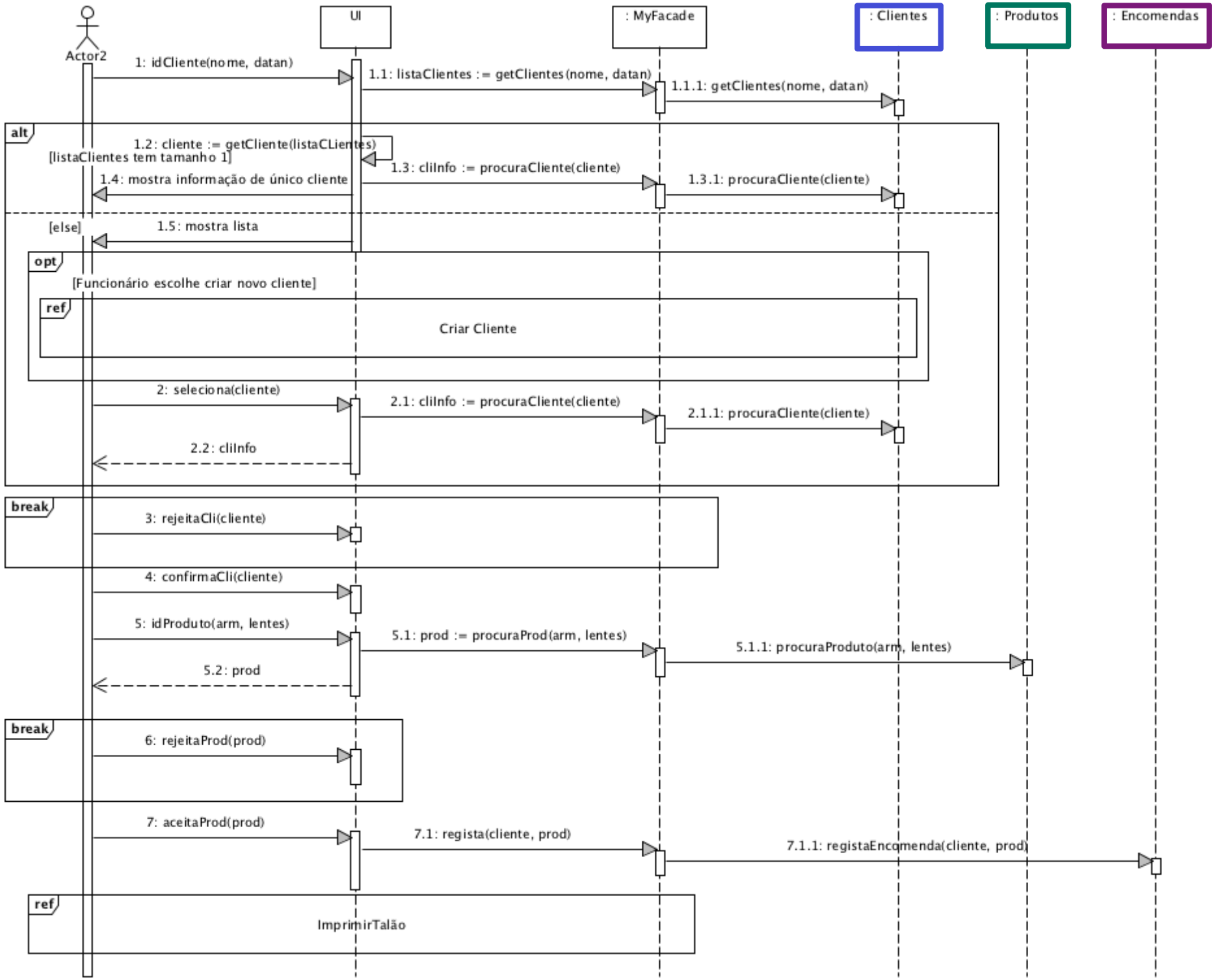
Desenvolvimento de Sistemas Software

Aula Teórica 18: Estratégia de Modelação Estrutural

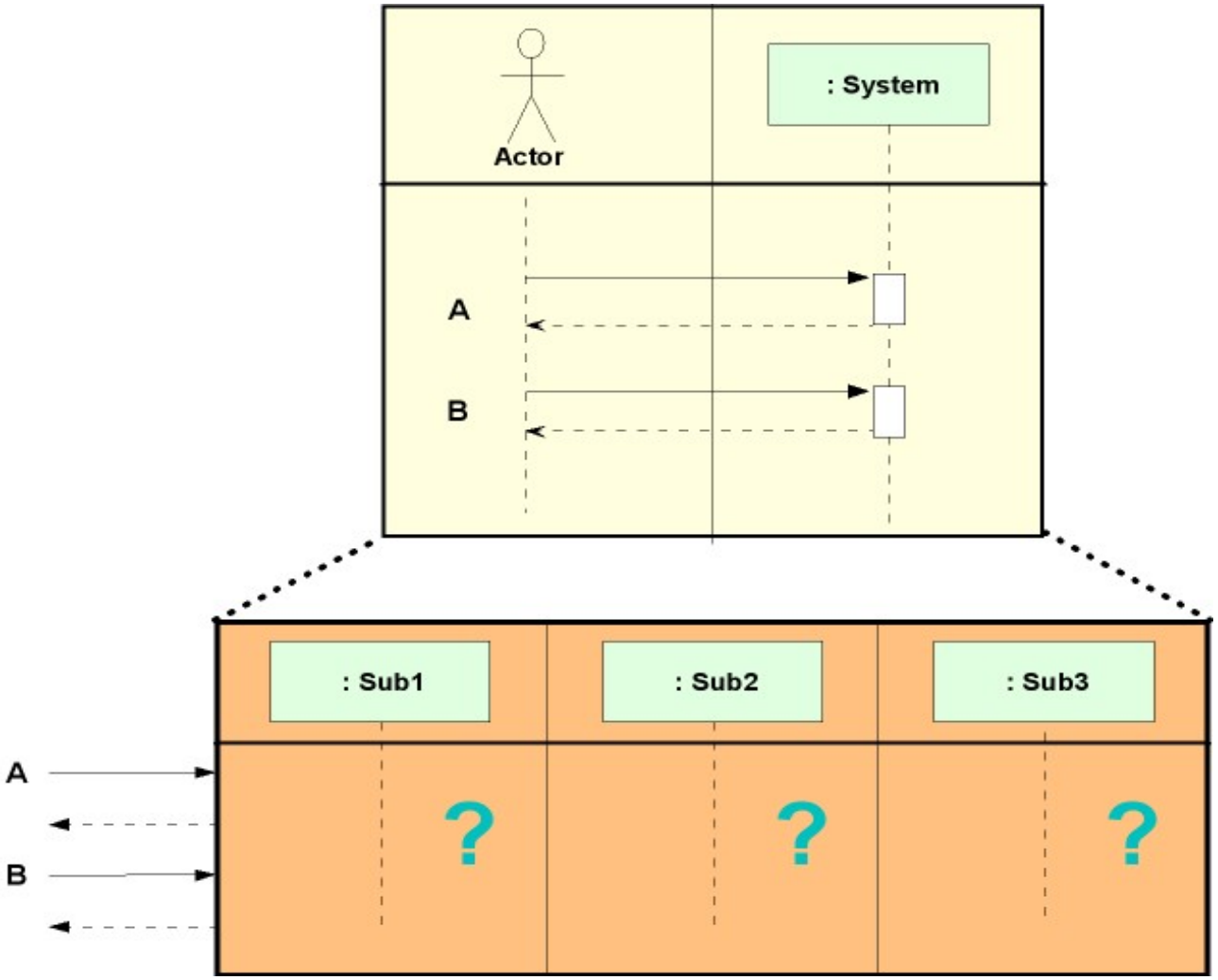
Desenvolvimento de Sistemas Software
António Nestor Ribeiro, José Creissac Campos, F. Mário Martins

Ponto da situação…

- Temos o suficiente para modelar estrutura e comportamento de um sistema
- Já identificamos requisitos funcionais
  - Sabemos *o que* o sistema deve fazer
- Que fazer agora com a linguagem?
  - Código ➔ diagramas – documentação
  - Diagramas ➔ código – programar?
- A questão é: *como* deve o sistema fazê-lo?
  - Usamos a linguagem como suporte à concepção de um sistema que responda aos requisitos funcionais identificados.
- Duas questões
  - Que classes?
  - Como (a que nível) modelar?


Actor2
UI
: MyFacade
: Clientes
: Produtos
: Encomendas
1: idCliente(nome, datan)
1.1: listaClientes := getClientes(nome, datan)
1.1.1: getClientes(nome, datan)
alt
1.2: cliente := getCliente(listaClientes)
[listaClientes tem tamanho 1]
1.3: cliInfo := procuraCliente(cliente)
1.4: mostra informação de único cliente
1.3.1: procuraCliente(cliente)
[else]
1.5: mostra lista
opt
[Funcionário escolhe criar novo cliente]
ref
Criar Cliente
2: seleciona(cliente)
2.1: cliInfo := procuraCliente(cliente)
2.1.1: procuraCliente(cliente)
2.2: cliInfo
break
3: rejeitaCli(cliente)
4: confirmaCli(cliente)
5: idProduto(arm, lentes)
5.1: prod := procuraProd(arm, lentes)
5.1.1: procuraProduto(arm, lentes)
5.2: prod
break
6: rejeitaProd(prod)
7: aceitaProd(prod)
7.1: regista(cliente, prod)
7.1.1: registaEncomenda(cliente, prod)
ref
ImprimirTalão

Subsistemas: Identificação



Temos que refinar o comportamento ou funcionalidade mas com que base e com que regras ?

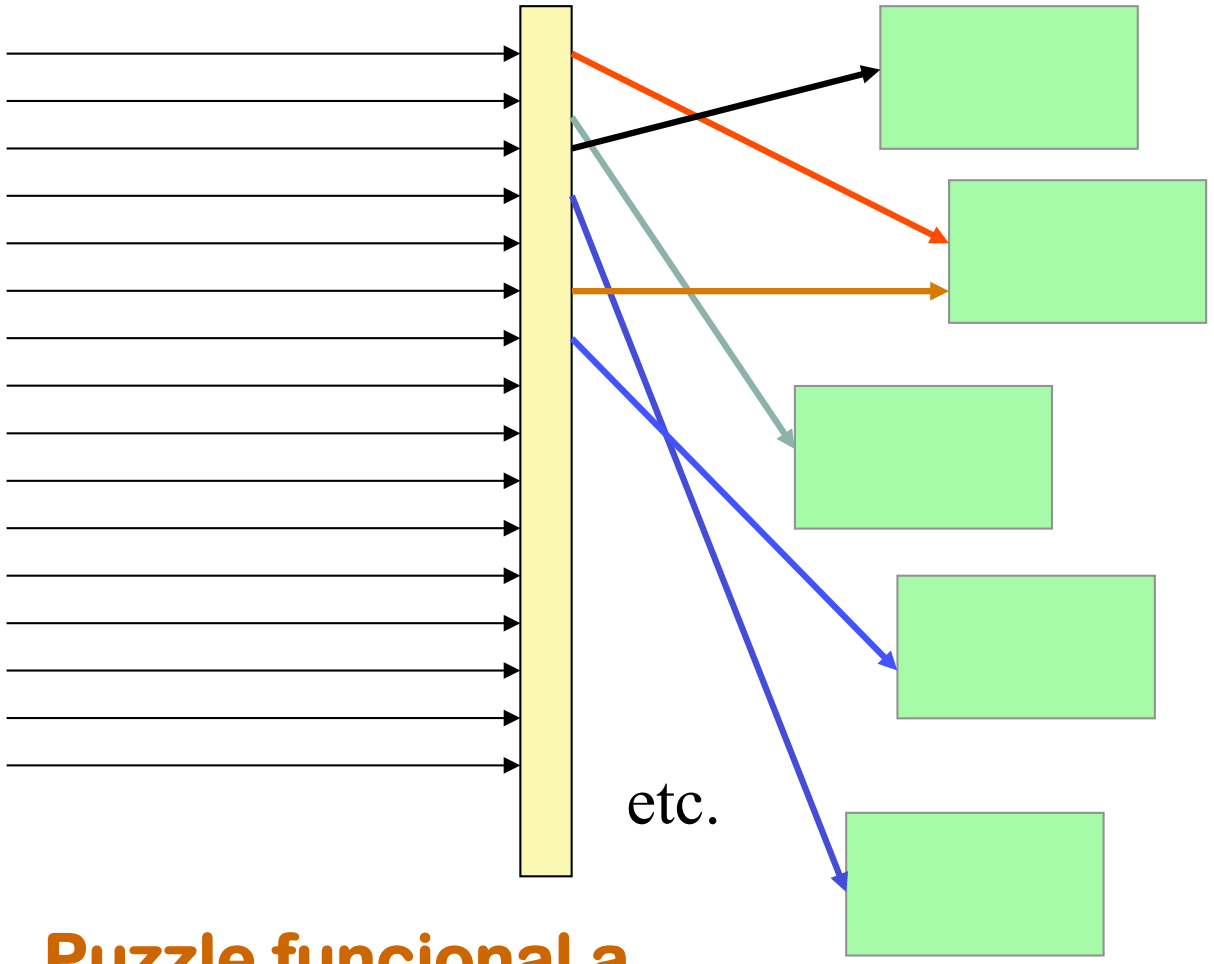
Procura de uma estratégia

- Um DSS representa, para cada UC, as interacções (mensagens) entre os actores e o sistema;
- O sistema é, a este nível, uma "black box", abstraindo-se da sua estrutura interna e do modo como realiza a sua funcionalidade;
- Mas, o total da funcionalidade "requisitada" e observável do sistema a desenhar, pode ser vista como todas as respostas do sistema às mensagens que os actores "enviam" ao sistema na sua interacção com ele, tal como especificado nos DSS obtidos dos UCs;
- Assim, deveríamos olhar para cada mensagem/evento que os actores iniciam para obter do sistema as suas "mais valias", e catalogá-las sob a forma de "responsabilidade funcional total" do sistema;
- Um bom modelo de domínio seria fundamental para uma tal abordagem ser "naive" mas coerente (cf. classes do domínio importantes no desenho do sistema são as "mesmas", mas refinadas, ao nível do desenho).

Procura de uma estratégia

O que temos então de momento ?

Uma carteira bem identificada de "responsabilidades funcionais", ou seja, de operações que as classes que vão representar o sistema terão que implementar em métodos.



Puzzle funcional a resolver

Concepção OO: a base

- **Questão:** Como se concebe a camada lógica de uma aplicação usando objectos ?

- **Resposta1:** Cria-se uma classe **:System** e nela se colocam todos os métodos necessários para que se tenha a funcionalidade pretendida

☹ **ERRADO, É EXACTAMENTE ISTO QUE NÃO QUEREMOS**

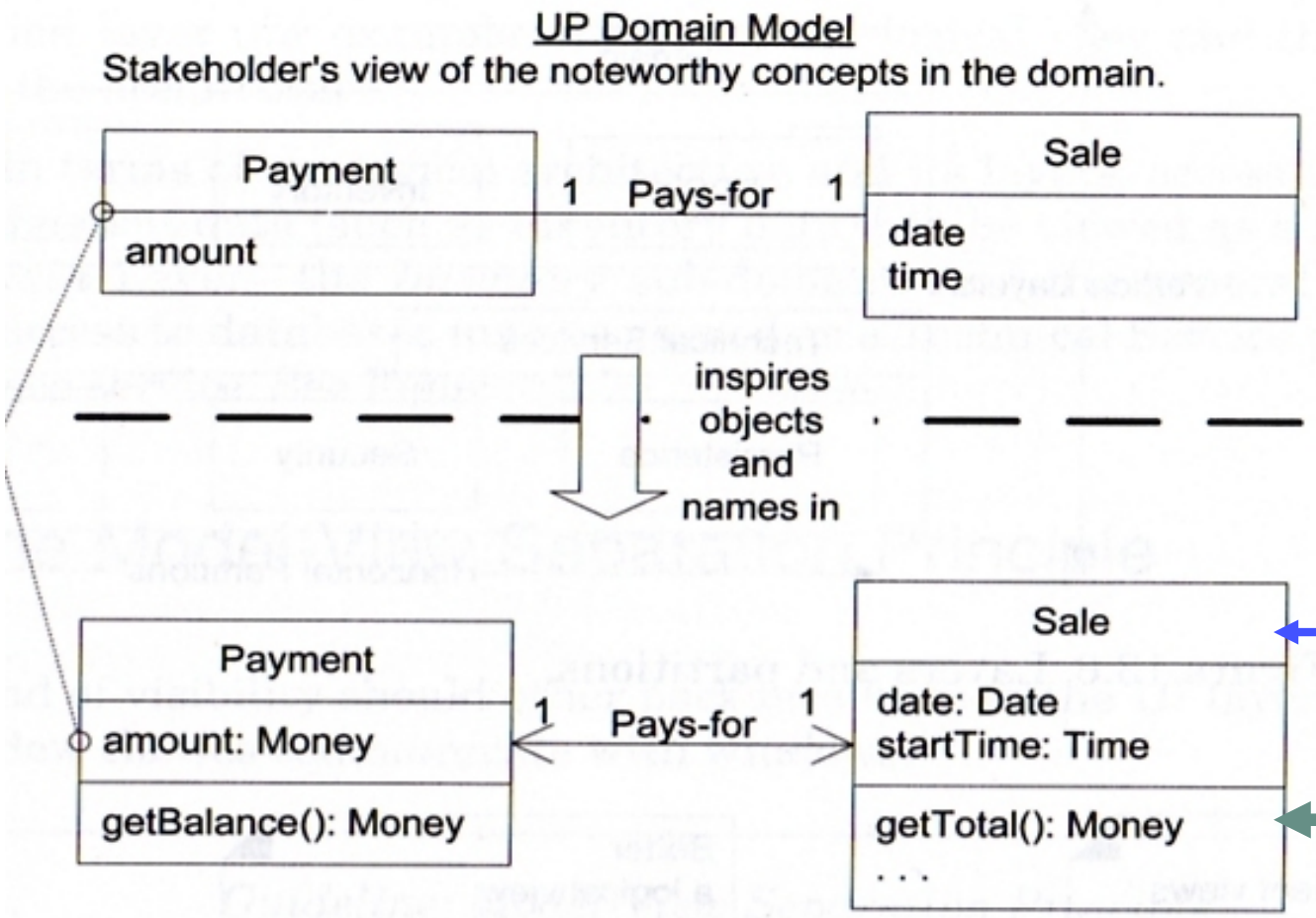
- **Resposta2:** Criam-se **"objectos de software"** com os mesmos nomes e atributos similares aos dos objectos/entidades identificadas no mundo real, que estão representados no **modelo do domínio**, e **atribuem-se-lhes responsabilidades funcionais (ou seja, métodos adequados).**

Estes objectos de software designam-se por **"objectos do domínio"** pois representam uma "coisa" importante do domínio do problema.

Ou seja, **concebem-se classes que possam criar tais objectos**.

Procura de uma estratégia

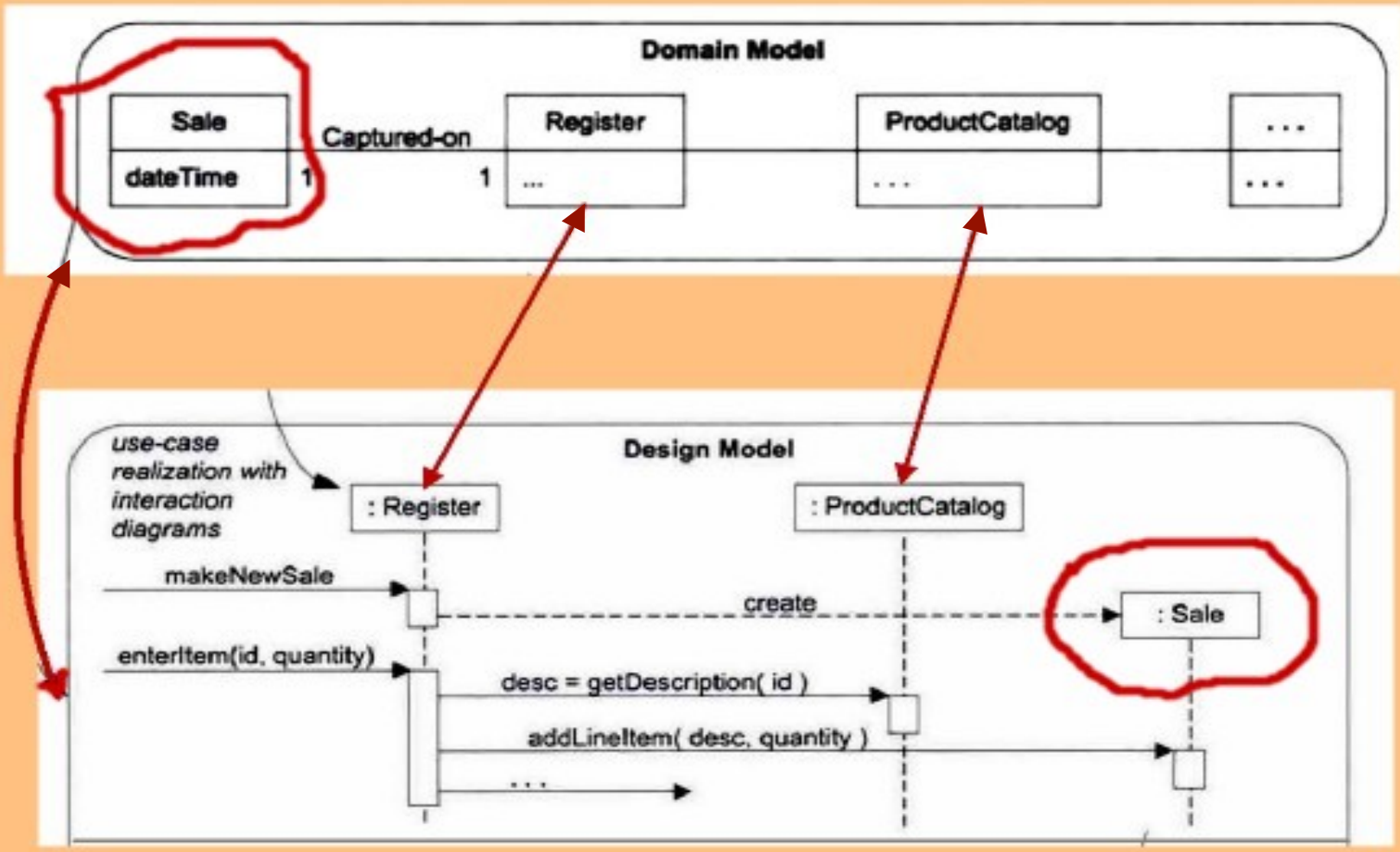
▣ **A identificação prévia de um conjunto de classes que fazem parte do Modelo do Domínio, sendo certo que algumas destas se irão tornar classes software do sistema software.**



Na concepção da camada lógica

Responsabilidades funcionais

Importância do Modelo de Domínio vs. Coerência dos projectos



Na criação dos Diagramas de Sequência de mais baixo nível

Estratégia (cont.)

- **Questão:** Como se identificam outras classes necessárias à implementação de tal **carteira de responsabilidades funcionais**, ou seja, que possuam métodos que implementem as operações de sistema que foram **contadas e identificadas** nos DSS ?

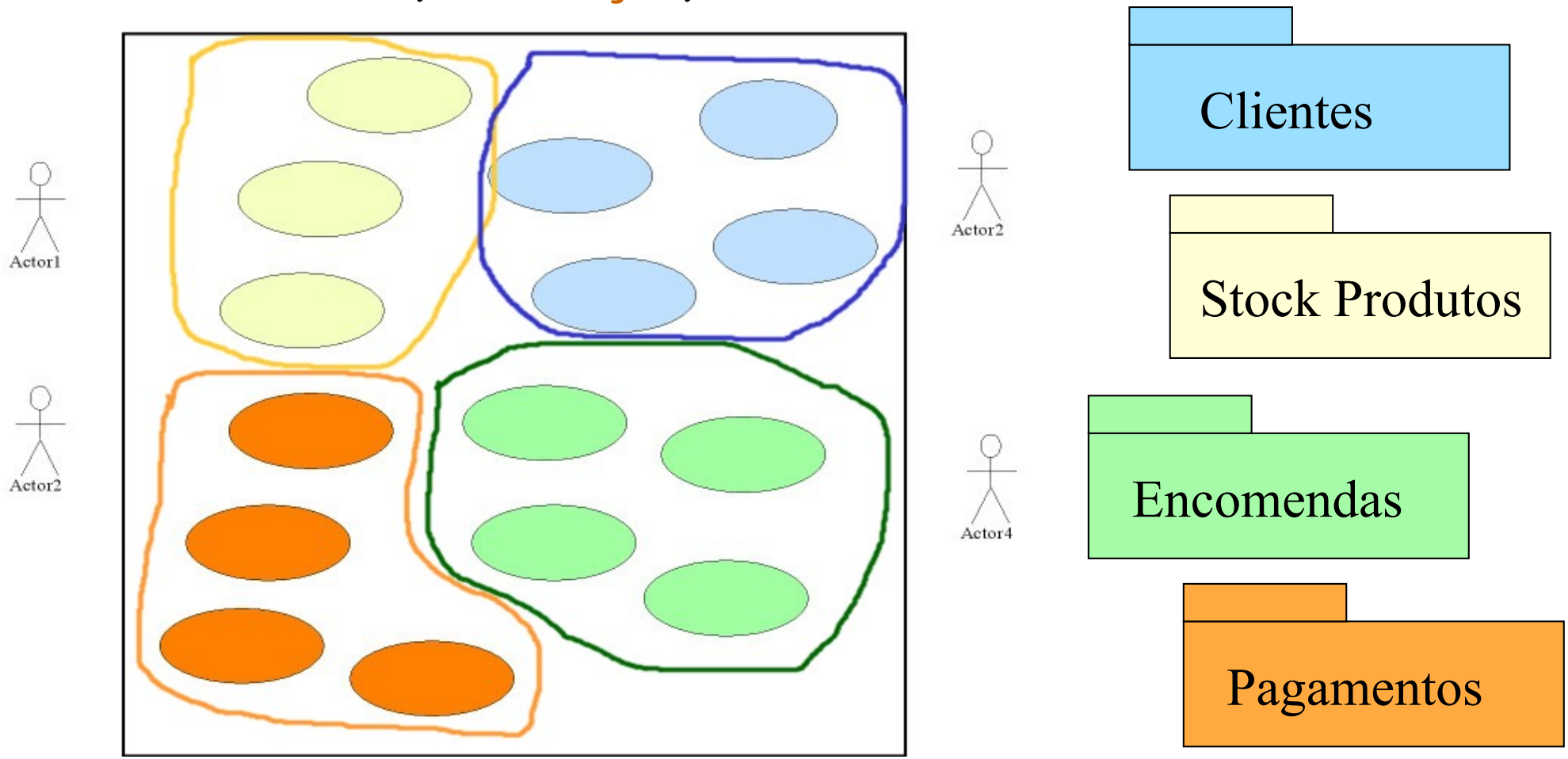
Resposta1: De forma intuitiva. **(☹ só "feeling"? É difícil).**

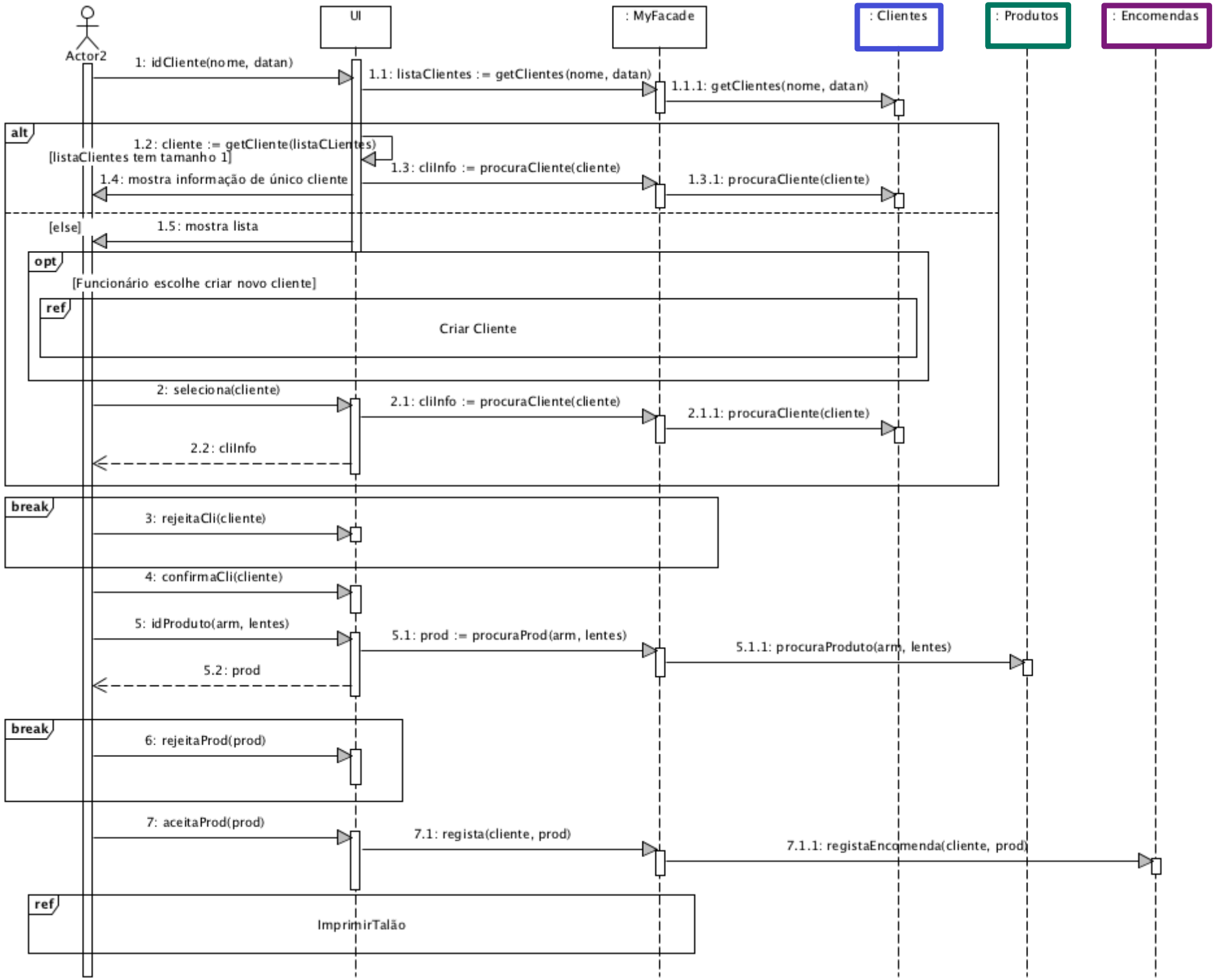
Resposta2: De forma racional usando os modelos que, **supostamente de forma coerente**, foram já desenvolvidos e, idealmente, seguindo os seguintes passos:

1) **Procurando identificar em primeiro lugar grandes grupos de funcionalidades, que se relacionam com "aspectos" do sistema ou até da arquitectura lógica escolhida, sabendo-se que, como se viu atrás, os "packages" podem muito bem representar, sob a forma de agrupamentos de classes, tais "grupos de funcionalidade";**

Procura de uma estratégia

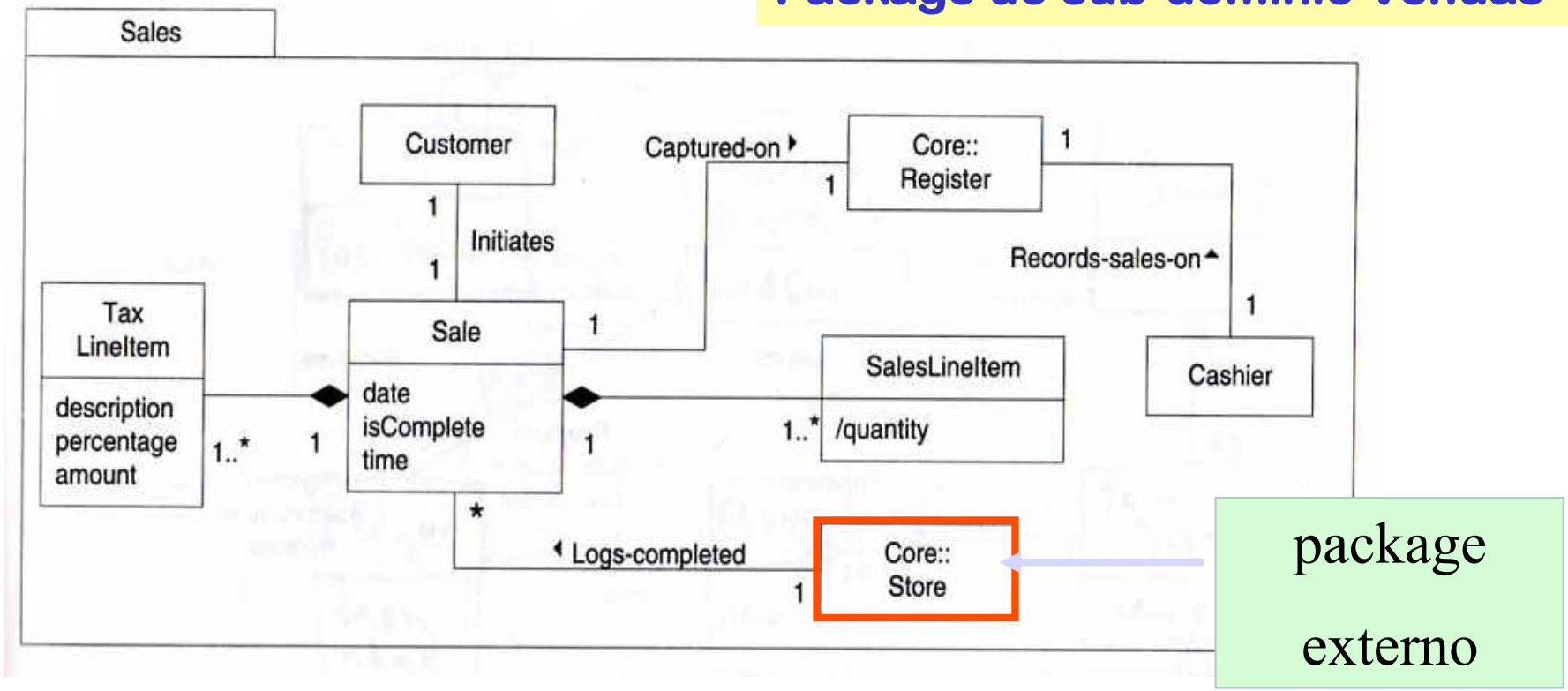
- **Podemos usar um (ou um conjunto) de Diagrama de Use Cases para identificar tarefas que, pela sua afinidade, possam ser agrupadas num grupo de funcionalidade mais abstracta, cf. Vendas, Gestão de Clientes, Gestão de Produtos, Facturação, etc.**




Actor2
UI
: MyFacade
: Clientes
: Produtos
: Encomendas
1: idCliente(nome, datan)
1.1: listaClientes := getClientes(nome, datan)
1.1.1: getClientes(nome, datan)
alt
1.2: cliente := getCliente(listaClientes)
[listaClientes tem tamanho 1]
1.3: cliInfo := procuraCliente(cliente)
1.4: mostra informação de único cliente
1.3.1: procuraCliente(cliente)
[else]
1.5: mostra lista
opt
[Funcionário escolhe criar novo cliente]
ref
Criar Cliente
2: seleciona(cliente)
2.1: cliInfo := procuraCliente(cliente)
2.1.1: procuraCliente(cliente)
2.2: cliInfo
break
3: rejeitaCli(cliente)
4: confirmaCli(cliente)
5: idProduto(arm, lentes)
5.1: prod := procuraProd(arm, lentes)
5.1.1: procuraProduto(arm, lentes)
5.2: prod
break
6: rejeitaProd(prod)
7: aceitaProd(prod)
7.1: regista(cliente, prod)
7.1.1: registaEncomenda(cliente, prod)
ref
ImprimirTalão

▣ Em seguida, se assumirmos esta abordagem "top-down" a partir dos diagramas de Use Cases, cada "package" deve agora ver a sua estrutura especificada usando um diagrama de classes, ficando da forma-exemplo:

Package do sub-domínio Vendas



Responsabilidades: CRC-cards

Responsability-based modeling

A análise da "responsabilidade" (o que deve fazer; o que deve "saber") de cada entidade no funcionamento global de um sistema, permite identificar subsistemas e suas formas de colaboração.

CRC-cards permitem melhor "desenhar" as nossas classes.

CRC-cards são usados para tentar analisar a criação de possíveis subsistemas

Responsabilidades

- O que sabe (atributos)
- O que faz (métodos)

Técnica: CRC-cards

Class Name:	
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators

Coisas que sabe

Coisas que faz

Classes que devem colaborar com esta para que esta possa cumprir as suas responsabilidades

Técnica: CRC-cards

CRC Card Sample

A CRC card is an index card that is use to represent the responsibilities of classes and the interaction between the classes.

BankAccount

Super Classes :

Sub Classes : SavingAccount, MarginAccount

Description :
Store the transcation record, customer data, balance, etc.

Attributes :

Name	Description
accountNumber	A unique value to identify the accounts

Responsibilities :

Name	Collaborator
Keep the latest value of the balance	Bank controller, Transcation records

BankController

Super Classes :

Sub Classes : AccountController, TranscationController, ATMController

Description :
Control the interactions between the customer and the bank system.

Attributes :

Name	Description
status	identify the status of the controller

Responsibilities :

Name	Collaborator
withDraw	Withdraw money from the bank acco

SavingAccount

Super Classes : BankAccount

Sub Classes :

Description :
Store the cash information of the customer record.

Attributes :

Name	Description
cashBalance	latest value of the cash balance

Responsibilities :

Name	Collaborator
getBalance	TranscationController, AccountControl

ATMController

Super Classes : BankController

Sub Classes :

Description :
Control the interactions between customer and the ATM terminals.

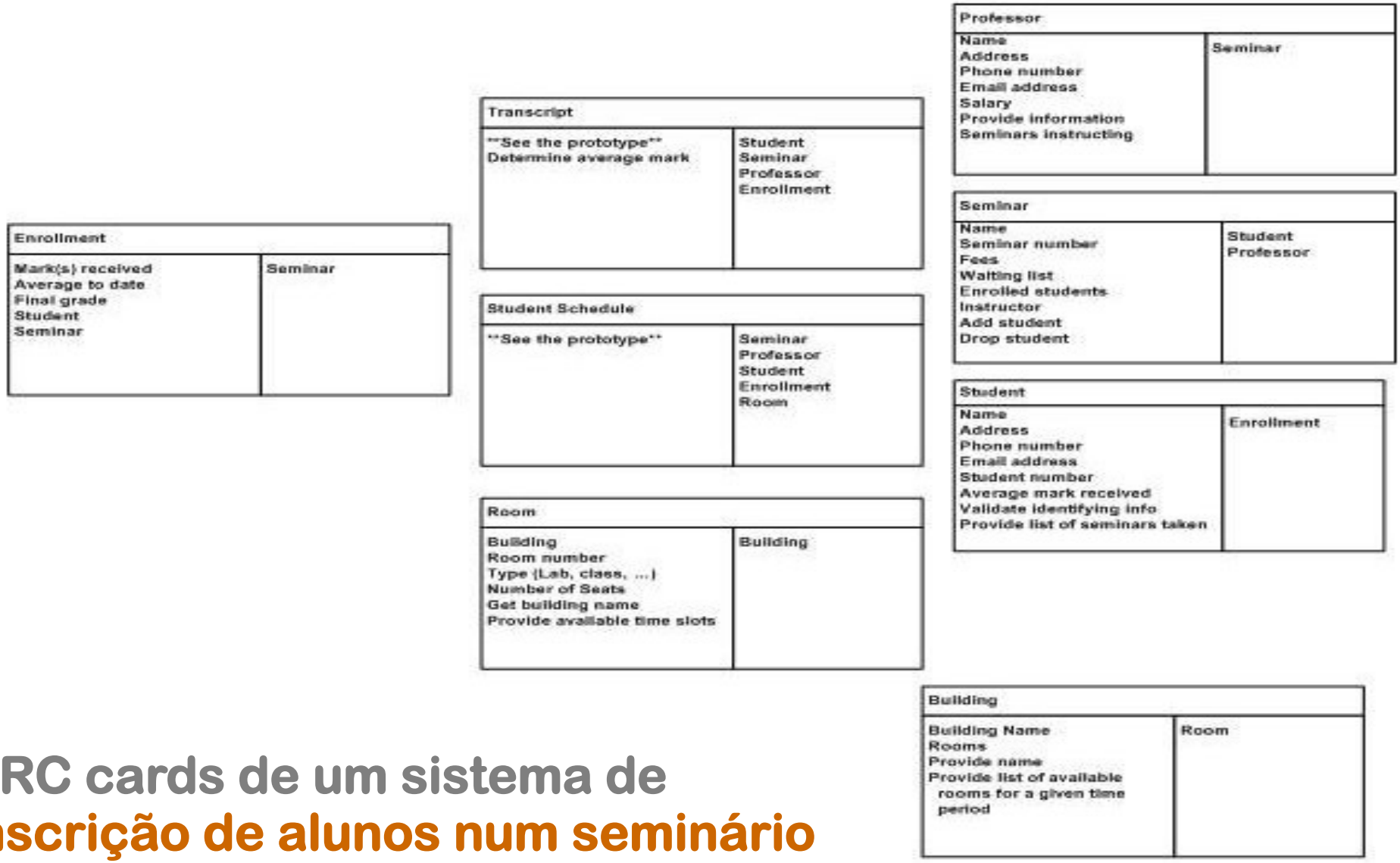
Attributes :

Name	Description
machineType	identifiy the type of the ATM terminal

Responsibilities :

Name	Collaborator
checkingPassword	

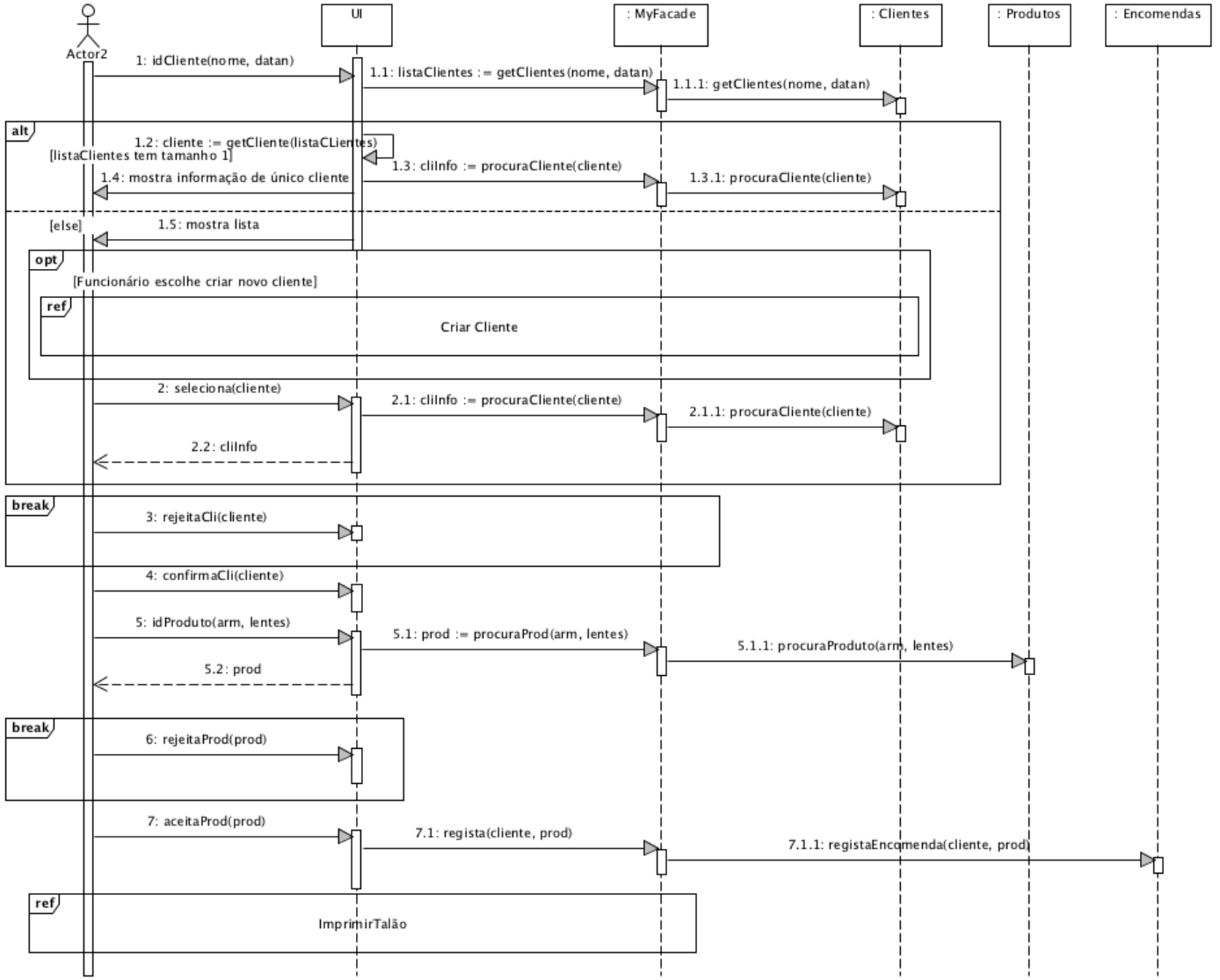
Técnica: CRC-cards



CRC cards de um sistema de **inscrição de alunos num seminário**

Novamente o exemplo…

- Que classes para cada subsistema?



Estratégia de Modelação Comportamental

Sumário

- Estratégia para a identificação das classes de um sistema
- *CRC cards (Class-responsibility-collaboration)*
- Exemplo ilustrativo da estratégia a seguir
  - identificação de responsabilidades
  - definição do modelo de classes
  - diferentes níveis de abstração na modelação