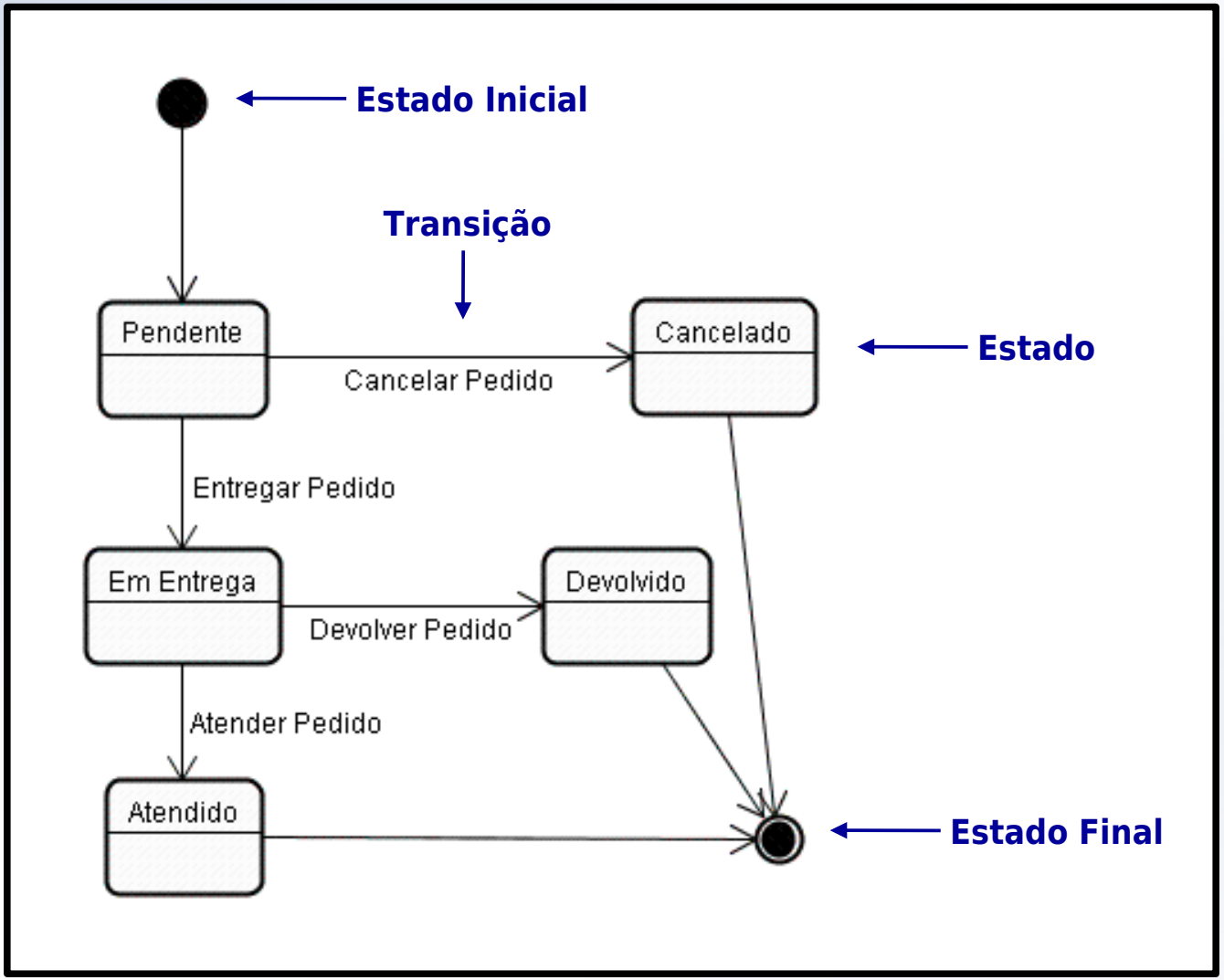
- Modela as funcionalidades do sistema;
- Captura típicas interações usuário – sistema;
- Usuários são atores;
- Atores e casos de uso são associados;
- Cada caso é descrito em detalhes separadamente.



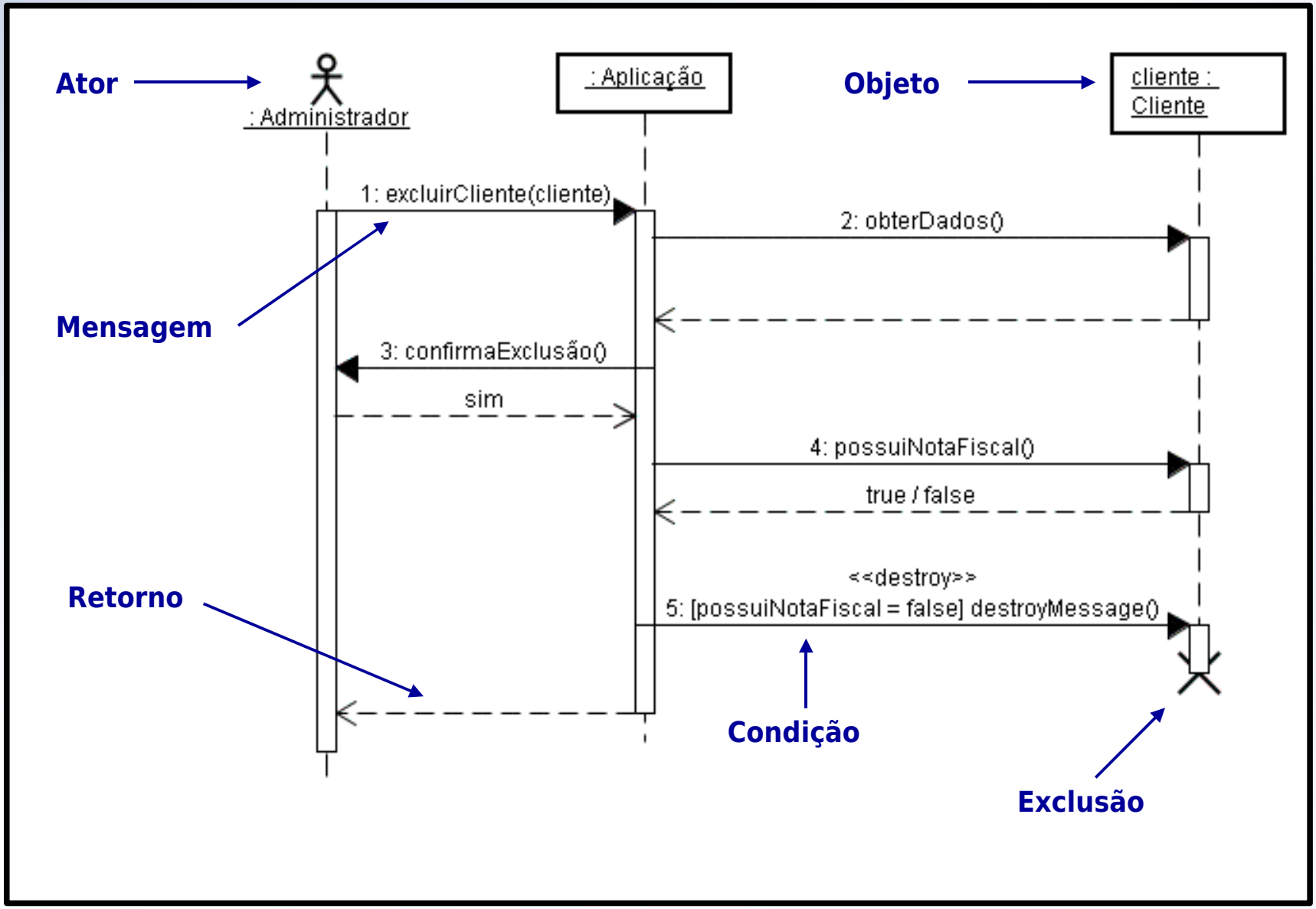
Diagramas de classes



Diagramas de estados



Diagramas de seqüência



Modelagem estática

Prof. Vítor Souza

Análise e Projeto Orientado a Objetos

Departamento de Informática
Univ. Federal do Espírito Santo

11/04/2006

Análise orientada a objetos

- Modelagem estática:
  - Identificação de classes;
  - Especificação de hierarquias de generalização/especialização;
  - Identificação de subsistemas;
  - Identificação de associações e atributos.
- Modelagem dinâmica:
  - Determinação do comportamento;
  - Definição de operações.

Classes

- Em última instância, um sistema OO é um conjunto de objetos que se comunicam;
- Objetos são categorizados em classes;
- Não modelamos objetos, modelamos classes;
- Logo, o processo central da Análise OO é a descoberta de classes que devem ser incluídas no modelo.

Como identificar classes?

- Internalize os conceitos OO;
- Analise os requisitos (documento e casos de uso) – estude o domínio do problema;
- Facilitadores:
  - Faça uma análise gramatical dos casos de uso;
  - Observe o ambiente do problema;
  - Procure ouvir atentamente os especialistas;
  - Verifique resultados de análise OO anteriores;
  - Observe outros sistemas;
  - Consulte fontes bibliográficas.

Possíveis candidatos à classes

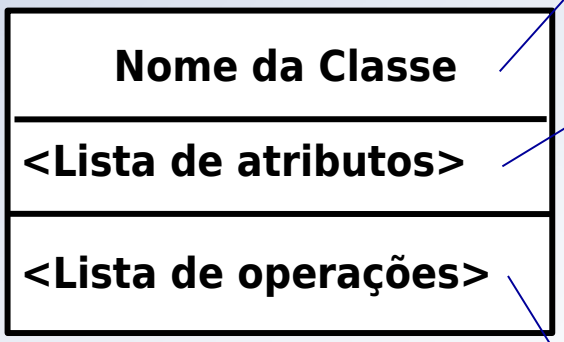
- Coisas que são parte do domínio do problema:
  - Ex.: pessoas, clientes, turmas, produtos, etc.
- Papéis desempenhados pelas diferentes pessoas que interagem direta ou indiretamente com o sistema:
  - Ex.: piloto, atendente, gerente etc.
- Ocorrências ou eventos que precisam ser registrados e lembrados pelo sistema:
  - Ex.: reclamação do cliente, reunião, etc.
- Locais físicos ou geográficos e lugares que estabelecem o contexto do problema:
  - Ex.: loja, aeroporto, etc.
- Unidades organizacionais (departamentos, divisões etc) que possam ser relevantes para o sistema:
  - Ex.: local para entrega, setor, etc.

Critério para seleção

- Uma classe deve:
  - Reter informações úteis;
  - Prover serviços que são requeridos pelos usuários (e inerentes à fase de análise);
  - Ter mais de um atributo;
  - Ter mais de uma instância;
  - Ter atributos e operações que são comuns a todas as suas instâncias.

Como modelar classes?

Representação em UML



Se estiver em itálico, a classe é abstrata.

Sintaxe: `<escopo> <nome> : <tipo> = <valor default>`
Escopo:
- privado
+ público
# protegido

Tipo: ignorado na fase de análise.

Sintaxe: `<escopo> <nome> (<parâmetros>) : <tipo>`

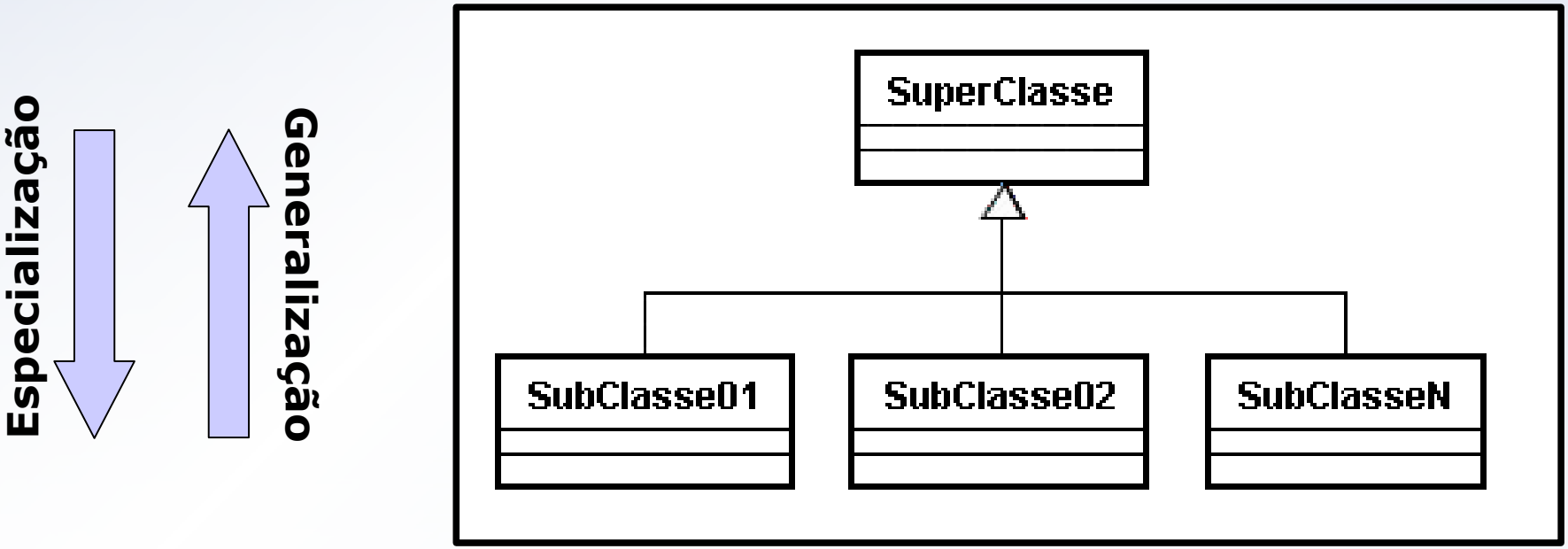
`<parâmetros>` = lista de pares "`<nome> : <tipo>`", separada por vírgula.

Tipo: ignorado na fase de análise.

Após identificadas, são dispostas em diagramas de classes.

Hierarquias

- Um dos principais mecanismos de estruturação de conceitos;
- Captura similaridades entre classes.



Diretrizes para herança

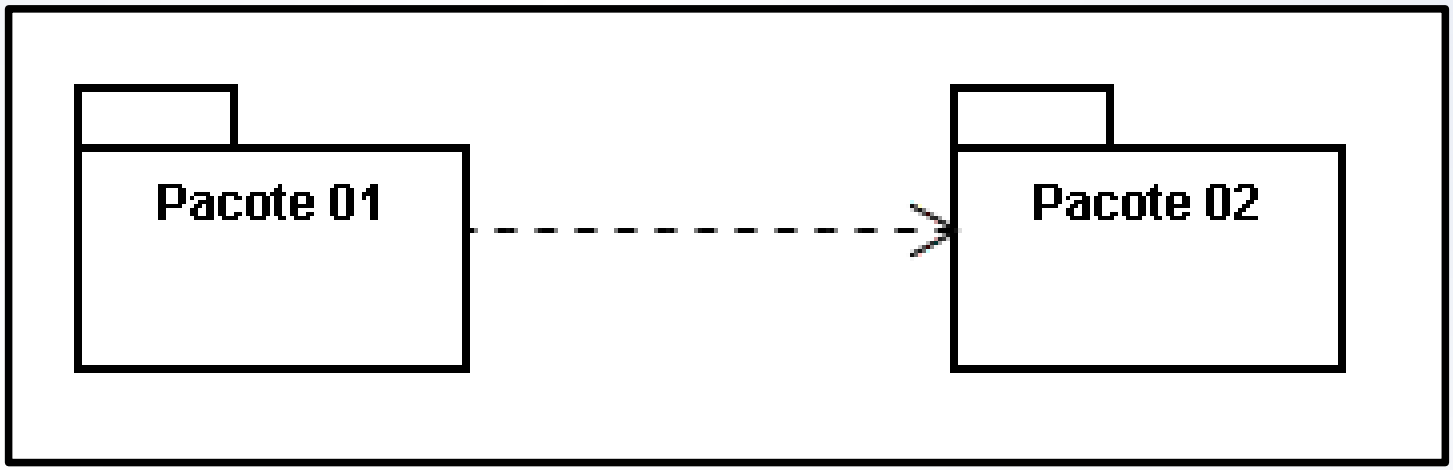
- Deve modelar relações “é-um-tipo-de”;
  - Instâncias da subclasse são por definição instâncias da superclasse. Tudo que é dito para esta última vale para a primeira.
- Subclasses devem suportar toda a funcionalidade das superclasses e possivelmente mais;
- Funcionalidade comum a diversas classes deve estar o mais alto possível na hierarquia;
- A herança deve estar no domínio do problema e fazer parte das responsabilidades do sistema;
- Classes que não adicionam funcionalidade nem redefinem funcionalidades existentes devem ser eliminadas.

Separação em subsistemas

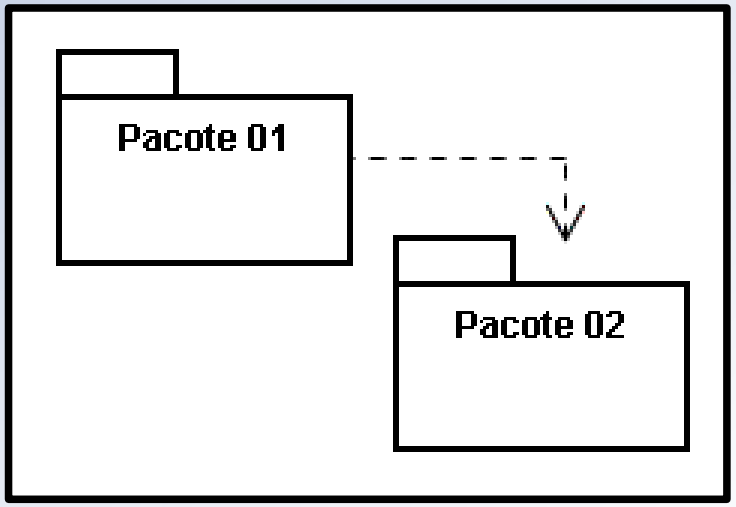
- Projetos grandes podem conter centenas de classes e estruturas diversas;
- Divisão das classes em pacotes:
  - Coleção de classes que colaboram entre si;
  - Conjunto coeso de responsabilidades;
  - “Caixa preta”.
- Vantagens:
  - Facilita o entendimento para leitores;
  - Auxilia na organização de grupos de trabalho;
  - Organiza a documentação.

Pacotes de classes x subsistemas

- Casos de uso podem ser um bom ponto de partida para definição do agrupamento;
- No entanto, não é obrigatório que a divisão seja a mesma;
- Apresenta-se um diagrama de pacotes e cada pacote possui um diagrama de classes próprio.

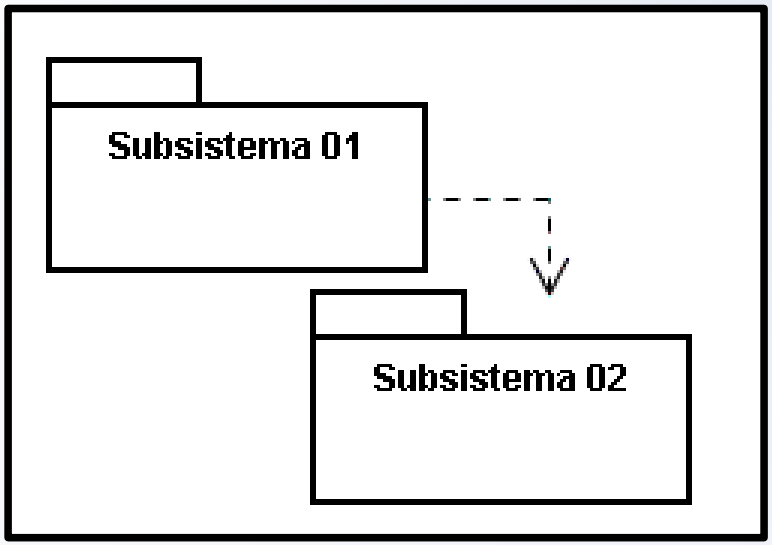


Associações entre pacotes



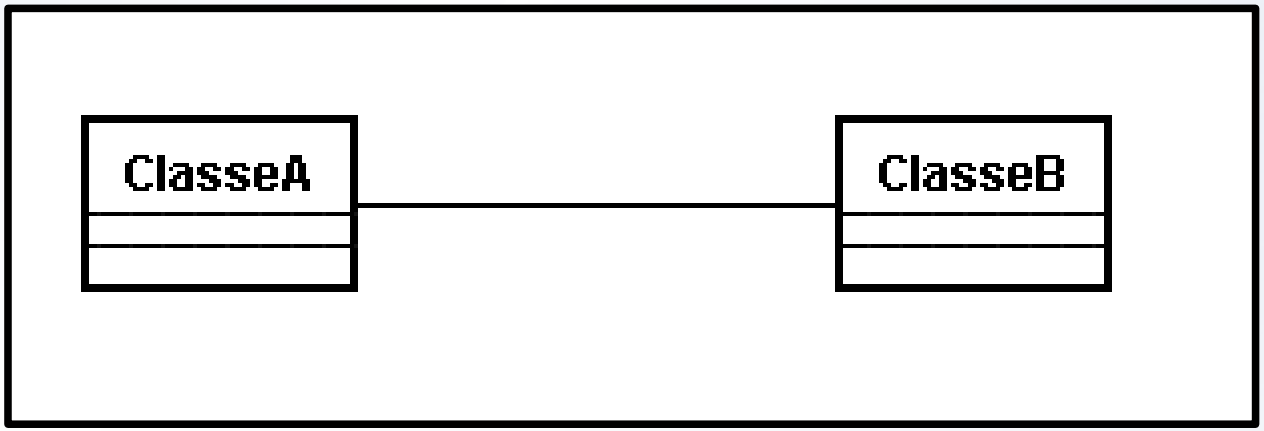
Nos pacotes de classes: classes do Pacote 02 aparecem no diagrama de classes do Pacote 01 (como se fossem "importadas").

Nos pacotes de casos de uso: casos de uso do Subsistema 01 dependem (<<include>>, <<extends>>, etc.) de casos de uso do Subsistema 02.



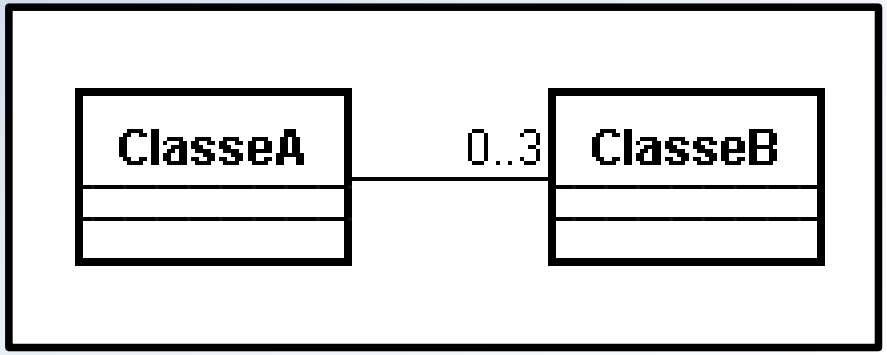
Identificando relacionamentos

- Classes se relacionam por meio de associações simples, agregações ou composições;
- Indicam possíveis ligações entre os objetos das classes associadas.

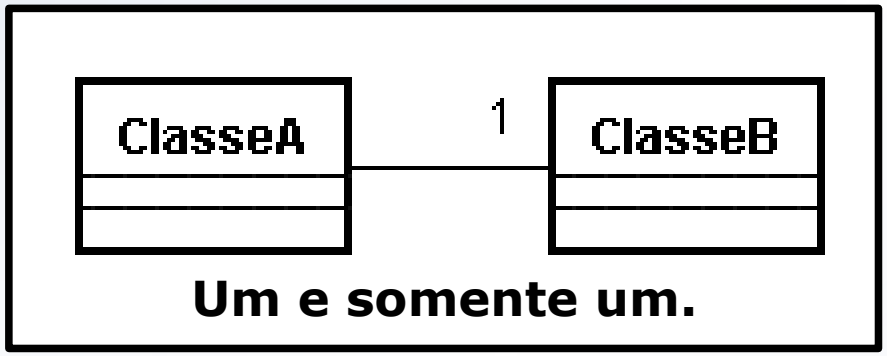


Cardinalidades

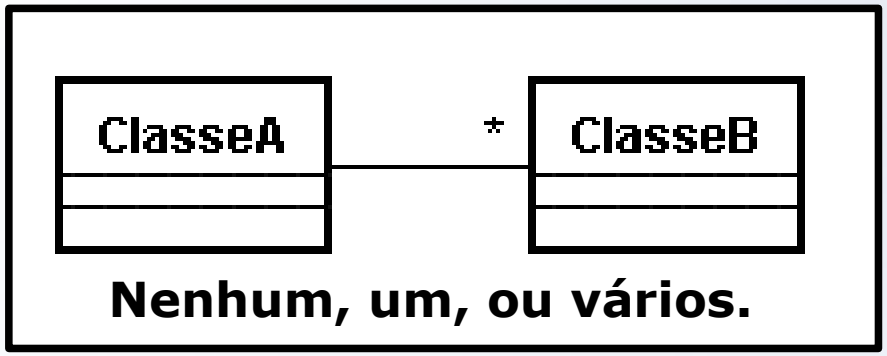
- Indicam quantos objetos podem participar de um determinado relacionamento.



Objetos da ClasseA podem se relacionar com no mínimo zero e no máximo três objetos da ClasseB.



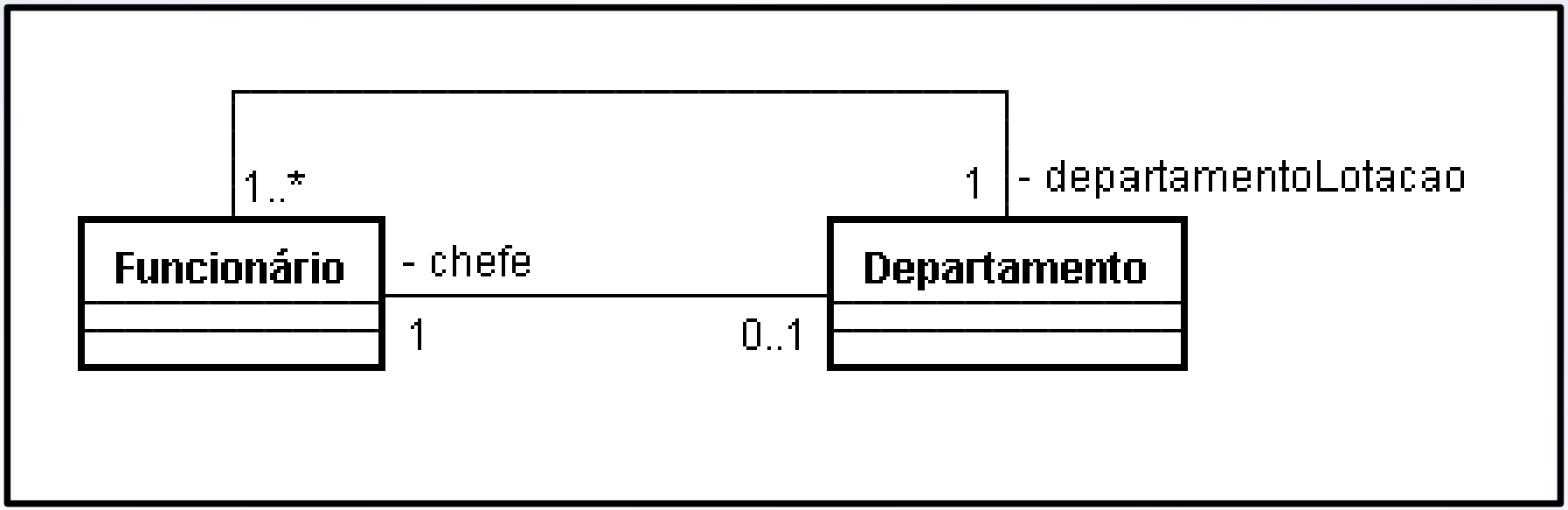
Um e somente um.



Nenhum, um, ou vários.

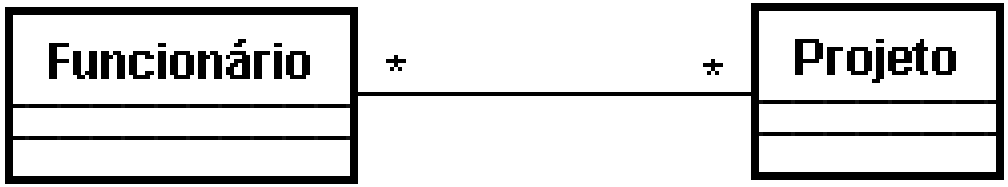
Papéis

- Indicam o papel que a classe desempenha na associação (são usados substantivos);
- É opcional, usado quando melhora o entendimento do modelo;
- Sintaxe: <escopo> <nome>.



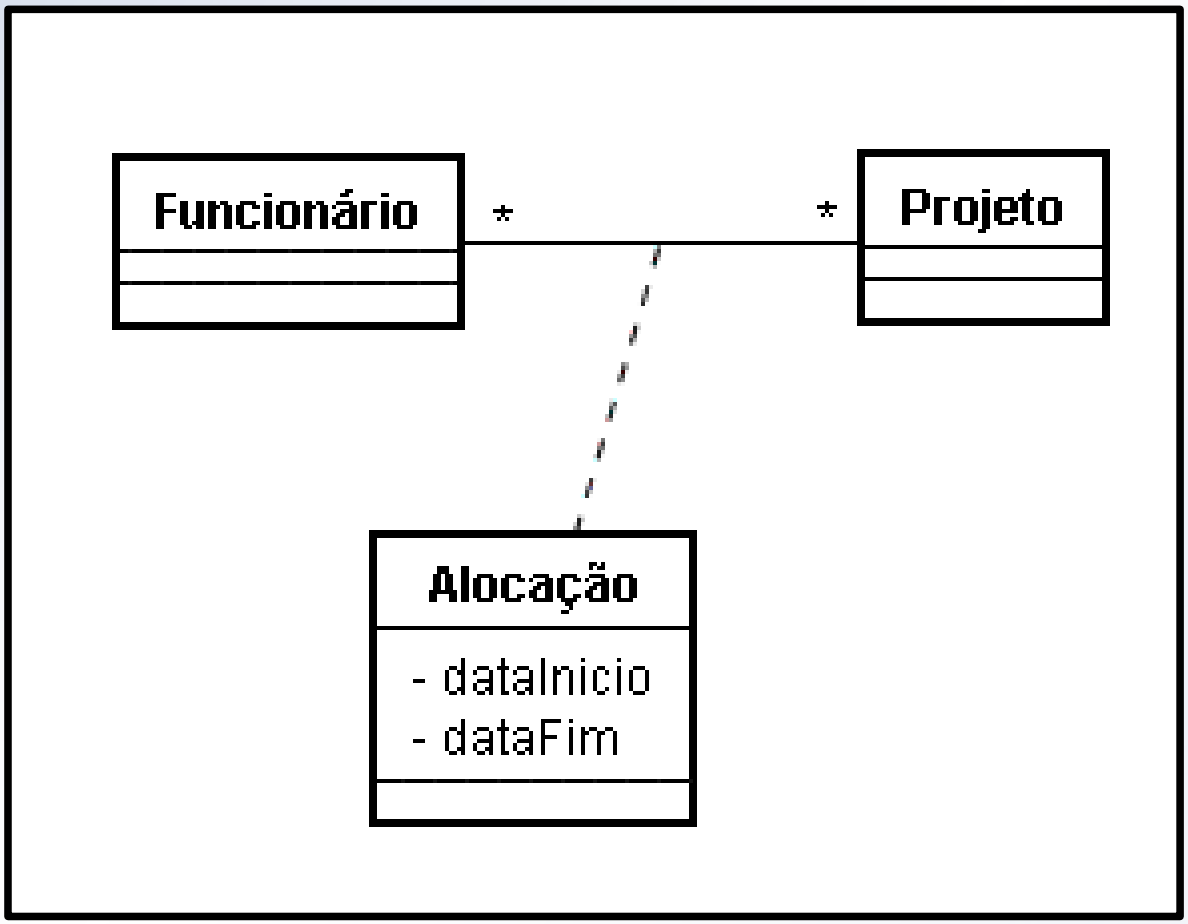
Relacionamentos “n-para-n”

- Perfeitamente legais;
- Requerem atenção à possíveis atributos do relacionamento (eventos a serem lembrados).



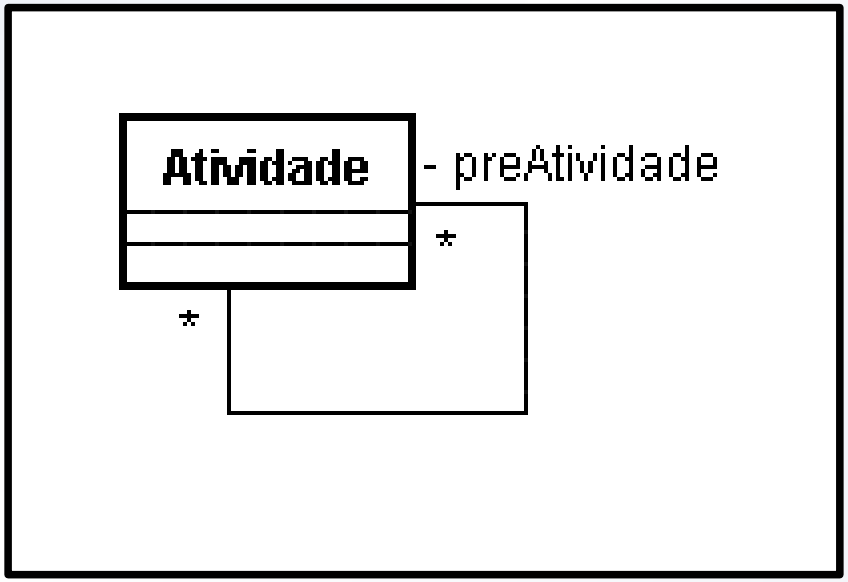
É requisito do sistema armazenar data de início e fim da alocação de um funcionário a um projeto?

Classes associativas



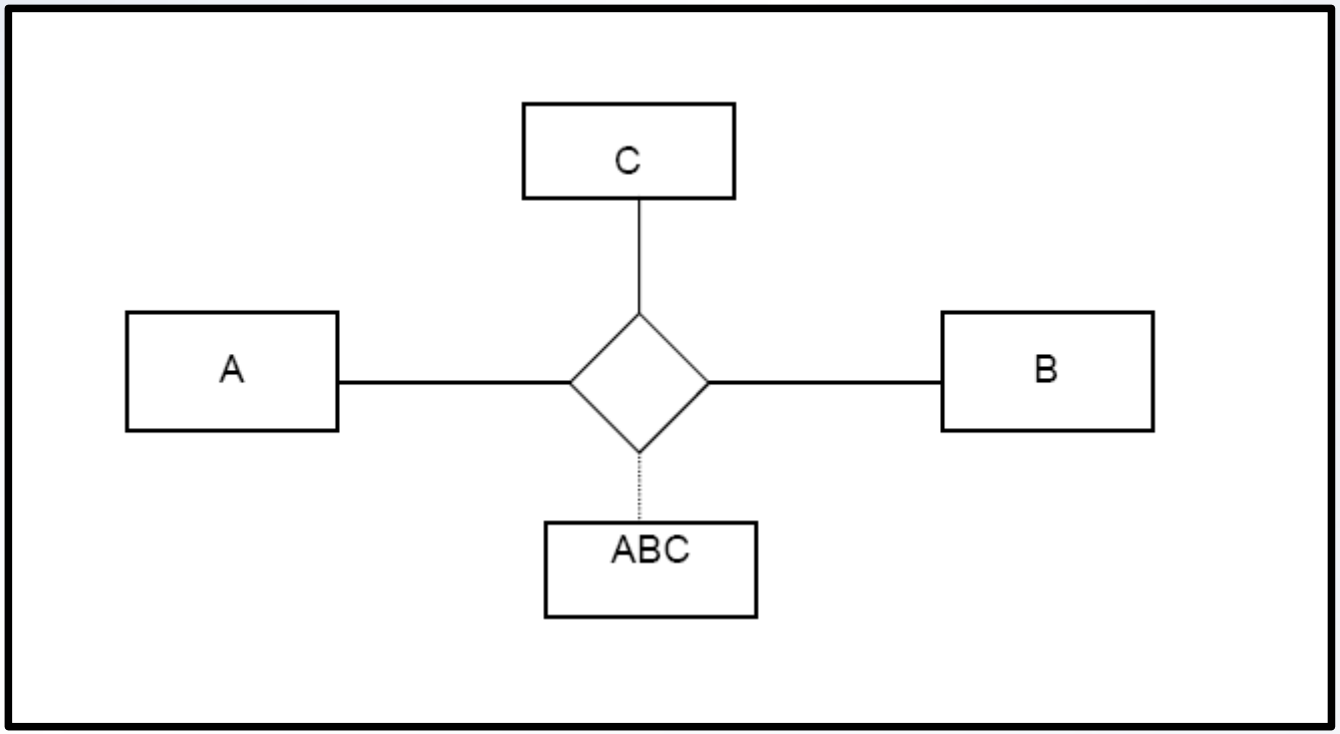
Relacionamentos recursivos

- Perfeitamente legais;
- Geralmente pedem definição de papéis.



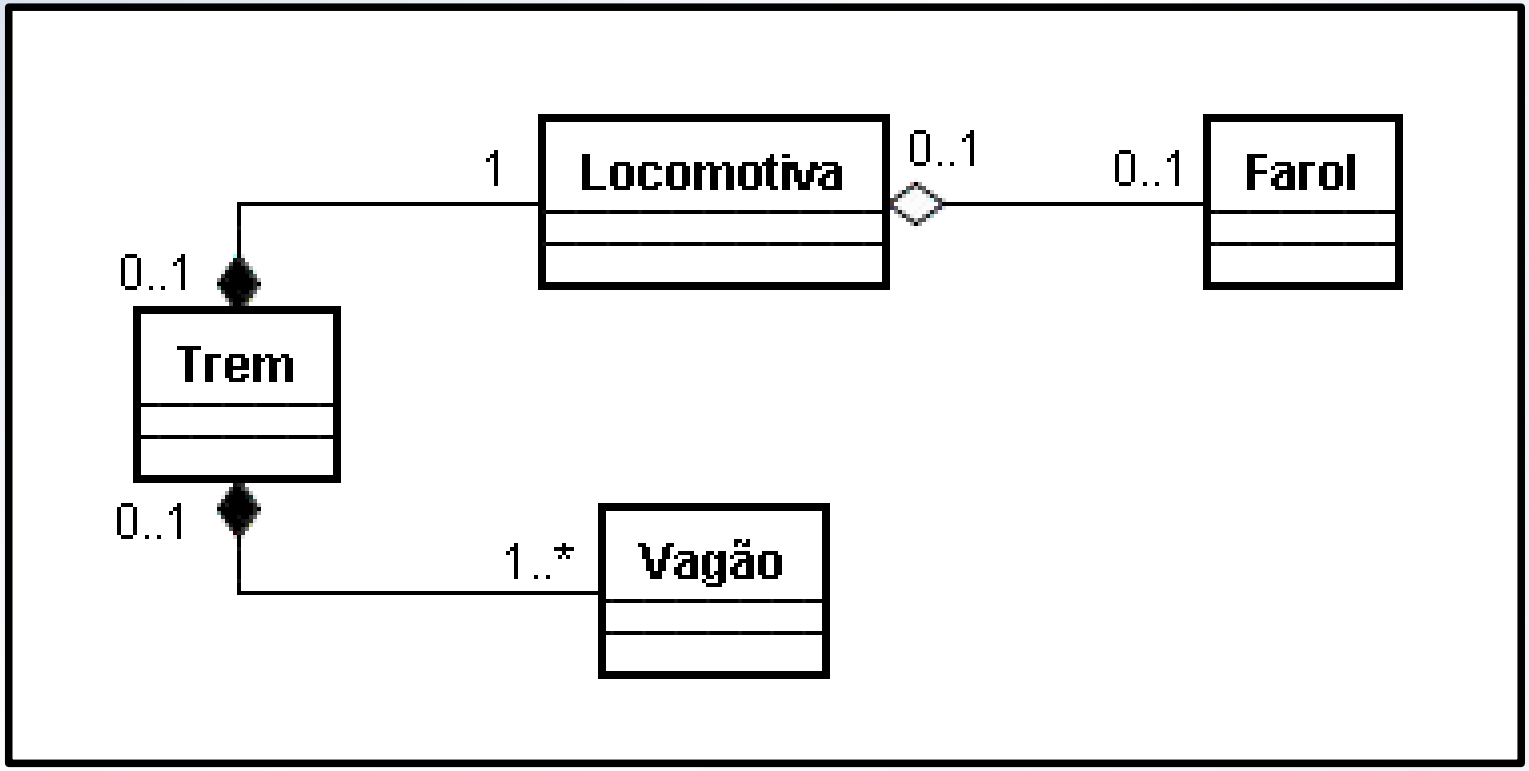
Associações n-árias

- Associações entre três ou mais classes;
- Extremamente raras, muitas vezes as ferramentas CASE nem dão suporte.



Agregação e composição

- Tipos especiais de associação, já estudados.



Qual a semântica da agregação

- Não há consenso;
- Uma visão:
  - Na agregação ou composição, as partes só podem existir como membros do todo;
  - A composição difere da agregação por restringir que a parte só participe de 1 todo.
- Outra visão:
  - Na agregação, as partes podem existir sem o todo e vice-versa;
  - A composição difere da agregação por restringir que o todo não vive sem as partes.

Atributos de classes

- Atributos são informações de estado (propriedades) para o qual cada objeto em uma classe tem seu valor;
- Muito similares às associações:
  - Como atributos têm um tipo, podemos considerar que são associações com um tipo;
  - Para tipos primitivos definimos atributos, do contrário modelamos uma associação;
  - Em última instância, associações e atributos são implementados da mesma forma;
  - Atributos e associações definem uma classe.

Como identificar atributos

- Quatro passos:
  - Descoberta de atributos;
  - Posicionamento dos atributos em uma hierarquia de classes;
  - Revisão da hierarquia de classes;
  - Especificação dos atributos.

Descoberta de atributos

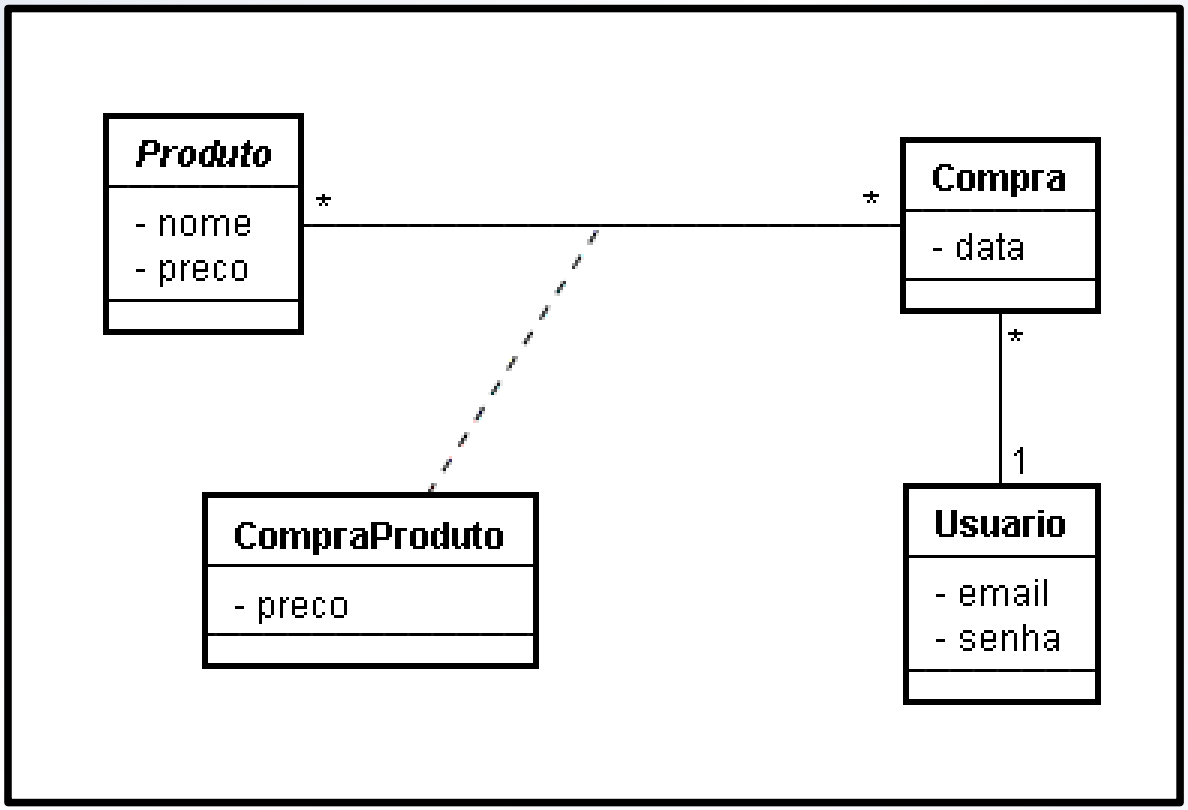
- Não há “mágica”. Usamos as mesmas técnicas da descoberta de classes;
- Características de um atributo:
  - Informação relevante no contexto do problema;
  - Captura um conceito atômico (do contrário seria uma classe).
- Muitas classes identificadas e que não passam nos critérios de classe podem vir a ser atributos.

Posicionamento de atributos

- Atenção à hierarquias de classes:
  - Atributos genéricos ficam mais acima na hierarquia;
  - Por outro lado, se ele não se aplica a algumas subclasses, deve ser trazido “para baixo”, somente para as classes apropriadas.
- Revisão da hierarquia:
  - Descoberta de atributos nos leva a um melhor entendimento, o que possivelmente implicará revisão de hierarquias.

Especificação de atributos

- Escolha um nome com significado;
- Siga um padrão de nomenclatura;
- Inclua-o na modelagem de classes:



Dicionário de dados

- Incluído na especificação de análise;
- Descreve cada atributo:
  - Seu significado no domínio;
  - Unidades de medida;
  - Intervalo / limite;
  - Enumeração;
  - Precisão;
  - Valor default;
  - Obrigatoriedade;
  - Etc.

Modelagem dinâmica

Prof. Vítor Souza

Análise e Projeto Orientado a Objetos

Departamento de Informática
Univ. Federal do Espírito Santo

11/04/2006

Comportamento dinâmico

- Os diagramas de classes representam apenas elementos estáticos, dados;
- É preciso, ainda, representar o comportamento da aplicação em função do tempo e de eventos específicos;
- Modelar o comportamento:
  - Indica como o sistema irá responder a eventos ou estímulos externos;
  - Auxilia o processo de descoberta das operações das classes do sistema.

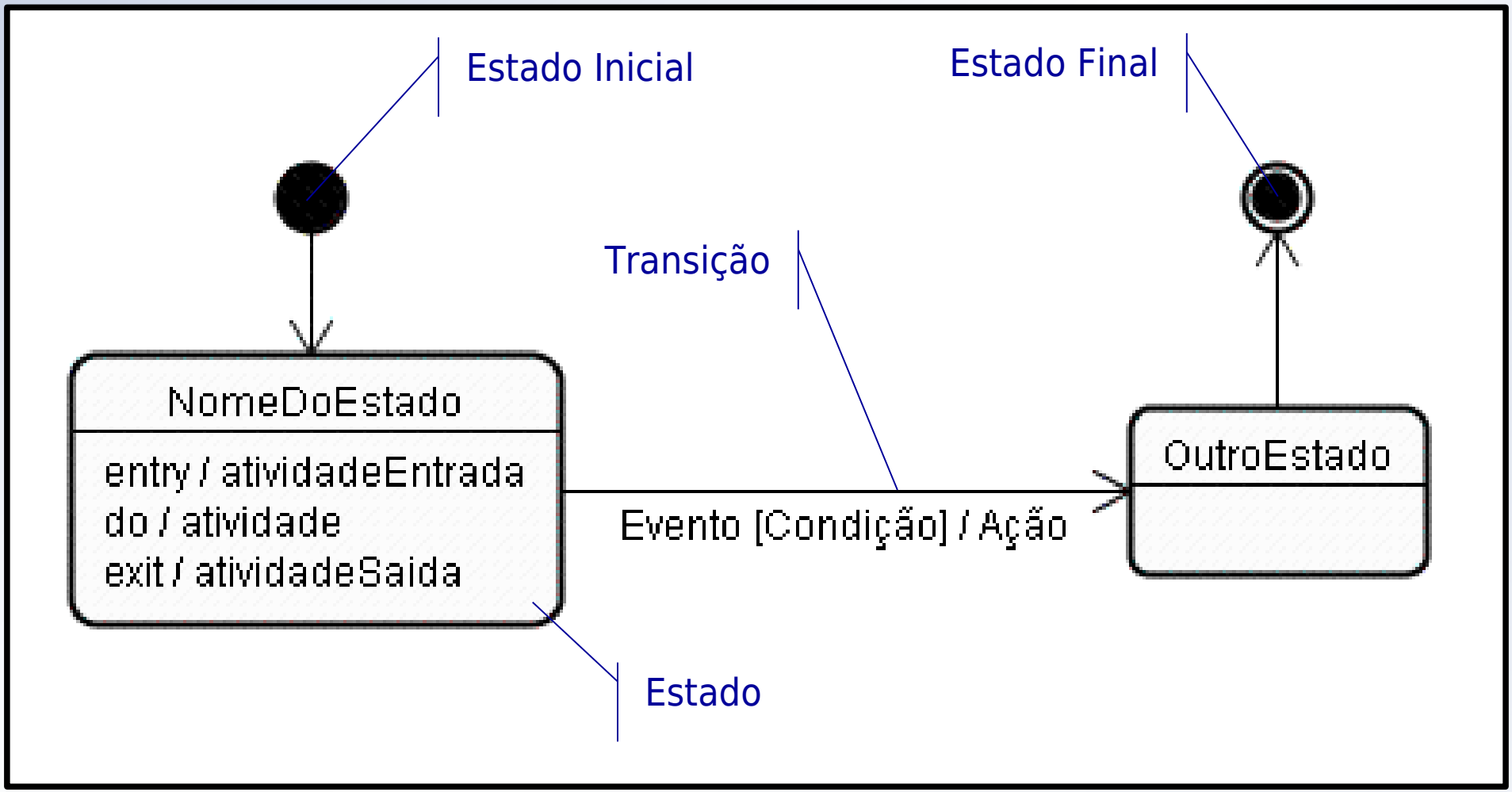
Modelos da UML

- Diagramas de Estados:
  - Descrevem os estados possíveis pelos quais um particular objeto pode passar e suas transições, estímulos e atividades;
- Diagramas de Interação:
  - Descrevem como grupos de objetos colaboram entre si em um certo comportamento.

Diagrama de Máquina de Estados

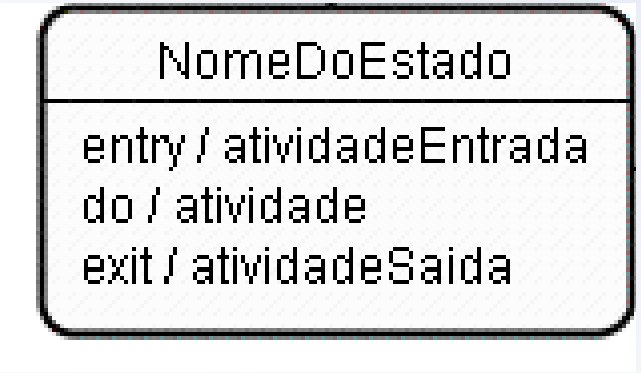
- Similar ao Diagrama de Transição de Estados (DTE), do paradigma Estruturado;
- Construído para uma única classe;
- Mostra o comportamento dos objetos desta classe ao longo do tempo:
  - Estados possíveis;
  - Eventos que alteram estado;
  - Características de cada estado;
  - Etc.

Notação


Estado Inicial
Estado Final
Transição
NomeDoEstado
entry / atividadeEntrada
do / atividade
exit / atividadeSaida
OutroEstado
Evento [Condição] / Ação
Estado

Estados

- Nome: deve descrever claramente o estado;
- Atividades:
  - De entrada: ocorrem ao entrarmos no estado;
  - De saída: ocorrem ao saírmos do estado;
  - Convencional: ocorrem enquanto o objeto estiver naquele estado.



Estados inicial e final

- Os estados ligados ao estado inicial são aqueles nos quais o objeto pode estar logo quando for construído;
  - Deve haver um e somente um estado inicial;
- Os estados ligados a um estado final são aqueles dos quais o objeto não mais sairá;
  - Não é obrigatório ter e pode ter mais de um;
  - Um estado ligado a um estado final não pode ter transições para outros estados.

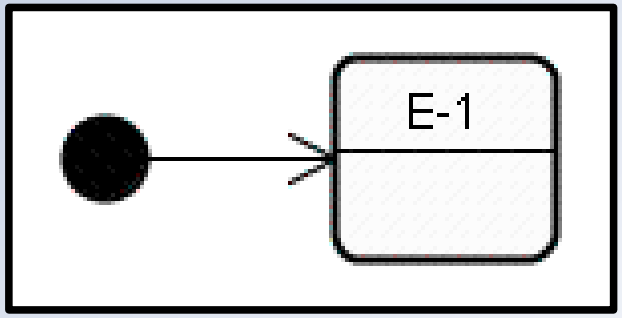
Eventos, condições e ações

- Indicam a possibilidade de ir de um estado a outro;
  - Evento: evento externo que motivou a transição;
  - Condição de guarda: condição necessária para efetuar a transição (além do evento);
  - Ação: ação realizada durante a transição.

Ação x atividade

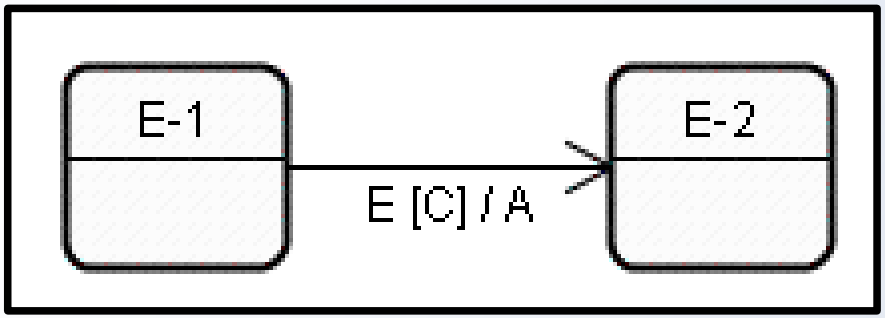
- Ação x atividade convencional:
  - Uma ação ocorre durante a transição (ela é atômica);
  - Uma atividade ocorre enquanto o objeto permanece no estado (quando ela acabar, ele muda de estado);
- Ação x atividades de entrada/saída:
  - Caso só haja uma transição de entrada/saída, não há diferença na prática.

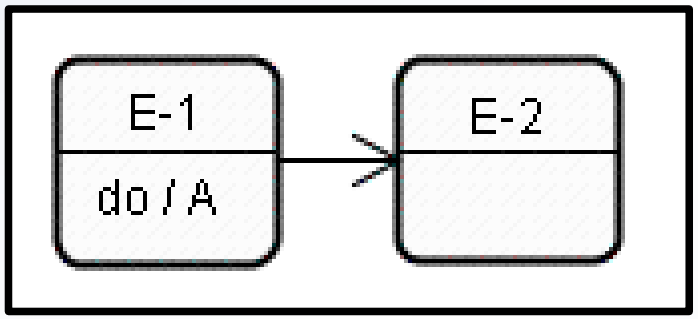
Exemplos



Ao ser criado, o objeto da classe em questão encontra-se no estado E-1.

Se o objeto estiver no estado E-1 e ocorrer o evento E, se a condição C = true, ocorre a ação A e o objeto passa para o estado E-2.





Enquanto estiver no estado E-1, o objeto realiza a atividade A. Ao terminar a atividade A, automaticamente ele muda para o estado E-2.

Como identificar estados

- Somente para algumas classes:
  - Que possuam comportamento variável no tempo – em cada estado se comportam de maneira diferente;
  - Nas quais seja possível identificar ao menos três estados distintos;
  - Cujo passado influencia no seu comportamento atual;

Um estado é uma situação durante a vida de um objeto durante a qual ele satisfaz alguma condição, realiza alguma atividade ou aguarda algum evento.

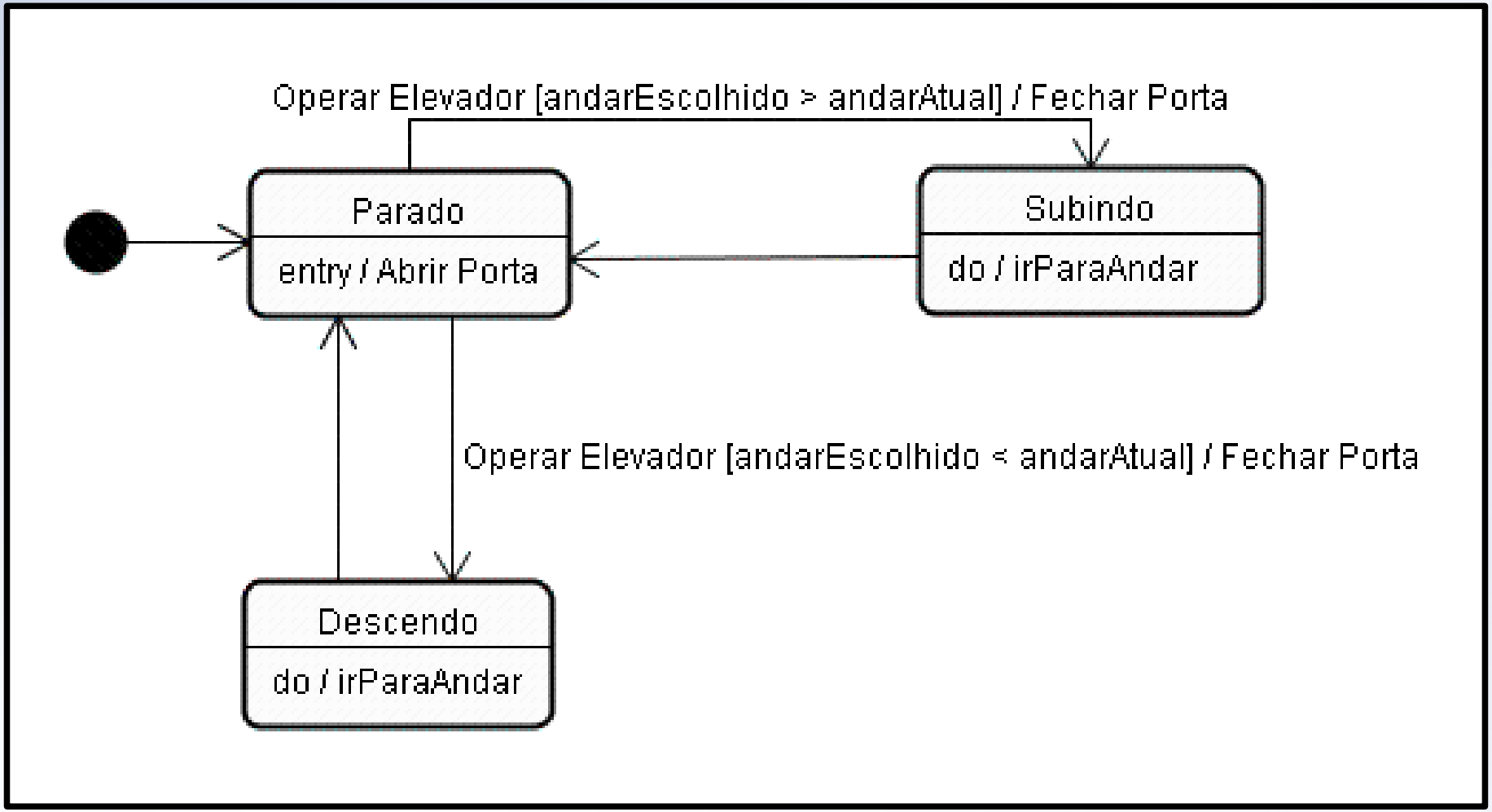
Como identificar transições

- Dado um estado, verificamos para quais outros estados ele pode passar e o evento em que isso ocorre;
- Na grande maioria das vezes, os casos de uso ou cenários são os eventos que acionam as transições;
- Após identificadas, avalie condições de guarda e ações.

Dicas práticas

- Preste atenção nas classes próximas, pois podem ser alvos de ações ou participar de condições;
- Identifique na especificação os eventos aos quais os objetos da classe devem responder;
- Se um estado for muito complexo, considere criar um sub-diagrama só para ele;
- Verifique se todas as ações estão condizentes com sua modelagem de classes;
- Simule o sistema em execução e procure estados não-alcançáveis, loops ou deadlocks.

Mais um exemplo



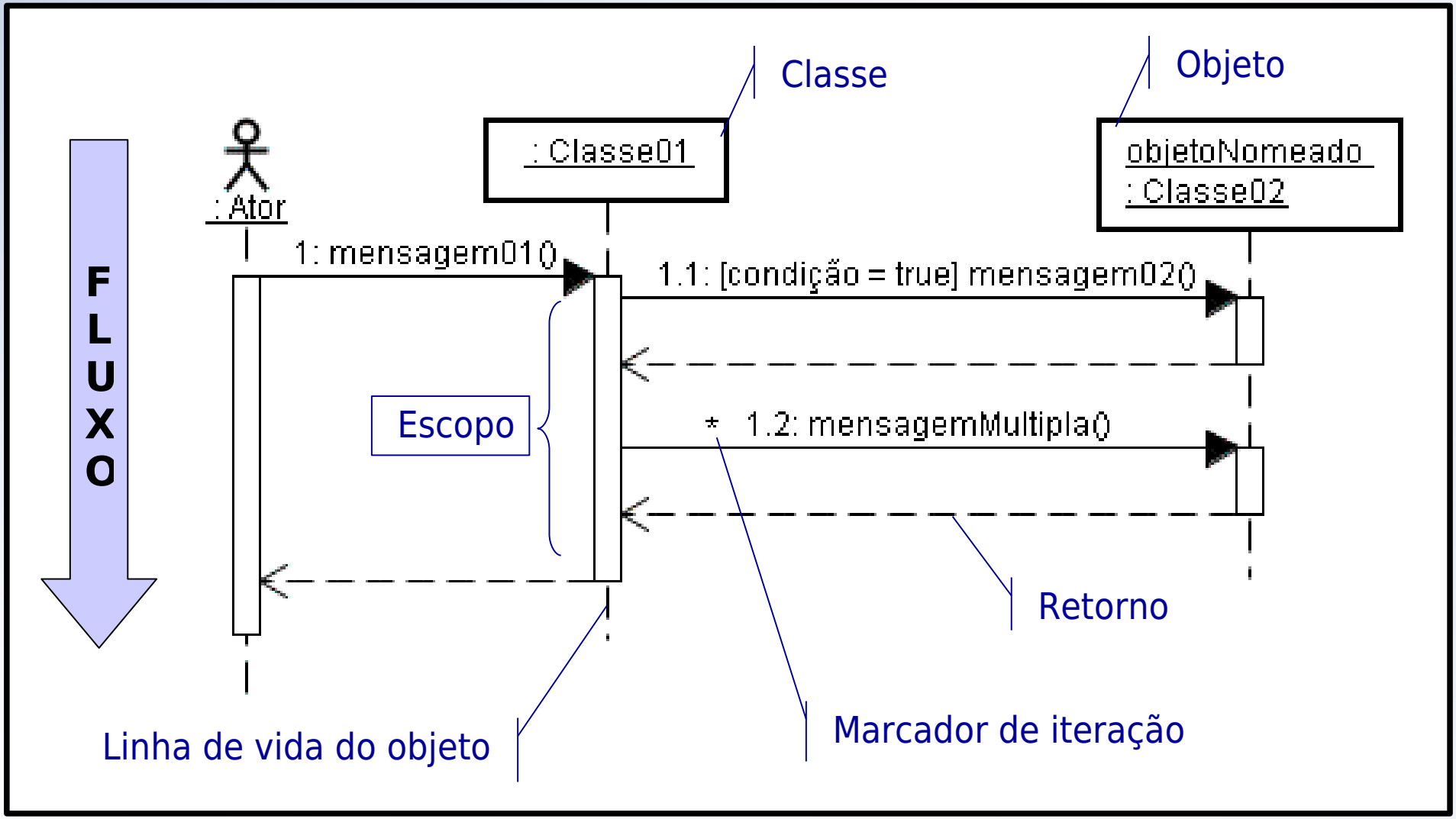
Diagramas de interação

- Ajudam a compreender e capturar o fluxo global de controle;
- Modelam um conjunto de objetos e as mensagens que trocam;
- Foco em uma funcionalidade específica (caso de uso);
- Dois tipos de diagrama:
  - Diagrama de Sequência (temporal);
  - Diagrama de Colaboração (estrutural).

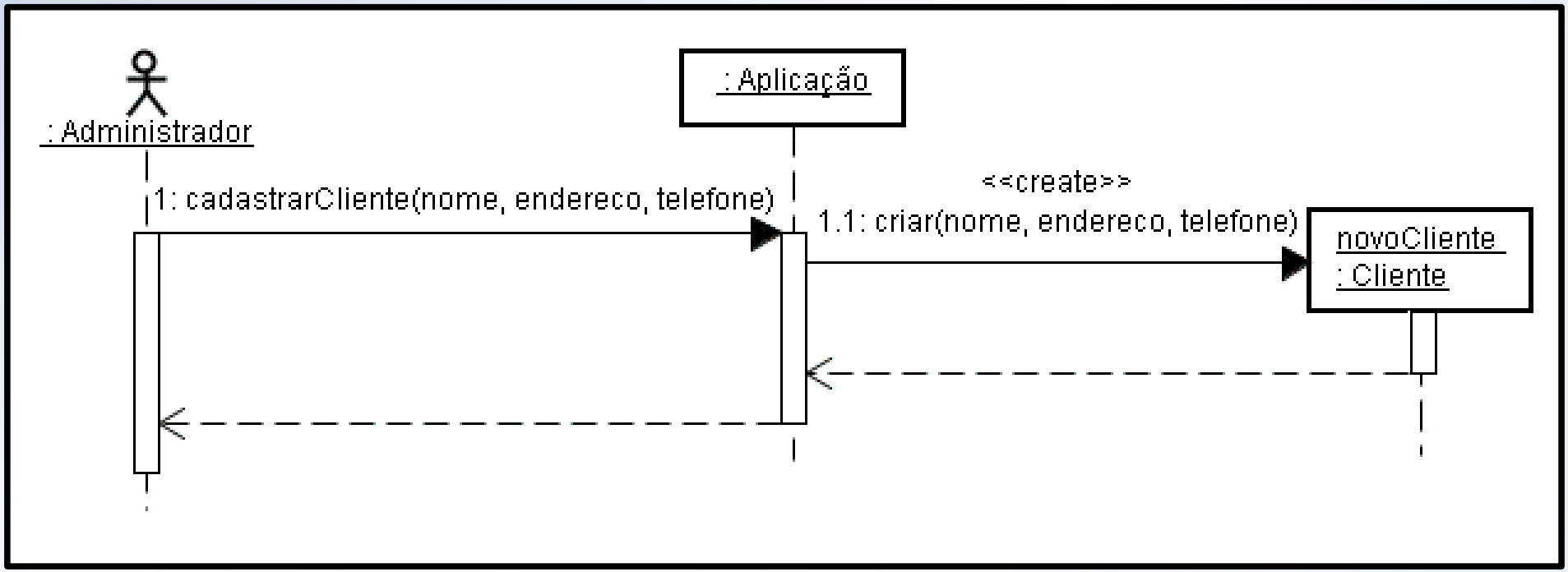
Diagramas de seqüência

- Foco no tempo e no controle;
- Usaremos como uma formalização de um fluxo de um caso de uso / cenário.

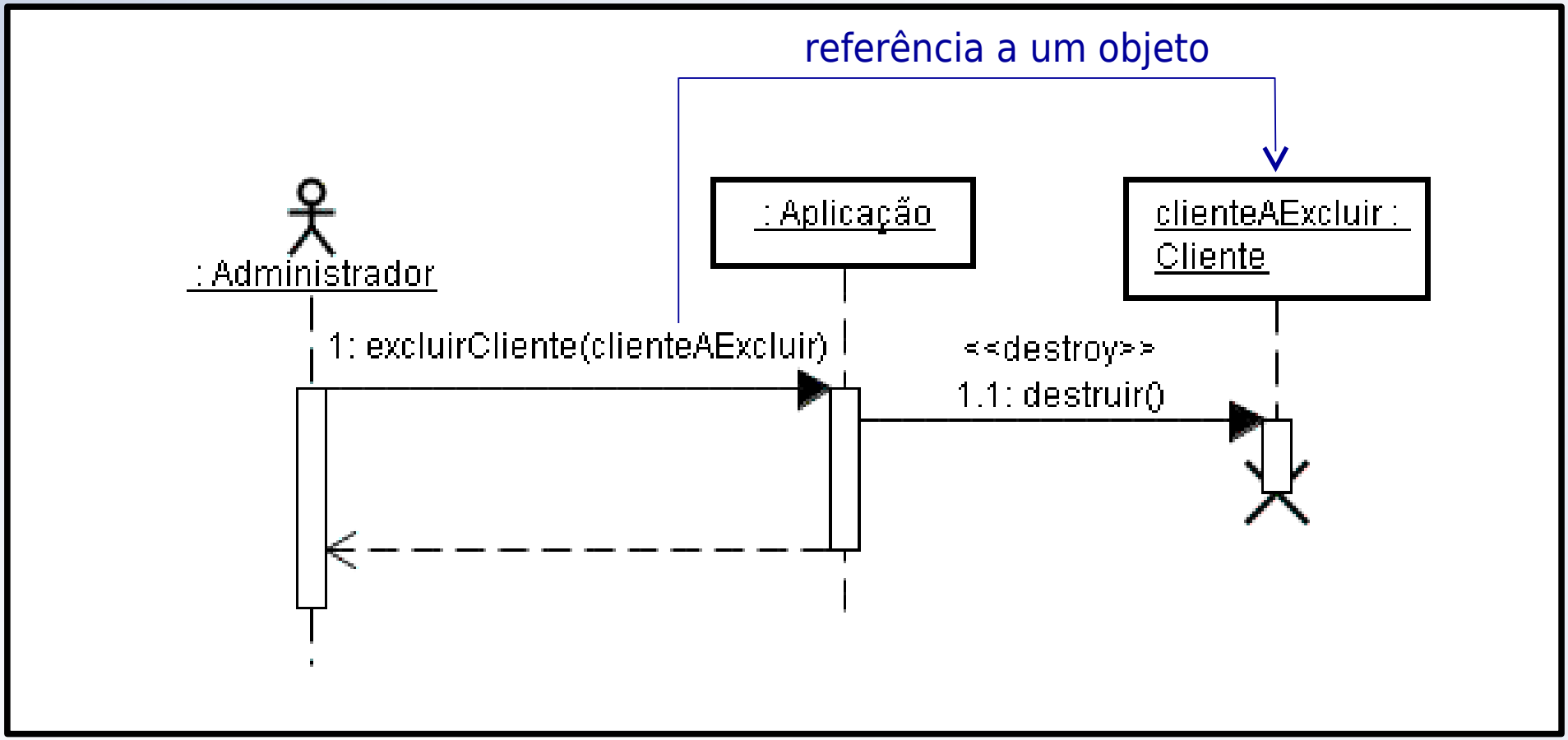
Notação geral



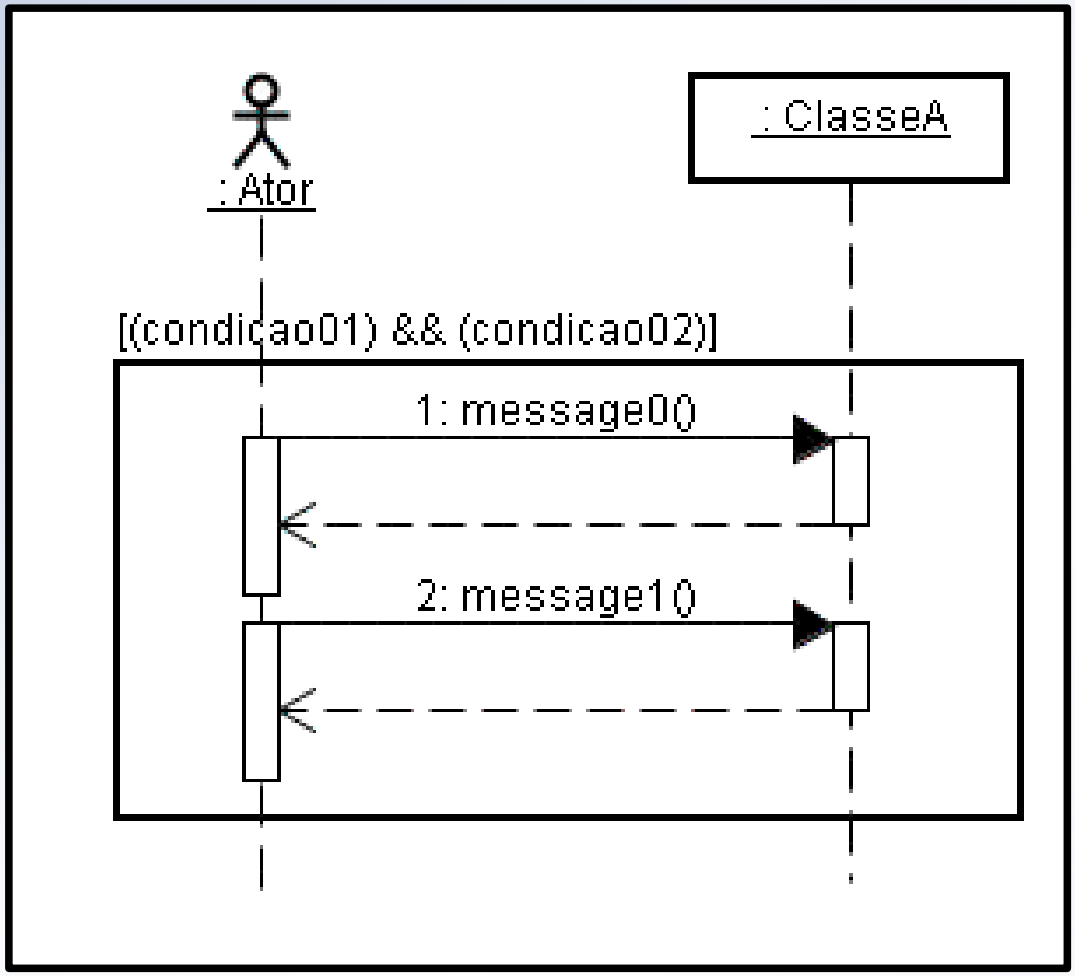
Criação de objetos



Exclusão de objetos



Condicionais e loops



Para loops:

```
[for i = 0; i < 10; i++]
```

ou

```
[for each obj in lista]
```

UML 2.0

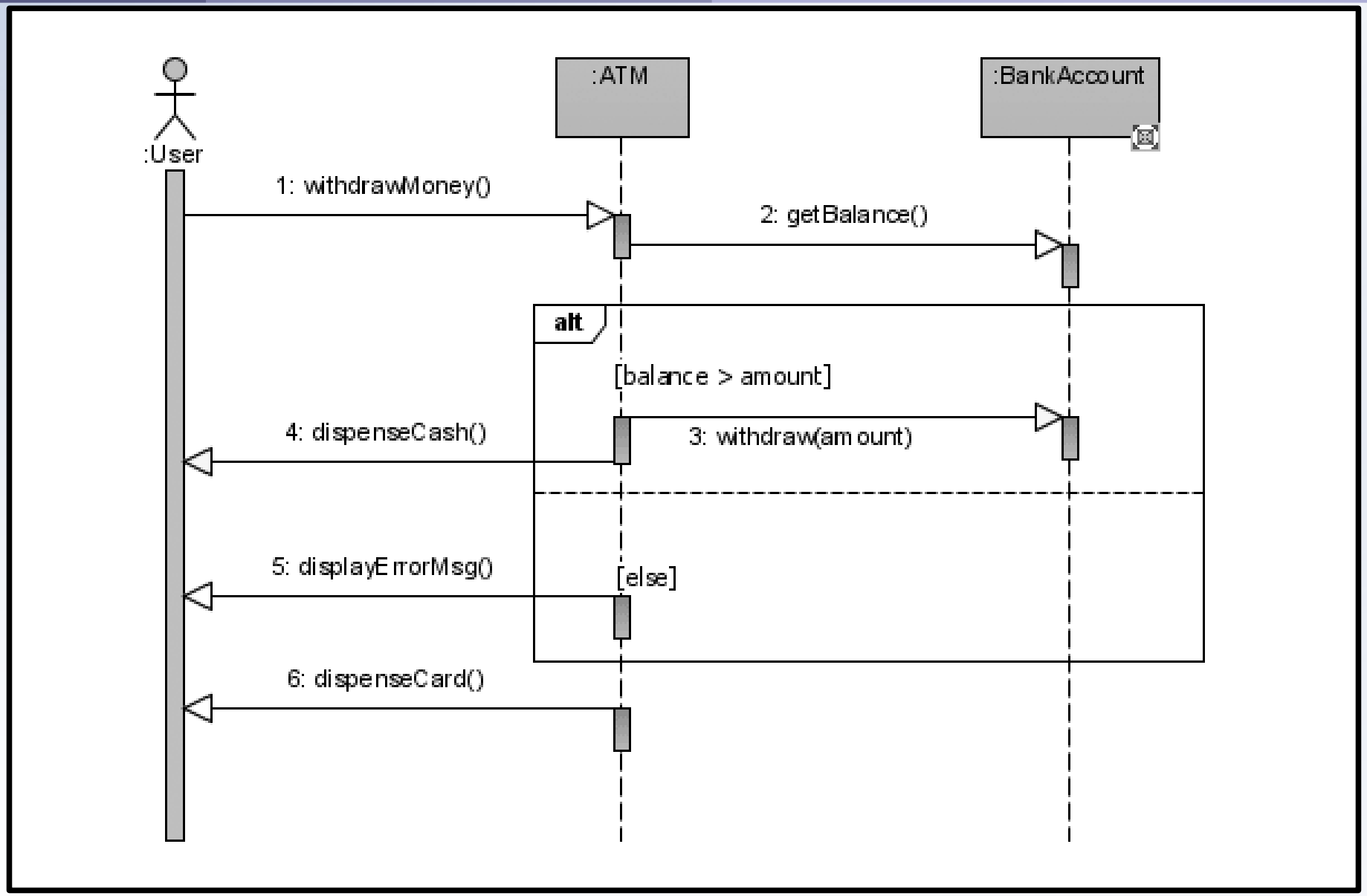
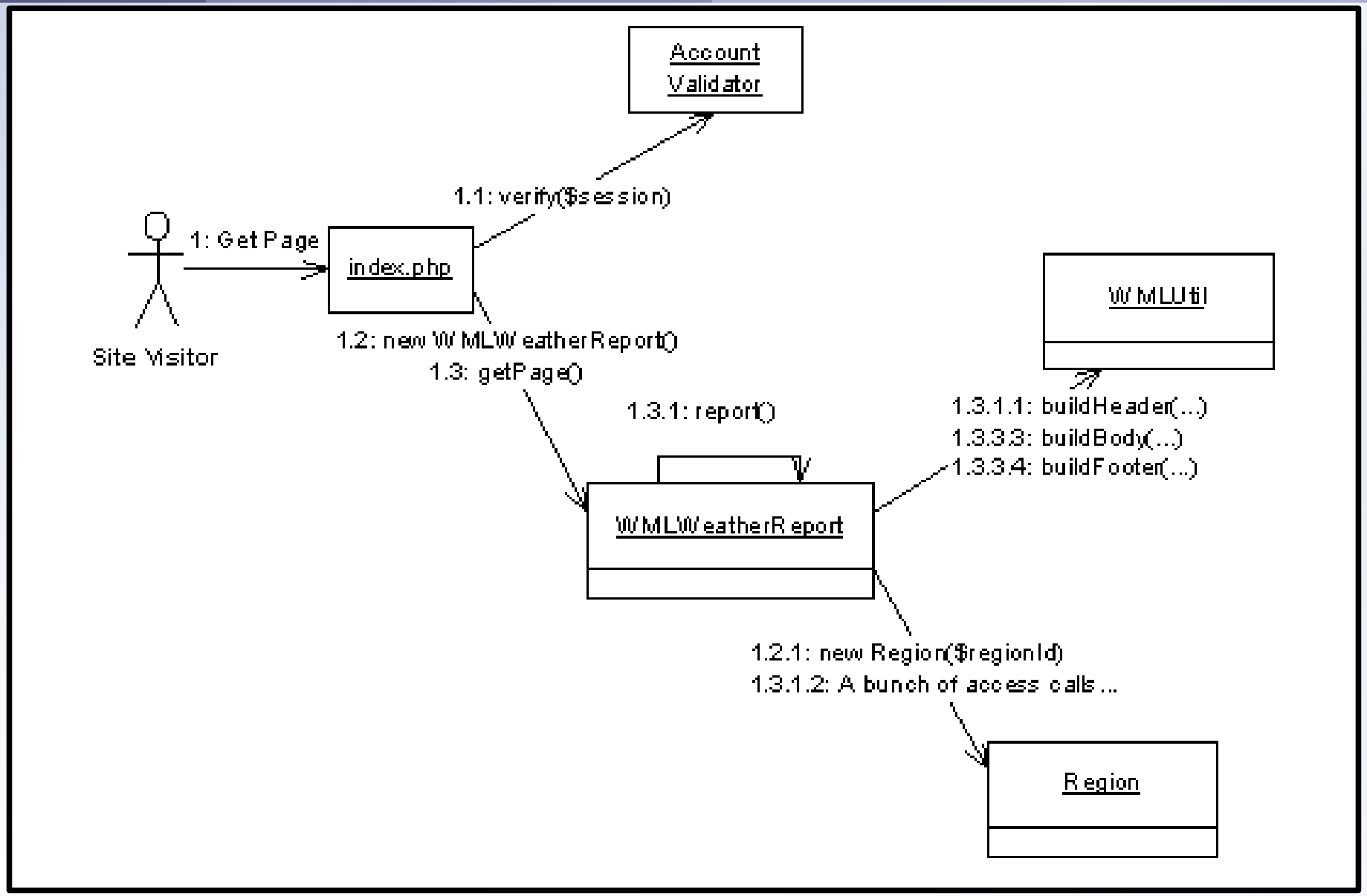

:User
:ATM
:BankAccount
1: withdrawMoney()
2: getBalance()
alt
[balance > amount]
4: dispenseCash()
3: withdraw(amount)
5: displayErrorMsg()
[else]
6: dispenseCard()

Diagrama de colaboração

- Ênfase na organização dos objetos que participam da interação;
- Notação similar a grafos: objetos são vértices e mensagens são arestas.

Exemplo


Account Validator
1.1: verify($session)
1: Get Page
index.php
Site Visitor
1.2: new WMLWeatherReport()
1.3: getPage()
WMLUtil
1.3.1: report()
1.3.1.1: buildHeader(...)
1.3.3.3: buildBody(...)
1.3.3.4: buildFooter(...)
WMLWeatherReport
1.2.1: new Region($regionId)
1.3.1.2: A bunch of access calls...
Region