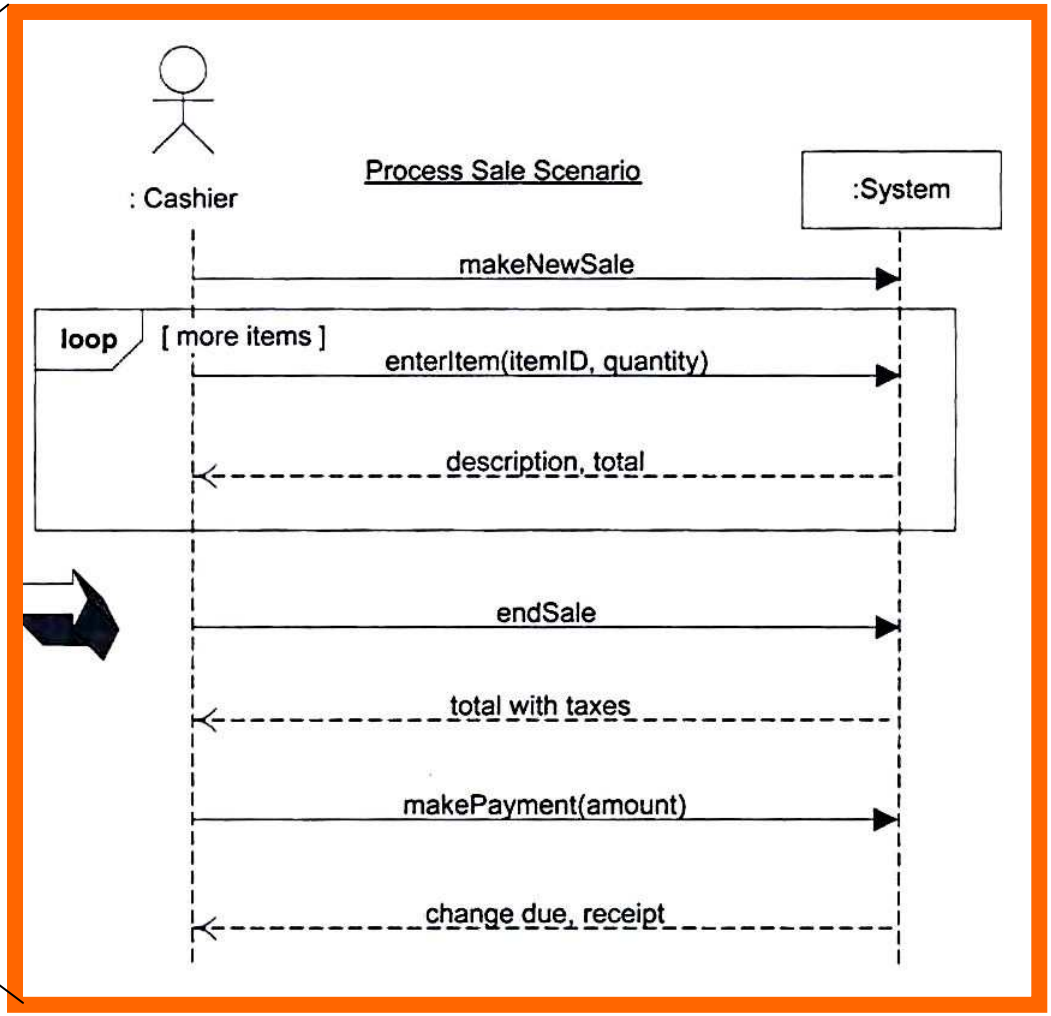
DIAGRAMAS DE SEQUÊNCIA

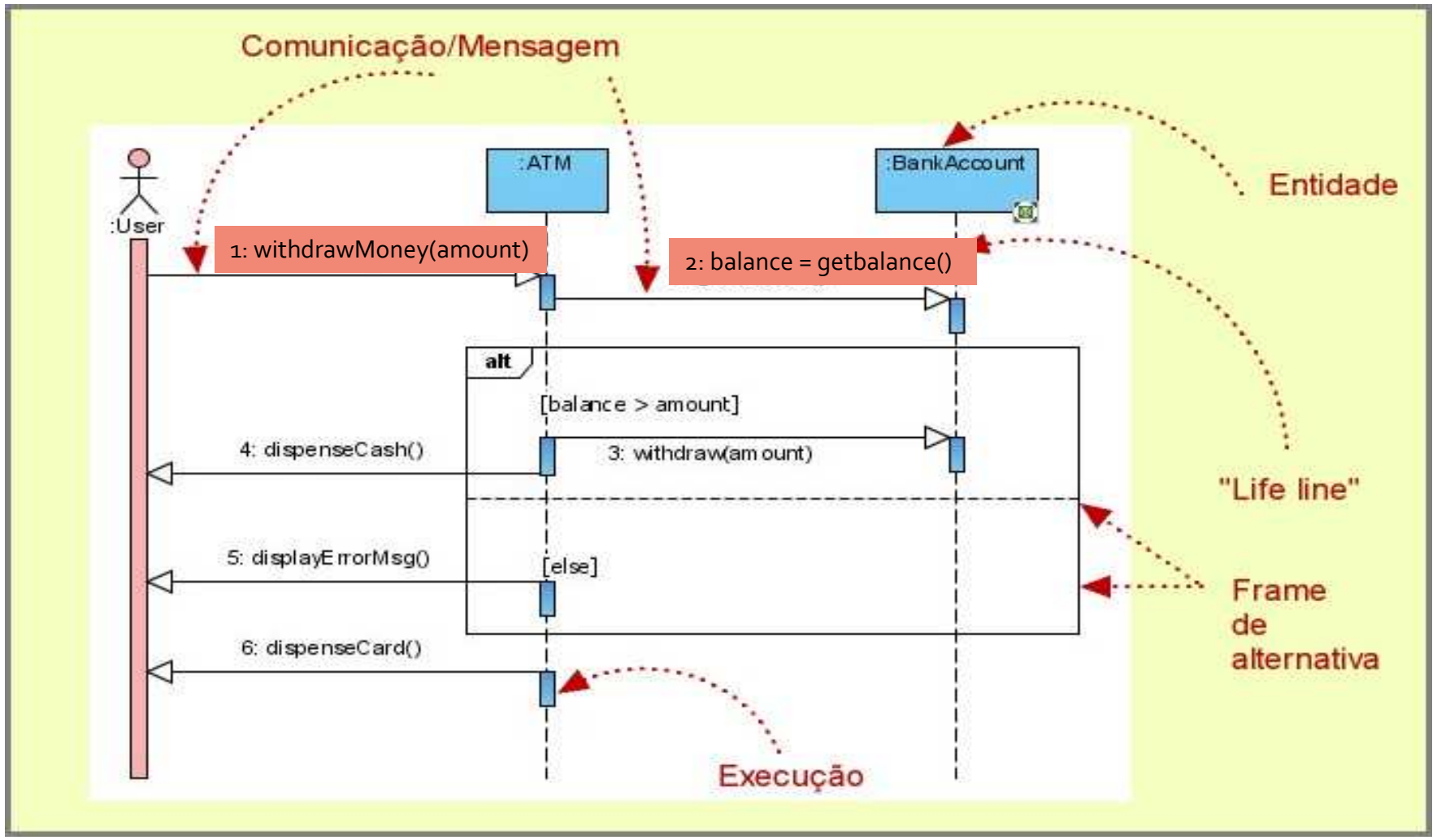
Simple cash-only Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...



Extraem-se dos UCs

DIAGRAMAS DE SEQUÊNCIA



DIAGRAMAS DE SEQUÊNCIA

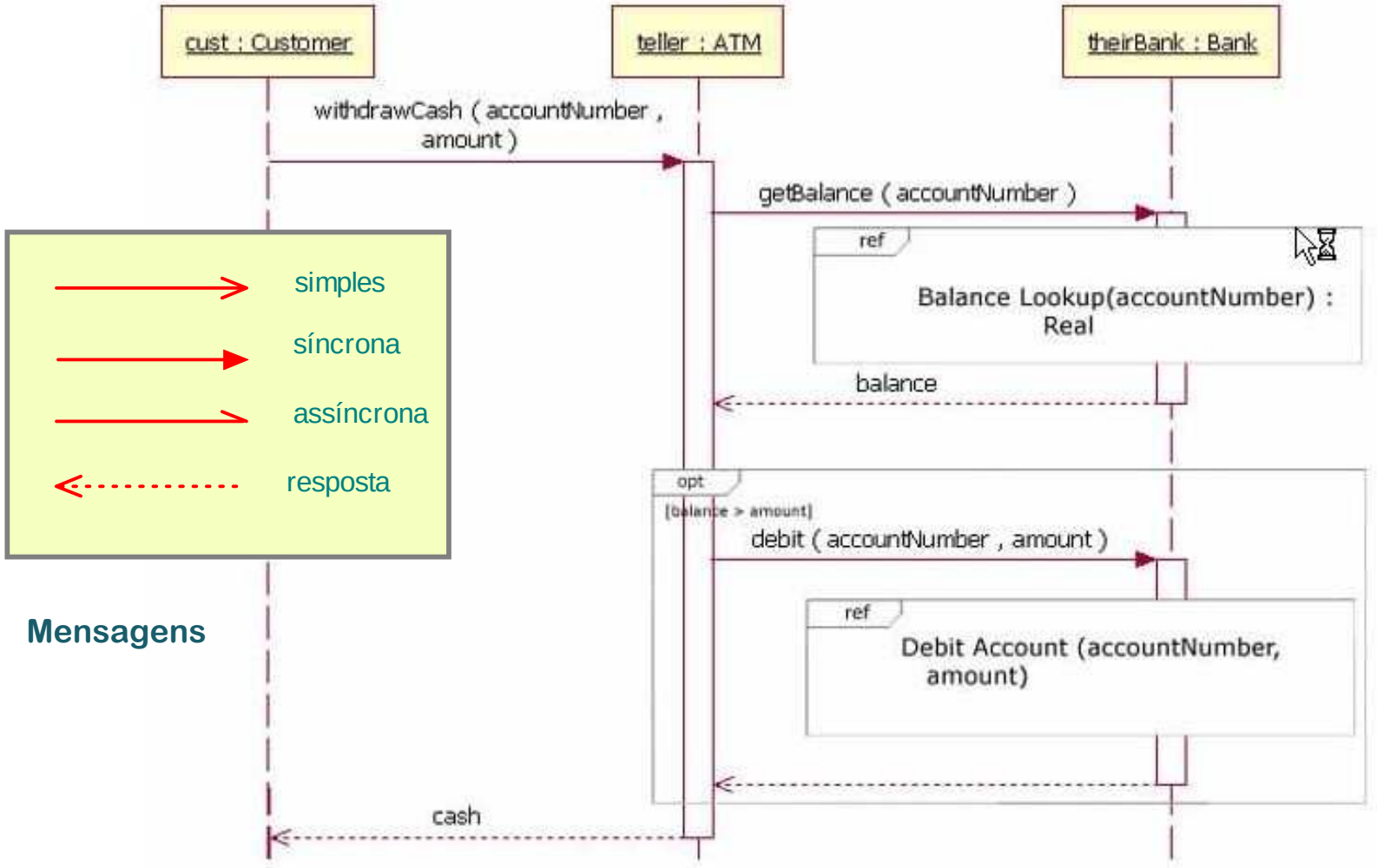
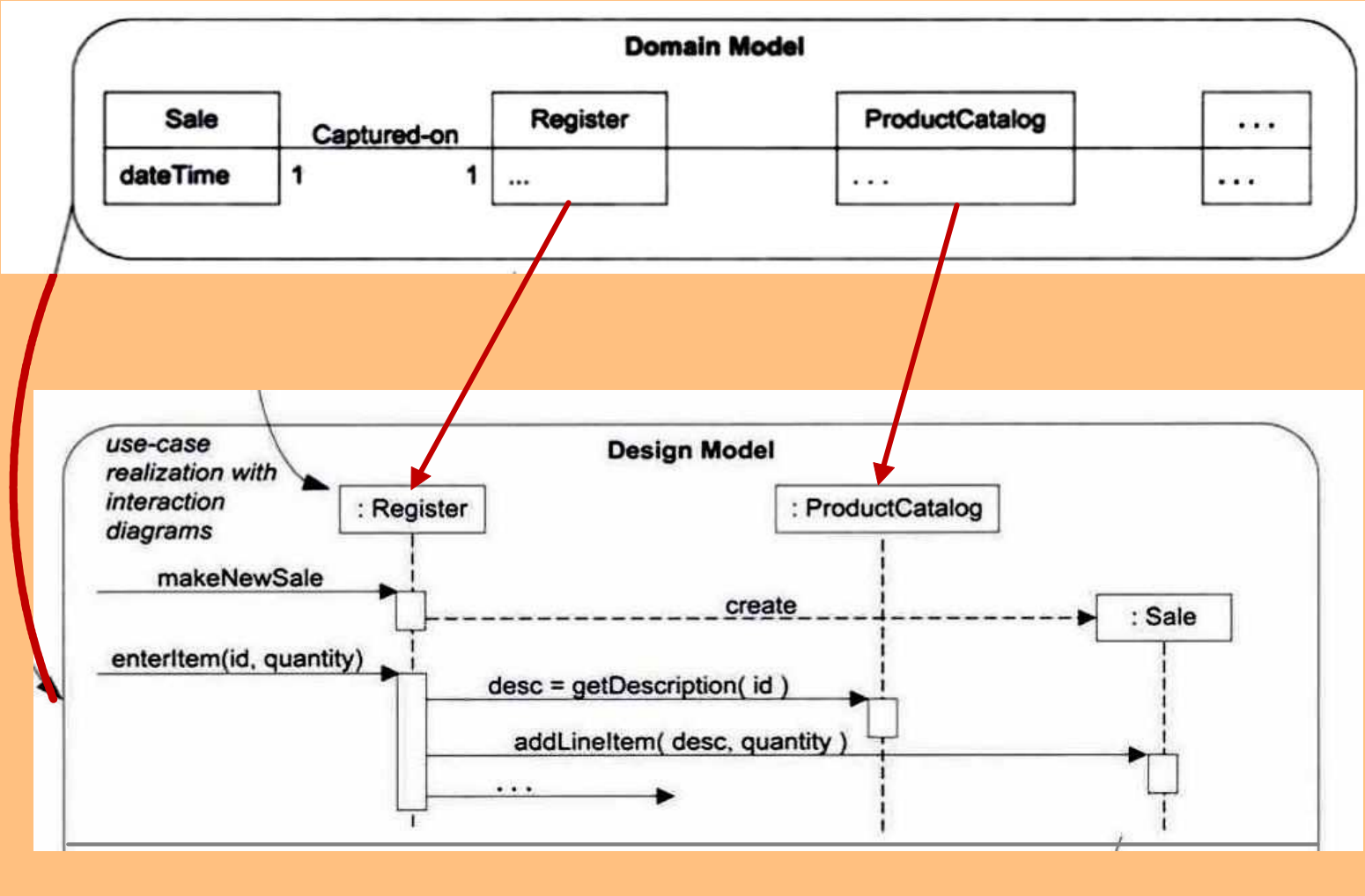


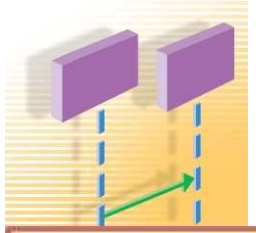
Diagrama de Sequência que referencia dois outros diagramas

DIAGRAMAS DE SEQUÊNCIA

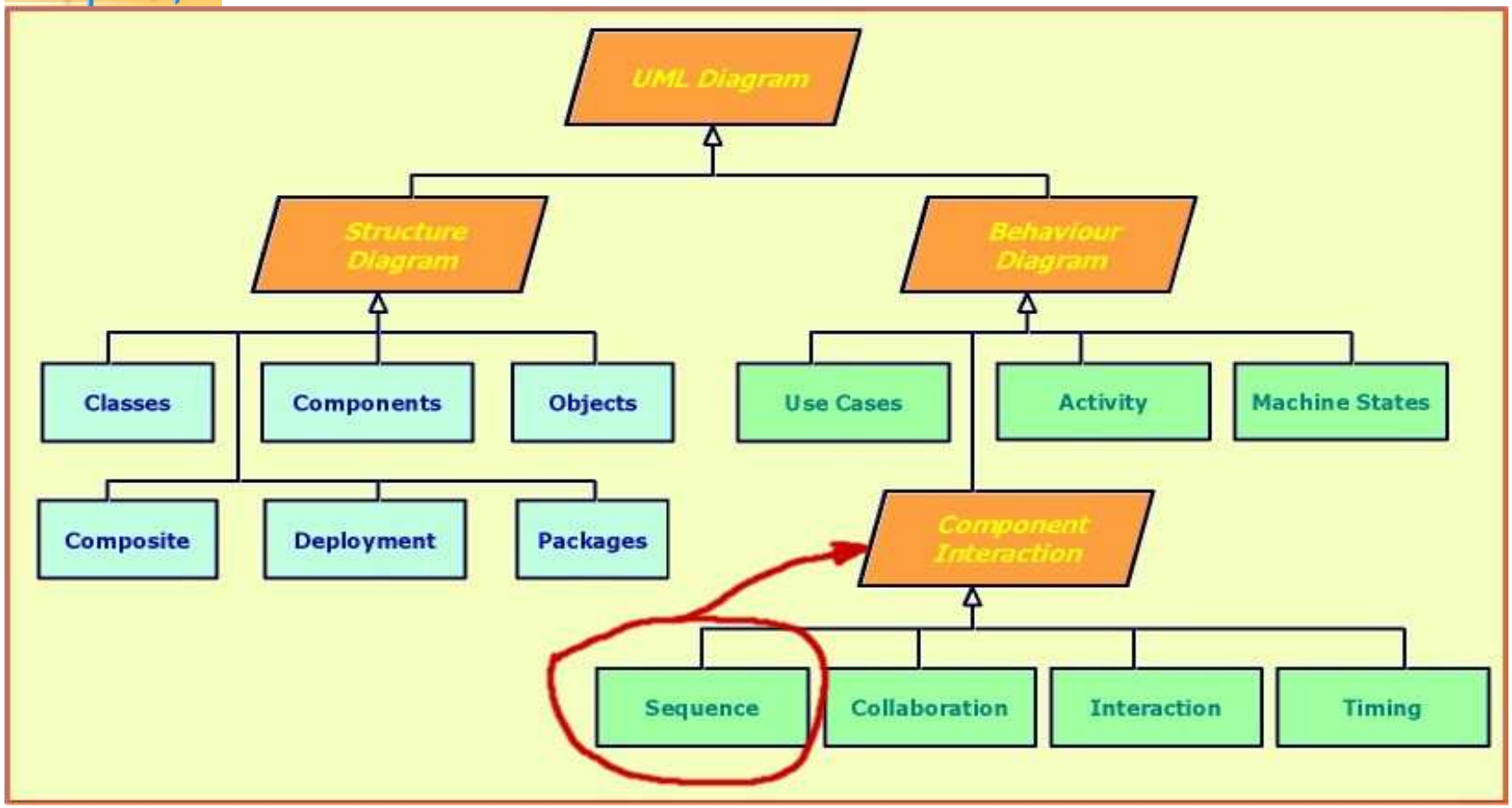


A coerência dos conceitos é sempre um aspecto importante

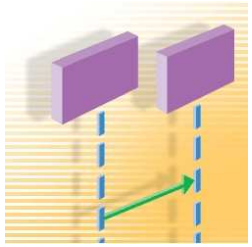
DIAGRAMAS DE SEQUÊNCIA



Vamos introduzir agora uma parte da notação UML que é destinada à criação de modelos de **comportamento** dos sistemas usando Diagramas de Sequência.



DIAGRAMAS DE SEQUÊNCIA

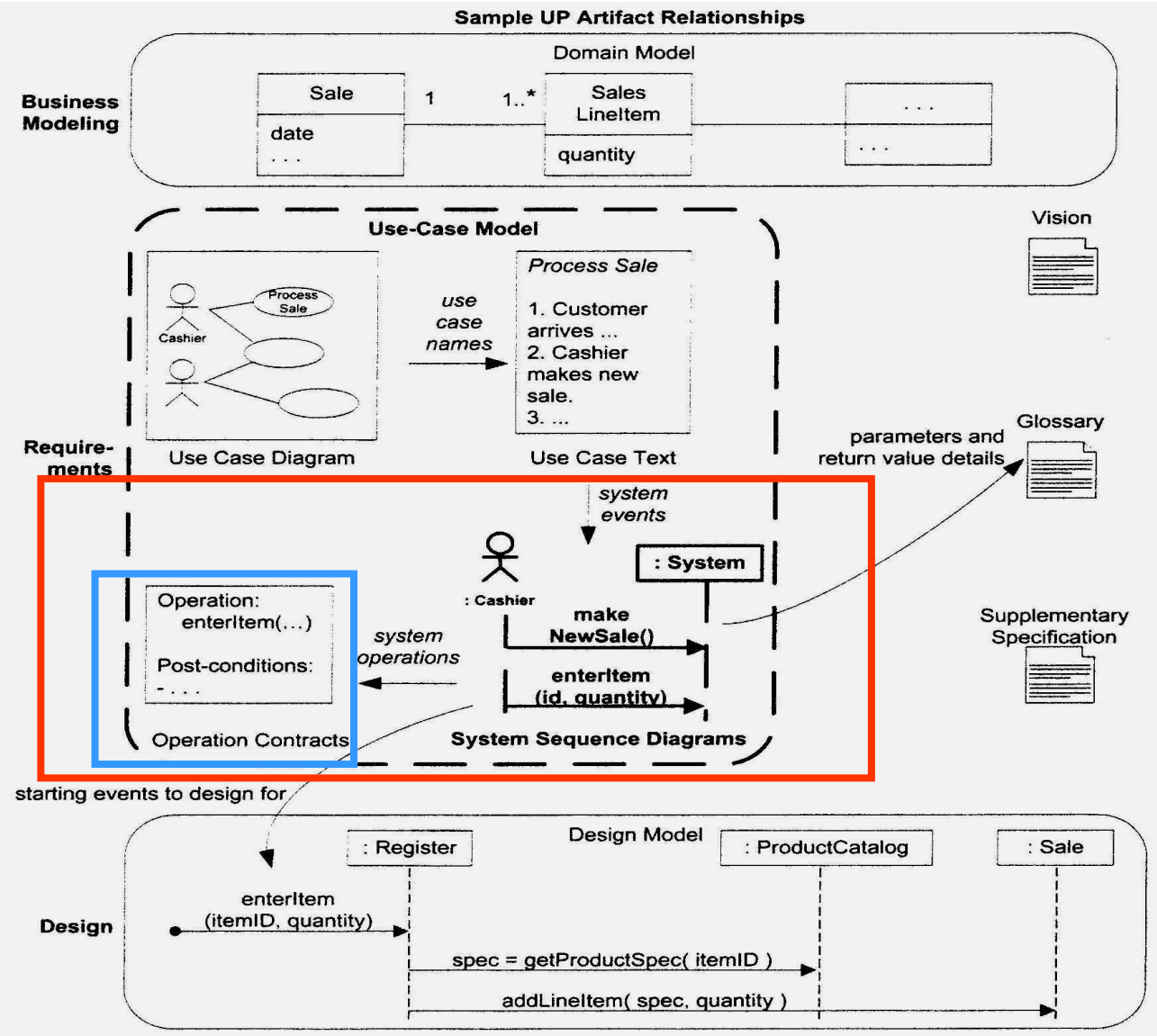


▣ **DIAGRAMAS DE INTERACÇÃO** são diagramas que visam em UML a modelação do comportamento de componentes dos sistemas, em especial oferecendo uma representação gráfica e para as interacções entre os objectos.

▣ **OS DIAGRAMAS DE SEQUÊNCIA**, em particular, representam as interacções entre objectos através das mensagens que são trocadas entre eles, especificando ainda qual o respectivo encadeamento temporal correcto (sequência temporal).

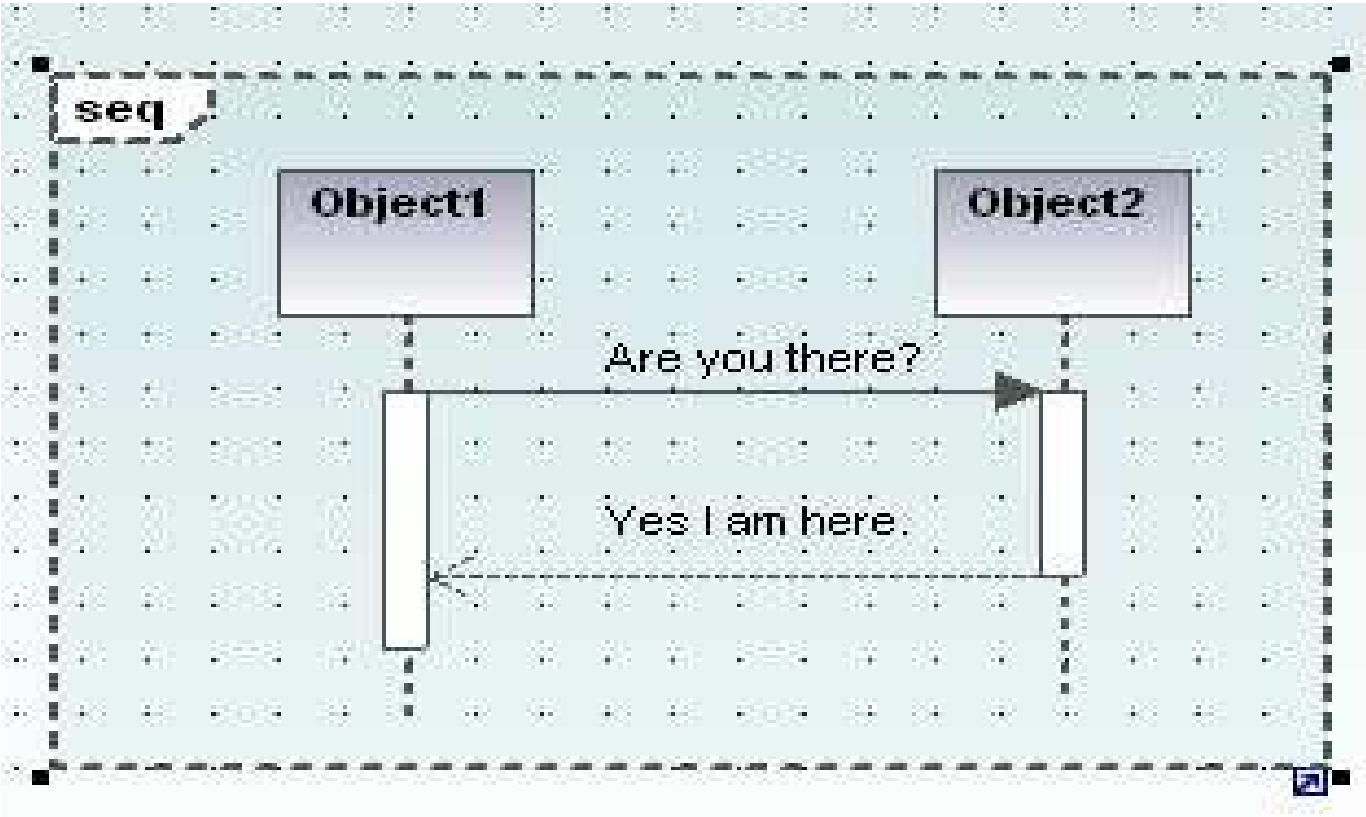
▣ **OS DIAGRAMAS DE SEQUÊNCIA** de mais alto nível, são diagramas da fase de análise e não de implementação, representando o sistema como uma "black-box", ou seja, sem indicar ainda sub-sistemas, representando os eventos que os actores geram, e as respostas comportamentais do sistema (operações do sistema). Por isso, designam-se, por vezes, DSS (Diagramas de Sequência de Sistema).

DIAGRAMAS DE SEQUÊNCIA



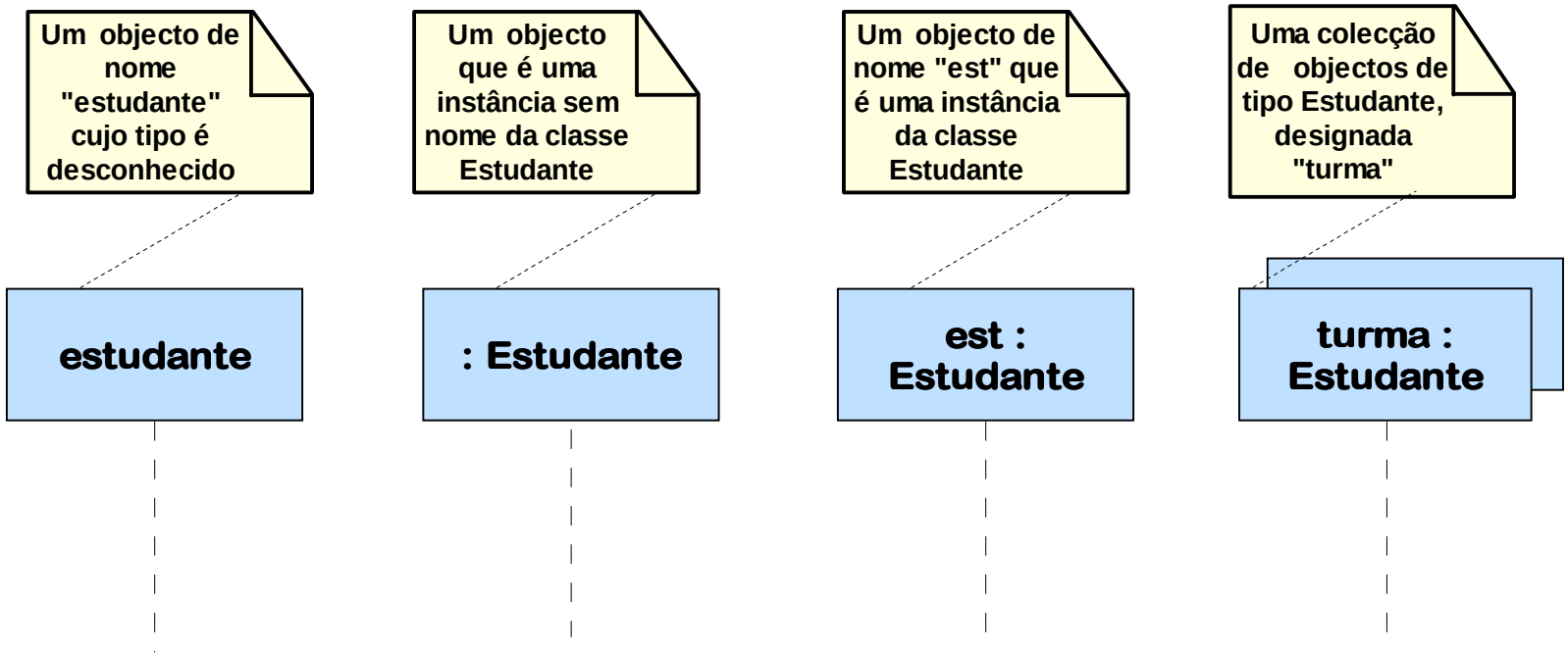
A partir da especificação textual dos UC, os passos dos Actores são vistos como eventos externos que iniciam operações do sistema.

O "Hello World" dos Diagramas de Sequência …



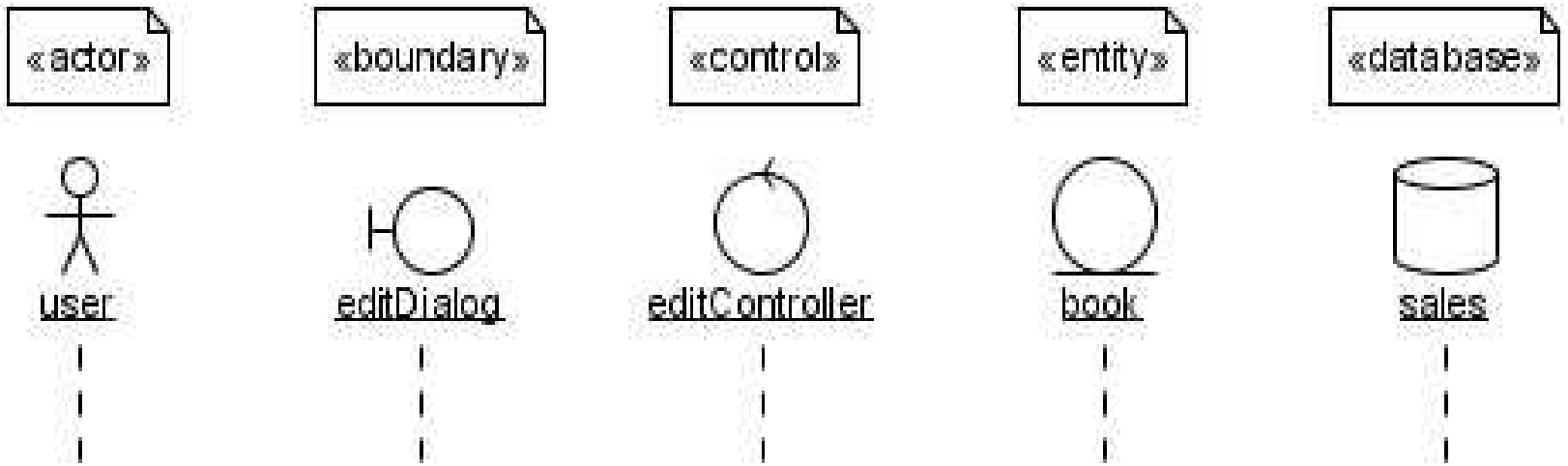
Enfim, 2 objectos "falam entre si" através de mensagens e respostas …

Outras formas de representação de objectos



DS: NOTAÇÃO ESSENCIAL

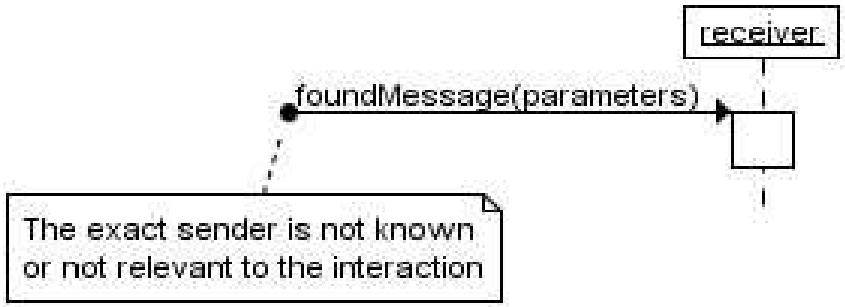
- Outras formas de representação (classificação) de objectos são baseadas no uso de estereótipos de UML e/ou ícones especiais, alguns deles já associados à fase de implementação.



Veremos mais tarde, na fase de concepção e implementação, a sua importância e interesse na identificação de entidades do Sistema Software.

As mensagens têm semântica muito própria.

☑ Mensagens de origem incógnita:



☑ Mensagens assíncronas:

O objecto que envia a mensagem não espera pelo resultado e continua o seu comportamento

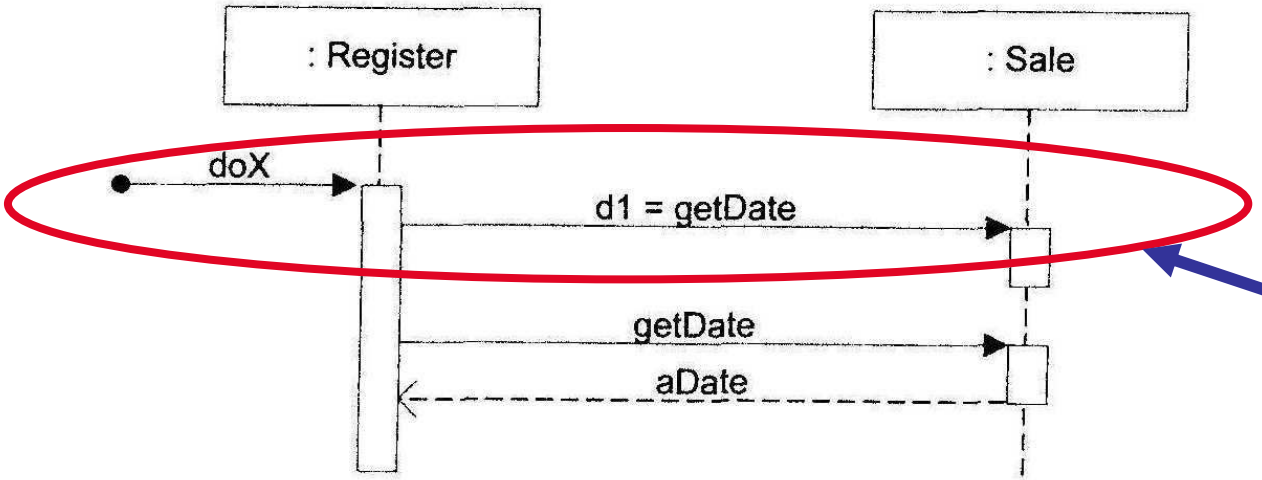
☑ Mensagens síncronas (as usuais):

O objecto que envia a mensagem e espera pelo resultado

There are two ways to show the return result from a message:

1. Using the message syntax *returnVar = message(parameter)*.
2. Using a reply (or return) message line at the end of an activation bar.

Both are common in practice. I prefer the first approach when sketching, as it's less effort. If the reply line is used, the line is normally labelled with an arbitrary description of the returning value. See Figure 15.8.



Usaremos este formato

© Applying UML and Patterns, Craig Larman, 3td Ed., Prentice Hall, 2007

DS: NOTAÇÃO (FRAMES)

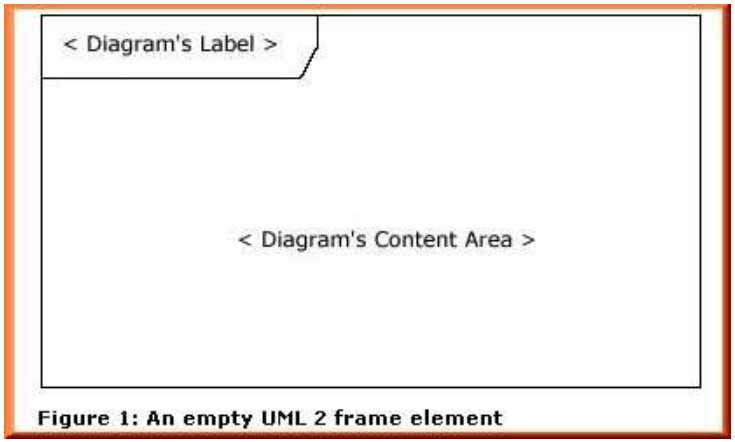


Figure 1: An empty UML 2 frame element

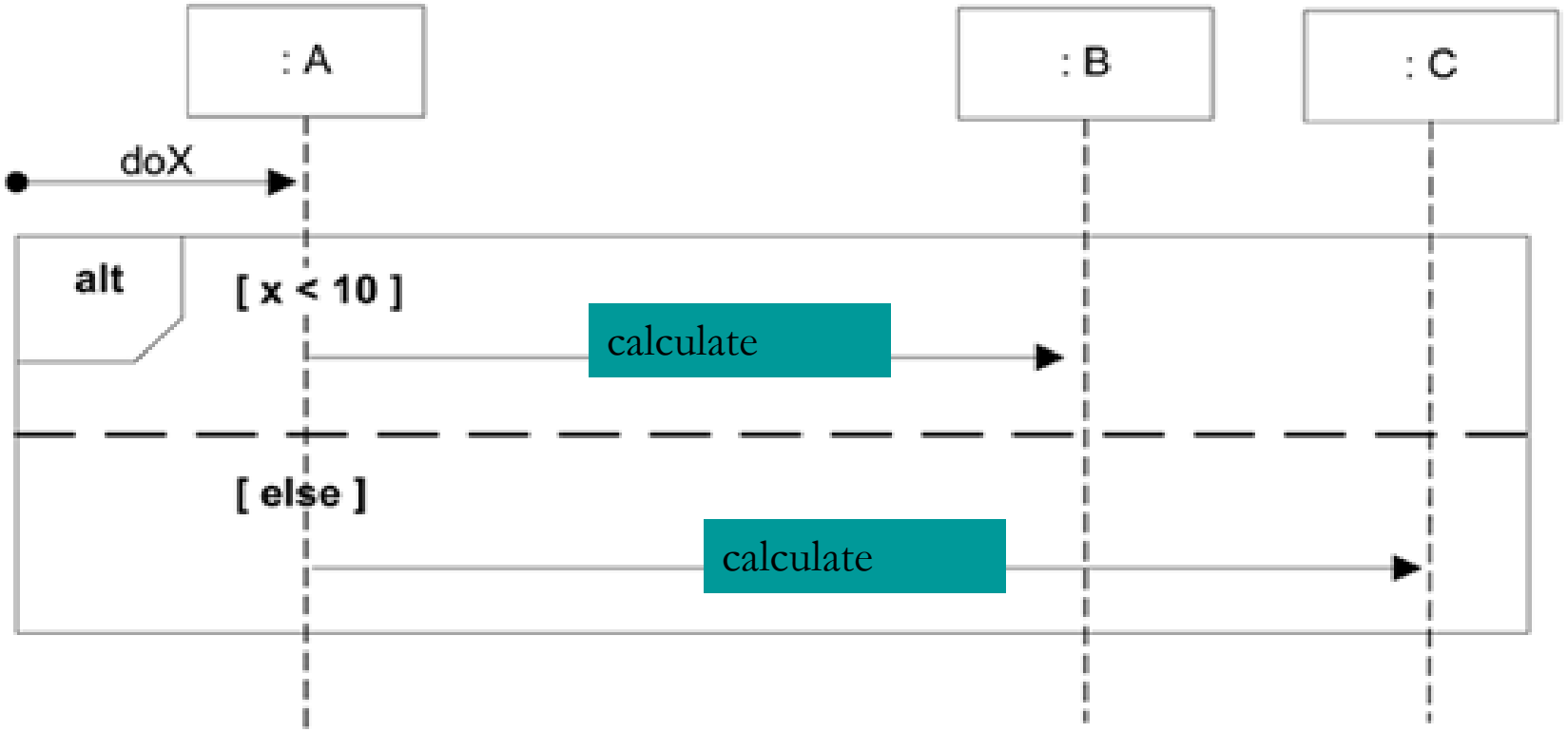
DIAGRAM FRAMES de UML, são regiões ou fragmentos dos diagramas. Servem para assinalar fragmentos condicionais, loops, regiões críticas ou até outros diagramas.

As condições chamam-se **guardas**.

Frames que são usados com muita frequência em DS, como instrumentos de modularidade e estruturação

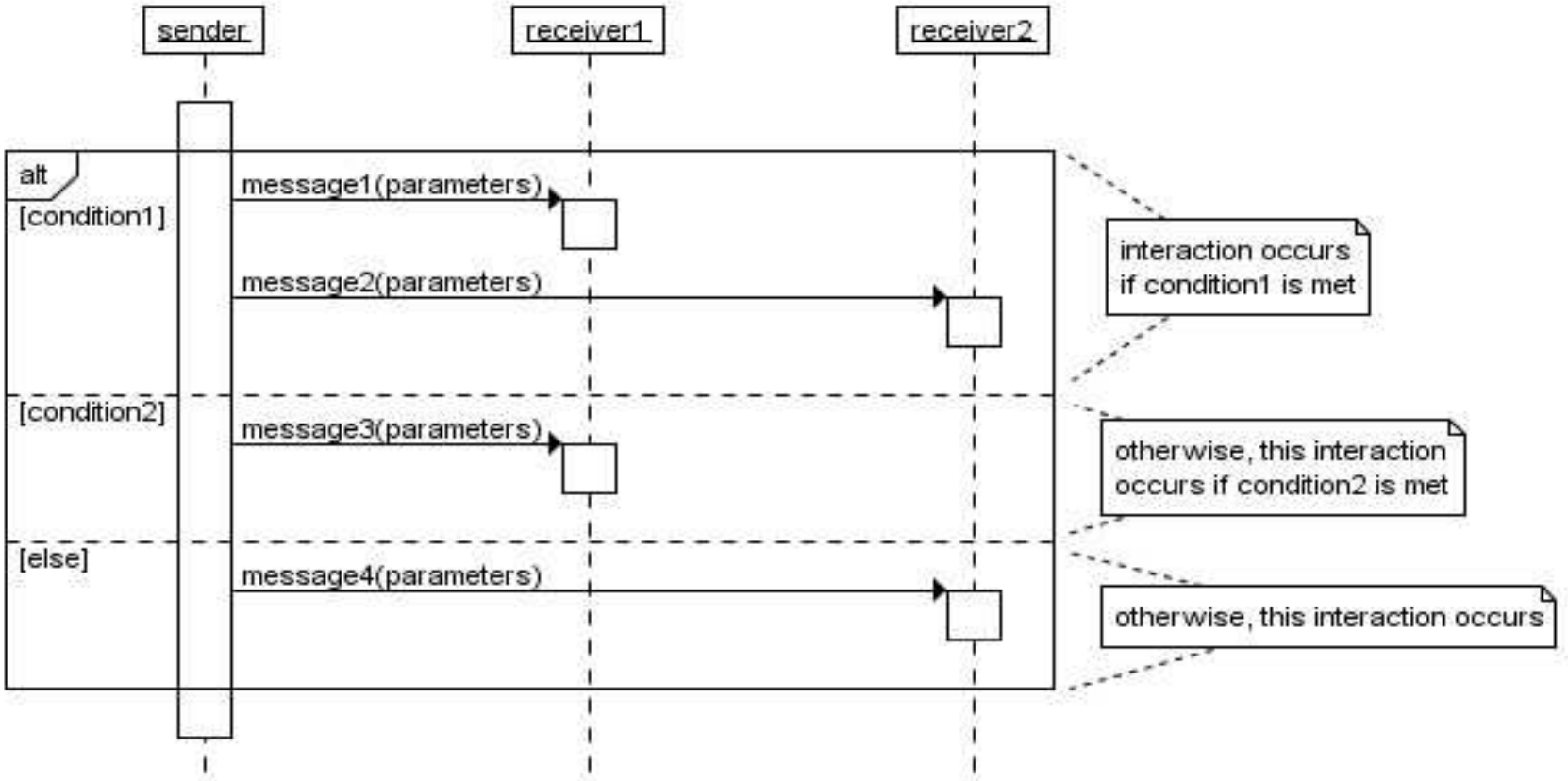
Frame	Semântica
alt	Define dois fragmentos mutuamente exclusivos cuja condição lógica de exclusão (ou não) é expressa numa guarda escrita entre [...]
loop loop(n)	Fragmento cíclico que vai ser repetido enquanto a guarda for verdadeira.
opt	Fragmento opcional que é executado apenas se a guarda for verdadeira
par	Fragmentos que são executados em paralelo
region	Região crítica de execução (não partilhável)
ref	Referencia a um outro diagrama (ou até método)

DS: FRAME alt (if-then)

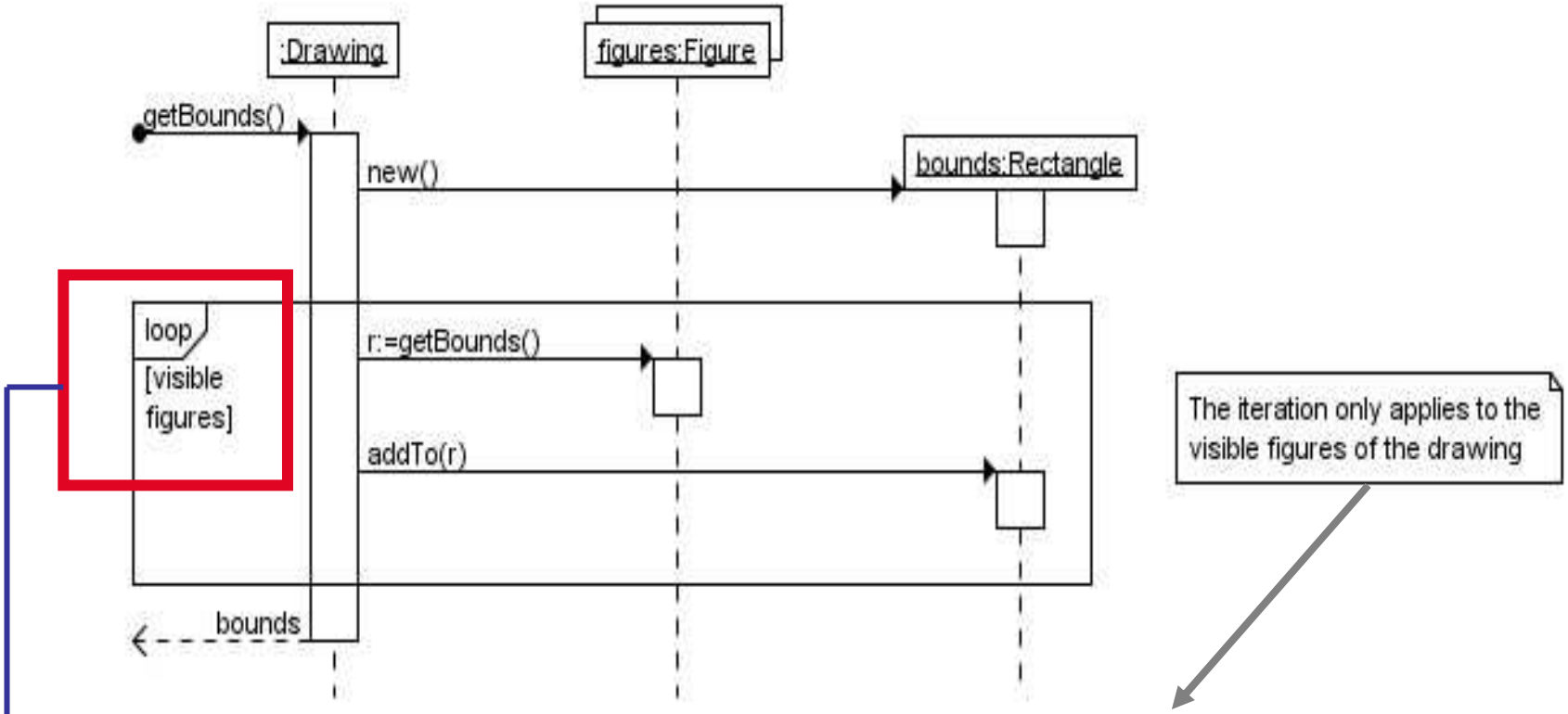


Os 2 fluxos possíveis (conforme o valor da guarda) são mutuamente exclusivos, pelo que apenas um deles será seguido.

DS: FRAME alt

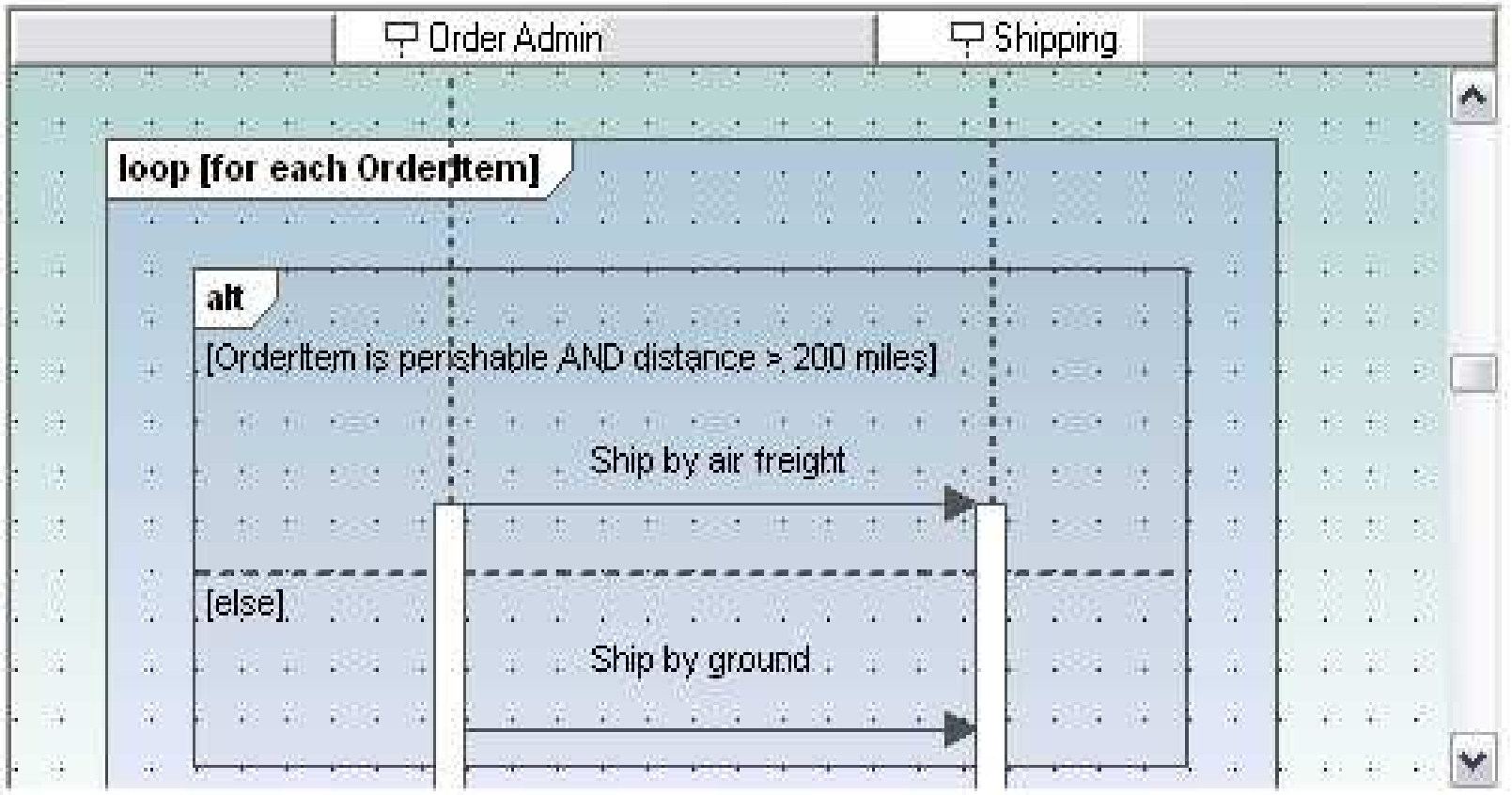

sender
receiver1
receiver2
alt
[condition1]
message1(parameters)
message2(parameters)
[condition2]
message3(parameters)
[else]
message4(parameters)
interaction occurs if condition1 is met
otherwise, this interaction occurs if condition2 is met
otherwise, this interaction occurs

DS: FRAME loop



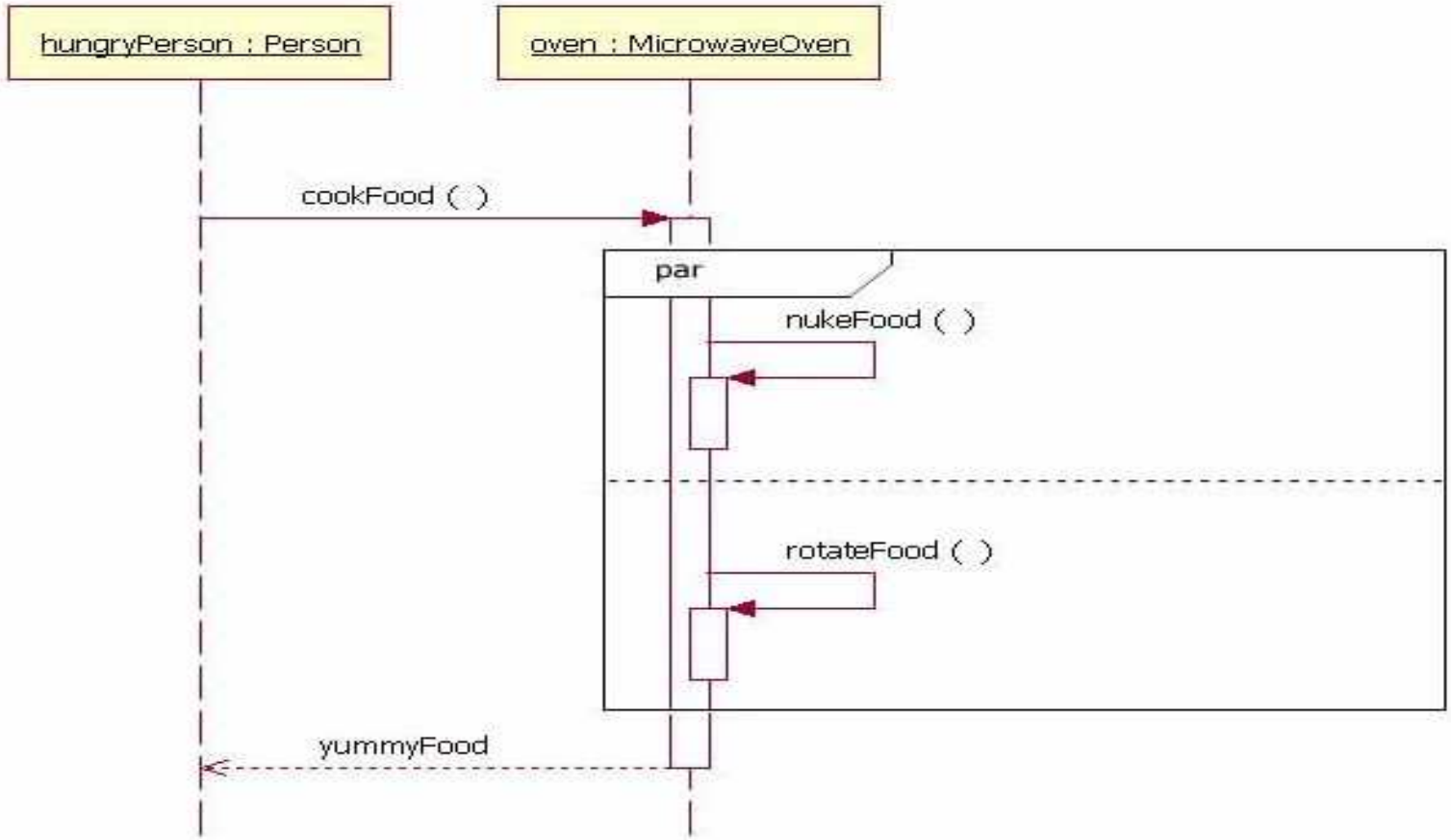
Ler: Para cada uma das figuras do conjunto de figuras que seja visível, determinar os seus limites (cf. r) e adicionar tal rectângulo ao rectângulo que está a ser calculado (limites totais do desenho).

DS: FRAME loop



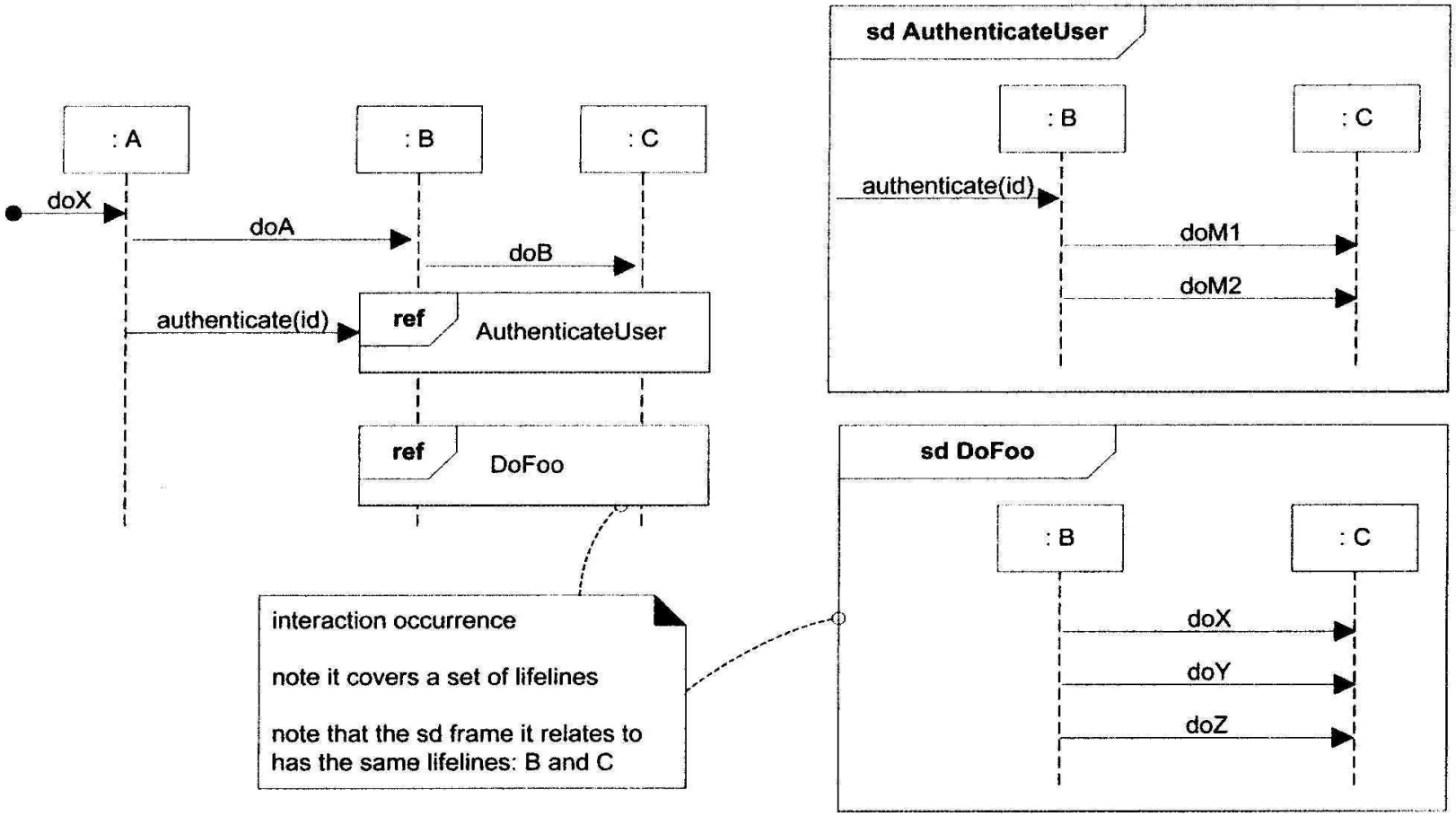
Para cada produto da Encomenda, decidir se é enviado por via aérea ou terrestre.

DS: FRAME par



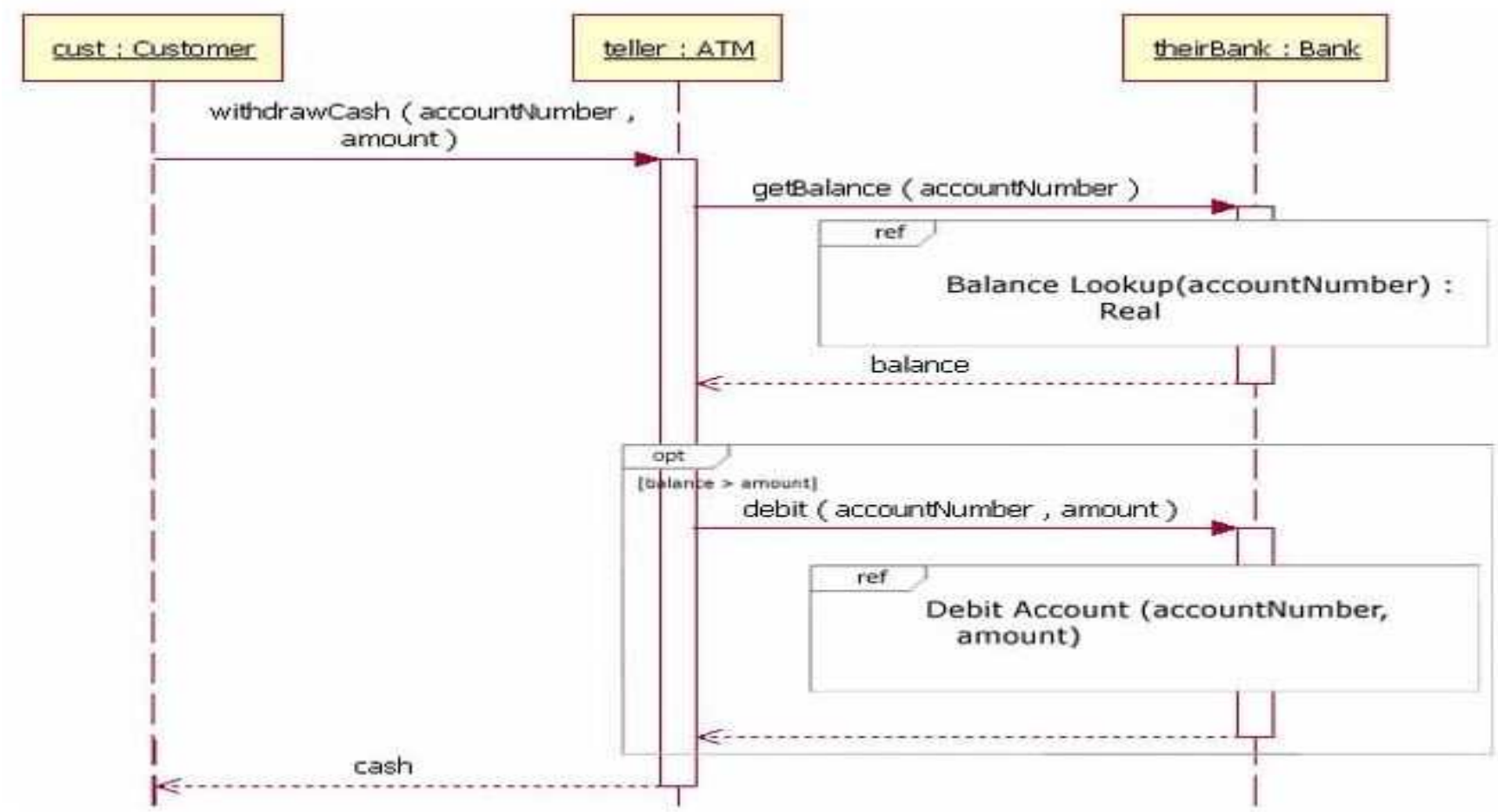
Uma pessoa esfomeada envia a um micro-ondas uma mensagem para cozinhar uma refeição. O micro-ondas envia a si próprio duas mensagens, uma para "bombardear" e outra para "rodar" a comida, tarefas que são realizadas em paralelo. Quando ambas estiverem concluídas, a esfomeada pessoa recebe como resultado comida "nham-nham" (ie. deliciosa).

DS: MODULARIDADE

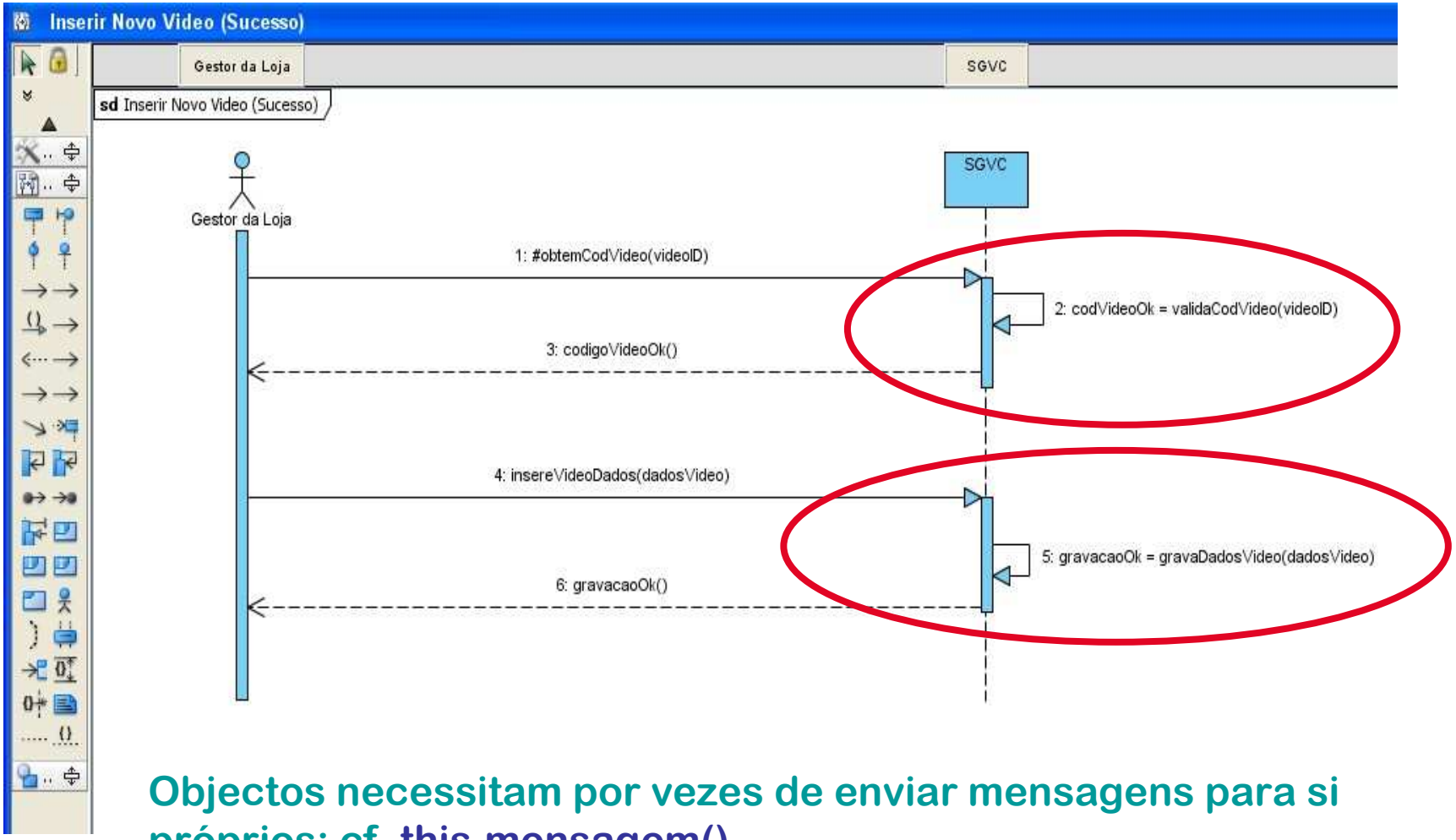


A estruturação de diagramas pode ser feita identificando diagramas de sequência usando sd nome e referenciando-os usando o frame ref.

DS: MODULARIDADE

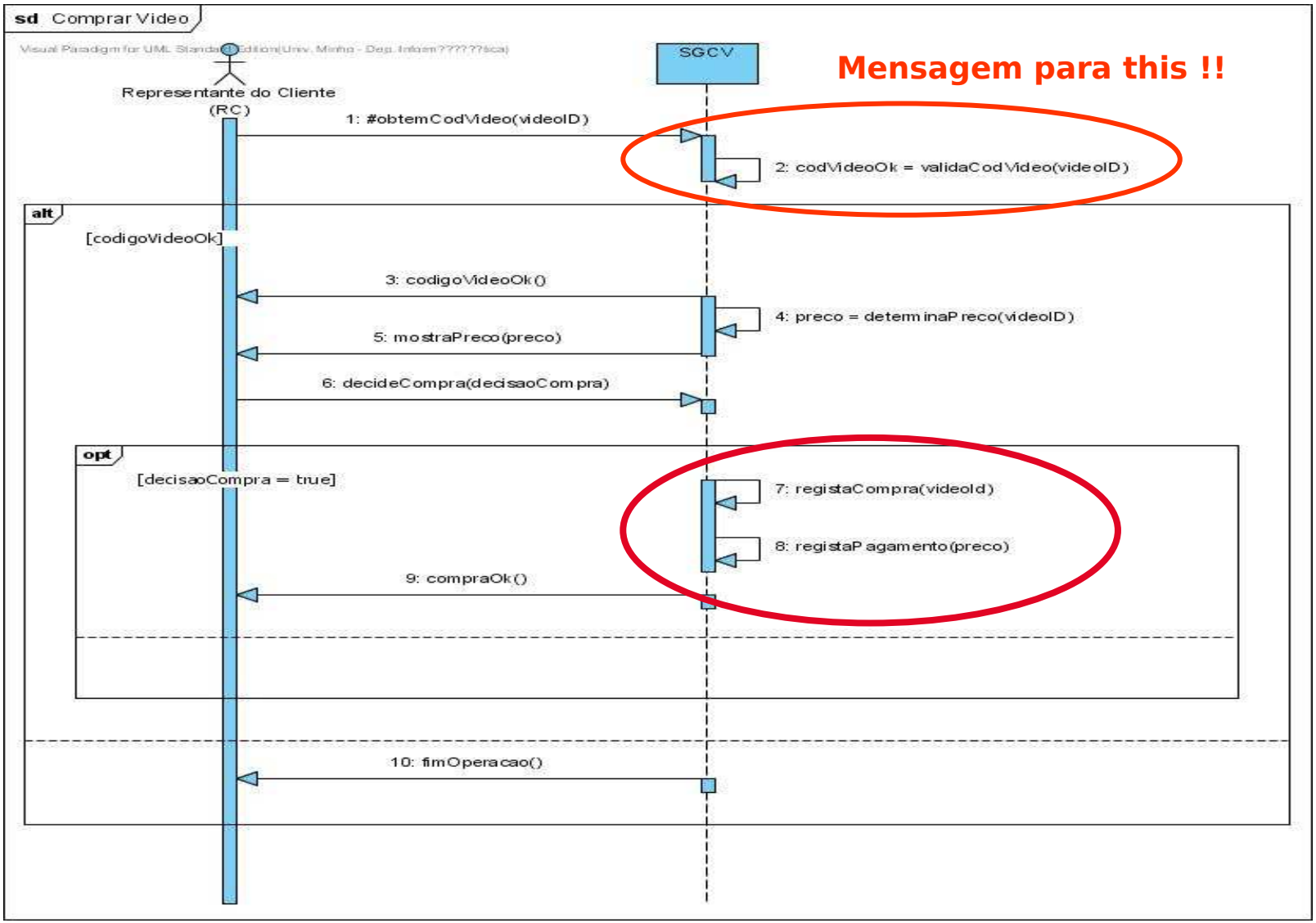


DS: EXEMPLOS

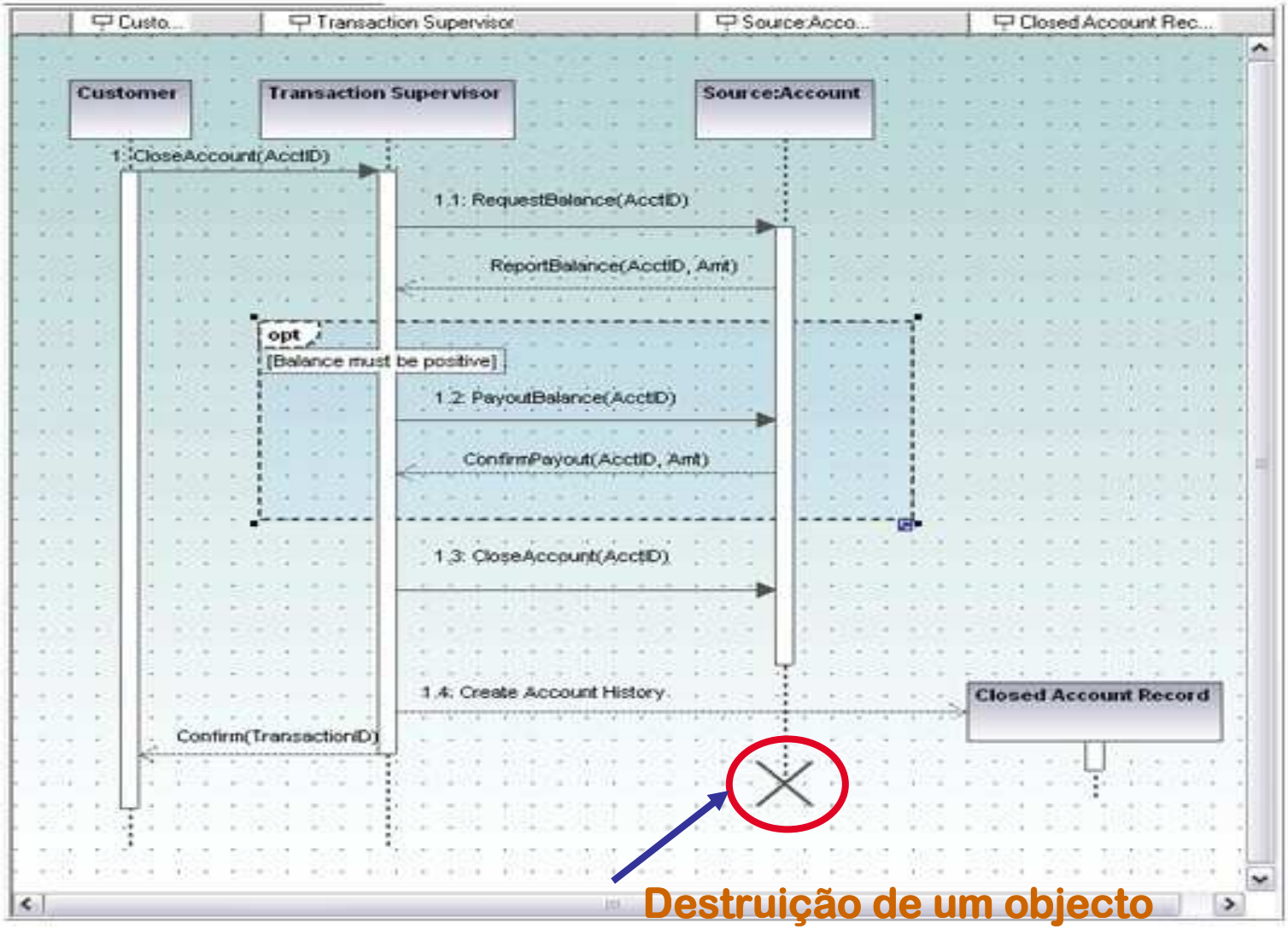


Objectos necessitam por vezes de enviar mensagens para si próprios: cf. this.mensagem()

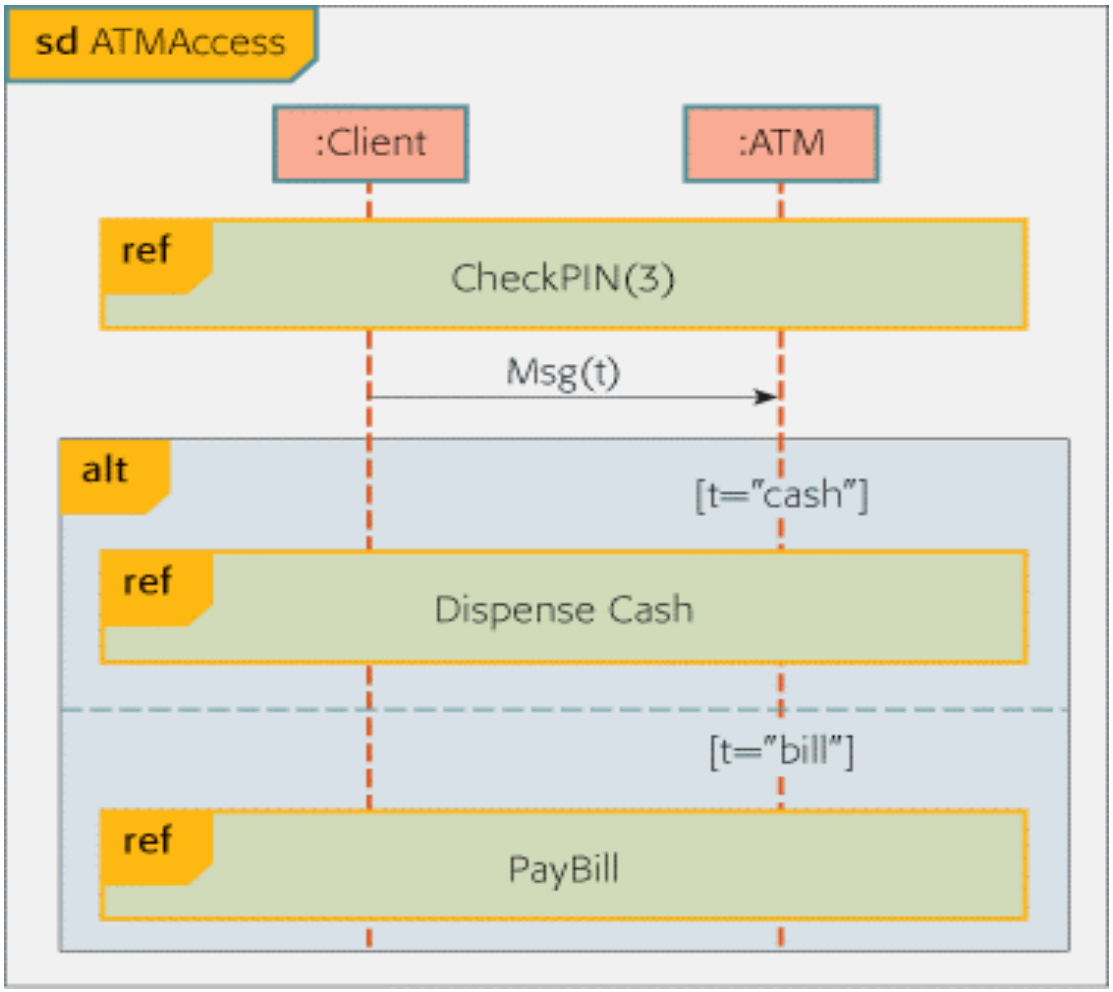
DS: EXEMPLOS


sd Comprar Video
Representante do Cliente (RC)
SGCV
Mensagem para this !!
1: #obtemCodVideo(videoID)
2: codVideoOk = validaCodVideo(videoID)
alt
[codigoVideoOk]
3: codigoVideoOk()
4: preco = determinaPreco(videoID)
5: mostraPreco(preco)
6: decideCompra(decisaoCompra)
opt
[decisaoCompra = true]
7: registaCompra(videoId)
8: registaPagamento(preco)
9: compraOk()
10: fimOperacao()

DS: EXEMPLOS


Custo...
Transaction Supervisor
Source:Acco...
Closed Account Rec...
Customer
Transaction Supervisor
Source:Account
1: CloseAccount(AcctID)
1.1: RequestBalance(AcctID)
ReportBalance(AcctID, Amt)
opt
[Balance must be positive]
1.2: PayoutBalance(AcctID)
ConfirmPayout(AcctID, Amt)
1.3: CloseAccount(AcctID)
1.4: Create Account History
Closed Account Record
Confirm(TransactionID)
Destruição de um objecto

DS: EXEMPLOS

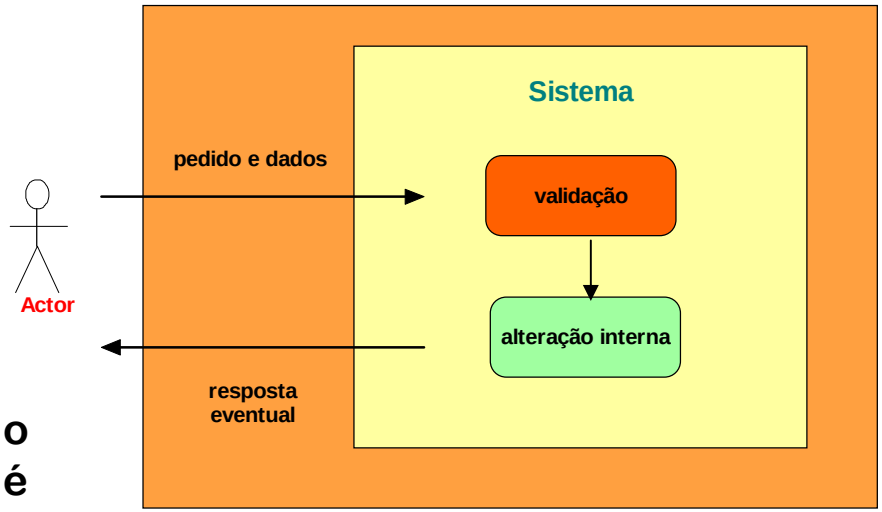


A possibilidade de termos diagramas de sequência com modularidade permite-nos agora gerir tal modularidade, ou seja, pensar em regras que nos digam "por onde" devemos "partir" estas sequências, isto é, com que critério ou método.

Necessitamos de ter um método sistemático de passar dos UC para os DS !

MÉTODO: UCs → DS

- Use Cases não são operações do sistema mas incluem operações do sistema;
- Jacobson diz que: cada use case constitui uma série completa de eventos iniciados por um actor e especifica a interacção que tem lugar entre o actor e o sistema. Um use case é, portanto, uma sequência especial de transacções relacionadas, executadas por um actor e pelo sistema em diálogo.



TRANSACÇÃO

(para o actor é atómica !)

Cada transacção destas pode muito bem ser analisada, individualizada e transposta de forma sistemática para o Diagrama de Sequência

MÉTODO: UCs → DS

- Aceites todas estas definições "standard" do que se deve entender ser um UC, o que se deve entender por **<<include>>** e por **<<extend>>**, como se devem escrever UC textuais, etc., torna-se agora importante passar dos diagramas de UC e das descrições textuais de Ucs, para especificações mais detalhadas do ponto de vista comportamental.

- **Problema:** Como passar sistematicamente, ou seja com método, dos UC para os Diagramas de Sequência que especificam, com mais detalhe, como o Sistema em desenvolvimento responde às mais diversas solicitações dos actores, tal como anteriormente identificado no Diagrama de Use Cases ? **Resposta:** Não há qualquer metodologia, quer no RUP, quer noutro processo qualquer OO para o fazer.

- **Vamos assim introduzir um método inovador**, não documentado ainda em livros nem artigos, desenvolvido visando uma coerente e correcta resolução desta fase de transição do RUP, de forma a que, seguindo certas regras, se atinjam certos objectivos de forma coerente e, portanto, sendo os passos reversivelmente identificáveis.

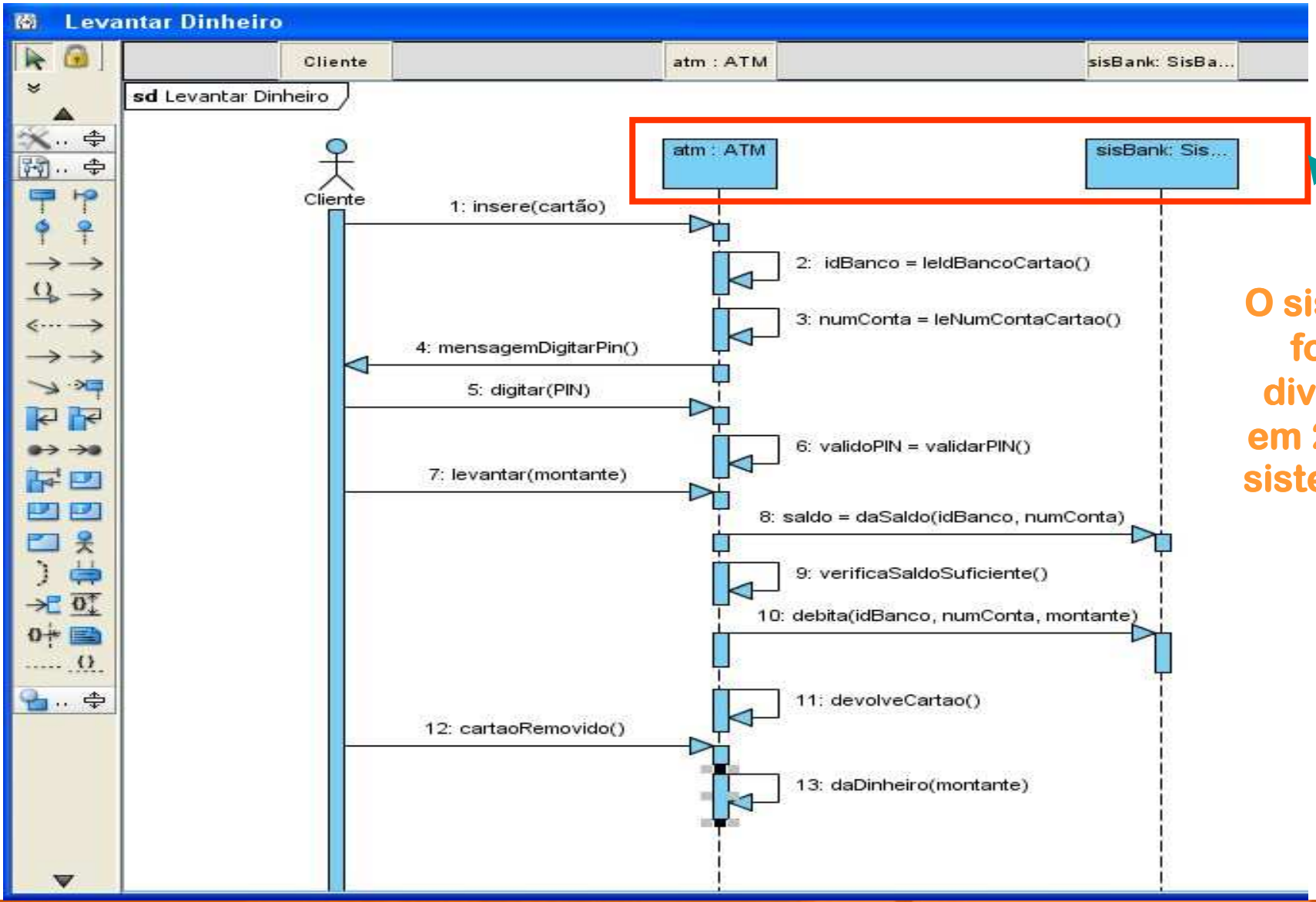
MÉTODO: UCs → DS

PASSO 1 : Identificar as transacções no UC (por exemplo com cores)

Use Case: Levantar Dinheiro			
Cenário de Sucesso			
	Actor	Sistema	
1	Cliente identifica-se introduzindo o seu cartão		T1
2		Lê do cartão identificação do Banco e número da Conta	
3		Sistema pede o PIN ao Cliente	
4	Cliente introduz o PIN		T2
5		Sistema verifica se o PIN é válido	
6	O Cliente solicita o levantamento de uma dada importância		
7		Sistema verifica se a conta tem saldo suficiente	T3
8		Sistema subtrai a importância que vai disponibilizar ao saldo da Conta	
9		Sistema devolve cartão	
10	Cliente retira Cartão		
11		Sistema disponibiliza a importância em notas	T4

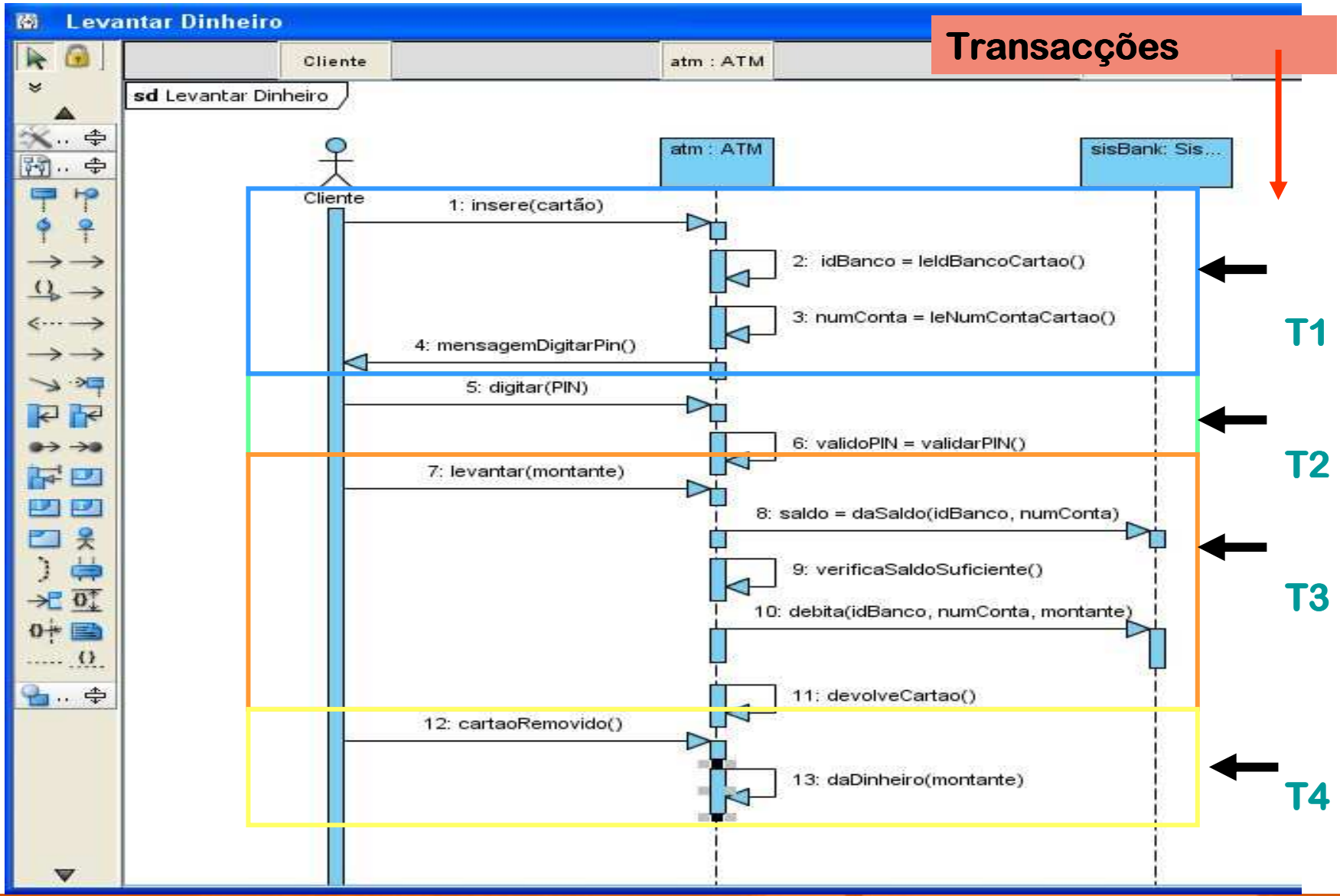
Algumas são do tipo Acção -> Reacção (ie. possuem apenas 2 passos)

MÉTODO: UCs → DS

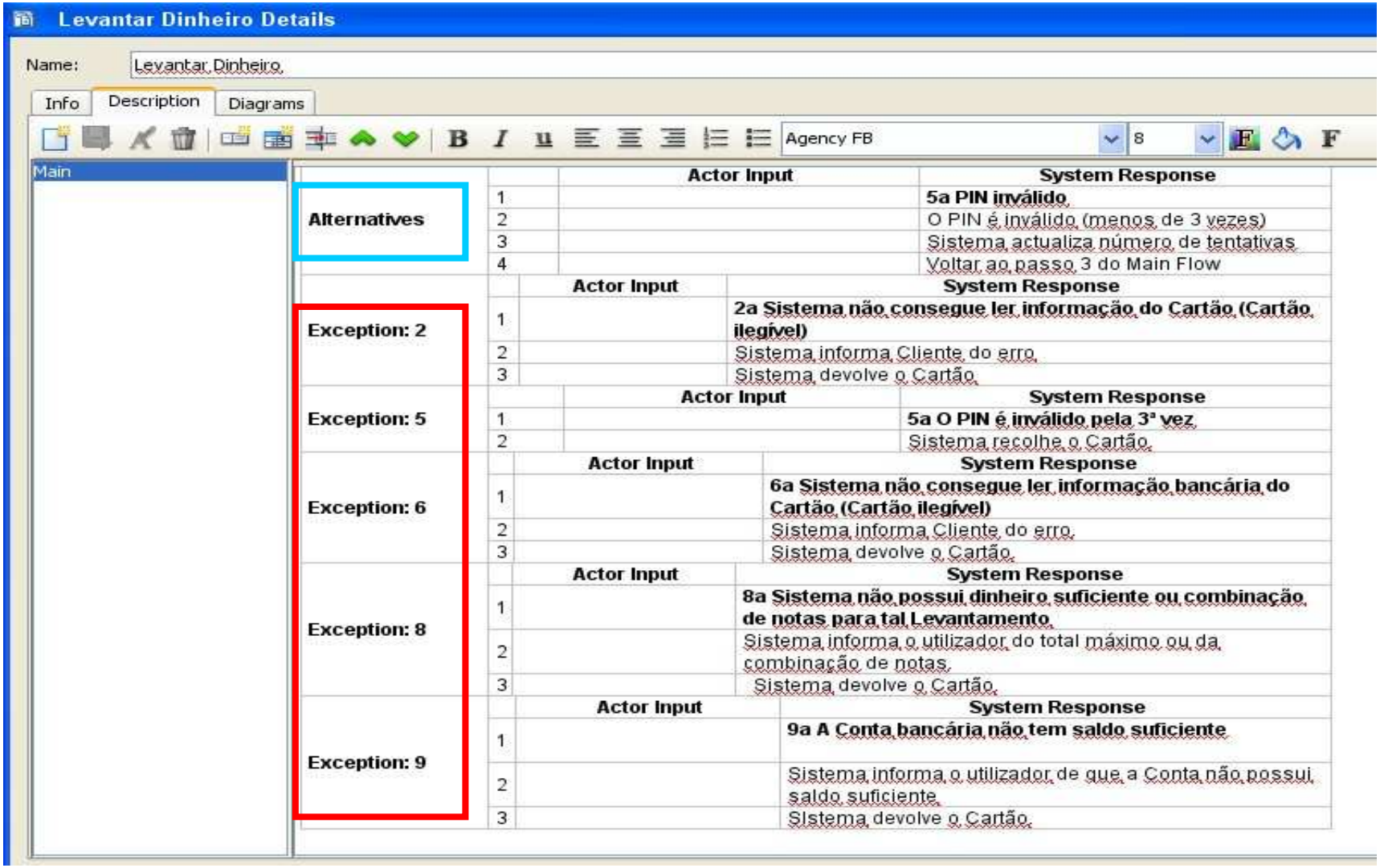


O sistema foi já dividido em 2 sub-sistemas !

MÉTODO: UCs → DS



LEVANTAMENTO DE UMA ATM: Alternativas e Excepções. Como fazer ?



		Actor Input	System Response
Alternatives	1		**5a PIN inválido**
	2		O PIN é inválido (menos de 3 vezes)
	3		Sistema actualiza número de tentativas
	4		Voltar ao passo 3 do Main Flow

		Actor Input	System Response
Exception: 2	1		**2a Sistema não consegue ler informação do Cartão (Cartão ilegível)**
	2		Sistema informa Cliente do erro
	3		Sistema devolve o Cartão

		Actor Input	System Response
Exception: 5	1		**5a O PIN é inválido pela 3ª vez**
	2		Sistema recolhe o Cartão

		Actor Input	System Response
Exception: 6	1		**6a Sistema não consegue ler informação bancária do Cartão (Cartão ilegível)**
	2		Sistema informa Cliente do erro
	3		Sistema devolve o Cartão

		Actor Input	System Response
Exception: 8	1		**8a Sistema não possui dinheiro suficiente ou combinação de notas para tal Levantamento**
	2		Sistema informa o utilizador do total máximo ou da combinação de notas
	3		Sistema devolve o Cartão

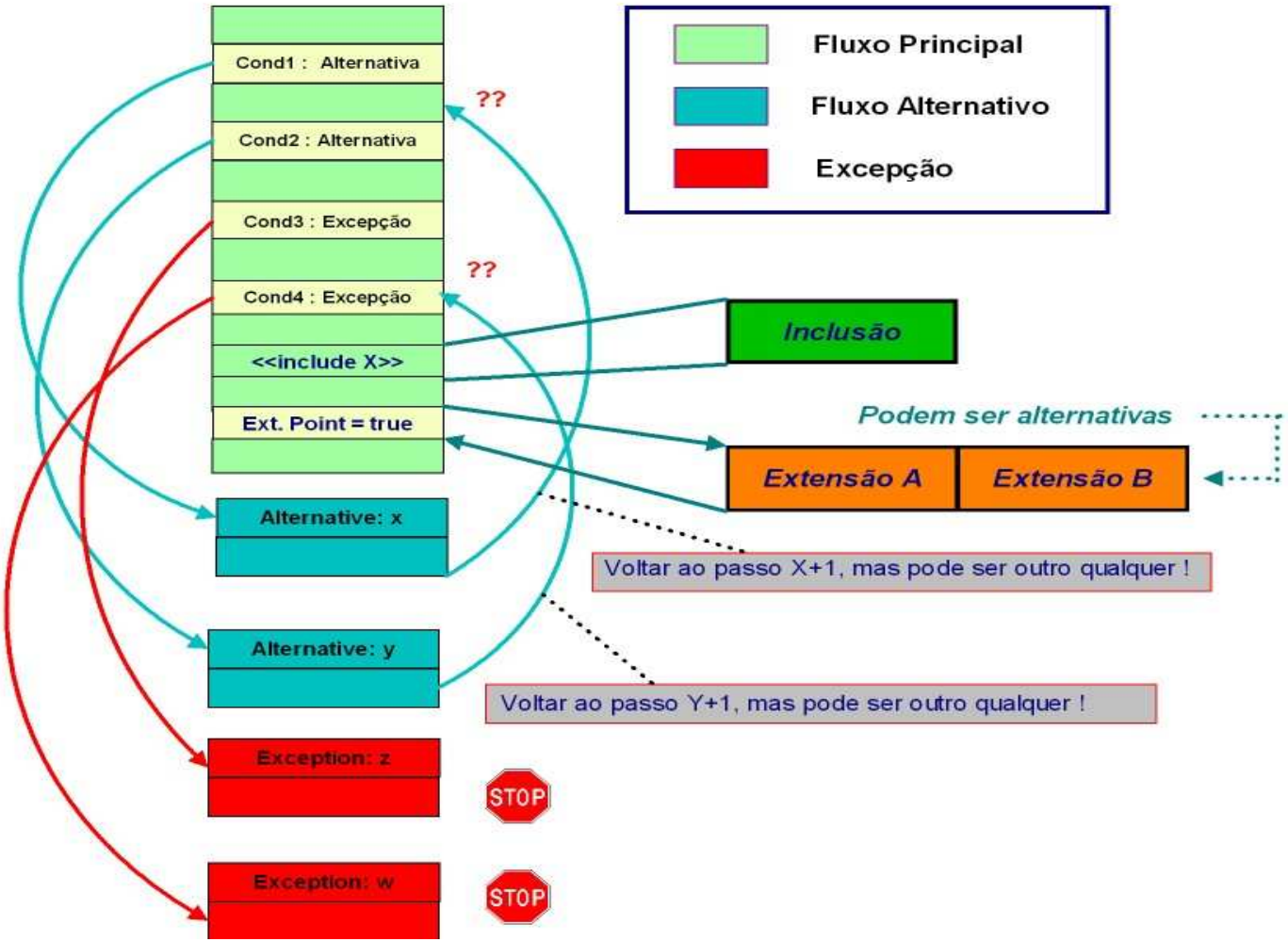
		Actor Input	System Response
Exception: 9	1		**9a A Conta bancária não tem saldo suficiente**
	2		Sistema informa o utilizador de que a Conta não possui saldo suficiente
	3		Sistema devolve o Cartão

MÉTODO: UCs → DS

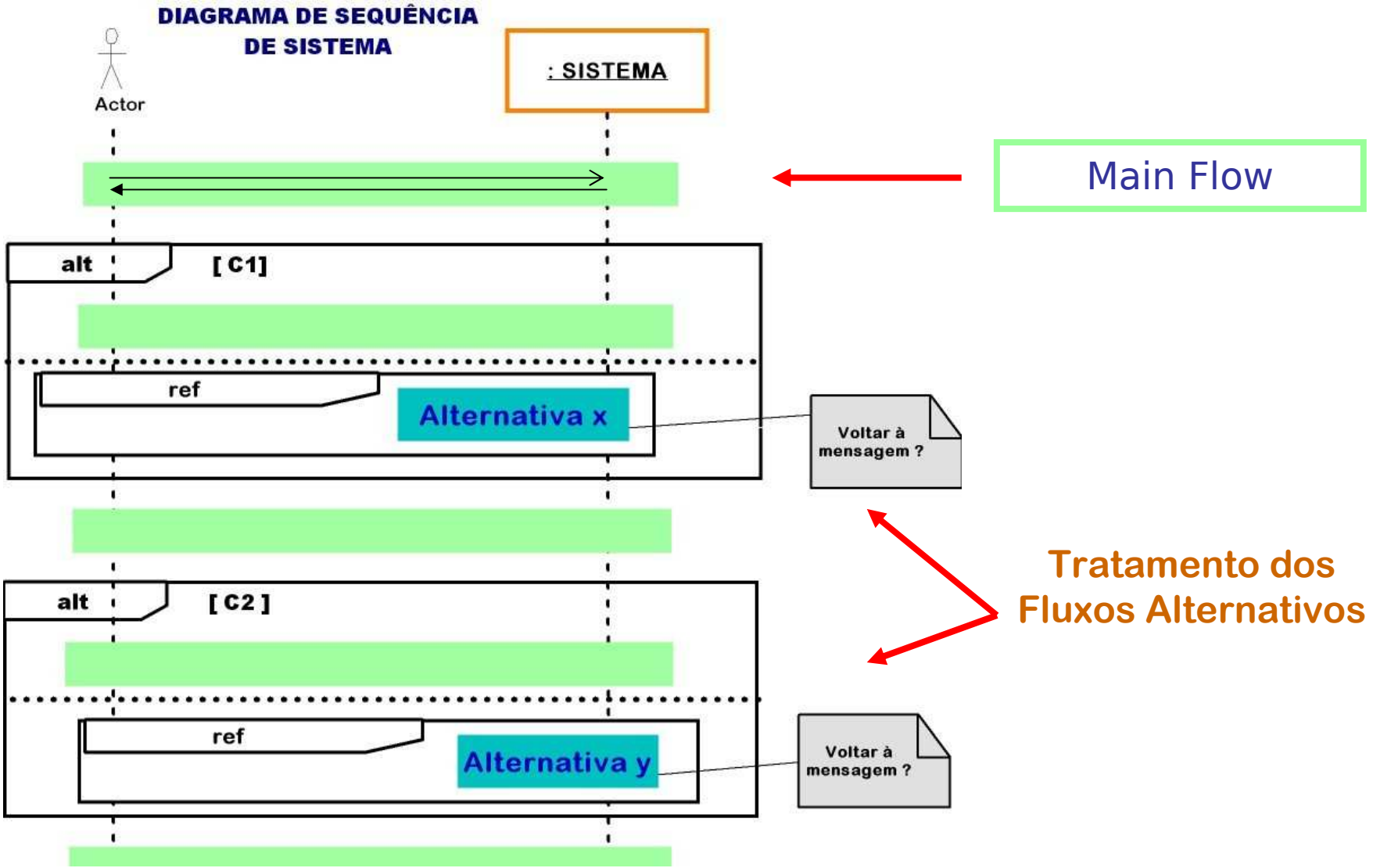
- Para o **Main Flow** (Cenário de Sucesso) sabemos já identificar as transacções definidas no UC textual e passar cada uma delas para o Diagrama de Sequência
- Precisamos agora de ter uma foma sistemática de representar no Diagrama de Sequência os outros dois tipos de fluxo que podemos ter num UC: **Alternativas** e **Excepções**.
- Precisamos ainda de saber tratar sistematicamente as situações em que o UC **<<include>>** outros Ucs ou **"é estendido"** por outros Ucs.
- Se criarmos um **"template"** geral do que pode ser um UC textual, talvez tal nos ajude a identificar todos os casos possíveis de transformação de um UC num DS.

TEMPLATE GERAL UCs

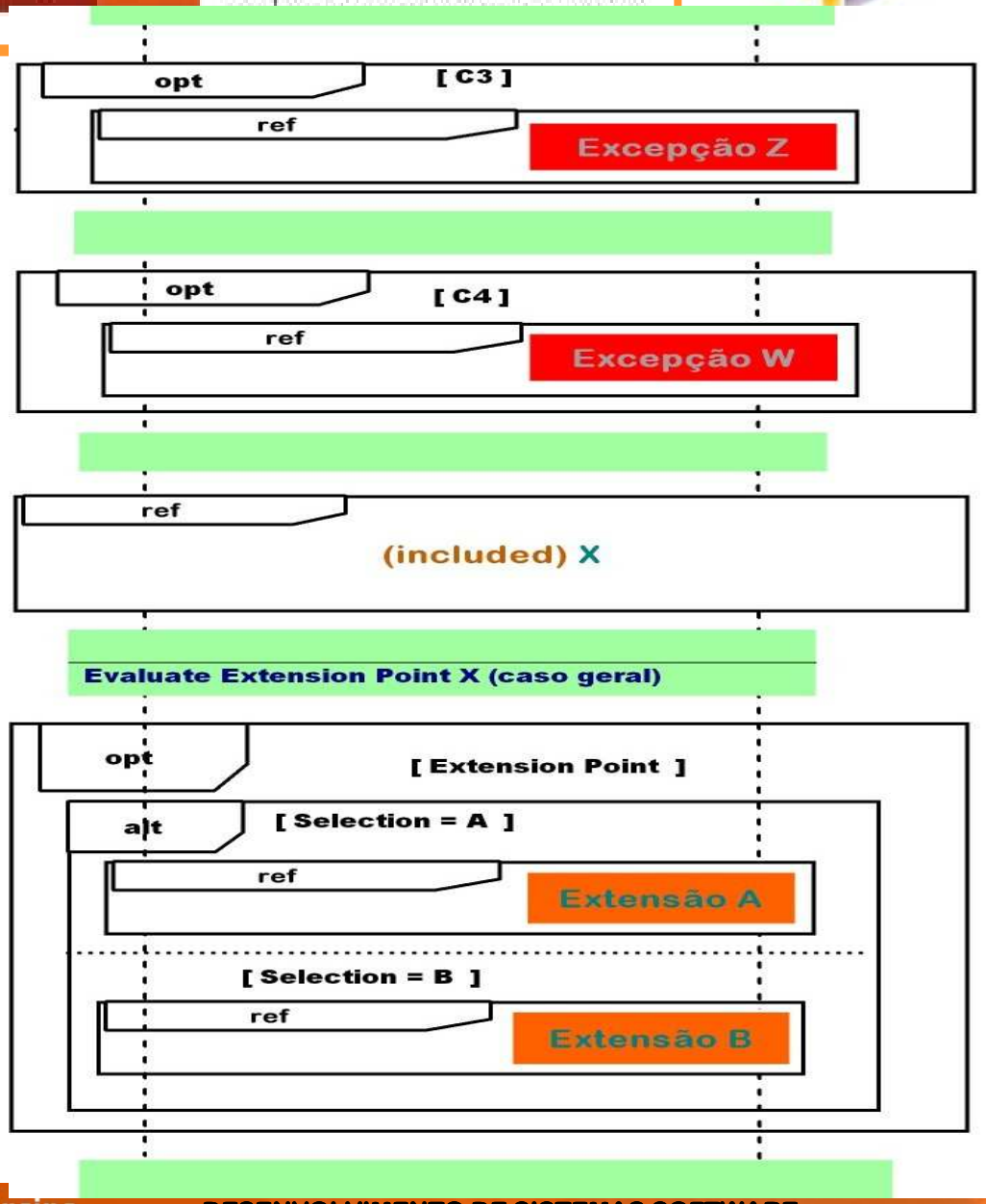
TEMPLATE GERAL COMPLETO DE UM USE CASE TEXTUAL



PASSAGEM PARA DS (1)



PASSAGEM PARA DS (2)

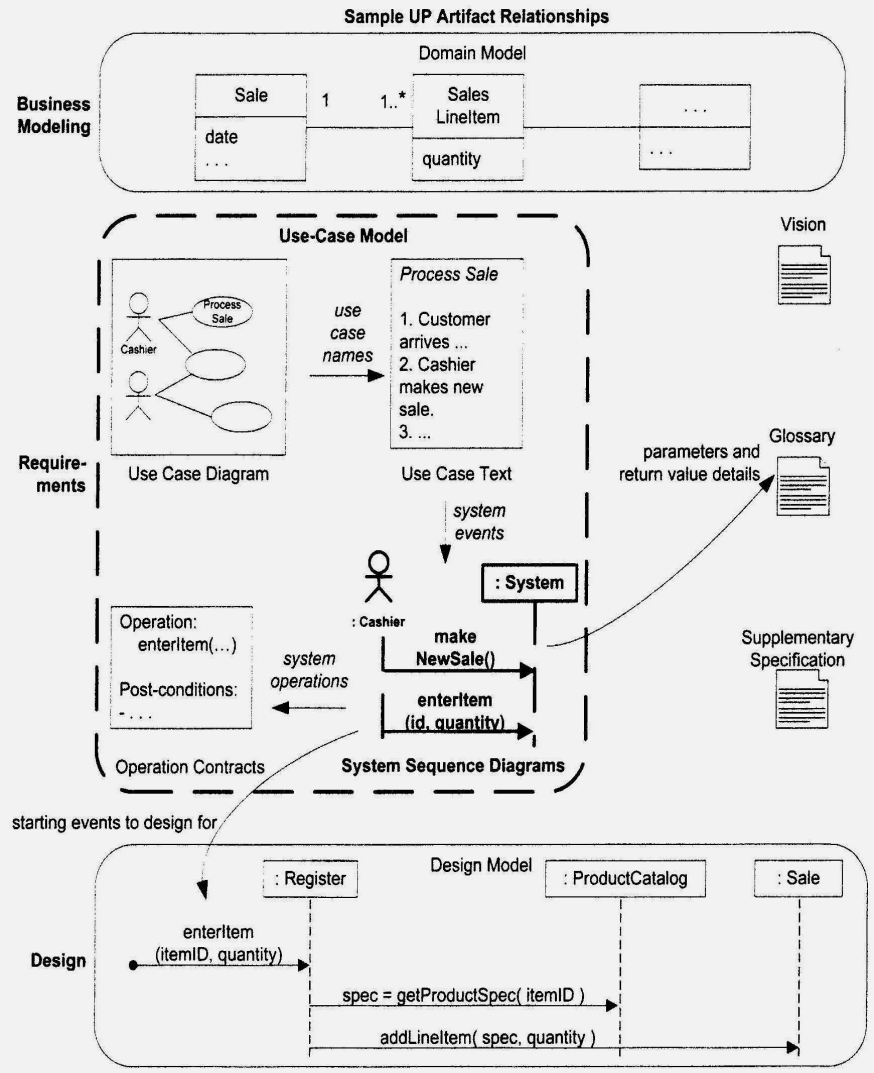


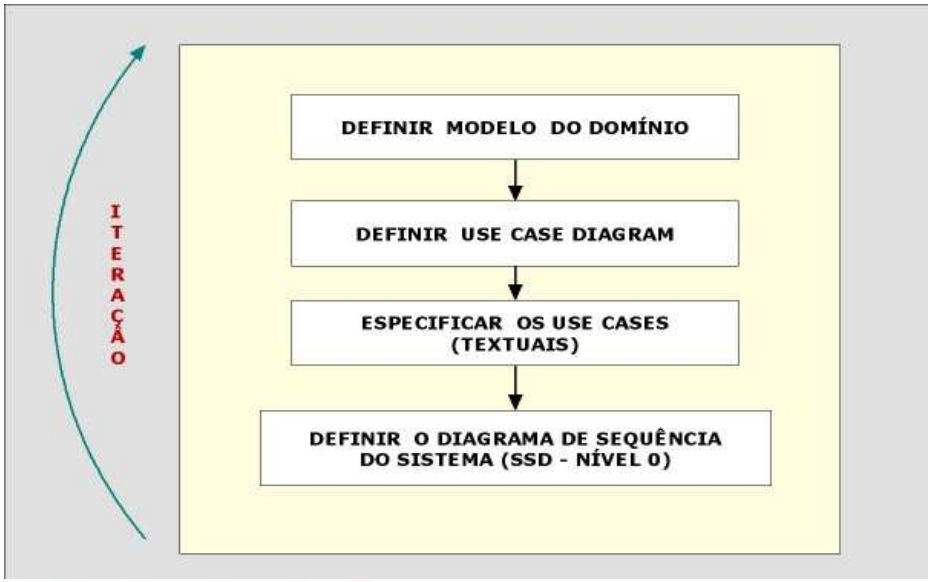
Tratamento dos Fluxos de Excepção,

Tratamento dos <<include>>

Tratamento dos <<entend>>, mesmo que tenham Extension Points alternativos

ONDE ESTAMOS RUP/DSS ?





What is an Iteration?

An iteration is a distinct sequence of activities based on an established plan and evaluation criteria, resulting in an executable release (internal or external)