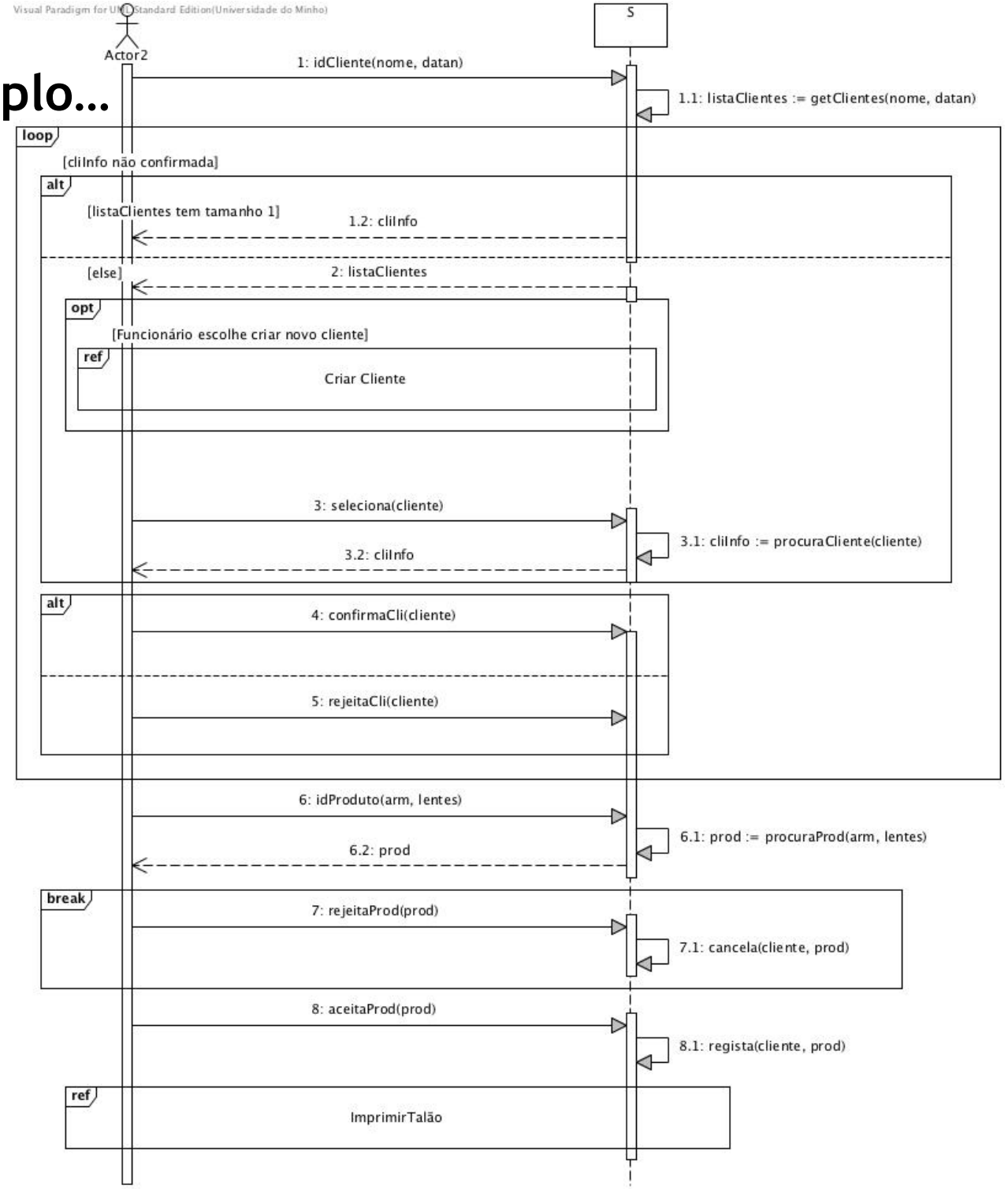
Desenvolvimento de Sistemas Software

Aula Teórica 15: Modelação comportamental com Diagramas de Sequência

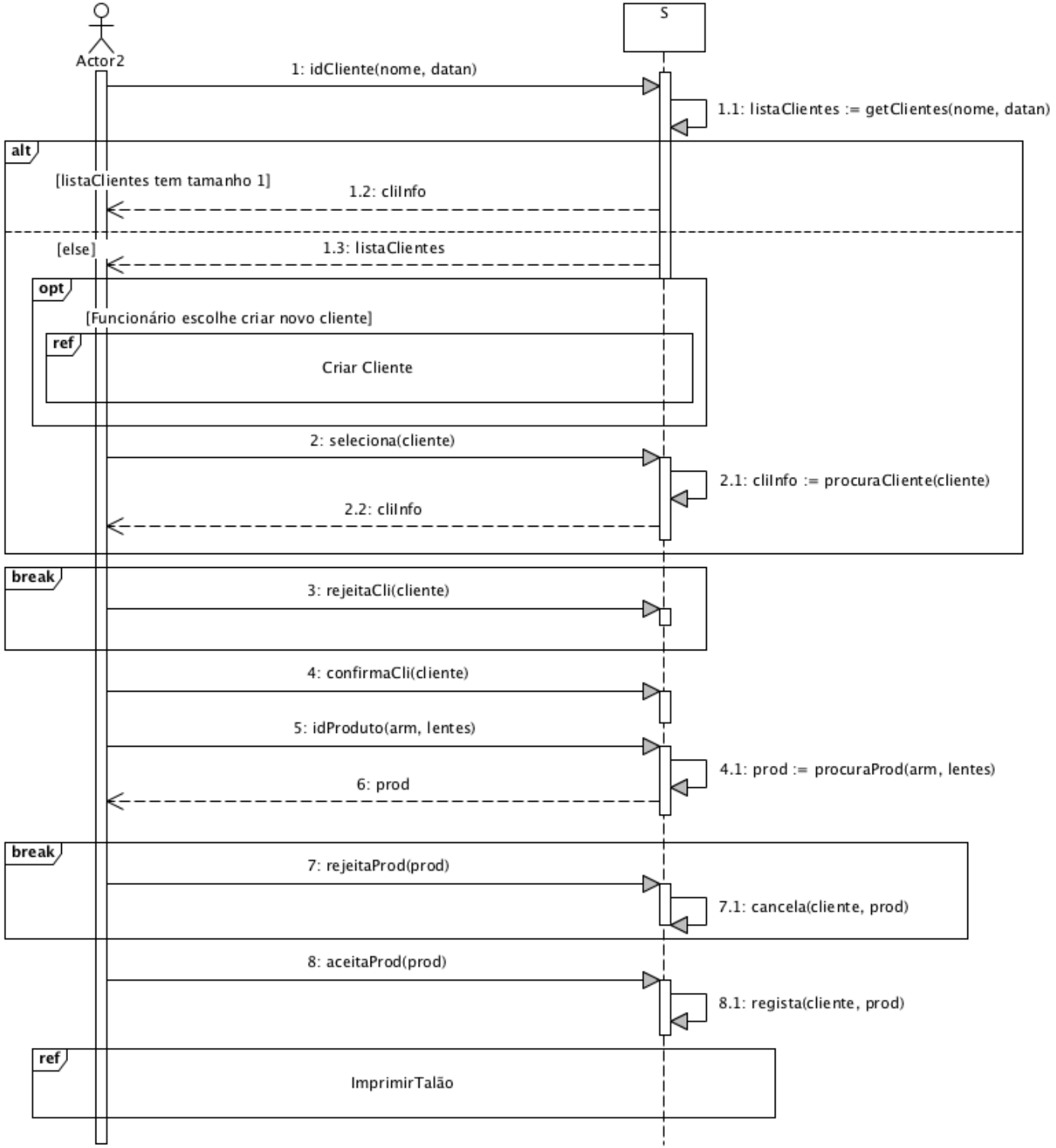
Desenvolvimento de Sistemas Software
António Nestor Ribeiro, José Creissac Campos, F. Mário Martins

Um exemplo…

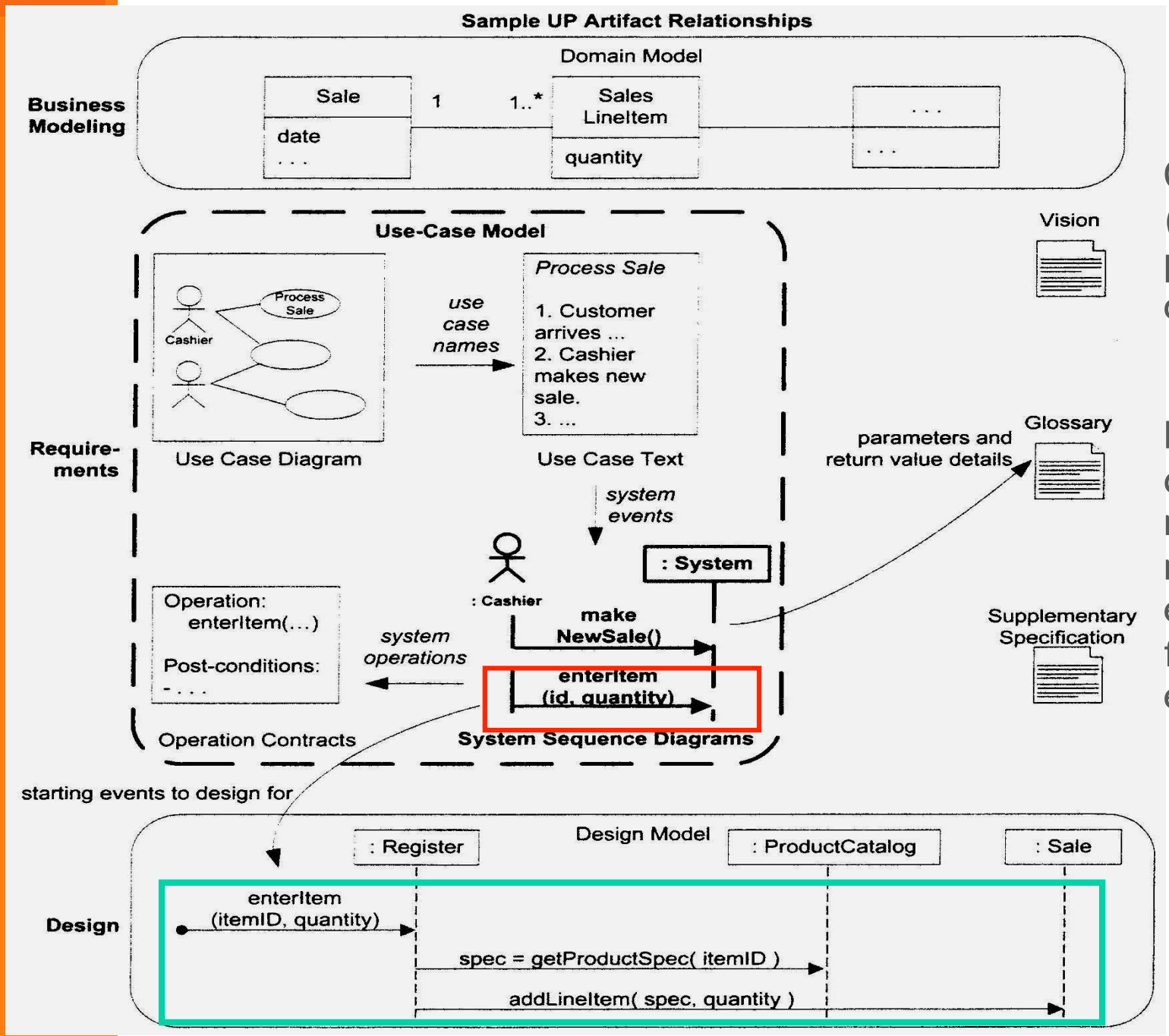


Mais interface que Use Case

DSS – DS ao nível do Sistema



ONDE ESTAMOS RUP/DSS ?



DS: Refinamento

Os DS a nível de Sistema (DSS) são o ponto de partida para os DS de concepção.

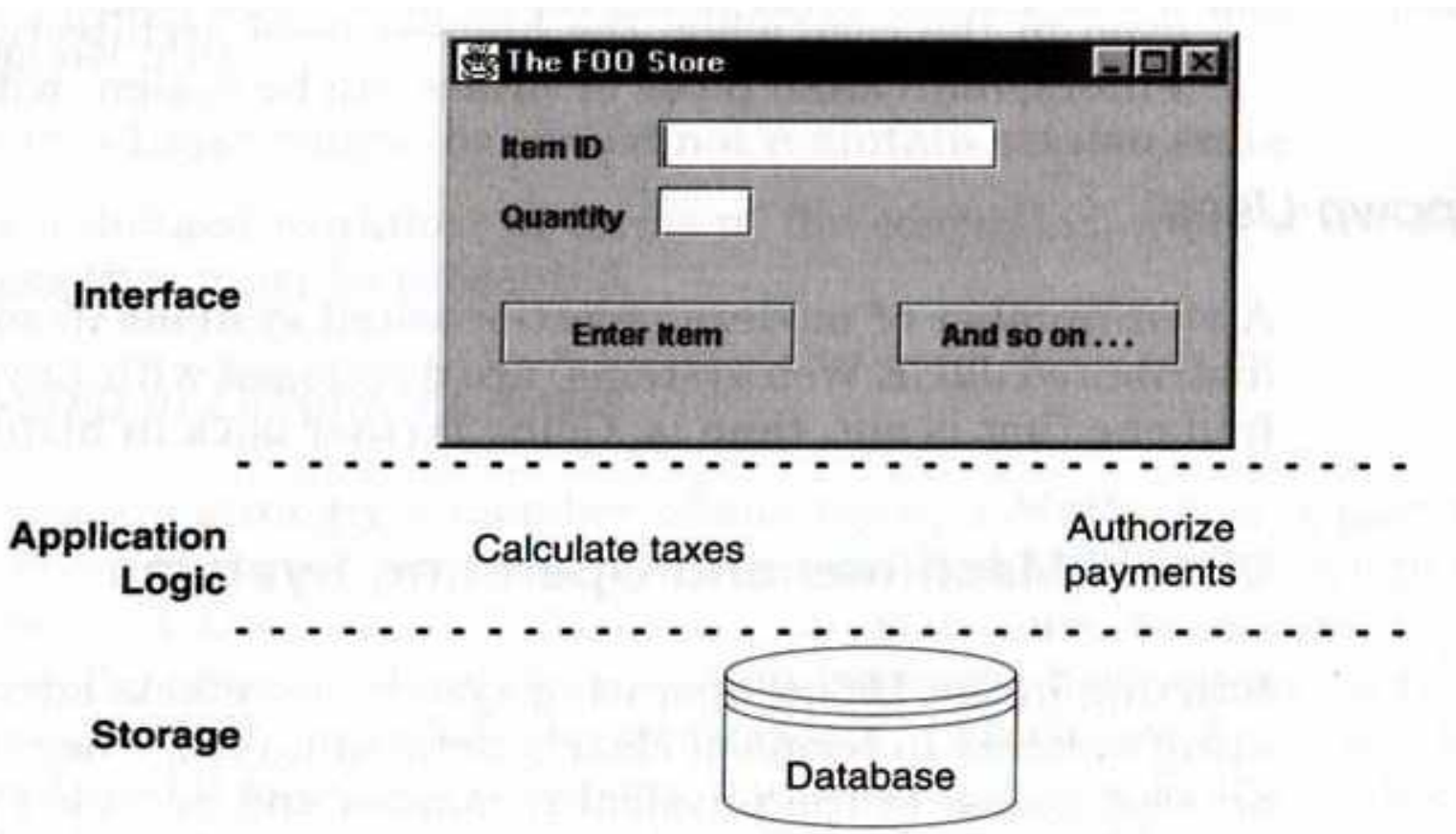
Porém, temos que saber dividir o objecto :System nos subsistemas, ou seja, nas entidades funcionais e estruturais que irão fornecer a funcionalidade especificada e pretendida.

Subsistemas: Importância

- **Subsistemas** são partições de um sistema que possuem algumas das seguintes propriedades e/ou características:
  - ☑ **Representam unidades independentes**, ou seja, que são autónomas, que podem ser internamente alteradas sem que tal implique alterações nas outras unidades, porque mantêm as API ou interfaces;
  - ☑ **Podem portanto ser independentemente desenhados, testados e instalados**; A suas mudanças, adaptações, etc. não implicam com as outras unidades do sistema software (mantendo a API);
  - ☑ **Podem representar sistemas externos** à concepção;
  - ☑ **Podem representar/modelar "componentes"**;
  - ☑ **Podem representar agrupamentos de classes (packages!)** que em conjunto representam funcionalidade de nível superior de granularidade, que são depois "usadas" através de uma API conjunta, encapsulando a sua estrutura interna, e que é capaz de criar instâncias em "run-time" !!

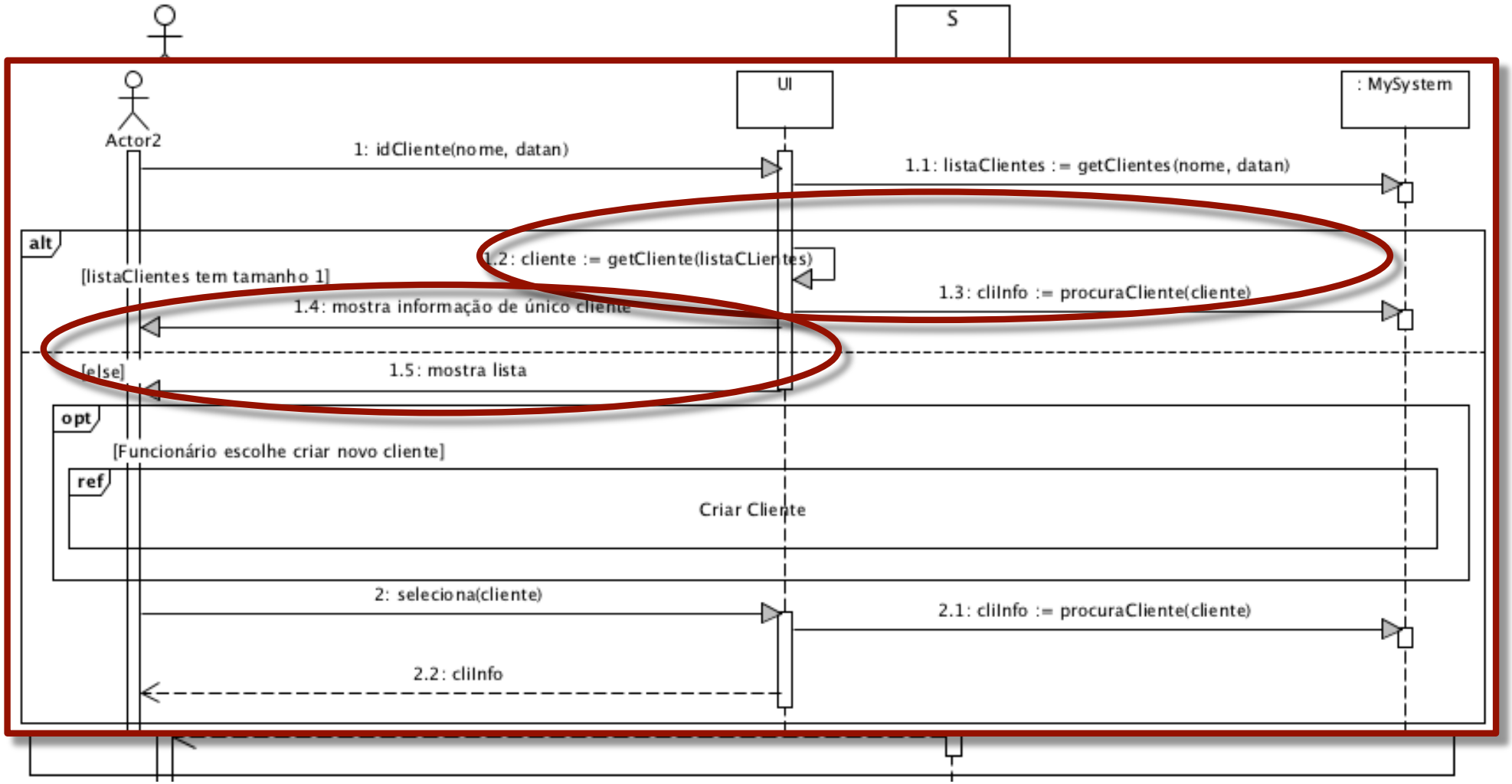
Como identificar subsistemas com estas propriedades na concepção ?

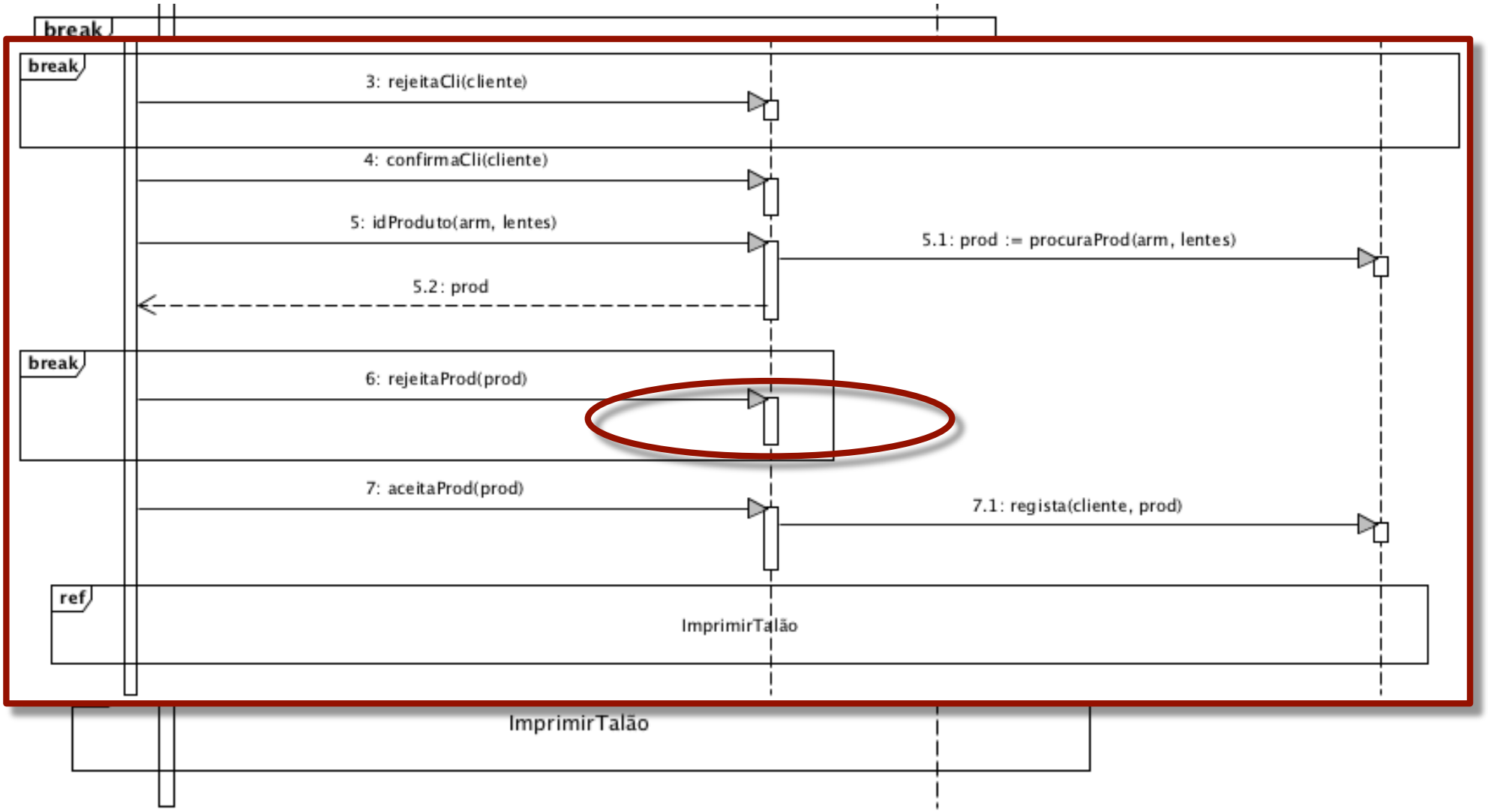
Subsistemas cf. Arquitectura

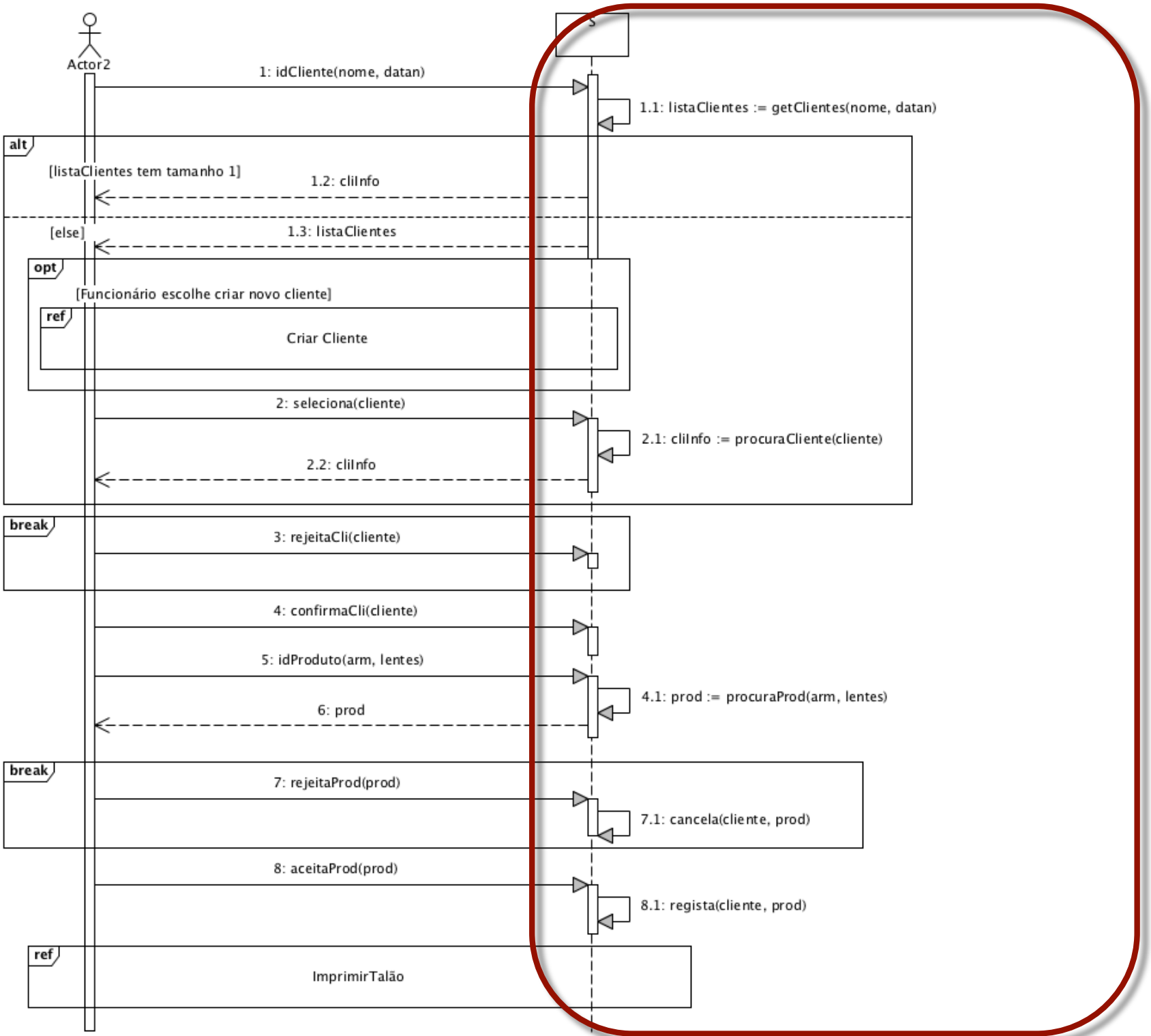


- Por exemplo, se como requisito tivermos por **"objectivo"** uma **arquitectura de 3 camadas**, então será natural que tentemos especificar tendo por base 3 grandes "packages".

Voltando ao exemplo...




break
break
3: rejeitaCli(cliente)
4: confirmaCli(cliente)
5: idProduto(arm, lentes)
5.1: prod := procuraProd(arm, lentes)
5.2: prod
break
6: rejeitaProd(prod)
7: aceitaProd(prod)
7.1: regista(cliente, prod)
ref
ImprimirTalão
ImprimirTalão


Actor2
1: idCliente(nome, datan)
1.1: listaClientes := getClientes(nome, datan)
alt
[listaClientes tem tamanho 1]
1.2: cliInfo
[else]
1.3: listaClientes
opt
[Funcionário escolhe criar novo cliente]
ref
Criar Cliente
2: seleciona(cliente)
2.1: cliInfo := procuraCliente(cliente)
2.2: cliInfo
break
3: rejeitaCli(cliente)
4: confirmaCli(cliente)
5: idProduto(arm, lentes)
4.1: prod := procuraProd(arm, lentes)
6: prod
break
7: rejeitaProd(prod)
7.1: cancela(cliente, prod)
8: aceitaProd(prod)
8.1: regista(cliente, prod)
ref
ImprimirTalão

Um DSS com :System deverá ser refinado num DS com :Subsytem1, etc…

Isto quer dizer que depois de termos especificado fundamentalmente comportamento (ou seja o **"yin"** do UML), cf. Use Cases, Diagramas de Actividade e Diagramas de Sequência, devemos agora preocuparmo-nos com o **"yang"**, ou seja, a estrutura.

Numa metodologia OO, as entidades que compõem um sistema e que representam a sua estrutura são modeladas, naturalmente, usando **CLASSES**, pois são estas que criam os objectos, que são as entidades computacionais.

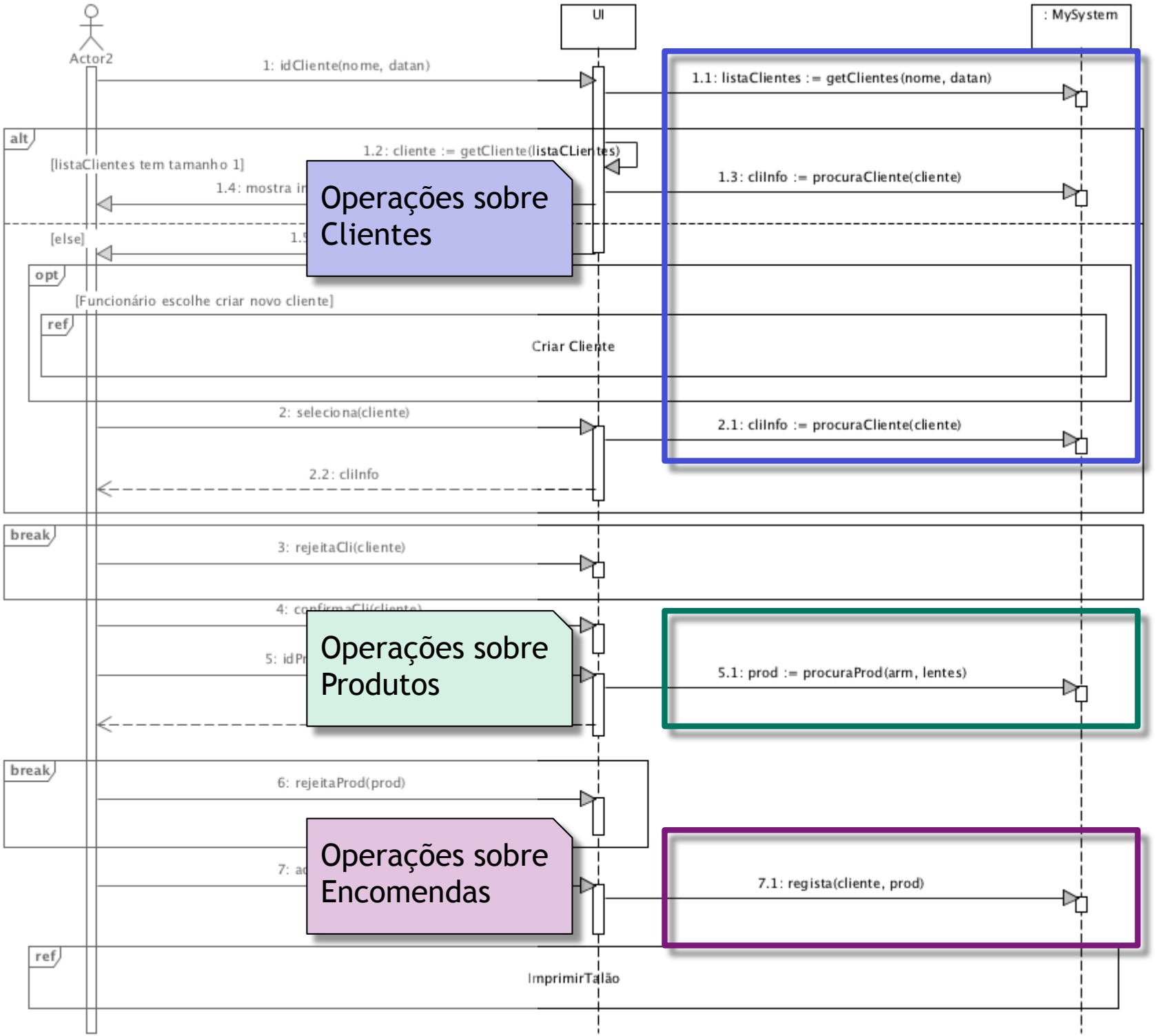
DSS: Refinamento

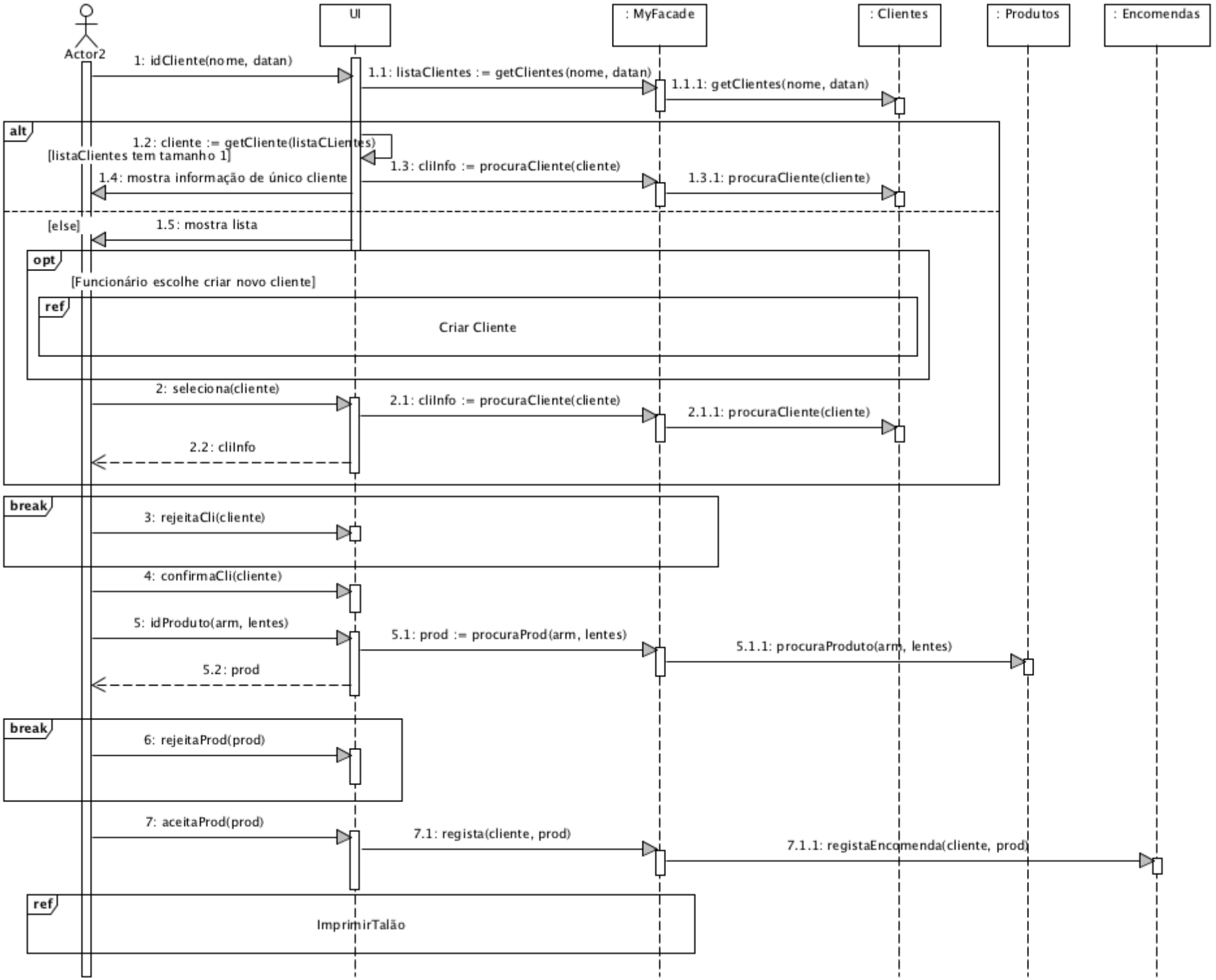
Questão: Qual o critério, ou método, para se passar de um DSS para um DS de concepção ?

Resposta: Dividindo a "blackbox" :System em subsistemas que possuam a "competência" (o que deve saber) e a "responsabilidade" (o que deve fazer), para responder a certas operações especificadas ao nível do Sistema;

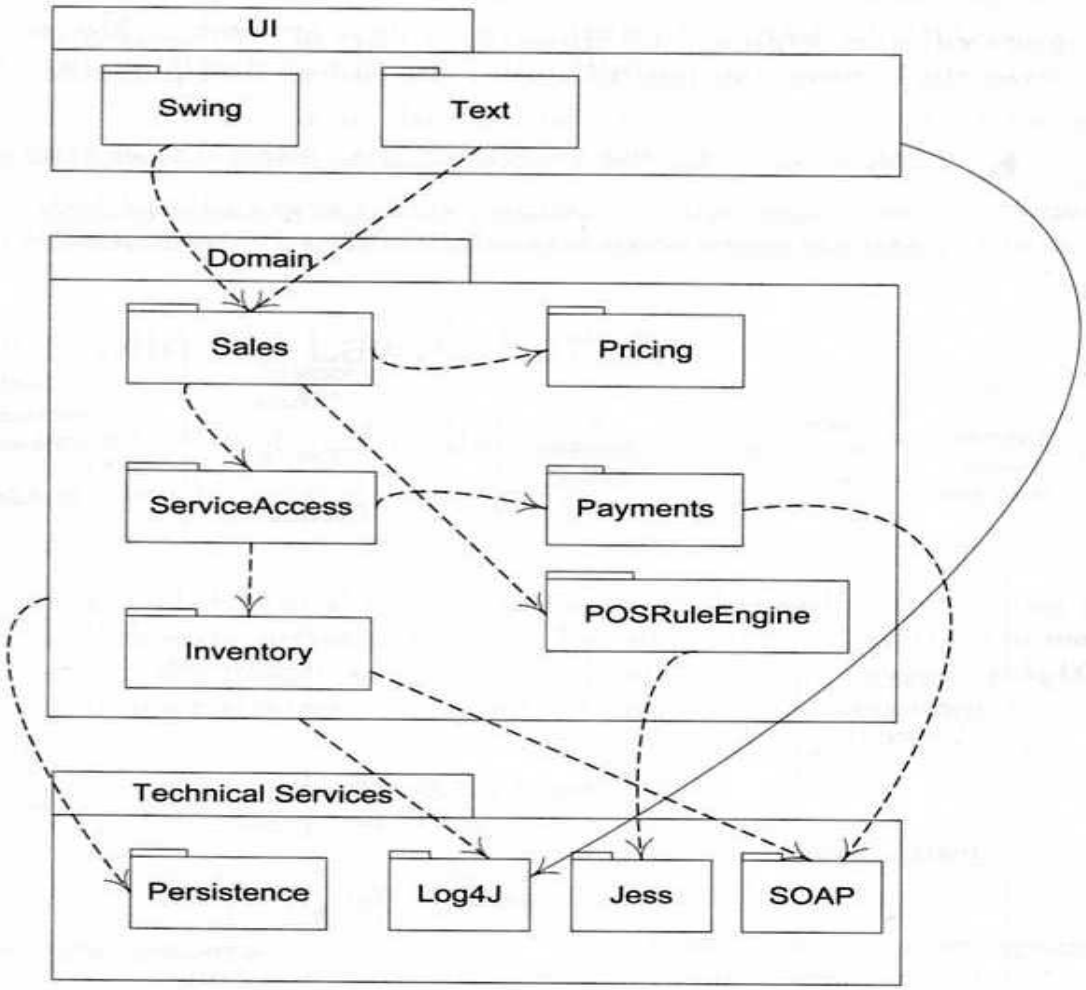
Questão: O que "são" tais subsistemas num DS ?
Resposta: Dado que num DS de UML só há "objectos", serão objectos, ainda que possam ser de "tipos" diferentes, ie., de classes diferentes;

Questão: Como especificar então as competências e as responsabilidades de um objecto a este nível ?
Resposta: Há várias técnicas. Nesta fase vamos agrupar as operações em função das entidades do modelo de domínio.


Actor2
UI
: MySystem
1: idCliente(nome, datan)
1.1: listaClientes := getClientes(nome, datan)
alt
[listaClientes tem tamanho 1]
1.2: cliente := getCliente(listaClientes)
1.3: cliInfo := procuraCliente(cliente)
1.4: mostra in
Operações sobre Clientes
[else]
opt
[Funcionário escolhe criar novo cliente]
ref
Criar Cliente
2: seleciona(cliente)
2.1: cliInfo := procuraCliente(cliente)
2.2: cliInfo
break
3: rejeitaCli(cliente)
4: confirmaCli(cliente)
5: idPr
Operações sobre Produtos
5.1: prod := procuraProd(arm, lentes)
break
6: rejeitaProd(prod)
7: ad
Operações sobre Encomendas
7.1: regista(cliente, prod)
ref
ImprimirTalão


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)
1.3.1: procuraCliente(cliente)
1.4: mostra informação de único 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 cf. Arquitectura



Assim, packages podem ser desenhados usando uma estratégia de 1 package por "responsabilidade", ou por camada específica da arquitectura e dos serviços requisitados.

Dentro de cada mega-package, e em função da análise, podemos criar sub-packages dentro dos quais as classes representam as informações e os comportamentos que, em colaboração entre elas, implementam os requisitos funcionais de alto nível.

Resumindo o exemplo...

Use Case: Receber receita
Descrição: Funcionário processa a receita de um cliente
Pré-condição: Existe papel para imprimir talões
Pós-condição: Pedido de óculos fica registado
Comportamento normal:

1. Funcionário indica nome e/ou data de nascimento do cliente
2. Sistema apresenta lista de clientes correspondentes
3. Funcionário selecciona cliente [ponto de extensão: seleção]
4. Sistema apresenta detalhes do cliente seleccionado
5. Funcionário confirma cliente
6. Funcionário indica código de armação e de lentes
7. Sistema procura produto e apresenta detalhes
8. Funcionário confirma produto
9. Sistema regista reserva
10. «include» Imprimir Talão

Comportamento Alternativo [lista de clientes correspondentes tem tamanho 1]

2.1. Sistema apresenta detalhes do único cliente da lista
2.2. regressa à 5

Exceção

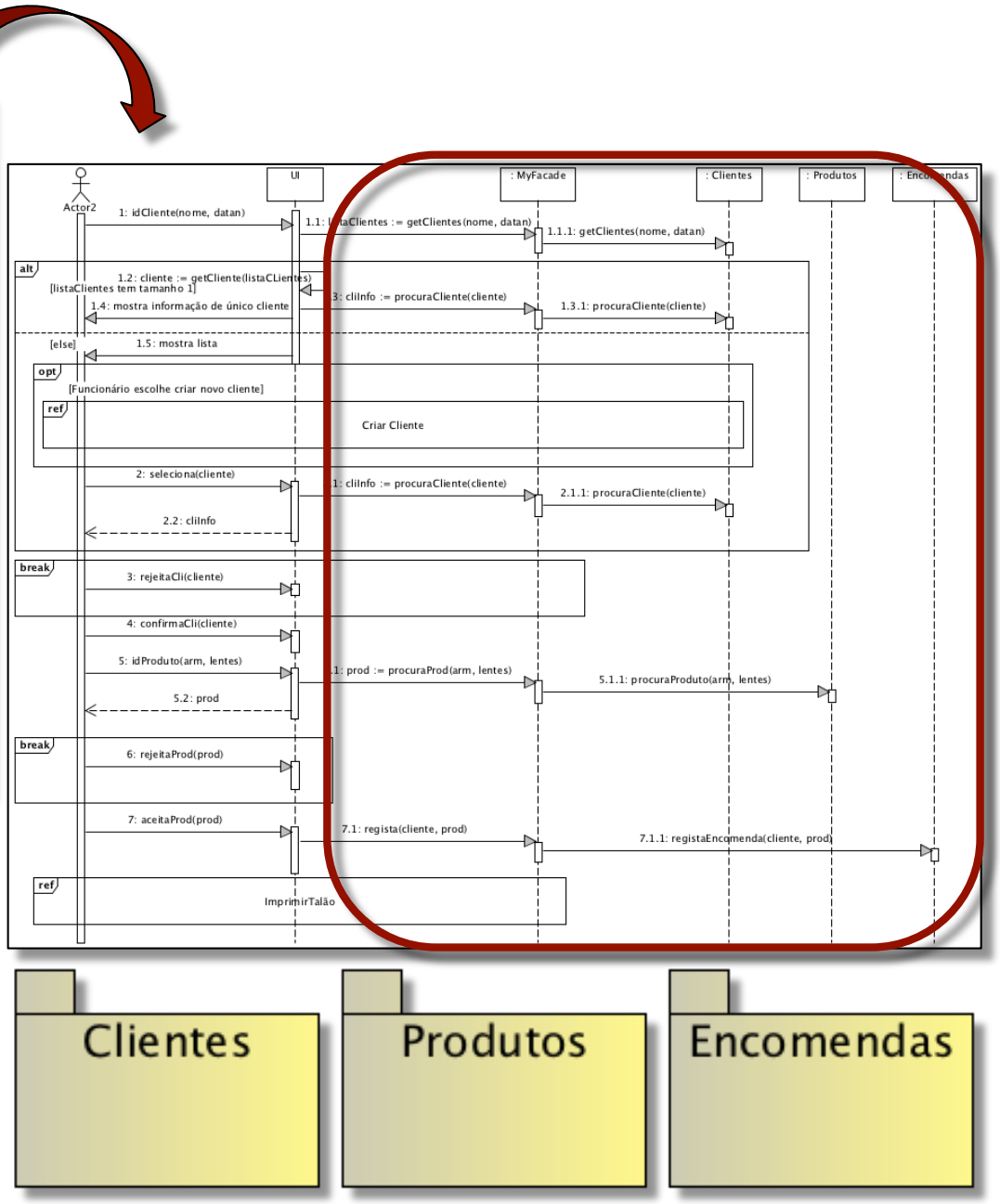
5.1. Funcionário não confirma cliente

Exceção

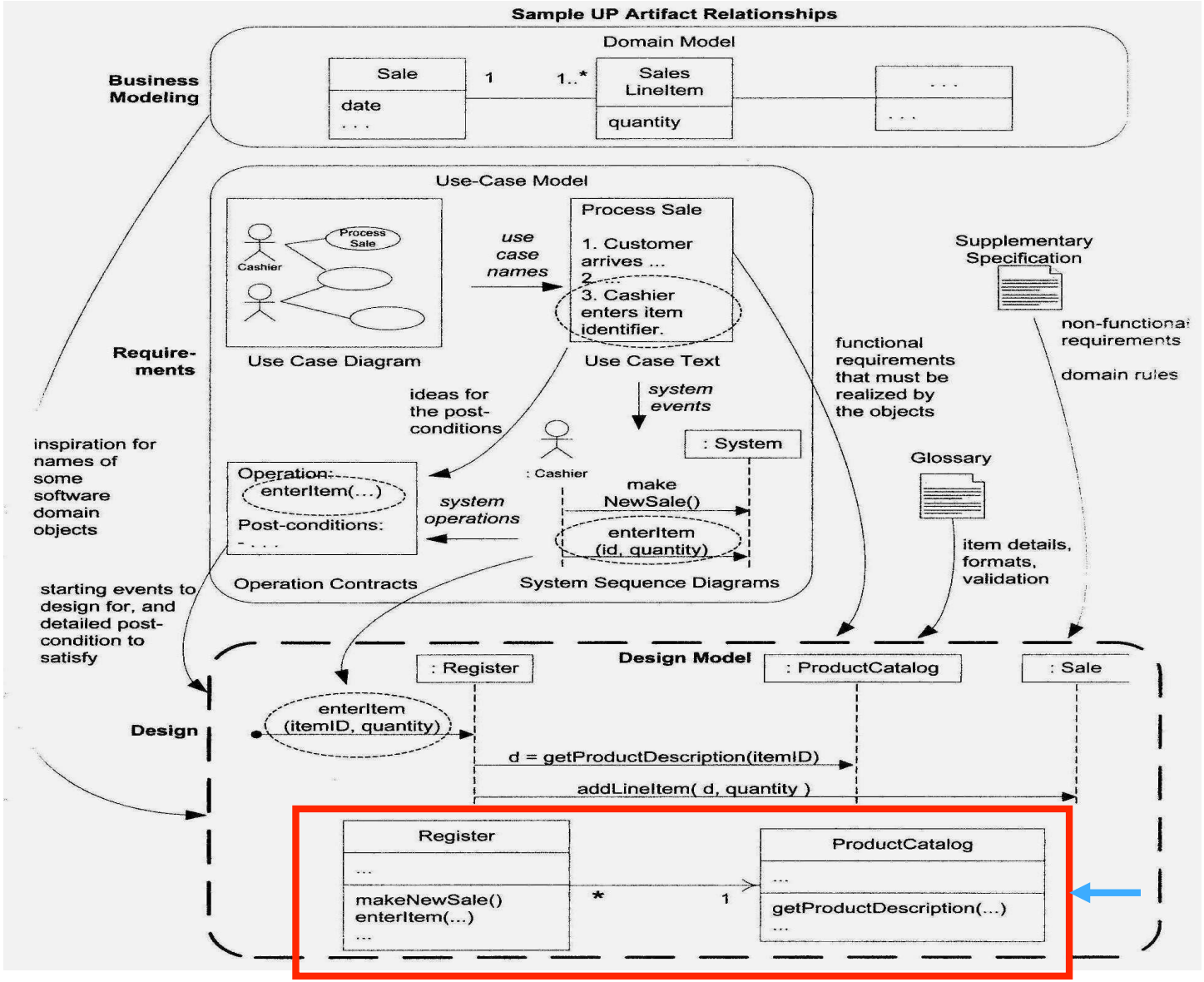
8.1. Funcionário não confirma produto
8.2. Sistema cancela reserva

Ponto de Extensão: seleção [funcionário escolhe criar novo cliente]

3.1. Extended by: Criar Cliente



PASSO SEGUINTE: Modelação Arquitectural



Numa abordagem OO, é natural que o Mega-Objecto **:System** seja em seguida dividido e estruturado em objectos menores com responsabilidades particulares inicialmente não clarificadas mas que agora temos que "desenhar" com maior detalhe.

Diagramas de Classe e/ou Packages

Modelação Comportamental

Sumário

- Diagramas de Sequência ao nível do Sistema (DSS)
- Refinamento do DSS – subsistemas e sua relevância
- Decomposição em camadas
- Decomposição por entidades/responsabilidades
- Necessidade da modelação estrutural