

Modelação Visual Com UML e Rational Rose 2000

Caso de Estudo

“Sistema de Gestão dos Cursos da Eastern
State University (ESU)”



José A. M. Cordeiro,
(j.cordeiro@computer.org)

Esc. Sup. Tec. de Setúbal – Janeiro 2001

Retirado do livro de **Terry Quatrani**
Visual Modeling with Rational Rose 2000 and UML

Modelação Visual, UML e Rational Rose 2000

◆ Objectivos

- Demonstrar com exemplos práticos a utilização da **linguagem UML** nas várias fases de desenvolvimento de um projecto
- Mostrar uma possível **metodologia de desenvolvimento** de uma aplicação seguindo os princípios da orientação por objectos
- Rever a utilização e a composição dos vários **diagramas UML**
- Fornecer algumas pistas práticas para a identificação e construção dos vários **elementos (*things*) e relações do UML**
- Ilustrar a utilização e organização do *Rational Rose 2000*

Modelação Visual, UML e Rational Rose 2000

♦ Objectivos (cont...)

■ NOTAS:

- Não se pretende desenvolver a aplicação completa uma vez que se perderia muito tempo com os detalhes
- Utiliza-se o processo unificado da Rational (RUP). A exemplificação ou descrição deste processo não estão entre os objectivos desta apresentação
- Não se vai demonstrar completamente a utilização e as capacidades do *Rational Rose 2000*



Caso de Estudo Gestão dos cursos da ESU



- ◆ Gere as disciplinas de um curso permitindo a atribuição de professores e a inscrição e matrícula de alunos

Resumo:

- Professores escolhem disciplinas a leccionar
- Produzida listagem de disciplinas e professores
- Alunos inscrevem-se e matriculam-se nas disciplinas
- Produzida listagem de disciplinas e alunos matriculados

Descrição do Problema

Gestão dos cursos da ESU

Sistema actual



- ◆ Atribuição de professores às disciplinas
 - Os professores definem quais as disciplinas que irão leccionar nesse semestre
 - A secretaria introduz os dados e emite uma listagem para cada professor com as disciplinas que vão leccionar
- ◆ Inscrição dos alunos
 - A secretaria produz uma listagem para os alunos das disciplinas disponíveis nesse semestre
 - Os alunos preenchem um formulário em que escolhem as disciplinas que pretendem ter (até 4) e entregam-no na secretaria

Descrição do Problema

Gestão dos cursos da ESU

Sistema actual


- ◆ Inscrição dos alunos (cont...)
 - A secretaria introduz os dados e um sistema automático atribui os alunos às disciplinas
 - No caso de haver problemas a secretaria contacta os alunos para obter escolhas adicionais
 - No final é entregue aos alunos uma listagem das disciplinas que irão frequentar para que eles efectuem as suas matrículas
- ◆ Listagem de Disciplinas
 - Após o período de inscrição os professores recebem a listagem das disciplinas a leccionar com a lista dos alunos matriculados





Descrição do Problema

Gestão dos cursos da ESU

Solução pretendida




- ◆ Professores - Gestão das disciplinas
 - Acesso *online* ao sistema para escolha das disciplinas a leccionar e para saber no final quais os alunos matriculados nas mesmas
- ◆ Alunos - Inscrição e Matricula
 - Recebem um catálogo do curso com a lista de disciplinas que inclui o docente, o departamento e os pré-requisitos necessários
 - Escolhem *online* até 4 disciplinas, e deverão indicar 2 opcionais
 - As disciplinas poderão ter no máximo 10 alunos e no mínimo 3 alunos (senão serão canceladas)



Descrição do Problema

Gestão dos cursos da ESU

Solução pretendida

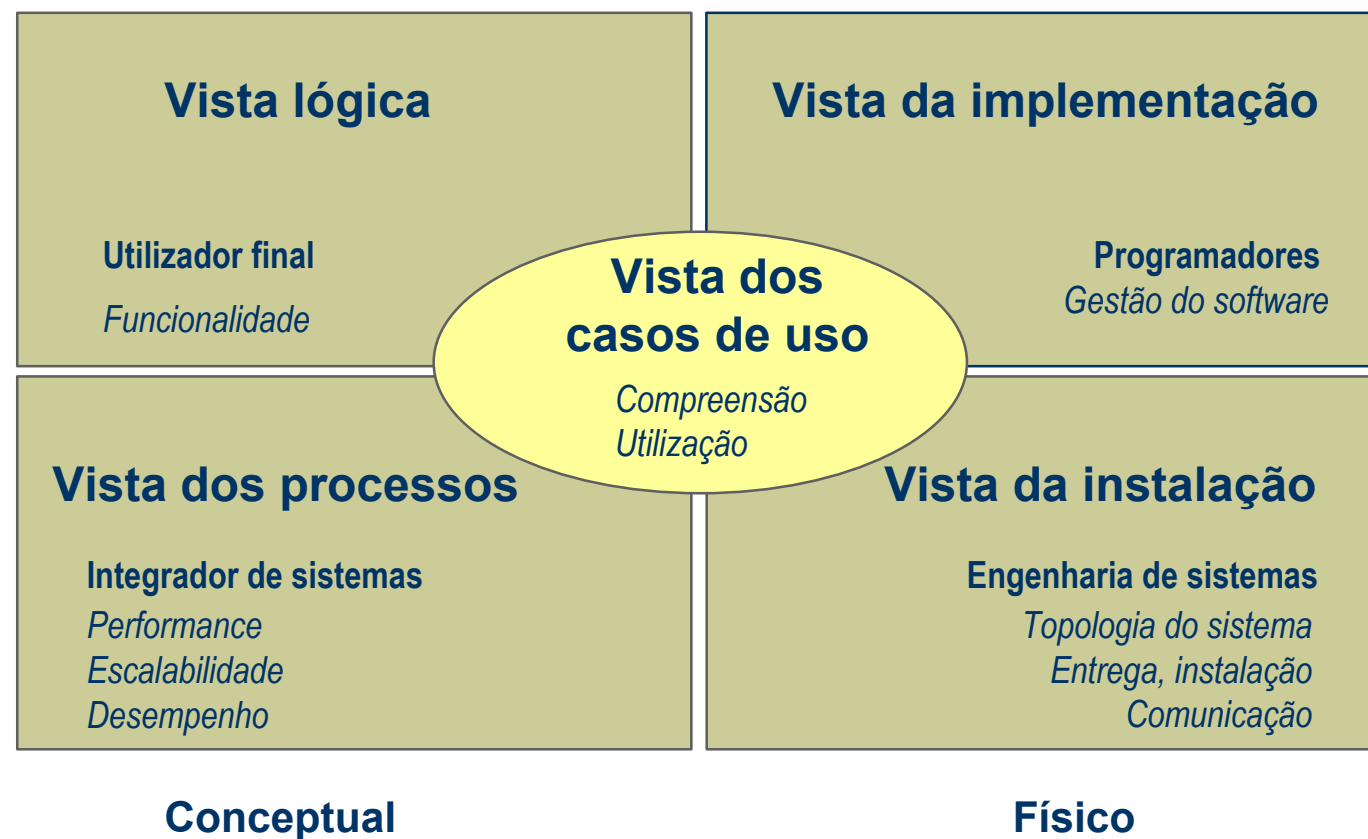


- ◆ Alunos - Inscrição e Matricula (cont...)
 - Terão acesso *online* ao sistema durante um certo período de forma a poderem adicionar e alterar disciplinas à sua selecção inicial
 - A matrícula ser-lhes-à cobrada através de um sistema de facturação externo. Este sistema irá receber a informação necessária a partir do sistema de gestão de cursos

Arquitectura da Solução

Modelo das 4+1 vistas da arquitectura

(Usado no *Rational Rose 2000*)



Arquitectura da Solução Vista dos Casos de Uso

- ◆ Vai **documentar o comportamento** do sistema, (ou seja a funcionalidade que o sistema irá disponibilizar aos seus utilizadores) principalmente através da identificação de:
 - actores,
 - casos de uso,
 - relações entre actores e casos de uso com a criação dos diagramas de casos de uso
- ◆ Facilita a comunicação com o cliente e com os utilizadores finais

Vista dos Casos de Uso

Actores

Actor: Representa alguém ou algo que necessita de interagir com o sistema, mas que não faz parte do mesmo

- ♦ **Interagem com o sistema** - Fornecem e/ou recebem informação do sistema
- ♦ **Não fazem parte do sistema**

Vista dos Casos de Uso

Identificação de actores

Pistas para identificação

- ◆ Colocar as seguintes **questões**:
 - A quem interessa determinado requisito?
 - Em que lugar da organização o sistema é usado?
 - Quem beneficia com a utilização do sistema?
 - Referindo determinada informação, quem a fornece, a utiliza e a apaga?
 - Quem mantém e dá apoio ao sistema?
 - O sistema usa algum recurso exterior?
 - Alguém desempenha papeis diferentes?
 - Um mesmo papel é desempenhado por várias pessoas?
 - O sistema interage com algum *legacy system*?

Vista dos Casos de Uso

Identificação de actores

Criação de actores

- ◆ É um **processo iterativo** – a primeira lista raramente é a definitiva
 - Ex: Um novo aluno representa um actor diferente de um aluno que retorna?
 - Ex: Criação de um actor para cada papel que alguém pode desempenhar. Caso do professor-assistente.
- ◆ **Documentar** os actores – a descrição deve identificar o papel que o actor representa quando interage com o sistema



Vista dos Casos de Uso **Identificação de actores** Eastern State University



- ◆ **Aluno** - alguém que se matricula para ter aulas na Universidade
- ◆ **Professor** - alguém certificado para dar aulas na Universidade
- ◆ **Funcionário da secretaria** – alguém responsável pela manutenção do sistema de gestão de cursos da Universidade
- ◆ **Sistema de Facturação** - sistema externo responsável pela cobrança das matrículas aos alunos da Universidade

Vista dos Casos de Uso

Casos de Uso

Caso de uso: É a descrição de uma sequência de acções realizadas por um sistema que originam um resultado mensurável com interesse para um determinado actor

- ◆ Modelam um diálogo entre um actor e o sistema: Representam as **funcionalidades disponibilizadas** pelo sistema ao actor
- ◆ O conjunto de todos os casos de uso define todas as maneiras de como o sistema pode ser usado

Vista dos Casos de Uso

Identificação dos casos de uso

Pistas para identificação

- ◆ Colocar as seguintes **questões**:
 - Quais as tarefas de cada actor?
 - Algum actor cria, guarda, altera, apaga ou lê informação no sistema?
 - Qual o caso de uso que cria, guarda, altera, apaga ou lê esta informação?
 - Algum actor necessita de informar o sistema de alterações externas súbitas?
 - Algum actor necessita de ser informado acerca de uma certa ocorrência no sistema?
 - Quais os casos de uso que mantêm e apoiam o sistema?
 - Estarão todos os requisitos funcionais realizados nos casos de uso?

Vista dos Casos de Uso

Identificação dos casos de uso

Criação de casos de uso

- ◆ Não há resposta para a questão: *Qual a dimensão que um caso de uso deve ter?*

Método prático

“Um caso de uso normalmente define uma funcionalidade do sistema importante e completa com início e fim e deve entregar algo de valor para determinado actor”

Ex: O aluno escolhe as disciplinas, é adicionado aos alunos da disciplina e é-lhe cobrada a matrícula. Três casos de uso ou apenas um?

Ex: O funcionário da secretaria deve adicionar cursos, apagar cursos e modificar cursos. São três casos de uso ou apenas um?

- ◆ **Documentar** os casos de uso – descrever a funcionalidade providenciada pelo caso de uso





Vista dos Casos de Uso

Identificação dos casos de uso

Eastern State University




- ◆ Criar a lista de disciplinas do curso
- ◆ Matricular nas disciplinas
- ◆ Requerer a lista de disciplinas a leccionar
- ◆ Seleccionar as disciplinas a leccionar
- ◆ Manter a informação das disciplinas
- ◆ Manter a informação dos professores
- ◆ Manter a informação dos alunos



Vista dos Casos de Uso

Identificação dos casos de uso

Documentação pormenorizada



◆ Fluxo de acções

- Deve descrever os passos necessários para realizar determinada funcionalidade e deve ser definido no sentido daquilo que o sistema **deve fazer** e não da maneira **como o faz**
- Deve incluir:
 - Quando e como o caso de uso começa e acaba
 - Que interacções o caso de uso tem com os actores
 - Que dados são necessários ao caso de uso
 - A sequência normal de acções do caso de uso
 - A descrição de sequências de acções alternativas ou de excepção

Vista dos Casos de Uso

Identificação dos casos de uso

Documentação pormenorizada

- ◆ **Fluxo de acções (cont...)**
 - É um **processo iterativo** iniciando-se com uma descrição breve que depois se vai detalhando.
 - Sugestão de um **modelo** para a sequência de acções
 - X. Sequência de acções para o caso de uso <nome>
 - X.1 Pré-condições
 - X.2 Acções principais
 - X.3 Subacções
 - X.4 Acções alternativas

Vista dos Casos de Uso

Relações envolvendo casos de uso e actores

Associação, inclusão e extensão

- ◆ **Relações possíveis:**
 - **Associação** – entre um caso de uso e um actor
 - **Inclusão** (<<Include>>) – para incluir casos de uso que reúnem funcionalidades comuns a mais que um caso de uso
Ex: Validar utilizador
 - **Extensão** (<<Extend>>) – para mostrar comportamentos opcionais, comportamentos que só ocorrem mediante determinadas condições ou diferentes sequências que dependem da escolha do utilizador
Ex: Nome do utilizador inválido

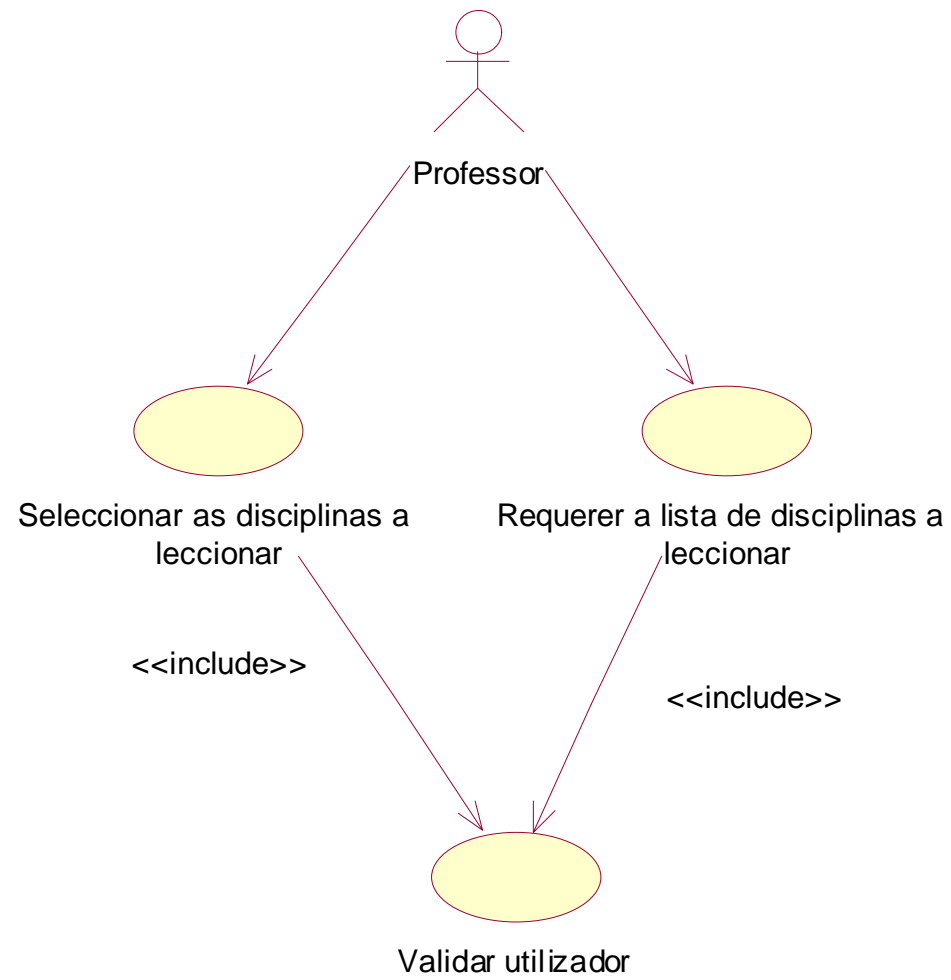
Vista dos Casos de Uso

Diagramas de casos de uso

Diagrama de casos de uso: Mostra um conjunto de actores, casos de uso, relações entre eles e opcionalmente colaborações

- ◆ Podem incluir relações de *herança* entre actores
- ◆ As **colaborações** definem as *realizações* dos casos de uso

Casos de uso do professor





Vista dos Casos de Uso

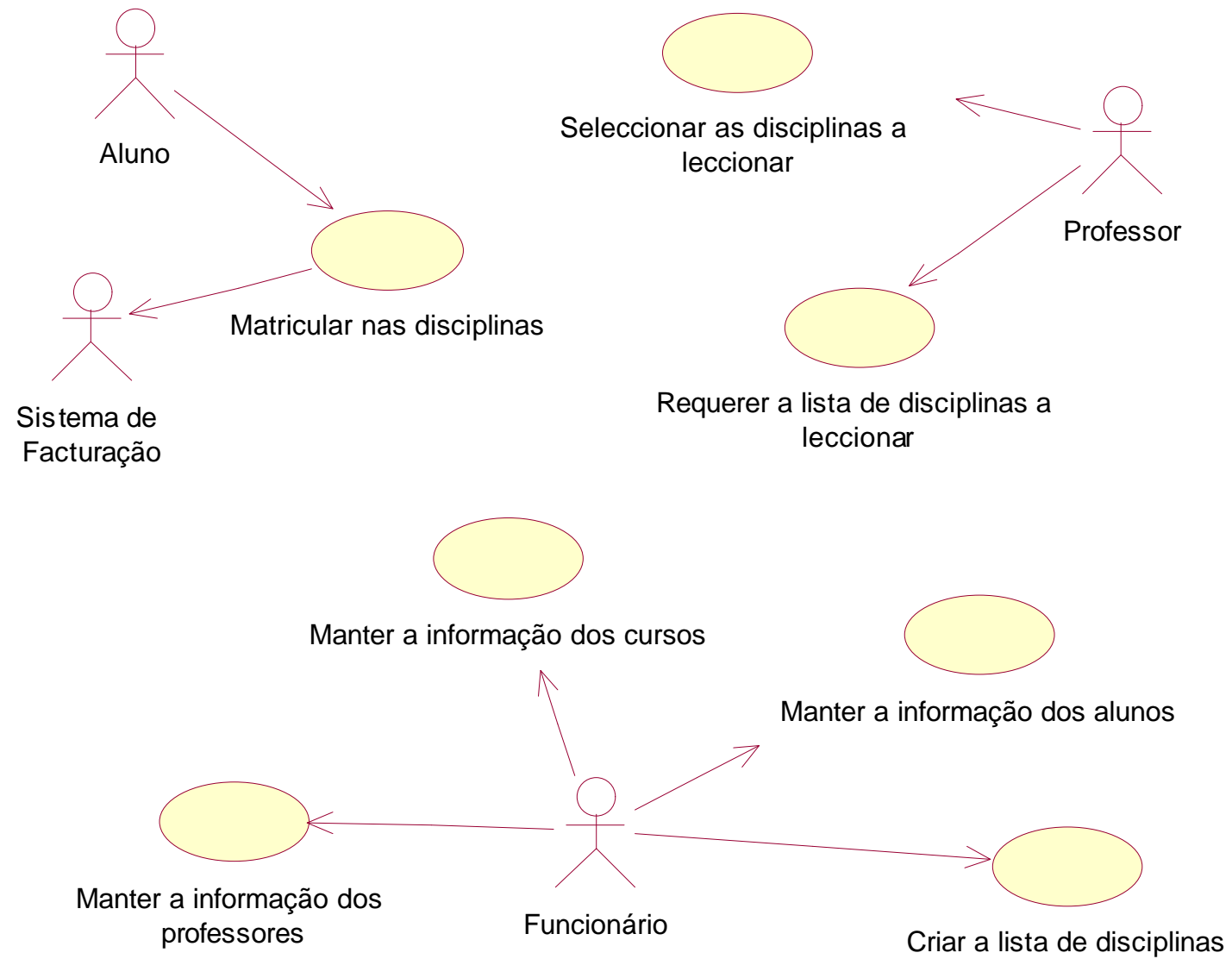
Diagramas de casos de uso

Pistas para utilização



- ◆ Em cada sistema existe normalmente um **diagrama principal** de casos de uso onde são mostradas as fronteiras do sistema e as funcionalidades principais
- ◆ Adicionalmente podem existir diagramas para:
 - Mostrar todos os casos de uso de um determinado actor
 - Mostrar todos os casos de uso a serem implementados numa interacção
 - Mostrar um caso de uso e todas as suas relações

Diagrama de casos de uso principal





Vista dos Casos de Uso **Diagramas de actividade**



Diagrama de actividades: Apresenta uma forma de fluxograma dando ênfase na sequência de actividades

Vista dos Casos de Uso

Diagramas de actividade

Composição

- ◆ Elementos UML incluídos nos diagramas de actividade:
 - **Actividades** – um conjunto de acções atómicas
 - **Transições automáticas** – entre as diferentes actividades
 - **Pontos de decisão** – permitem definir caminhos diferentes dependentes de uma condição (*guard condition*)
 - **Actividades inicial e final** – marcam o início e o fim das acções
 - **Barras de sincronismo** – permitem definir o local de início e fim de conjuntos de actividades que decorrem em paralelo
 - ***Swinlanes* (Pistas)** – permitem atribuir um responsável a um determinado conjunto de actividades
 - **Objectos** – por vezes são também incluídos

Vista dos Casos de Uso

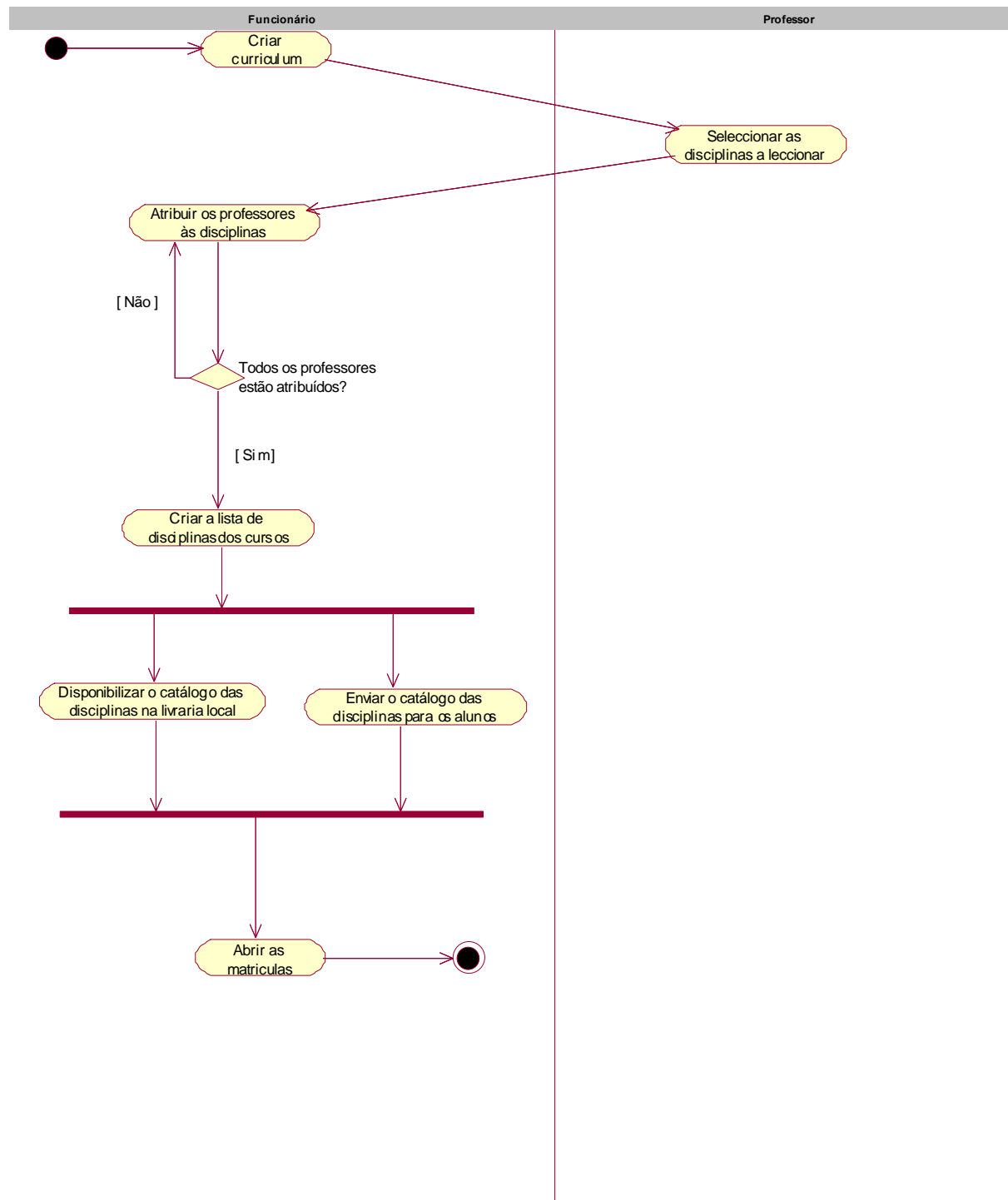
Diagramas de actividade

Pistas para utilização

- ◆ Descrição da dinâmica do sistema (os **processos de negócio**) mostrando a sequência de actividades, o seu paralelismo e a existência de caminhos alternativos

Nota: A sequência de actividades é representada do ponto de vista dos actores que colaboram com o sistema. Pode envolver vários casos de uso ou apenas um determinado caso de uso
- ◆ Descrição do fluxo de acções (o detalhe de um algoritmo) relativo a uma determinada operação

Diagrama de actividades da criação do curriculum



Arquitectura da Solução Vista Lógica

- ◆ Vai documentar o *design da solução* (ou seja a arquitectura do sistema), inclui a organização, a estrutura e as relações das classes
- ◆ Permite a comunicação com os peritos e entre os programadores e os analista de sistemas
- ◆ O *design* do sistema, irá ser representado principalmente através de:
 - Objectos,
 - Classes,
 - Relações entre Classes



Vista Lógica Objectos



Objecto: Representa uma entidade real ou conceptual. Possui três importantes características: estado, comportamento e identidade.

Vista Lógica Objectos Composição

- ◆ **Estado:** pode mudar com o tempo e é definido por um conjunto de propriedades (**atributos**)
Ex: disciplina: aberta (< 10 alunos) e fechada (=10 alunos))
- ◆ **Comportamento:** diz como um objecto responde a outro objecto. É implementado por um conjunto de **operações**
Ex: disciplina: Adicionar aluno e Apagar aluno
- ◆ **Identidade:** cada objecto é único mesmo que tenha o seu estado igual ao de outro objecto
Ex: disciplina: Álgebra 101, secção 1, Álgebra 101, secção 2

Vista Lógica Classes

Classe: É a descrição de um conjunto de objectos com propriedades comuns, comportamento comum, relações com outros objectos comuns e semântica comum.

- ♦ Uma classe funciona como um **modelo para a criação de objectos**
 - ♦ Cada objecto é uma instância de uma classe e cada objecto não pode pertencer a mais do que uma classe
- Ex: disciplina: local, hora (atributos) e
 obter local, adicionar aluno (Operações)

Vista Lógica

Identificação de classes

Pistas para identificação

- ♦ Uma classe deve capturar apenas uma abstracção, ou seja ter **um tema principal**.
Ex: (errado) classe que mantém a informação das disciplinas que um aluno fez e a informação do aluno
- ♦ Os nomes devem ser vir do vocabulário do problema
- ♦ A distinção entre objecto e classe pode não ser óbvia
- ♦ Documentação das classes – apresentar o propósito da classe e não a estrutura. Dificuldade em escolher o nome pode representar a presença de uma má abstracção
Ex: (errado) classe aluno – O nome, endereço e número de telefone do aluno

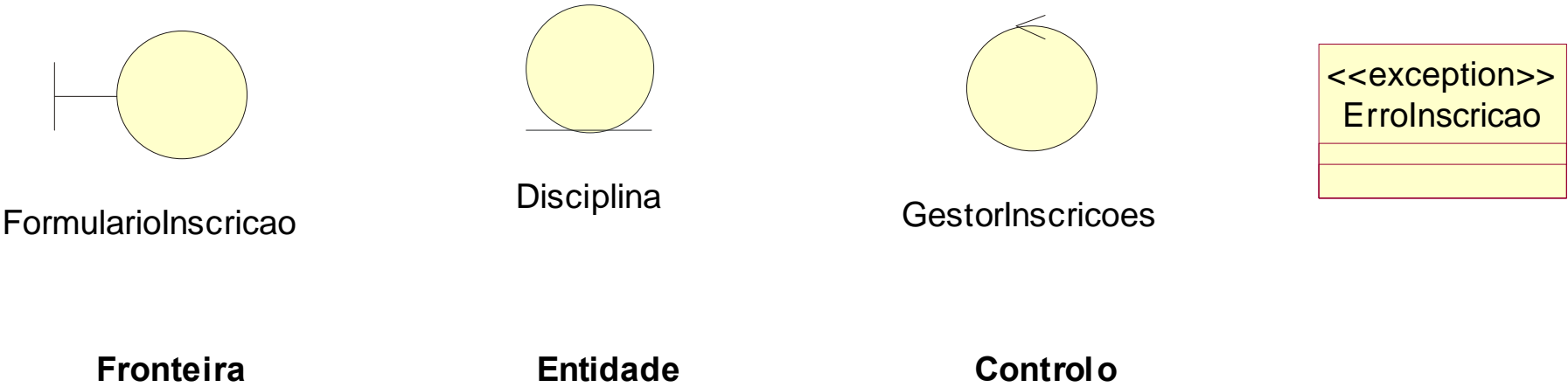
Vista Lógica

Identificação de classes

Método RUP

- ◆ Tipos de classes
(Seguem o princípio **Modelo-Visualização-Controllo**)
(representados por **estereótipos** com ícones próprios definidos dentro do *Rational Rose*):
 - **Entidade** (<<entity>>) – modela informação e comportamentos associados que são de longa duração
 - **Fronteira** (<<boundary>>) – tratam da comunicação entre o sistema e o exterior.
 - **Controlo** (<<control>>) – modelam o comportamento sequencial específico a um ou mais casos de uso.

Diagrama de classes de Inscrição nas disciplinas



Vista Lógica

Objectos e classes

Eastern State University

- ◆ Resultantes do cenário “Adicionar disciplina a leccionar” do caso de uso “Seleccionar disciplinas a leccionar”
 - **Classes fronteira**
 - OpcaoAdicionarDisciplina,
 - OpcoesDisciplinasProfessor
 - **Classes entidade**
 - Professor
 - Disciplina
 - Curso
 - **Classes controlo**
 - GestorDisciplinasProfessor



Vista Lógica

Pacotes

Agrupar classes



- ◆ Usados na vista lógica para agrupar um conjunto de classes e/ou outros pacotes
- ◆ Contêm habitualmente um conjunto de classes públicas que constituem (realizam) a interface do pacote, o resto são classes de implementação do pacote



Vista Lógica Diagramas de classes



Diagrama de classes: Mostra um conjunto de classes, interfaces, colaborações e as suas relações. Pode incluir pacotes e as suas relações.

Vista Lógica

Diagramas de classe

Pistas para utilização

- ◆ Usos comuns - pode existir um diagrama para:
 - Visualizar as classes implementadas num determinado pacote
 - Visualizar a estrutura e comportamento de uma ou mais classes
 - Visualizar uma hierarquia de herança
- ◆ Usados também para mostrar **colaborações***, esquemas de bases de dados e grupos de pacotes, incluindo relações
 - * sociedades de classes, interfaces e outros elementos que trabalham em conjunto de modo a fornecer um comportamento cooperativo

Vista Lógica

Cenários e Colaborações

Cenários

◆ **Cenário:**

- Capturam a funcionalidade dos casos de uso descrevendo possíveis sequências de acções que os concretizam. Representam uma possível sequência de acções dentro de um caso de uso
- Usam-se para ajudar a identificar objectos, classes e interacções entre objectos necessárias para executar determinada funcionalidade do sistema
- Documentam decisões de como as responsabilidades especificadas nos casos de uso se distribuem pelos objectos e classes do sistema
- Constituem um excelente meio de comunicação e discussão dos requisitos do sistema com os clientes



Vista Lógica

Cenários e Colaborações

Pistas para identificação



- ◆ Começar por encontrar os cenários principais referentes às acções e subacções dos casos de uso
- ◆ Quando se começarem a repetir com frequência os passos de outros cenários é indicação que se deve concluir
- ◆ Os cenários secundários, resultantes da análise dos casos “e-se” (“*what-if*”) vêm depois

Vista Lógica

Cenários e Colaborações

Realizações dos casos de uso

- ◆ Os **cenários** dizem como se **realiza** um caso de uso através de interacções dentro de sociedades de objectos - **colaborações**
(Nota: As realizações representam-se com diagramas de casos de uso)
Ex: *Criar um curso* dentro do caso de uso *Manter a informação dos cursos*
- ◆ Os cenários documentam-se com **diagramas de interacção**:
 - Diagramas de **sequência**
 - Diagramas de **colaboração**

Diagrama de casos de uso e das suas realizações

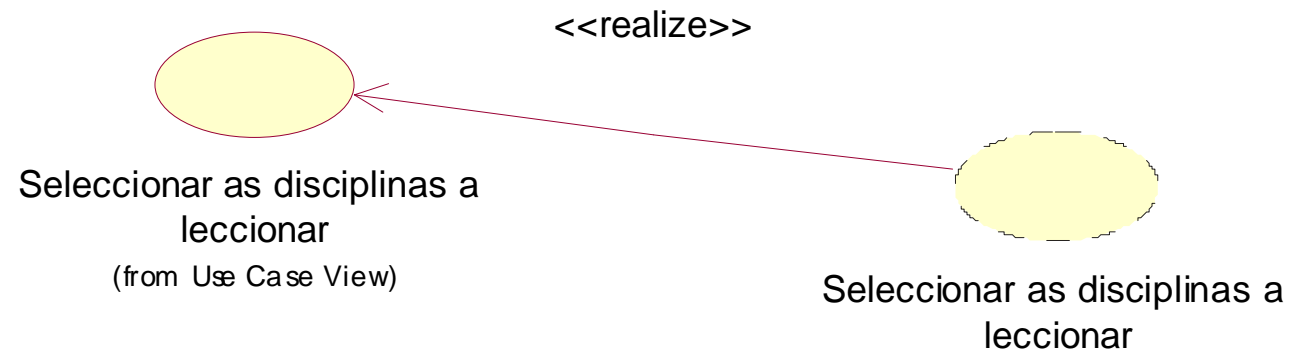
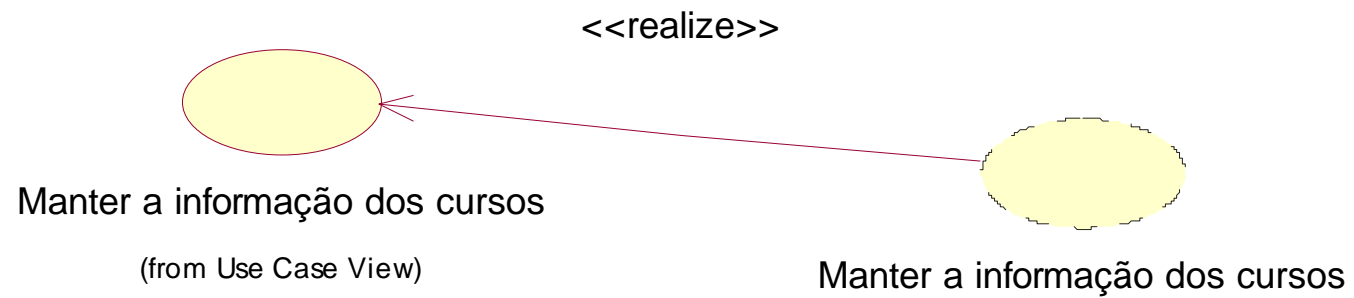
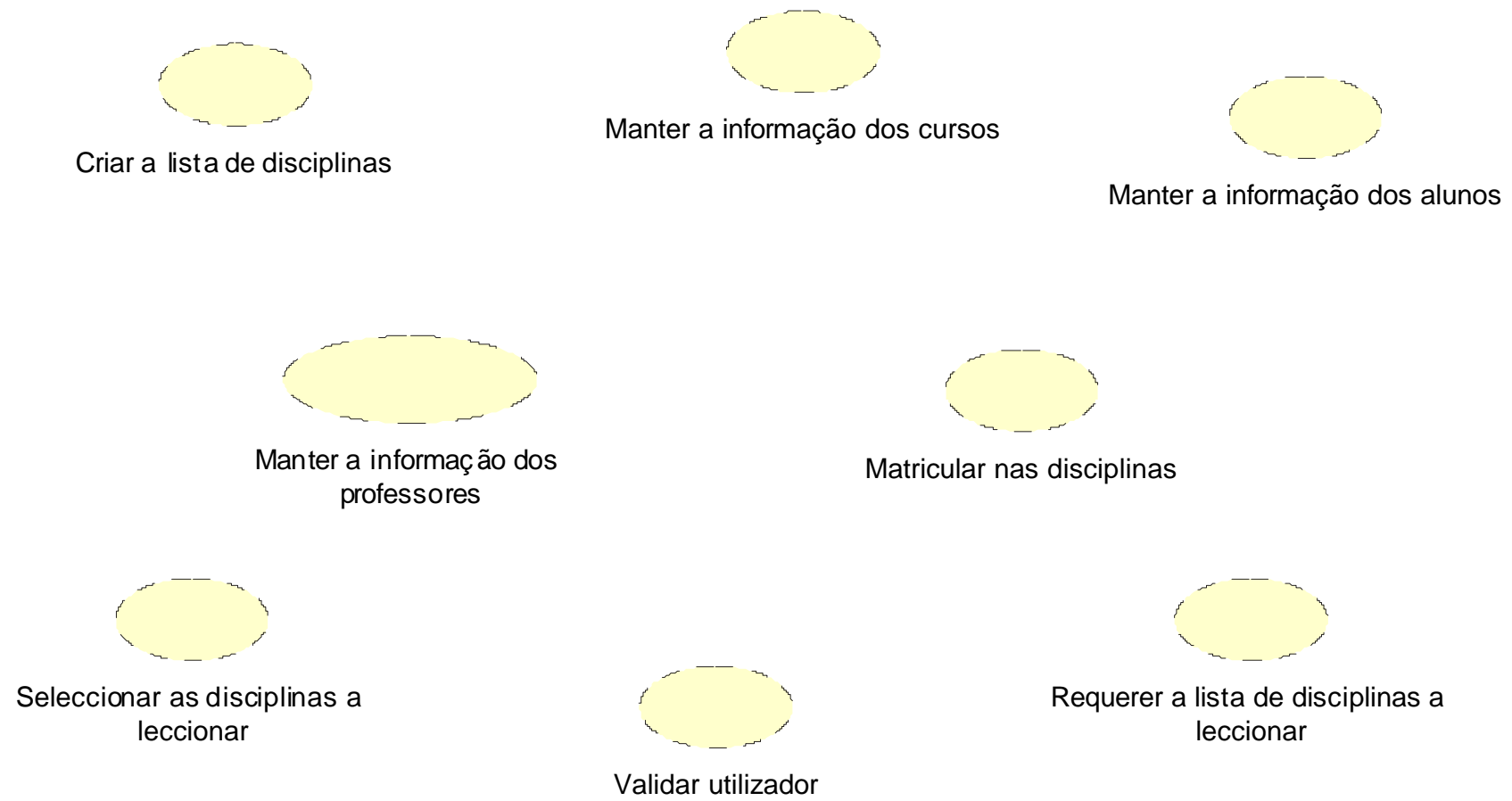


Diagrama de casos de uso das realições





Vista Lógica

Diagramas de sequência



Diagrama de Sequência: Mostra as interacções entre objectos evidenciando as suas sequências temporais. Contém normalmente objectos e mensagens.

Vista Lógica

Diagramas de sequência

Pistas para utilização

- ◆ Associados às **realizações** de casos de uso (**colaborações**)
- ◆ Para mostrar a sequência de acções de um cenário
- ◆ Devem-se acrescentar classes de fronteira para mostrar e documentar a interacção entre os actores e o sistema e não a implementação da interface
- ◆ Manter os diagramas de sequência **simples**
- ◆ Sugere-se incluir a lógica condicional apenas quando esta for simples, senão utilizar diagramas separados para o *if*, para o *then* e para o *else*

(Nota: em *Rational Rose* podem-se ligar os diagramas criados através de links colocados nas notas)

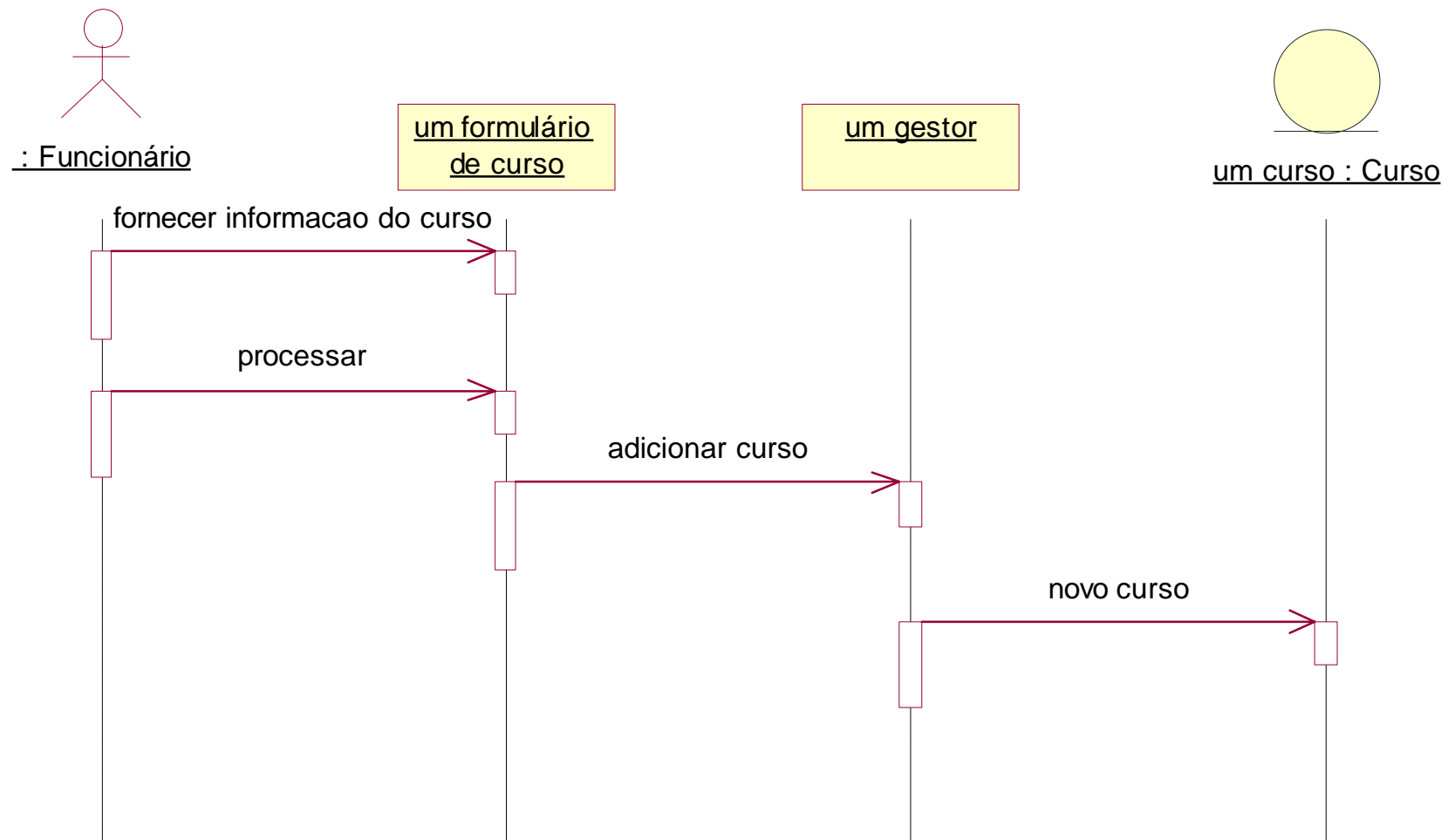
Vista Lógica

Diagramas de sequência

Eastern State University

- ◆ Resultante da análise do cenário “Criar um curso” do caso de uso “Manter a informação dos cursos”:
 - **Objectos:**
Um formulário curso, um gestor, um curso
 - **Actor:**
Funcionário
 - **Mensagens:**
Fornecer informação do curso, processar,
adicionar curso, novo curso

Diagrama de sequência para a criação de um curso



Vista Lógica

Diagramas de colaboração

Diagrama de colaboração: Mostra as interacções entre objectos evidenciando as ligações entre eles. Contém Objectos, ligações (*links*) e mensagens

- ◆ São equivalentes aos diagramas de sequência mas mostram os cenários de uma maneira diferente dando ênfase à arquitectura (objectos e ligações)

Diagrama de colaboração da criação de um curso

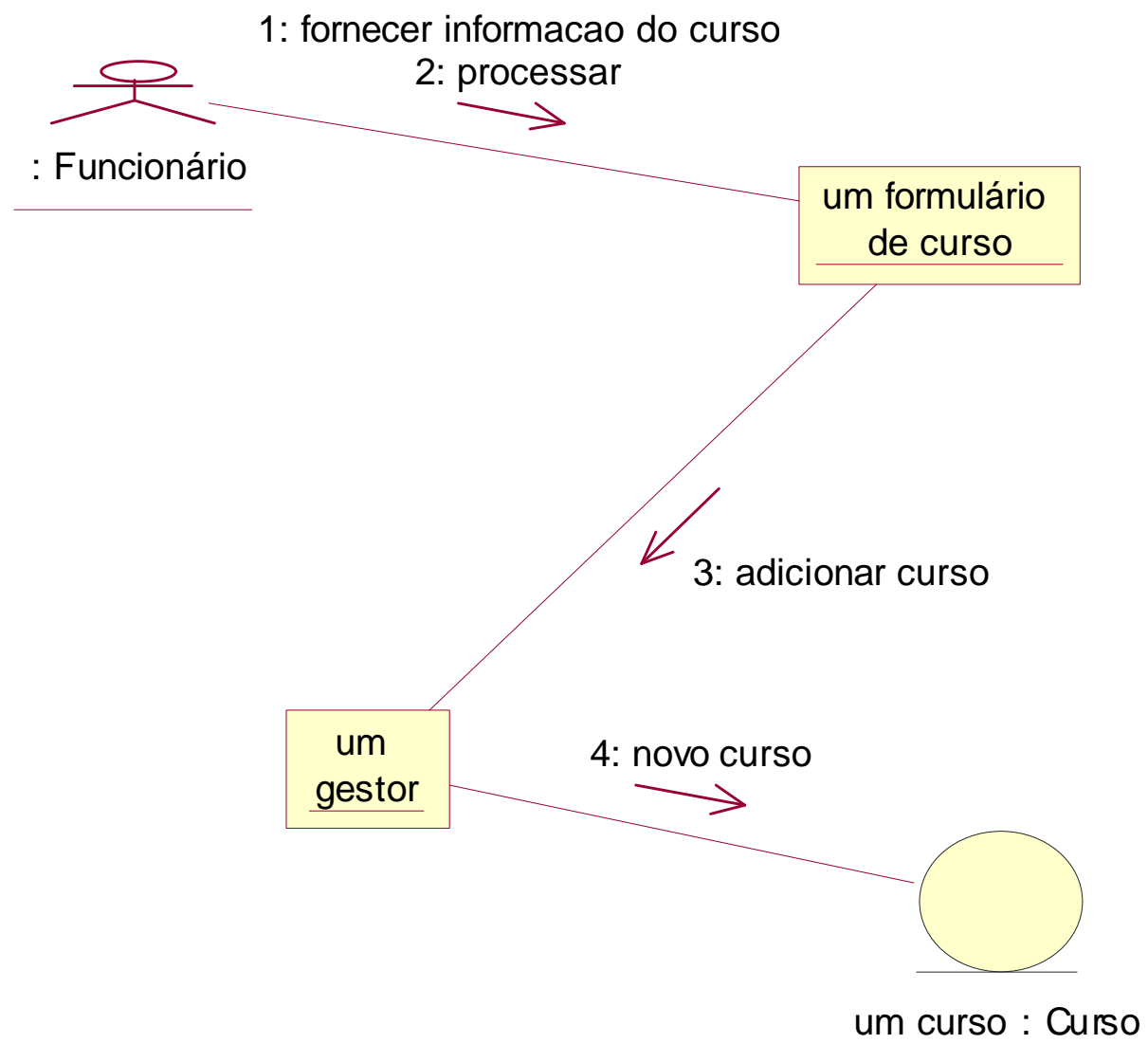


Diagrama de sequência de adicionar uma disciplina

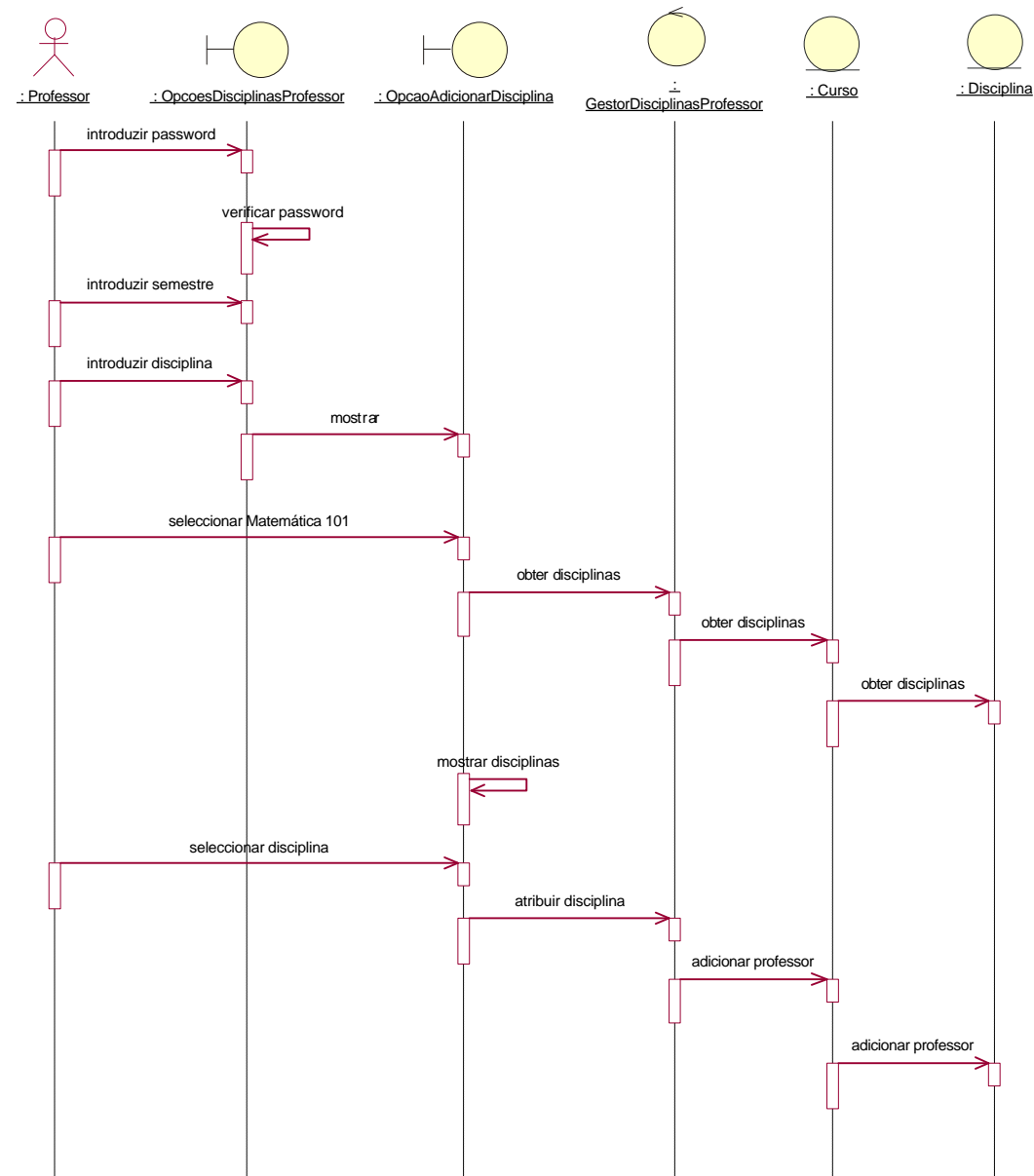
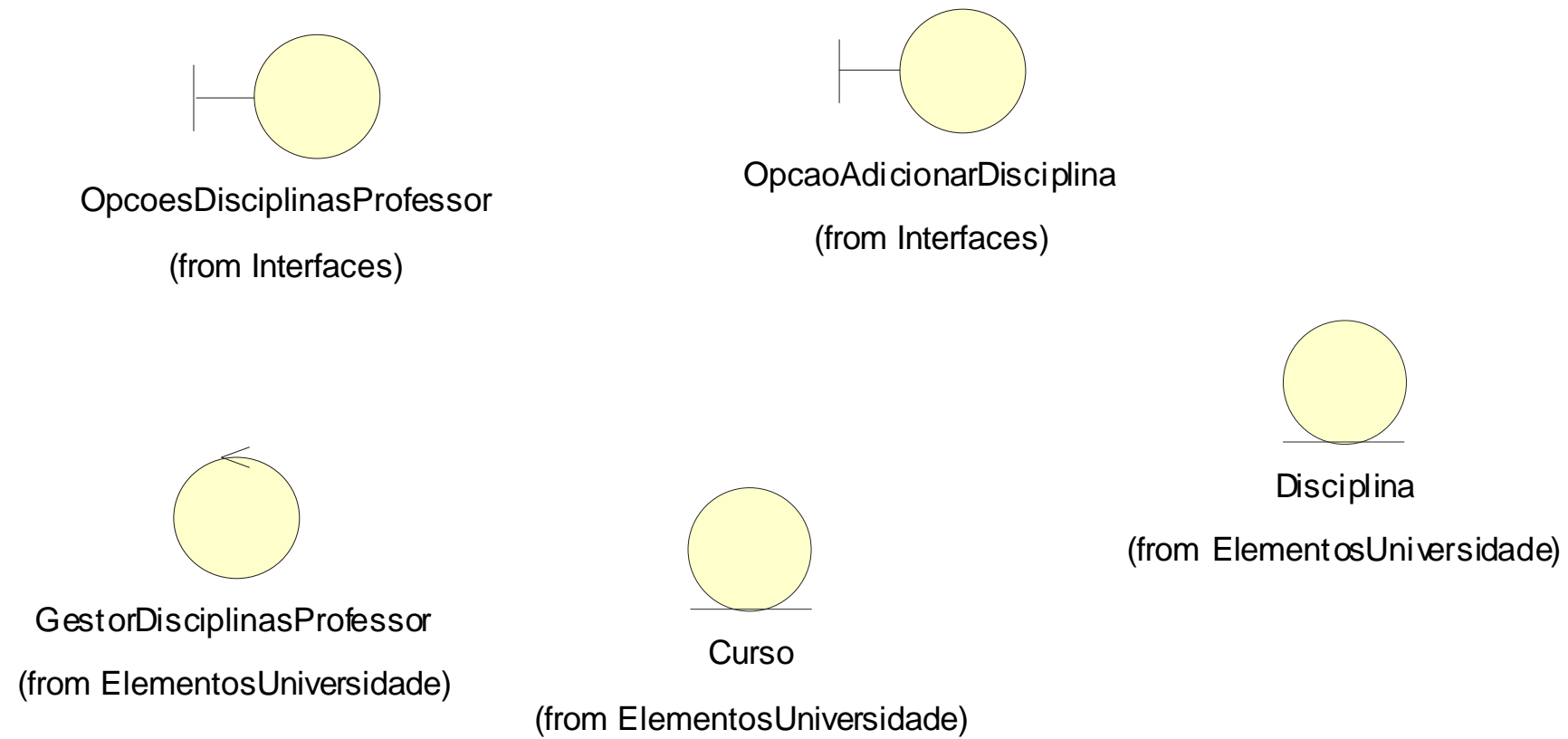


Diagrama de classes participantes de adicionar uma disciplina



Vista Lógica

Relações entre classes

Associação e agregação

- ◆ A análise de cenários revela relações entre classes
- ◆ O comportamento do sistema é obtido através de colaborações entre objectos. As *mensagens* trocadas entre objectos evidenciam relações entre as suas classes.
- ◆ Dois tipos de relações encontrados durante a análise:
 - **Associação** — é uma ligação bidireccional entre duas classes. Uma associação entre classes significa que existe uma ligação (*link*) entre os objectos dessas classes
 - **Agregação** — é uma forma especializada de associação onde um todo está relacionado com as partes. (Relação *part-off* ou *has-a*)

Vista Lógica

Associação e Agregação de Classes

Pistas para identificação

- ◆ Não é sempre clara a decisão entre a associação e agregação de classes. Colocar as seguintes **questões**:
 - É utilizada a frase “é parte de” na descrição da relação?
 - Algumas operações no todo são automaticamente aplicadas às partes?
 - Existe alguma assimetria intrínseca onde uma classe está subordinada a outra?
- ◆ Muitas vezes a decisão depende do contexto do problema.
Ex: Relação entre carro e pneus. Empresa pneus e concessionário.

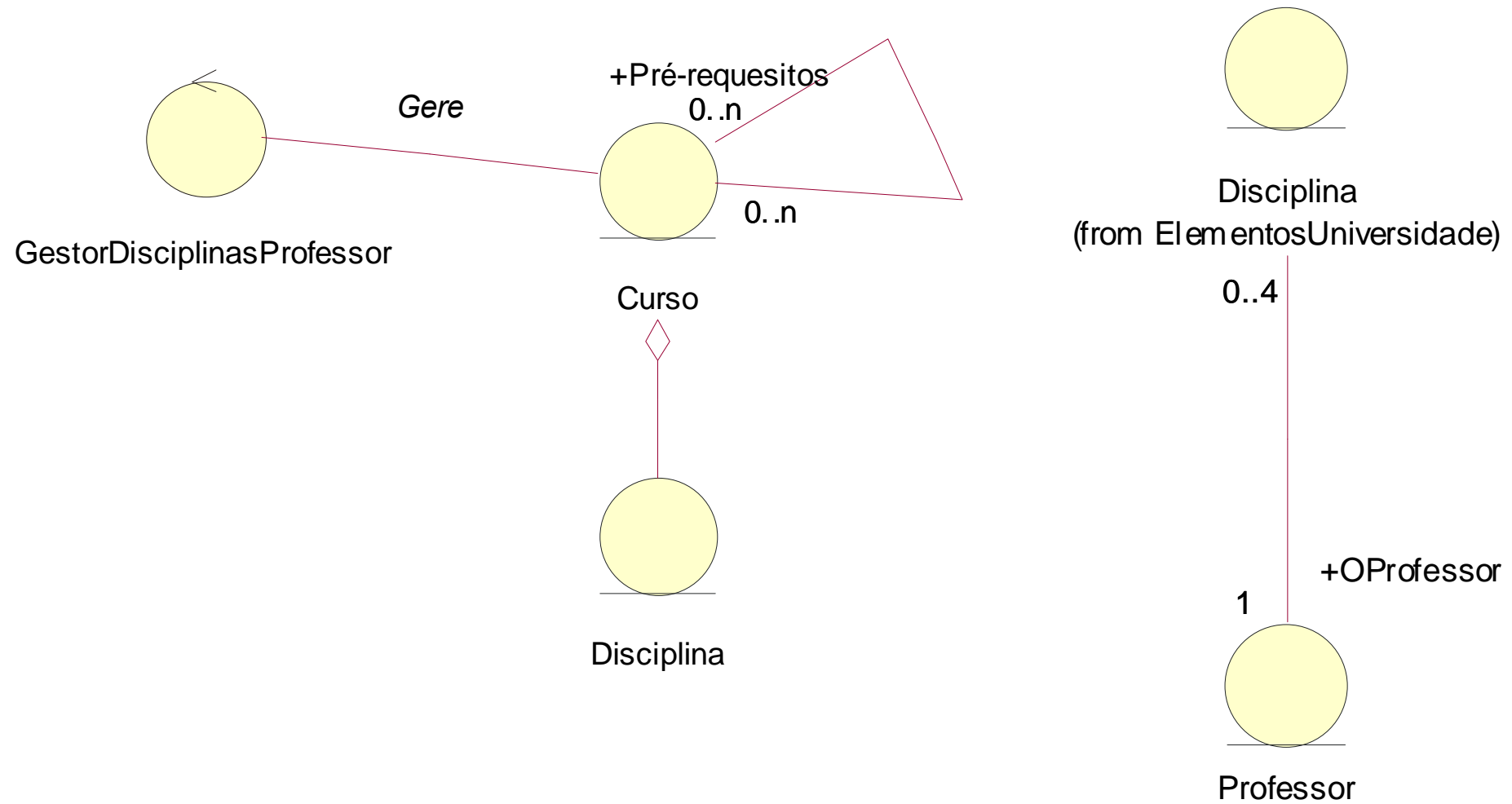
Vista Lógica

Associação e Agregação de Classes

Composição

- ◆ Podem-se ainda colocar os seguintes adornos nas associações ou agregações:
 - **Nomes** - comunicam o significado duma associação
Ex: Professor ensina curso
 - **Papeis** das classes envolvidas – usados habitualmente como alternativa a nomear uma associação
 - Indicadores de **Multiplicidade** – definem o número de objectos que participam na associação ou agregação
 - **Relações reflexivas** – quando múltiplos objectos de uma classe têm que comunicar entre si

Diagrama de classes com características das associações



Vista Lógica

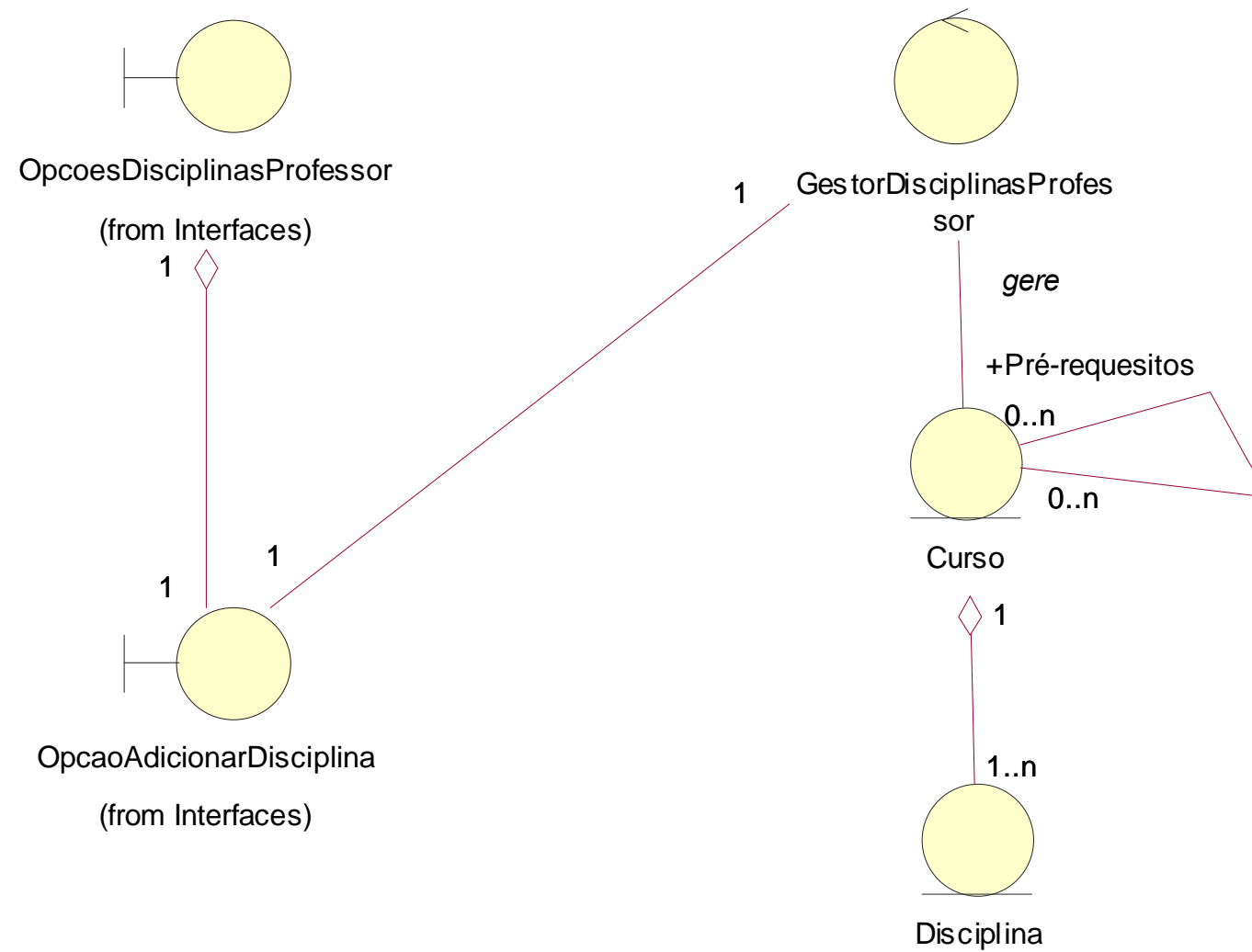
Associação e Agregação de Classes

Eastern State University

- ◆ Resultante do cenário “Adicionar uma disciplina” do caso de uso “Seleccionar as disciplinas a leccionar”:

Classe que envia a msg	Classe que recebe a msg	Tipo Relação
OpcoesDisciplinasProfessor	OpcaoAdicionarDisciplina	Agregação
OpcaoAdicionarDisciplina	GestorDisciplinasProfessor	Associação
GestorDisciplinasProfesoor	Curso	Associação
Curso	Disciplina	Agregação

Diagrama de classes do pacote ElementosUniversidade



Vista Lógica

Relações entre pacotes

- ◆ Existem também relações de **dependência** entre pacotes
Por exemplo o pacote A depende do Pacote B:
 - Implica que uma ou mais classes do pacote A iniciam uma comunicação com uma ou mais classes do pacote B
 - No caso anterior o pacote A é o pacote **Cliente** e o pacote B é o pacote **fornecedor**



Vista Lógica

Classes – operações e atributos

Comportamento e estrutura



- ◆ **Operações** – definem o comportamento da classe
 - As operações vão realizar as **responsabilidades** atribuídas à classe

Ex: Classe Disciplina – responsabilidade de adicionar e remover aluno
- ◆ **Atributos** – definem a estrutura da classe
 - Os atributos definem os **dados** que irão ser guardados por cada objecto da classe

Ex: Classe Disciplina – local e hora de cada disciplina

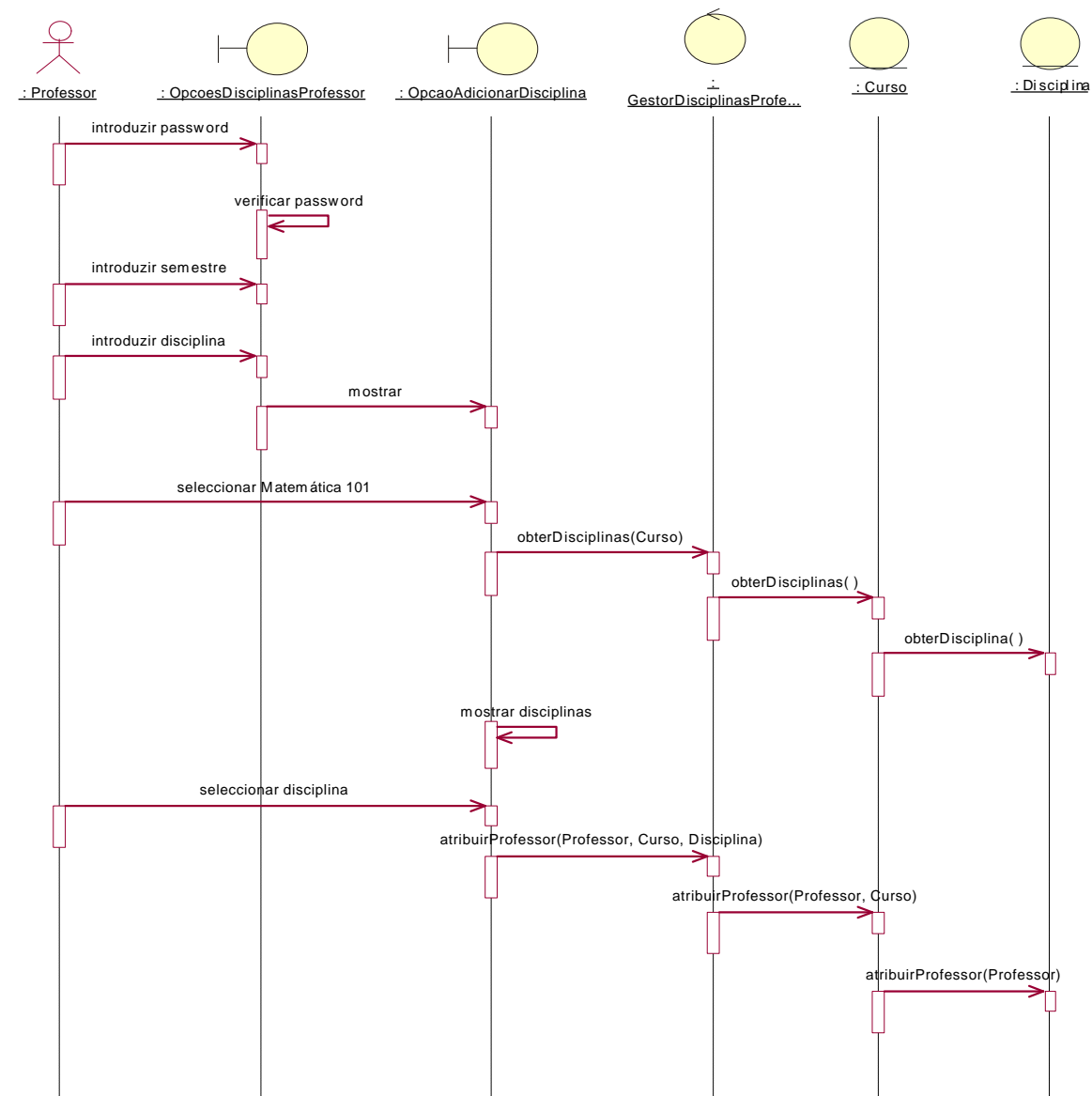
Vista Lógica

Classes – operações

Pistas para identificação

- As **mensagens** dos diagramas de sequência dão origem a operações nas classes dos objectos destinatários
- As mensagens não dão origem a operações quando são enviadas para uma classe de fronteira, ou quando têm origem ou destino num actor humano
- Os nomes devem ter em conta a classe que recebe a mensagem e não devem traduzir a implementação.
Ex: calcularNumeroAlunos e obterNumeroAlunos
- As operações devem ser **documentadas** em termos da funcionalidade disponibilizada. Devem indicar ainda os dados de entrada e saída.

Diagrama de sequência de adicionar uma disciplina com operações



Vista Lógica

Relação de Dependência entre Classes

Pistas para identificação

- ◆ **dependência** – relação uni-direccional entre duas classes. Classe depende de outra ou usa outra (Relação *using*)
 - Na dependência a classe da qual se depende não conhece a classe dependente (relação uni-direccional)
 - Pode ser identificada através da *assinatura* de uma operação quando é passado como argumento ou quando é retornado um objecto de outra classe
 - Pode ter origem ainda na criação e utilização de um objecto local de outra classe dentro de uma operação da classe dependente
 - Começa por ser modelada como associação

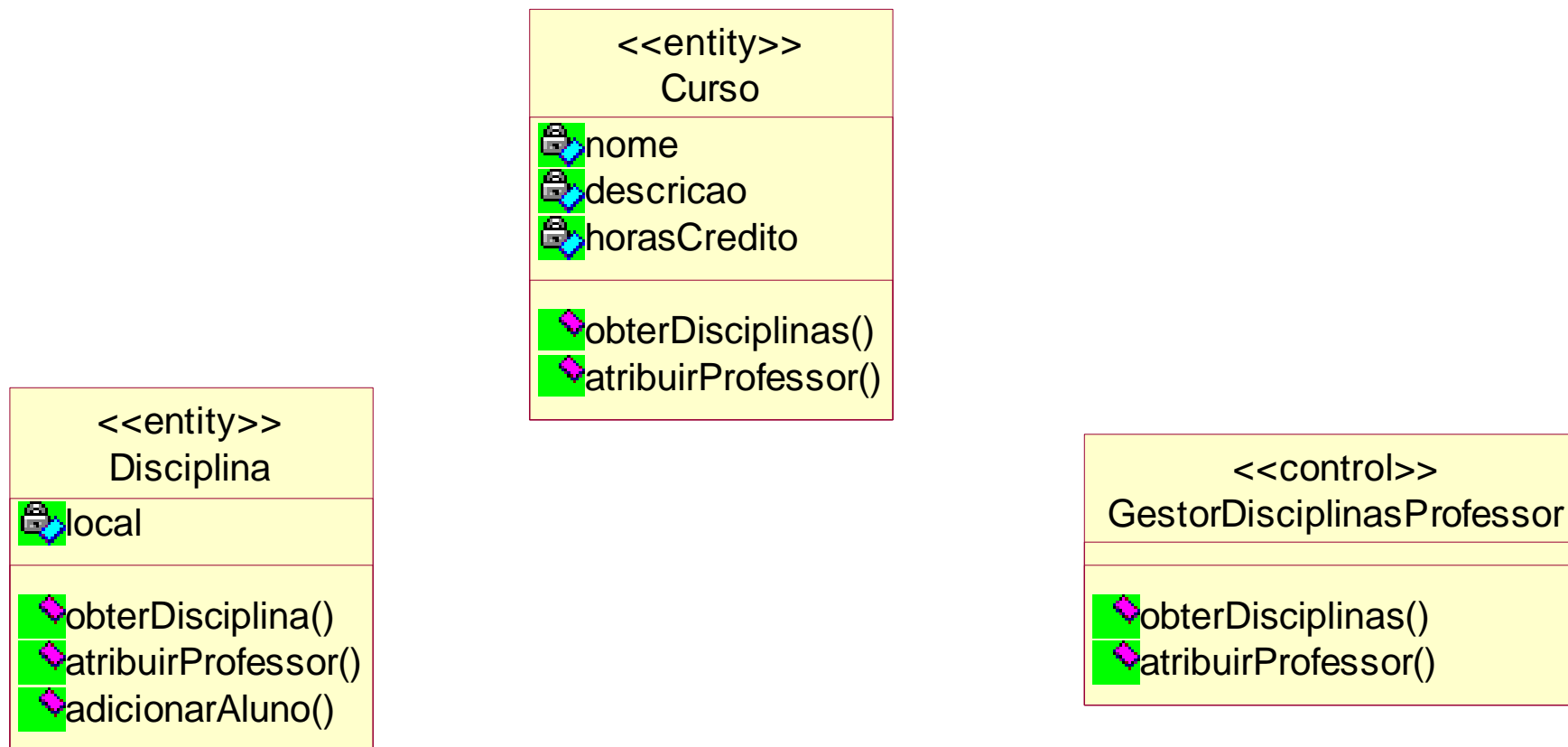
Vista Lógica

Classes – atributos

Pistas para identificação

- ◆ Podem ser obtidos a partir da descrição do problema, do conjuntos de requisitos ou da descrição das sequências de acções documentadas
- ◆ Devem ser **documentados** com definições claras e precisas.
 - Ex: (errado) – Nome do Curso: uma *string* de tamanho 15,
(correcto) – O título do curso tal como aparece nas publicações da universidade
- ◆ Podem ser criados **Diagramas de classe** apenas para mostrarem todos ou alguns atributos e operações de uma ou mais classes

Diagrama de classes do pacote ElementosUniversidade



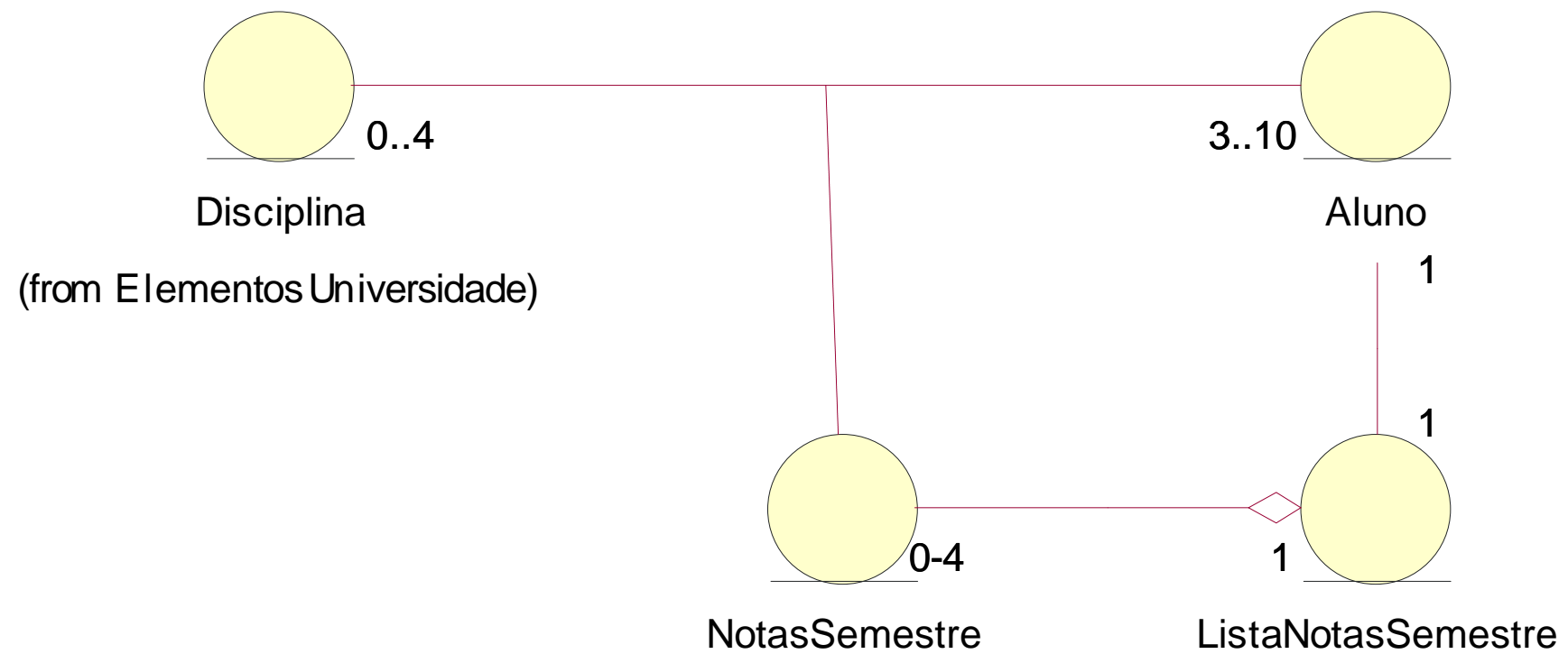
Vista Lógica

Classes de associação

Relações entre classes

- ◆ **Classes de associação** – obtidas a partir da associação entre duas classes quando esta relação tem estrutura e comportamento.
 - Ex: Aluno e disciplina. Onde colocar a Nota obtida?
 - Podem ter também associações com outras classes

Diagrama de classes das notas dos alunos





Vista Lógica

Herança entre classes

Relações entre classes



- ◆ **Herança** – define uma relação onde uma classe partilha a estrutura e comportamento de uma ou mais classes. A classe derivada vai herdar todos os atributos, operações e relações definidos nas classes que estão acima na hierarquia. (Relação *kind-of* ou *is-a*)
 - A relação de herança não necessita de ter nome, papéis ou mesmo multiplicidade
 - A herança é a **chave para a reutilização**

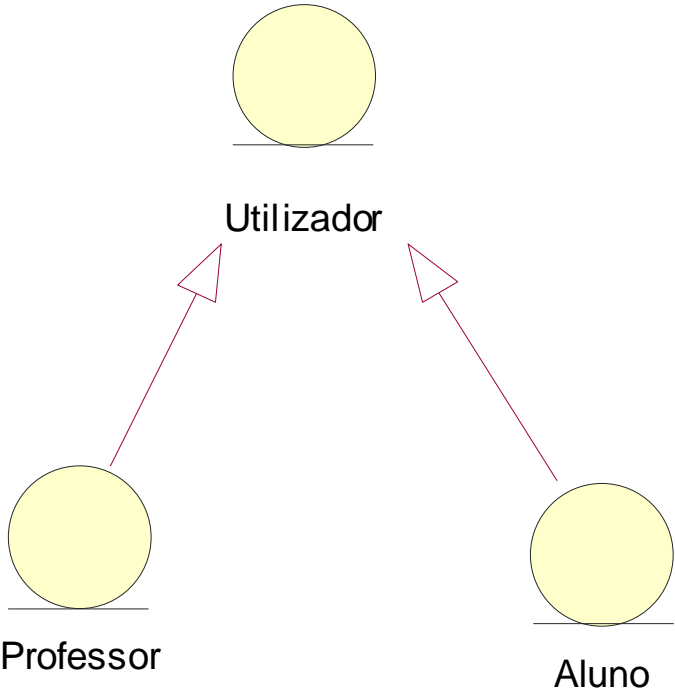
Vista Lógica

Herança entre classes

Pistas para identificação

- ◆ Por **Generalização** – criar uma classe base a partir de várias classes com comportamento e estrutura comuns
Ex: Aluno e professor: nome, endereço e telefone, professorID e alunoID
Utilizador com utilizadorID (se possível), incluir os outros atributos.
- ◆ Por **Especialização** – adicionar comportamento e estrutura a uma classe. Pode-se redefinir operações sem nunca restringir uma operação previamente definida
- ◆ Cuidado na utilização de discriminadores na criação de subclasses
Ex: Curso, CursoInterno, CursoExterno – discriminador é o local do curso.
E onde colocar por exemplo CursoObrigatorio?

Diagrama de classe das hierarquias do pacote InformaçãoPessoas





Vista Lógica

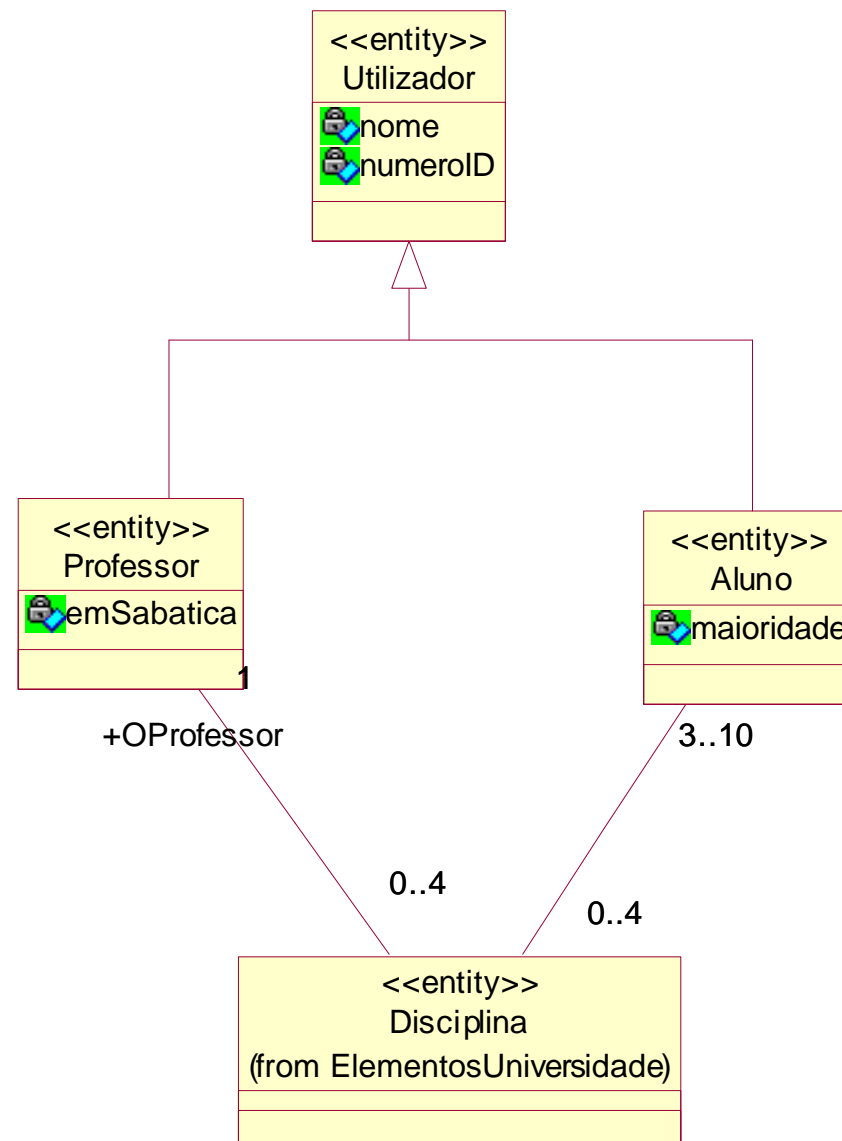
Herança múltipla

Pistas para utilização



- ◆ A herança múltipla origina diversos problemas devendo ser usada com muito cuidado
 - Ex: Colisões de identificadores, cópias múltiplas de características herdadas e código difícil de manter
- ◆ A agregação pode ser a alternativa correcta
 - Ex: classes Aluno, ALunoFullTime AlunoPartTime

Diagrama de classes da hierarquia do utilizador



Vista Lógica

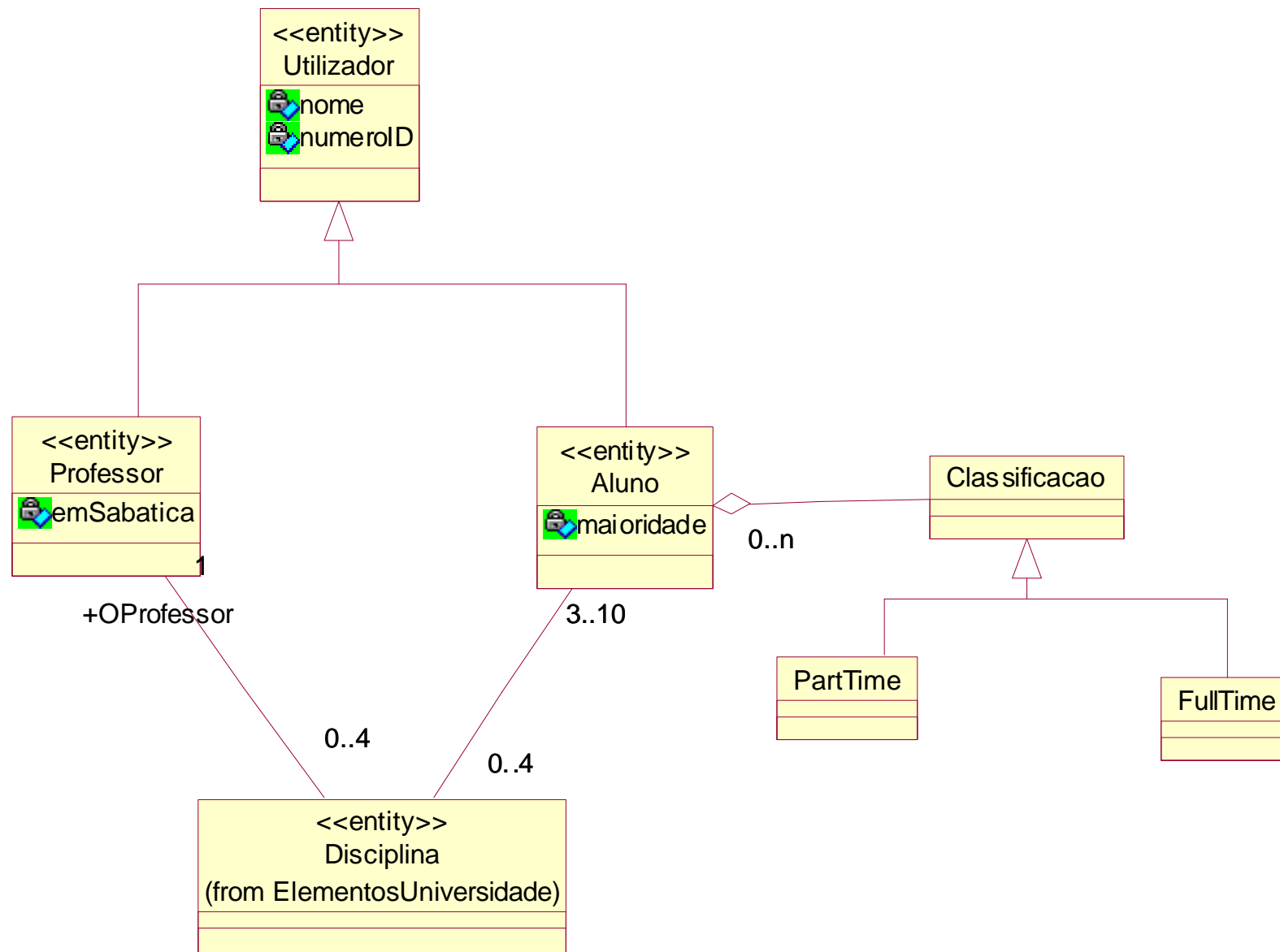
Herança e agregação entre classes

Pistas para utilização

- ◆ Não se deve abusar da utilização da herança apenas porque se diz que “a herança é muito boa”. Muitas vezes a agregação é a alternativa correcta

Ex: classes Aluno, AlunoFullTime, AlunoPartTime – e se um aluno de Full Time decide mudar para Part Time, o objecto muda de classe?, e se é adicionada uma nova dimensão AlunoComBolsa e AlunoSemBolsa?

Diagrama de classes da hierarquia do utilizador com classificação





Vista Lógica

Diagrama de estados



Diagrama de estados: Define máquinas de estados. Mostra os estados de um objecto, acontecimentos e mensagens que originam mudanças de estado e as acções resultantes

Vista Lógica

Diagrama de estados

Composição

- ◆ Elementos UML representados nos diagramas de estado:
 - **Estados** – quando um objecto satisfaz uma condição, executa uma acção ou aguarda um acontecimento. Pode incluir **acções** de entrada e saída e/ou **actividades** executadas durante o estado.
Ex: professor em sabática e a dar aulas.
 - **Transições entre estados** – mudança para o estado seguinte. Pode incluir uma **acção** ou uma **condição** associada e pode ainda originar um **acontecimento**. Pode ser automática (a actividade do estado anterior acabou) ou ser causada por um acontecimento
 - **Estados inicial e final** – Existe sempre um único estado inicial e podem haver um, nenhum ou vários estados finais

Vista Lógica

Diagrama de estados

Pistas para utilização

- ◆ Para mostrar o comportamento dinâmico duma classe que justifique, para investigar o comportamento de uma classe *todo* de uma agregação ou de uma classe de controlo
- ◆ Contem todas as mensagens enviadas e recebidas pelo objecto
- ◆ Os cenários representam um caminho através de um diagrama de estados
- ◆ Examinar os diagramas de sequência para a identificação de estados: o intervalo entre duas mensagens enviadas pelo objecto normalmente está associado a um estado

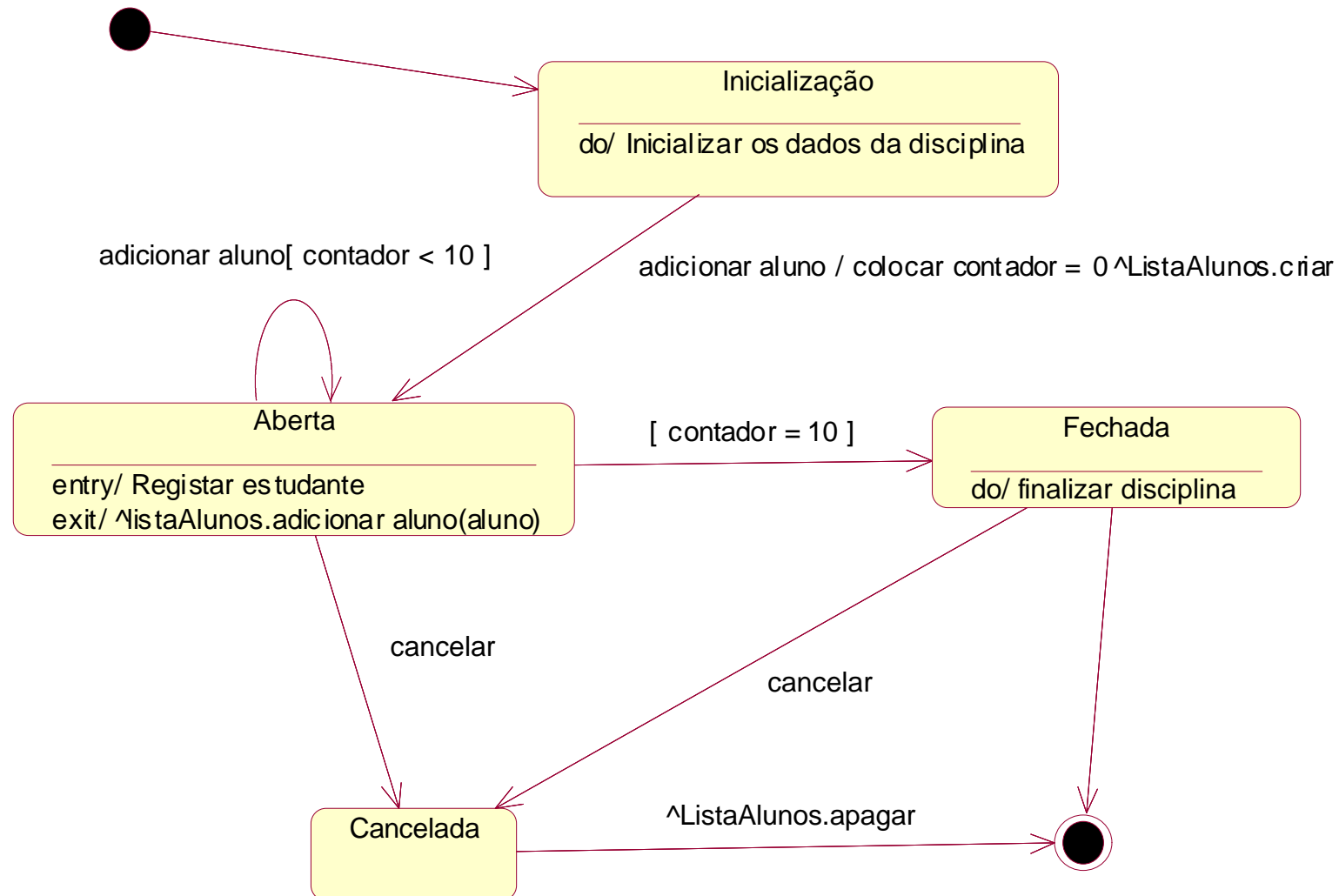


Vista Lógica **Diagrama de Estados** Eastern State University



- ◆ Resultantes da análise da classe “Disciplina” :
 - **Estados:**
 - Aberta
 - Fechada
 - Cancelada
 - Inicialização

Diagrama de estados da disciplina



Vista lógica

Homogeneização do design

- ◆ **Combinar classes** – Classes semelhantes que fazem a mesma coisa, classes de controlo com comportamentos semelhantes e que acedem à mesma informação
- ◆ **Dividir classes** – Quando não possuem um único tema principal, quando um atributo exhibe estrutura e comportamento associados
- ◆ **Eliminar classes** – Classes que não possuem qualquer estrutura e comportamento, classes que não participam em casos de uso
- ◆ **Verificar consistência**
 - Percorrer os cenários
 - Seguir os acontecimentos
 - Rever a documentação

Arquitetura da Solução

Vista da implementação


- ◆ Vai documentar a partir de diagramas de componentes a **organização modular do *software***
- ◆ Tem em conta requisitos derivados como sejam facilidade de desenvolvimento, gestão do *software*, reutilização e restrições das linguagens de programação utilizadas
- ◆ Os pacotes nesta vista representam partições físicas do sistema
- ◆ O *Rational Rose* inclui esta vista dentro da vista dos componentes definida no *browser* do programa

Vista da implementação


Diagrama de componentes

Diagrama de componentes: Mostra um conjunto de componentes e as suas relações. Pode incluir interfaces e pacotes

- ◆ Pode incluir componentes de **código fonte**, componentes **executáveis** e dependências entre componentes
- ◆ Pode incluir **pacotes** e dependências entre eles



Vista da implementação **Diagrama de componentes** Pistas para utilização



- ◆ Inclui habitualmente as classes do modelo lógico (quando estas estão implementadas em componentes)
- ◆ Outros usos:
 - Para modelar os ficheiros de código fonte. Inclui os componentes de trabalho
 - Para modelar a versão final do programa. Inclui executáveis, bibliotecas, tabelas, ficheiros e documentos
 - Para modelar bases de dados

Diagrama de componentes principal

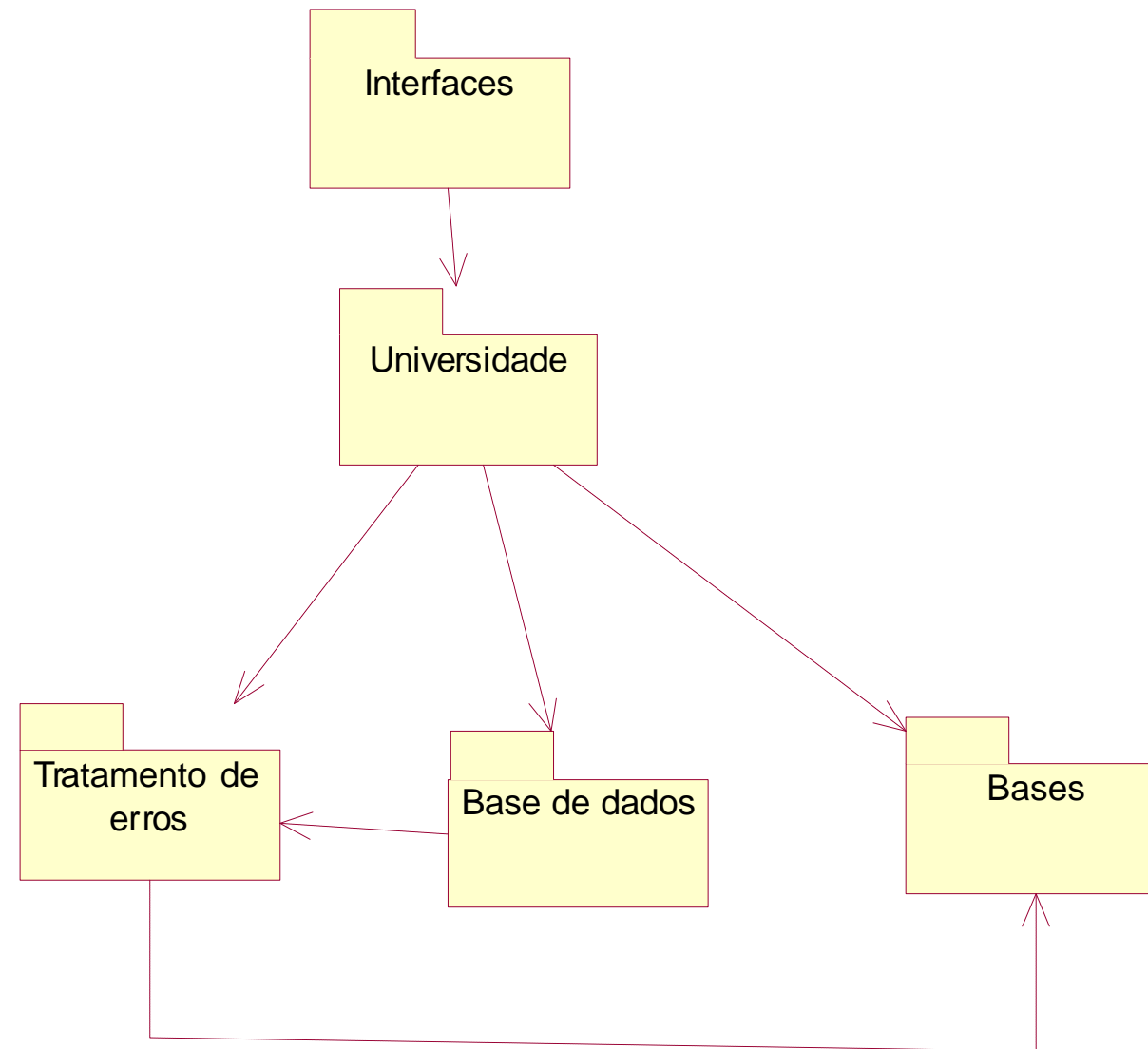
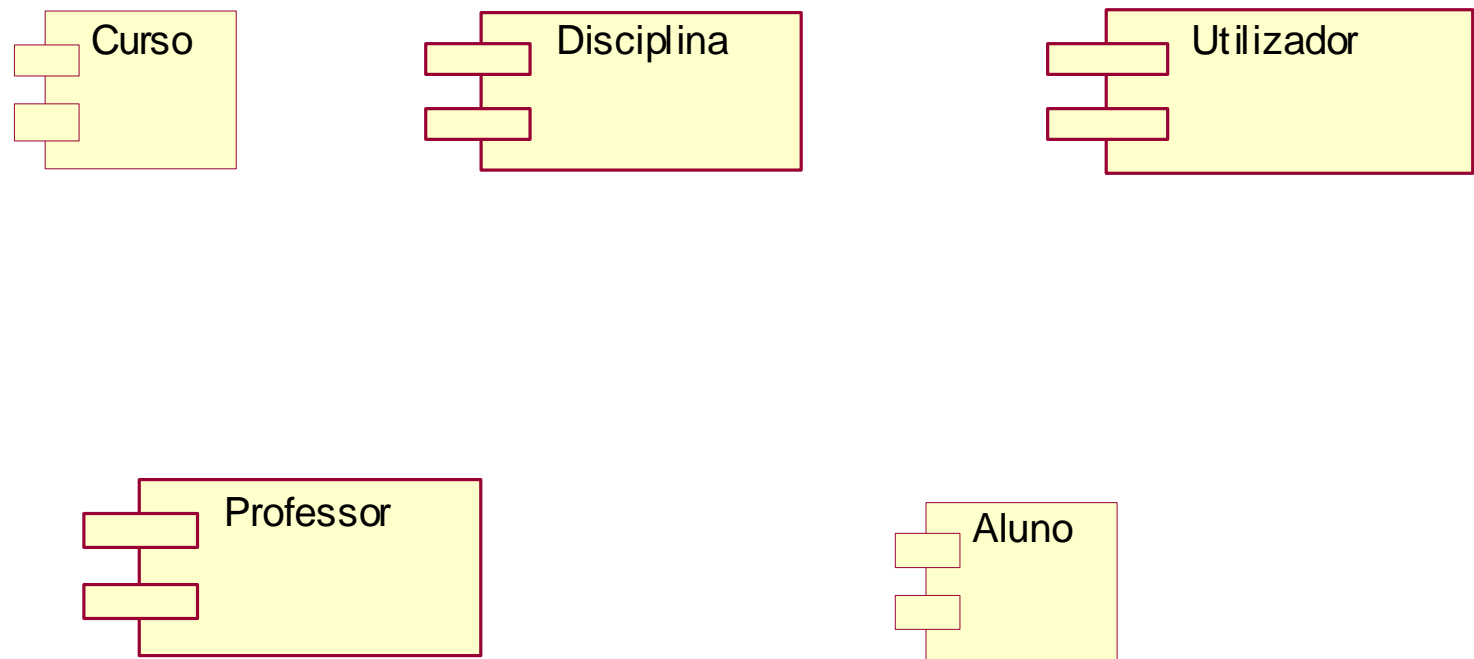


Diagrama de componentes da Universidade



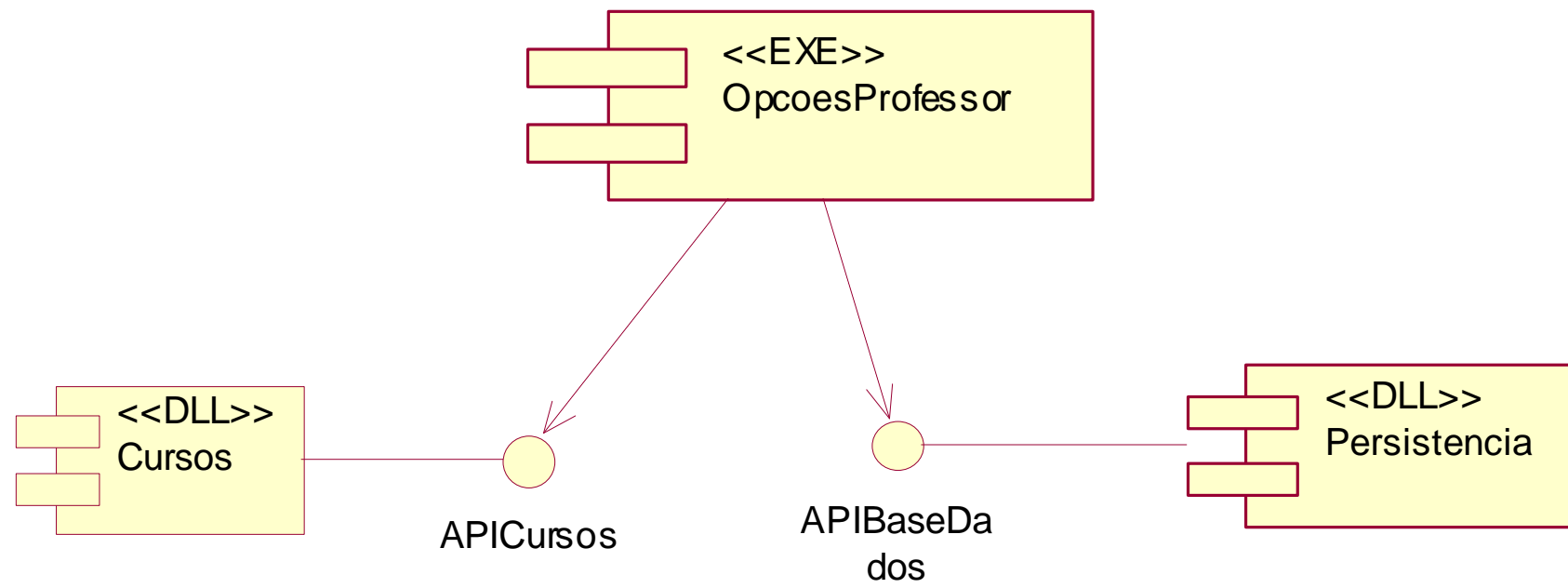


Arquitectura da solução **Vista dos processos**



- ◆ Vai documentar a partir de diagramas de componentes a estrutura da implementação *em execução* do sistema
- ◆ Tem em conta requisitos como sejam performance, fiabilidade, escalabilidade, integridade, gestão do sistema e sincronização
- ◆ O *Rational Rose* inclui esta vista dentro da vista dos componentes definida no *browser* do programa.

Diagrama de componentes executáveis do professor





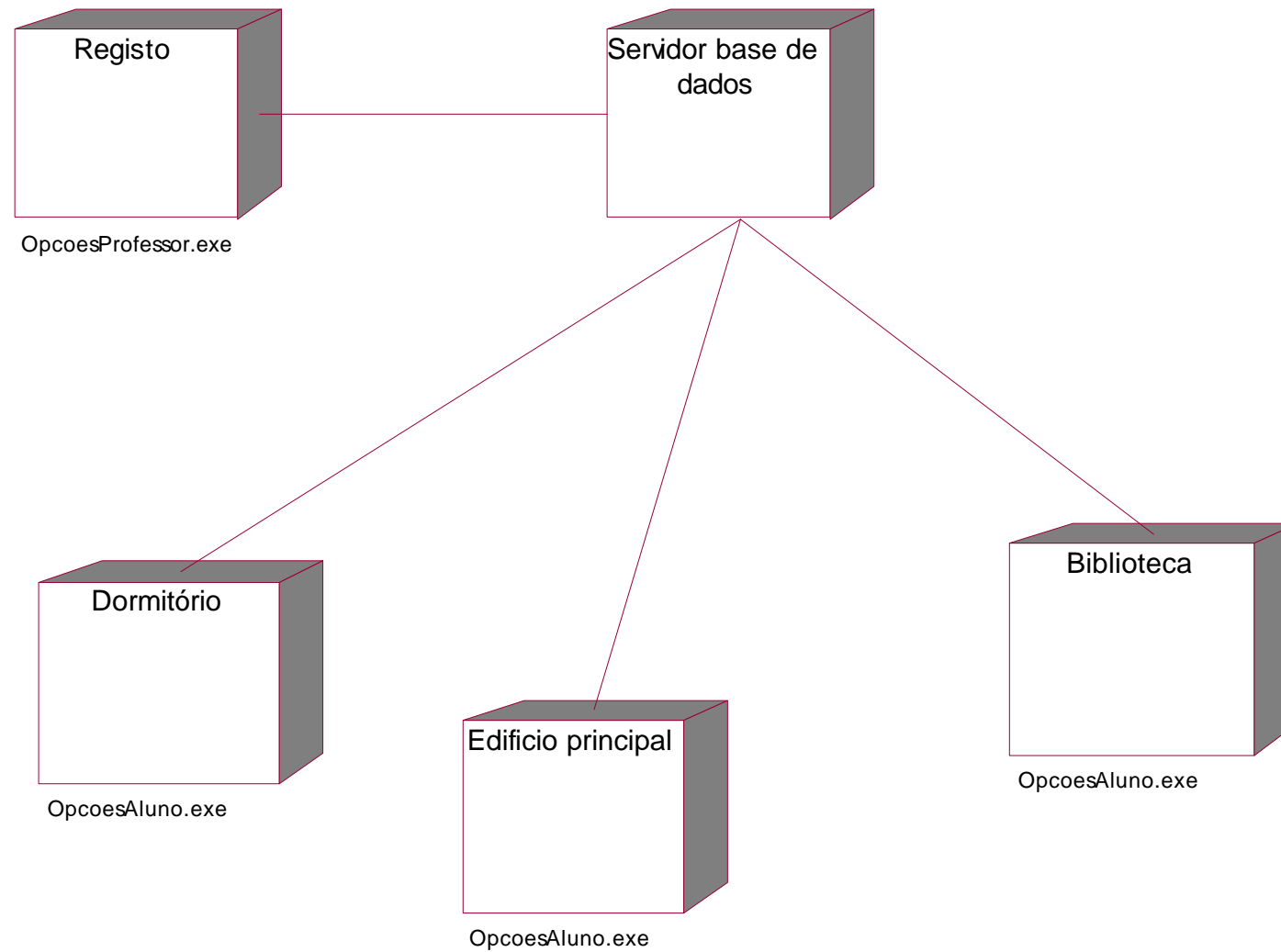
Arquitectura da solução

Vista da instalação



- ◆ Vai documentar a partir de diagramas de instalação a distribuição física dos elementos processadores e do *software* que incluem
- ◆ Define a **topologia do sistema**
- ◆ Tem em conta requisitos como sejam disponibilidade do sistema, fiabilidade, performance e escalabilidade

Diagrama de instalação



Vista da instalação

Diagrama de instalação

Diagrama de instalação: Apresenta uma visão dos elementos físicos que constituem o sistema. Mostra os nós, os componentes que existem dentro deles e as ligações com outros nós.



Vista lógica

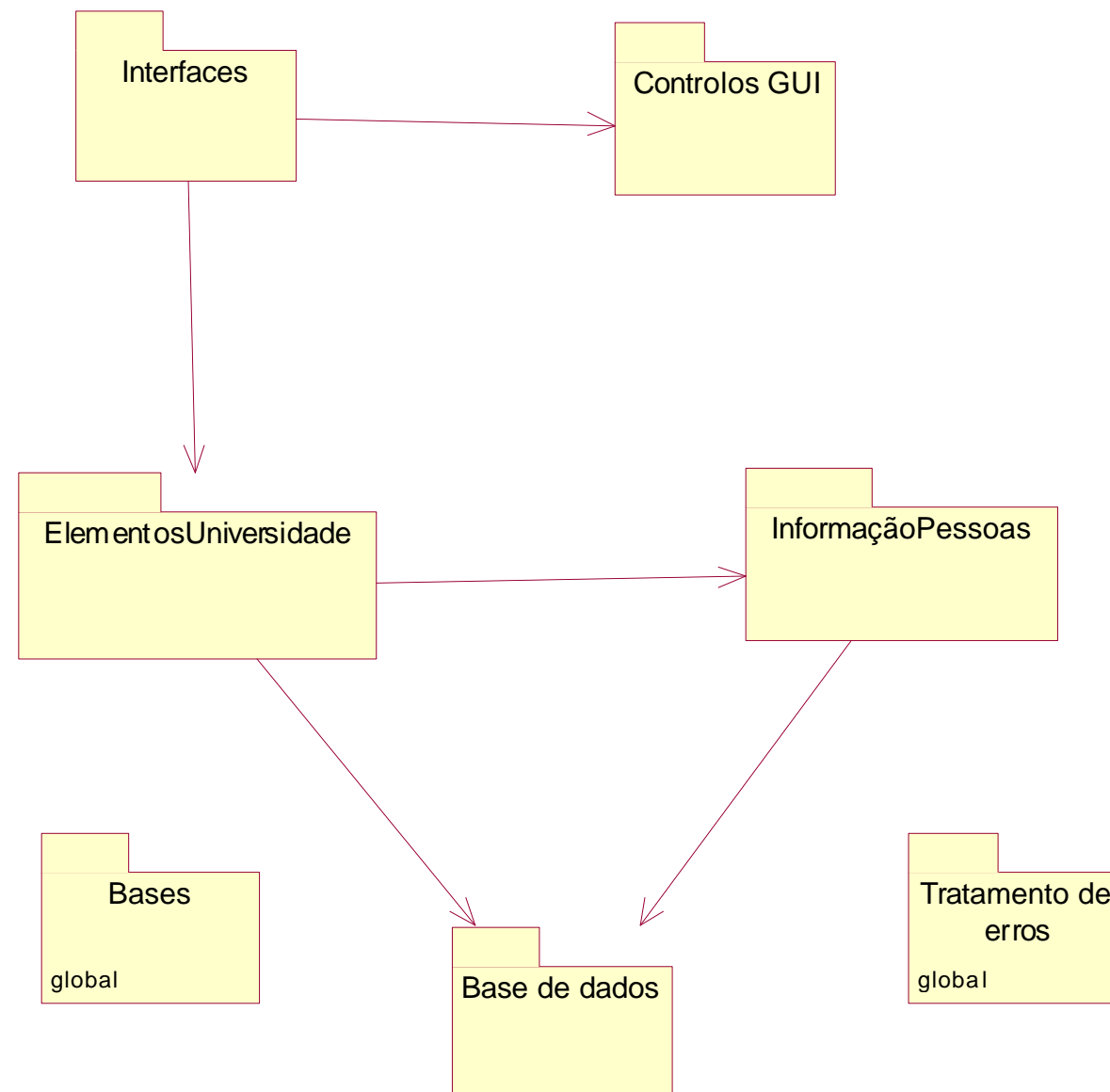
Mecanismos chave

Seleccção dos mecanismos chave



- ◆ É importante seleccionar os mecanismos chave que irão ser utilizados na arquitectura da solução que incluem:
 - Linguagem de implementação
 - Forma de armazenamento de dados
 - Interface do utilizador
 - Tratamento de erros
 - Mecanismos de comunicação
 - Distribuição e migração de objectos
 - Ligações em rede

Diagrama de classes principal



Vista lógica

Conclusão do Design

Propósito

- ◆ Passar da análise inicial da especificação do que o sistema **deve fazer** para **como fazer**.
- ◆ Completar e implementar o design
 - Definir e desenhar a interface do utilizador
 - Adicionar classes de design
 - Completar e refinar as relações definidas
 - Completar o design das operações e atributos
- ◆ Comunicar o design aos programadores

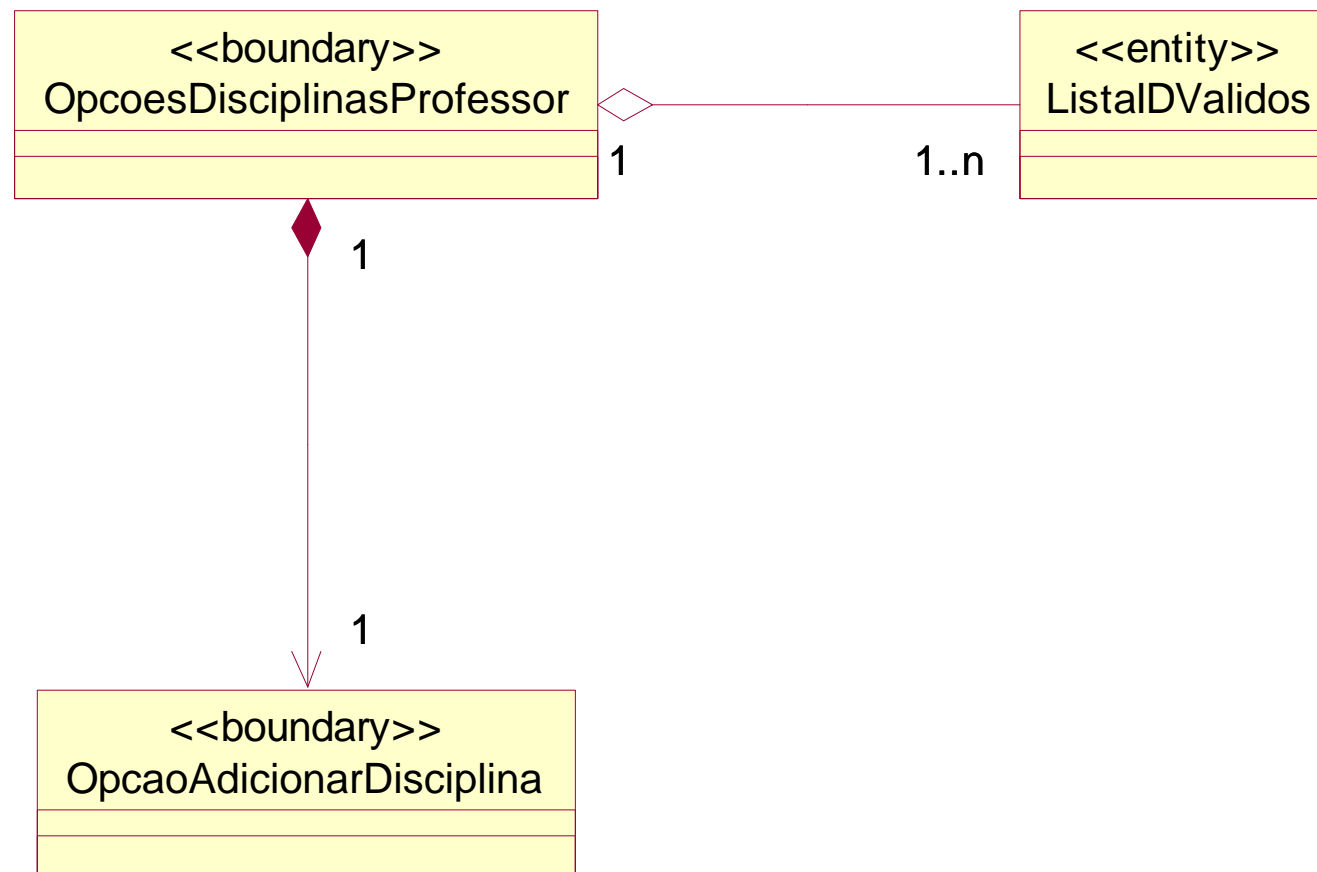
Vista lógica

Conclusão do Design

Interface com o utilizador e classes de design

- ◆ Definir e desenhar a interface do utilizador:
 - Examinar os cenários: é necessário fornecer a capacidade de enviar e receber mensagens dos actores
 - Implica a implementação das classes de fronteira definidas: criar janelas, definir o *layout* das janelas, tratar eventos do utilizador
Ex: Seleccionar disciplinas a leccionar – janela com password, janela de opções, botões, caixas edição de texto, etc.
- ◆ Adicionar classes de design
 - Para facilitar o **como fazer** (*how-to*) do sistema
 - Classes que aparecem por necessidade de implementação
Ex: ListaIDValidos - Classe que sabe como validar as *passwords*

Diagrama de classes principal do pacote das interfaces



Vista lógica

Conclusão do design

Refinação das relações entre classes

- ◆ Completar e refinar as relações definidas
 - **Navegação** – Rever as associações e agregações para determinar a uni-direccionalidade. Implica classes mais fáceis de manter
 - **Composição** – Quando a agregação implica a contenção exclusiva das partes. Implica contenção por valor (e não por referência)
Ex: Classes OpcoesDisciplinaProfessor e OpcaoAdicionarDisciplina
 - **Dependência** – Quando uma das classes numa associação não possui ou necessita de ter qualquer conhecimento da outra
Ex: Classes Disciplinas e BDDisciplinas
 - **Implementação da multiplicidade** – Quando a multiplicidade é superior a um usa-se normalmente **classes contentoras**.

Vista lógica

Refinação das relações entre classes

Eastern State University

- ◆ Resultantes do cenário “Adicionar uma disciplina” do caso de uso “Seleccionar as disciplinas a leccionar”:

Classe	Classe	Tipo Relação
OpcoesDisciplinasProfessor	OpcaoAdicionarDisciplina	Composição (unidir)
OpcoesDisciplinasProfessor	ListaIDValidos	Associação (unidir)
OpcaoAdicionarDisciplina	Curso	Associação (unidir)
Curso	Disciplina	Composição
Disciplina	Professor	Associação
Disciplina	BDDisciplinas	Dependência

Diagrama de classes dos cursos

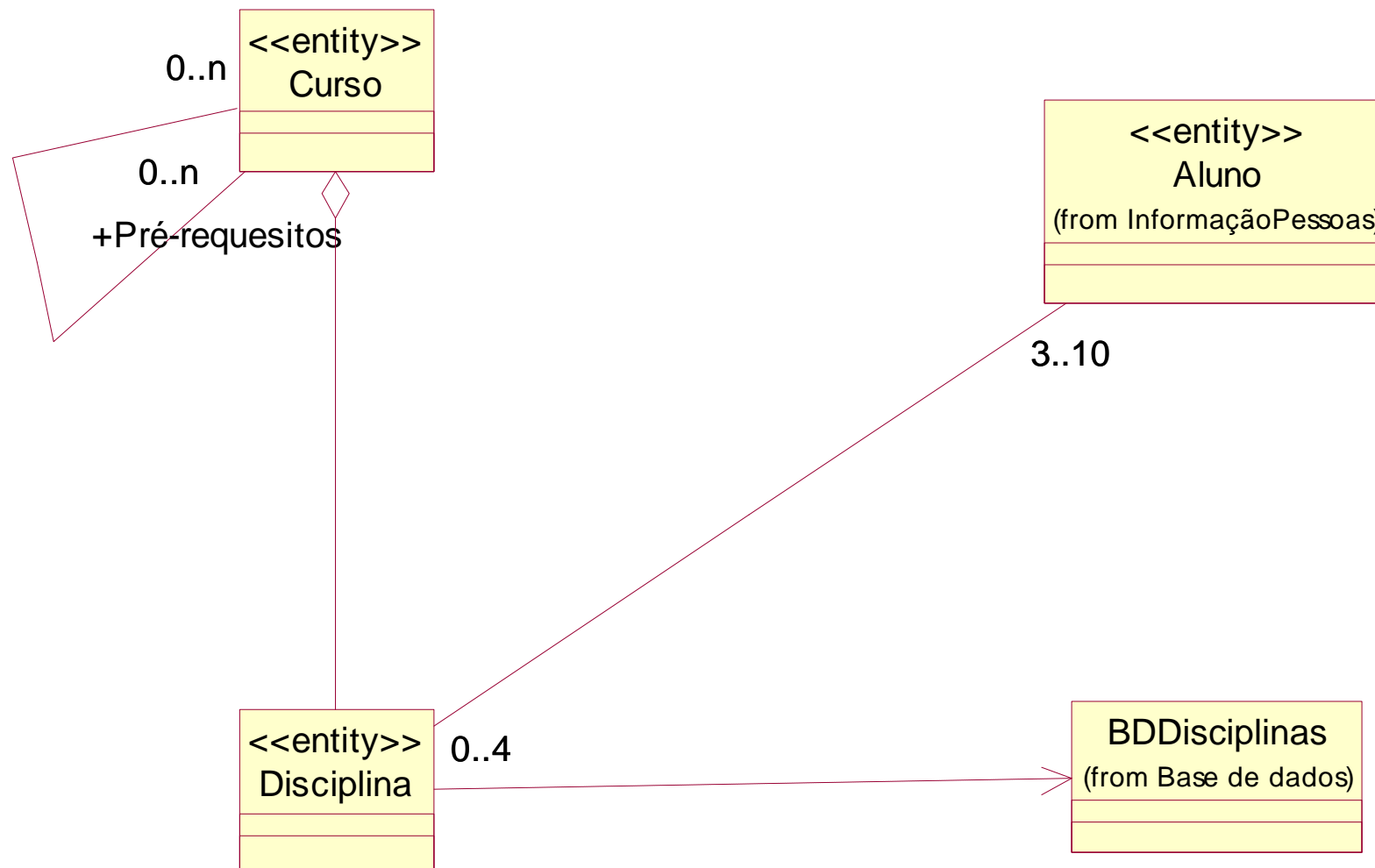
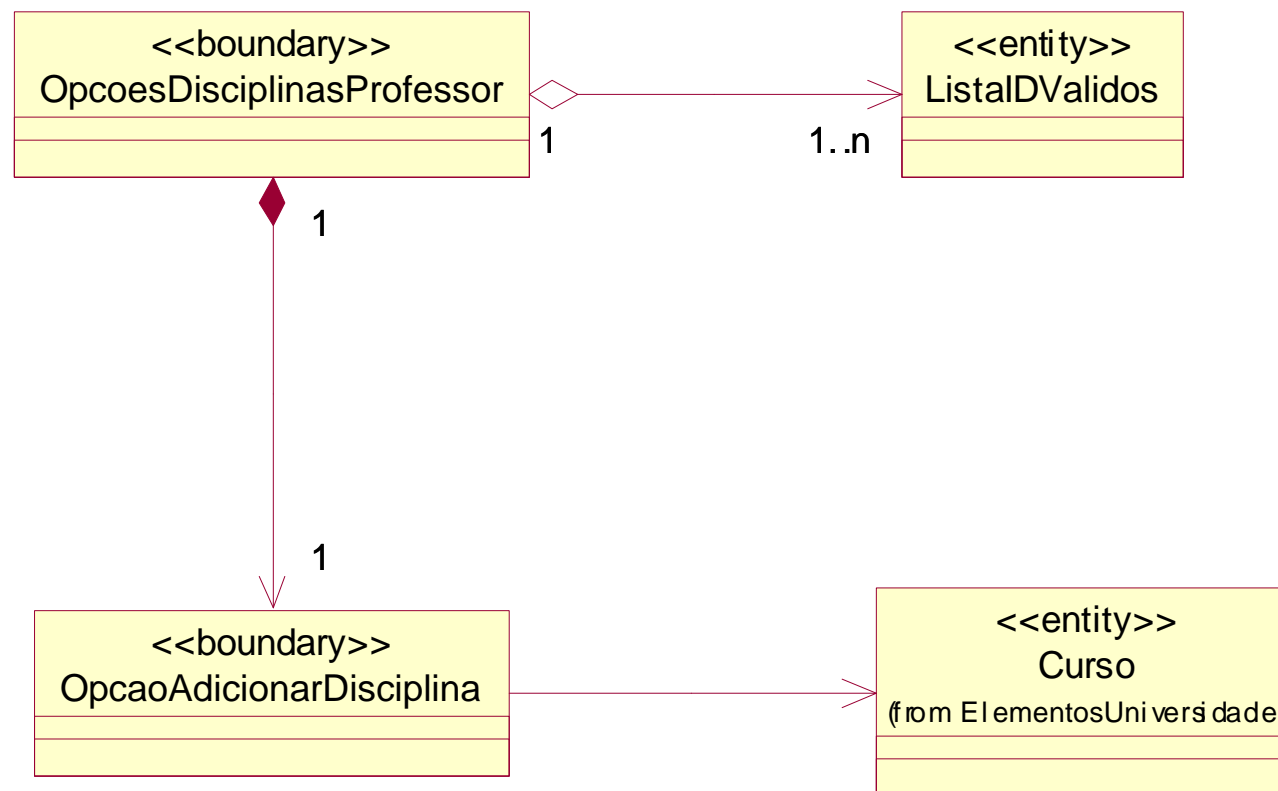




Diagrama de classes actualizado do pacote das interfaces







Vista lógica

Conclusão do design

Operações e atributos



- ◆ Completar o design das operações e atributos com:
 - **Tipos de dados** – são dependentes da linguagem
 - **Valores iniciais**
 - **Controlo de acesso** - privado, publico e protegido.

Diagrama de classes dos ElementosUniversidade com atributos e operações

