



## Diagramas de Interação

Metodologias e Desenvolvimento de Software

**Pedro Salgueiro**

pds@di.uevora.pt

CLV-256



- Diagramas de interação
  - Aspectos dinâmicos do sistema
  - Descrevem
    - colaboração entre vários objetos
    - num determinado comportamento
  - UML fornece vários diagramas de interação
    - **Diagramas de sequência**
    - **Diagramas de colaboração**



- Diagramas de interação
  - Descrevem/capturam o comportamento
  - Num **único** cenário de utilização
    - Conjunto de objetos
      - Mensagens trocadas entre objetos (no cenário que está a ser modelado)
- Diagramas de sequência
  - Focam-se no tempo
  - Ordenação temporal das mensagens
- Diagramas de comunicação
  - Ou diagramas de colaboração (UML 1.x)
  - Focam-se na organização dos objetos e nos dados

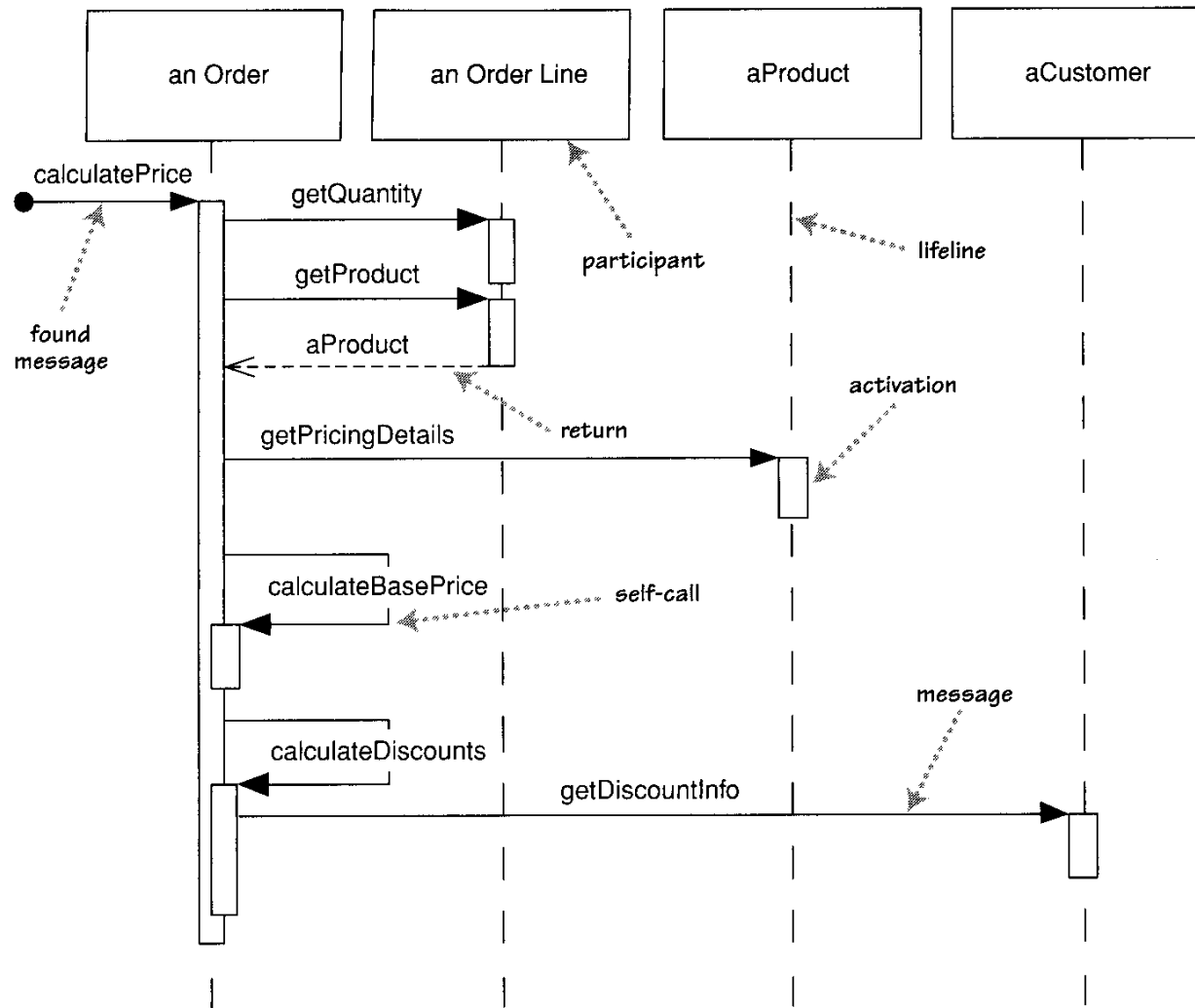


- Diagramas de sequência
  - Mostram
    - Interação entre participantes numa operação/cenário
    - Mensagens entre participantes
    - Focam-se no tempo
  - Cada participante
    - “Representado” por uma linha de vida (lifeline)
      - Linha vertical
    - Mensagens de/para linhas de vida
    - Ordenadas de cima para baixo



- Exemplo/cenário
  - Temos:
    - uma encomenda
  - Queremos
    - invocar uma operação para calcular o seu preço
  - Como?
    1. Analisar os itens de todas as linhas da encomenda
      - Determinar os seus preços
        - Regras relacionadas com os produtos
    2. Calcular o desconto da encomenda
      - Regras relacionadas com o cliente

# Diagramas de sequência



# Diagramas de sequência



- Instâncias (objetos) de (ou participantes)

- Order
- Order line
- Product
- Customer

- tipos de mensagens

- mensagem inicial

- CalculatePrice
  - “found message”

- invocação

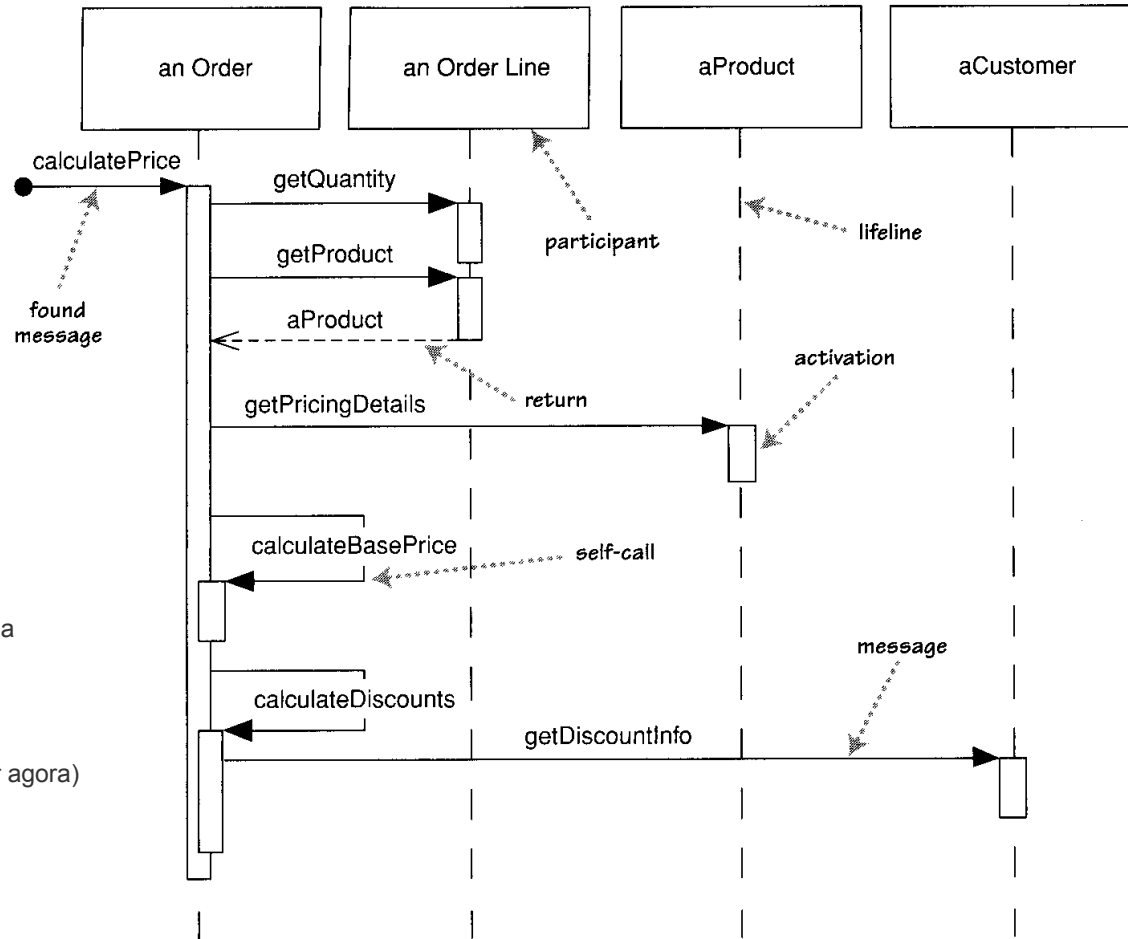
- `getQuantity`, `getProduct`,  
`getPrincipalDetails`  
e `calculateBasePrice`
  - **Devem** ser invocadas para cada linha da  
encomenda
- `CalculateDiscount`
  - Invocadas apenas uma vez
- Não se consegue distinguir no diagrama (por agora)

- self-message

- `calculateBasePrice`

- retorno

- `aProduct`





- Participantes
  - Representam objetos de classe
  - Notação
    - Não existe notação específica para os participantes
    - Duas alternativas:
      1. Sem referir tipos/classes
        - `anOrder, umaEncomenda`
      2. Referindo tipos/classes
        - `name : Class`
        - `joao : Customer`





- Linha de vida dos participantes
  - Representa o “tempo de vida” no cenário modelado
  - Caixa ou barra de ativação
    - Representa quando o participante está ativo
- Mensagens
  - Entre participantes
  - Mensagem inicial não parte de um participante
    - Tem uma origem desconhecida
    - Mensagem que dá início a toda a sequência
  - Nome das mensagens é importante
    - Ajuda a “relacionar” os participantes

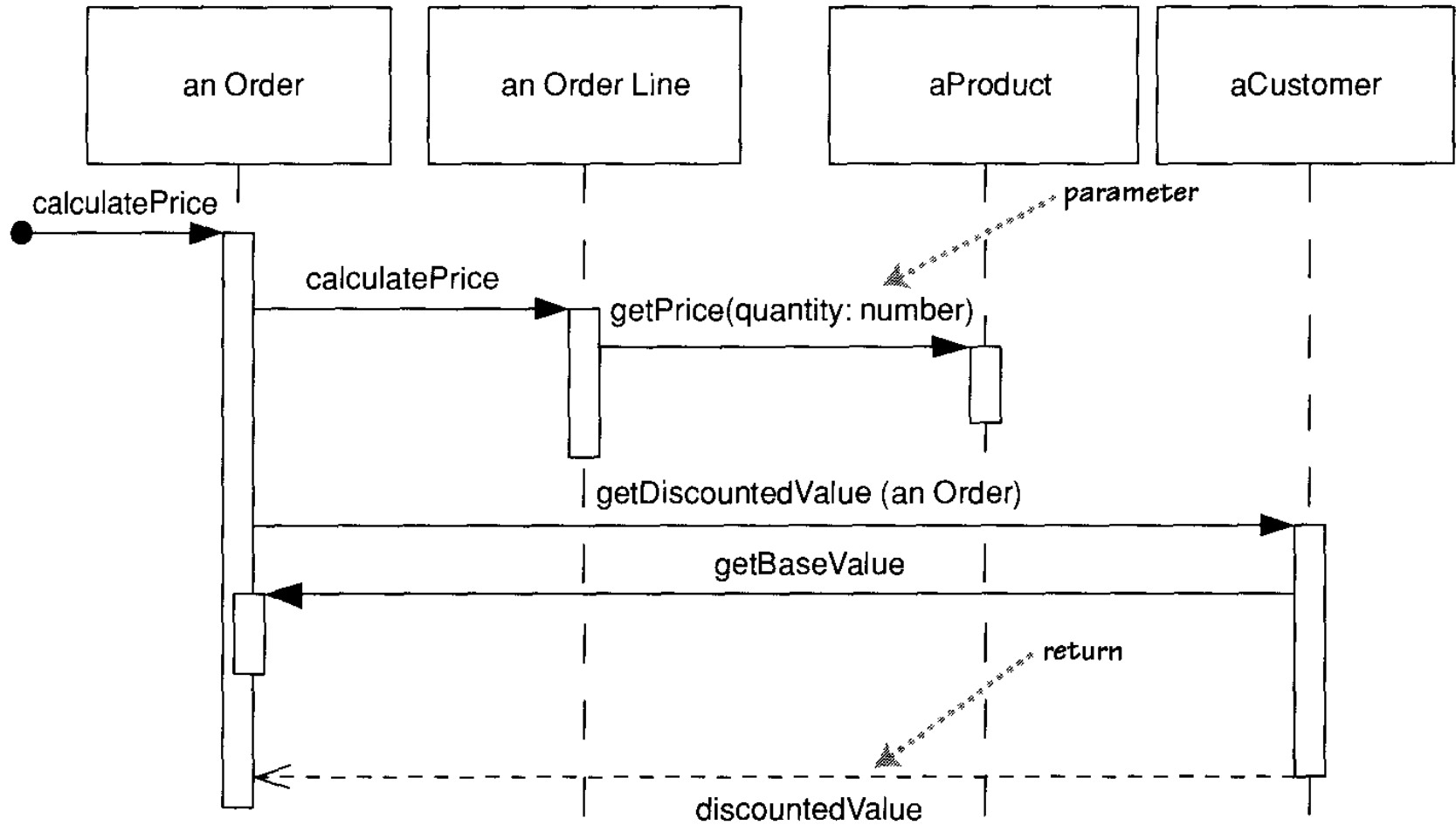
–



## Exemplo/cenário (pequena variação)

- `Order` pede a cada `OrderLine` para calcular o seu preço
- Cada `OrderLine` pede ao `Product` para calcular o seu preço
  - Indicamos a quantidade
- Para calcular o desconto, `Order` invoca um método ao `Customer`
  - Para calcular o desconto, `Customer` pede à `Order`, qual o valor base da encomenda
- Queremos
  - Invocar um comando para calcular o seu preço

# Diagramas de sequência





## Diferenças entre os dois cenários

- Forma como os participantes interagem entre si
  - Facilmente se percebe a interação entre os participante
  - Não mostra os detalhes do algoritmo, mas sim as mensagem trocadas entre os participantes
- Estilos de interação
  - Cenário 1
    - Controlo centralizado
    - Centralizado num participante:
      - Order
  - Cenário 2
    - Controlo distribuído
    - Processamento distribuído por todos os participantes



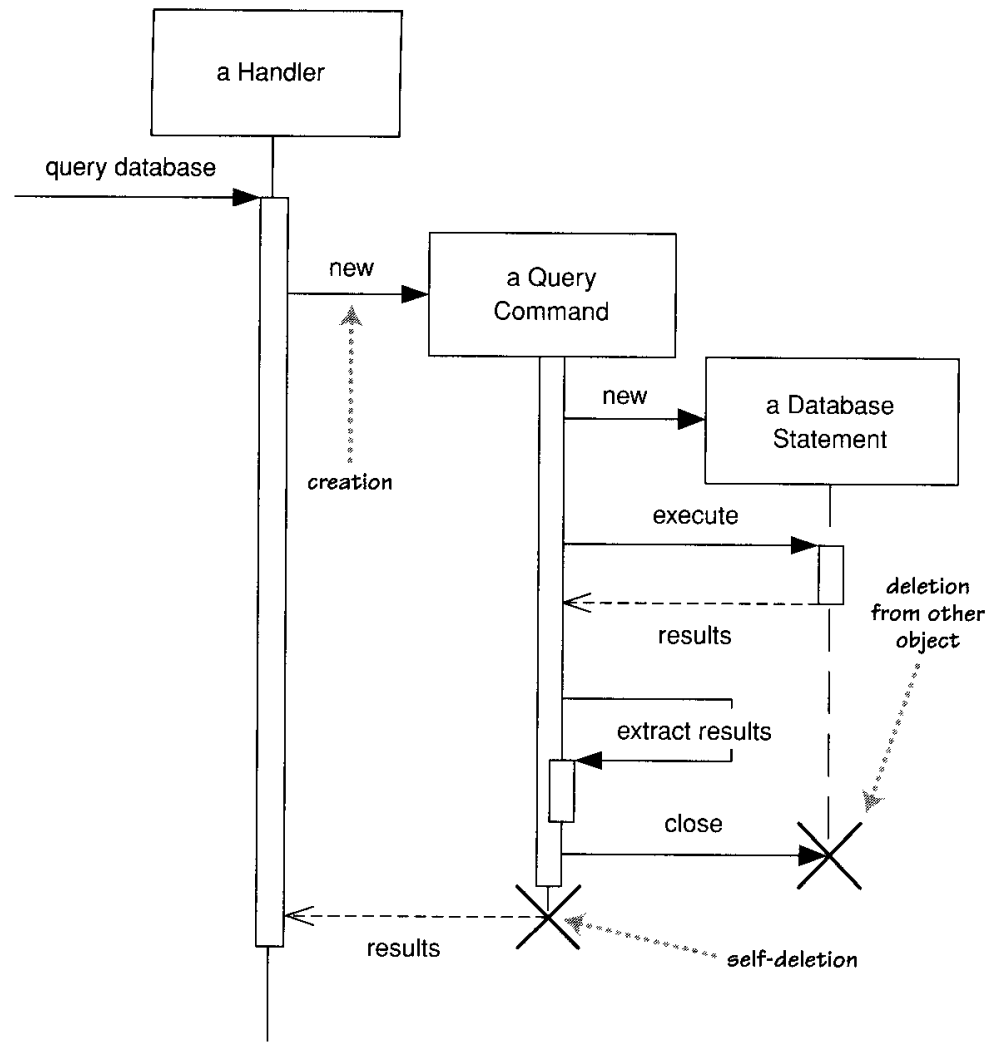
## Controlo centralizado vs distribuído

- Centralizado
  - Todo o processamento é feito no mesmo local
  - Mais simples
- Distribuído
  - Processamento distribuído por todos os participantes
    - Necessário “seguir” todos os objetos para “encontrar” o programa
  - Preferível
  - “Junta” dados com comportamentos
    - maior modularidade
    - mais hipóteses de usar polimorfismo
      - Exemplo: se cada tipo de produto tiver diferentes formas de calcular o seu preço, então o cálculo do preço é implementado pelas suas subclasses
  - Muito “orientado a objetos”



- Criar e apagar participantes (objetos)
  - Participantes que não existem durante toda a operação/cenário
- Criar participantes
  - Mensagem diretamente para a caixa do participante
    - Seta
    - Nome da mensagem é opcional
      - Normalmente: “new”
- Apagar participantes
  - Marcado/indicado com um “X grande”
  - Uma mensagem direta para o “X grande” indica que um participante apaga outro participante
  - “X grande” no fim de uma linha de vida, indica que o participante apagou-se a ele próprio

# Diagramas de sequência





- Apagar participantes, quando usar?
  - Num ambiente com *garbage collection*, não é necessário apagar objetos
    - Deve-se indicar que o objeto já não é preciso, usando o “X grande”
  - Para fechar ou terminar operações
    - Indicando que o objeto já não é preciso





## Ciclos e condições

- Diagramas de sequência
  - mostrar interações entre objetos
    - não são os mais indicados para modelar estruturas de controle
- Como modelar
  - ***interaction frames*** (caixas de interação)
  - marcar parte de um diagrama de sequências
  - parte do diagrama de sequências é “dividido” em vários fragmentos
  - Cada *frame* tem um operador e uma “guarda”



## Ciclos e condições

- Operadores
  - Tipo de operação associada ao frame
  - Ciclo: `loop`
  - Condições: `alt`
- Guarda
  - Quando é que o *frame* é “executado”



## Ciclos e condições

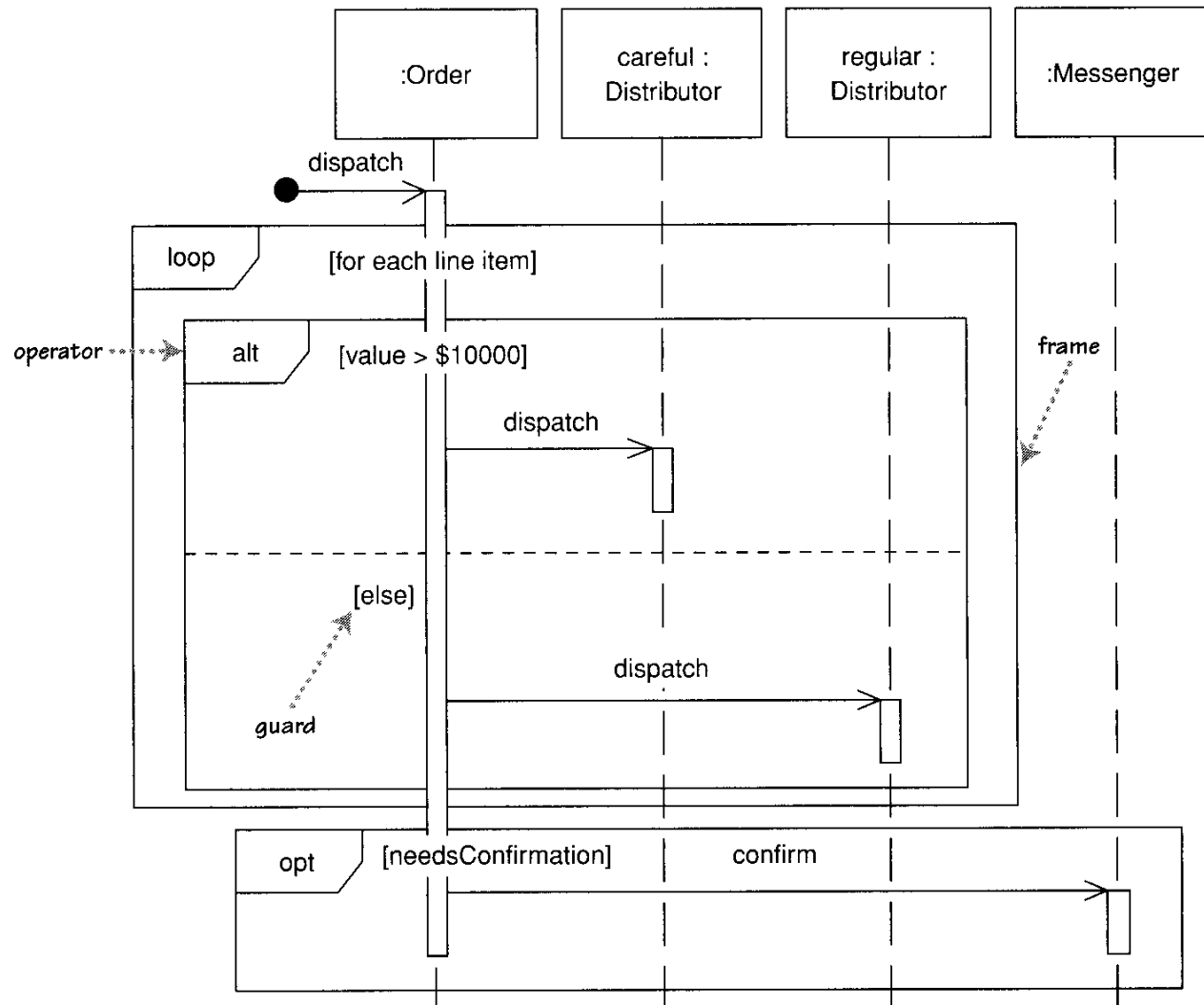
- Ciclos
  - Apenas um *frame*
  - Operador: `loop`
  - Guarda: Controlo do ciclo
- Condições
  - Vários *frames*
    - Um *frame* para cada condição
  - Operador: `alt`
  - Guarda: condição associada ao *frame*
    - Apenas os *frames* em que a guarda tenham um valor booleano verdadeiro são executadas



## Operadores comuns

- **alt**: múltiplos *frames* alternativos; apenas aqueles cuja condição é verdadeira, são executados
- **opt**: *frames* opcionais; o *frame* apenas é executado se a condição associada for verdade
- **par**: *frames* executados em paralelo;
- **loop**: ciclo; *frame* pode executar várias vezes; guarda indica como a iteração é feita
- **region**: região critica; o fragmento apenas permite executar uma *thread* ao mesmo tempo
- **neg**: fragmento indica uma interação que não é válida

# Diagramas de sequência



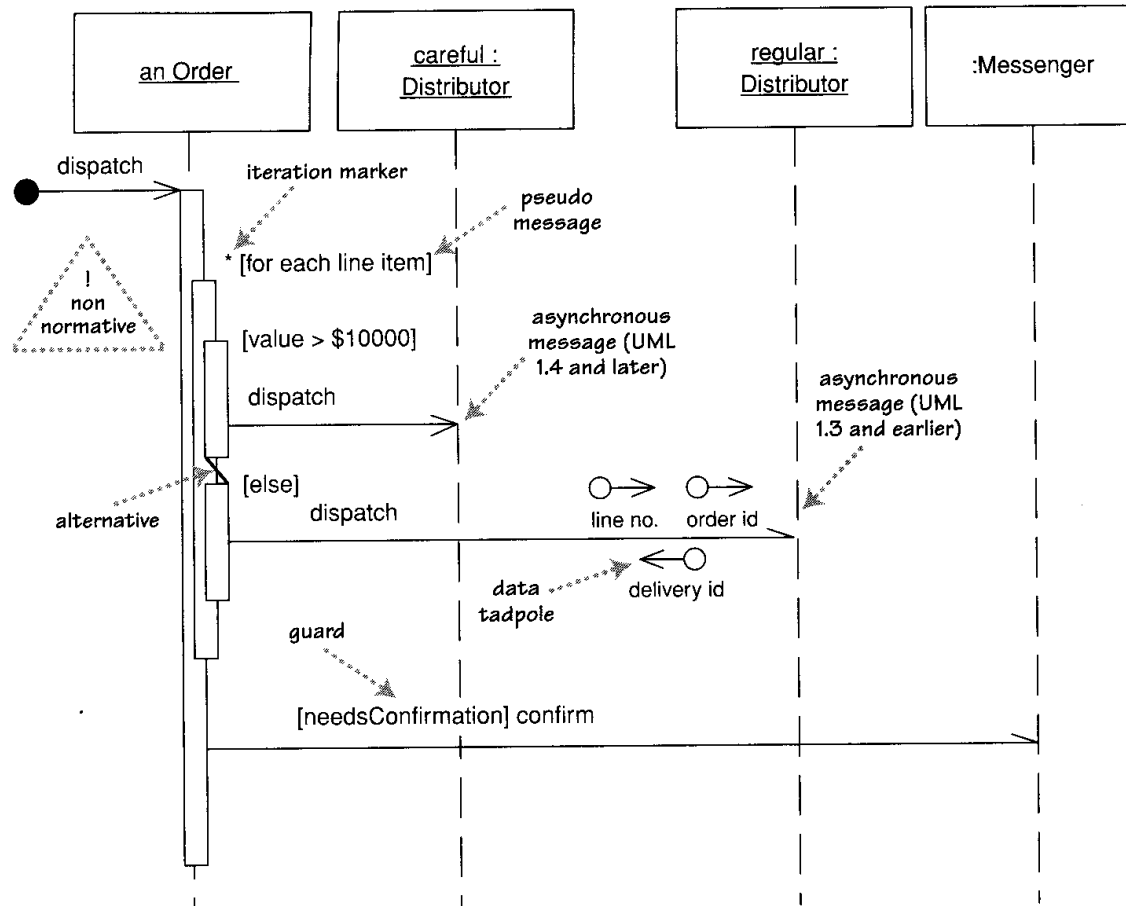


- Frames / segmentos
  - Notação introduzida no UML 2.x
- UML 1.x
  - marcadores de iteração (iteration markers)
    - \* adicionado à mensagem
    - texto entre [ ] para indicar a base da iteração
  - Guardas
    - expressões condicionais entre [ ]
    - mensagem apenas é enviada se a guarda for verdadeira
  - Notações que não devem ser usadas em UML 2.x
    - mas são permitidas nos diagramas de comunicação



- UML 1.x
  - Guardas
    - Não conseguem indicar que um conjunto de guardas são mutuamente exclusivas
  - Pseudo mensagens
    - Não são mensagens
    - Servem para indicar as bases dos ciclos
    - Ou as várias alternativas dos *ifs*
  - Marcadores de alternativas
    - Ajudar a indicar as diferentes alternativas de um *if*

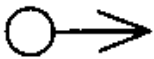
# Diagramas de sequência







## Passagem de dados

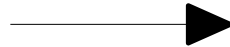
- Não existe notação standard
- Opção
  - Através de parâmetros nas mensagens
  - Setas de retorno
- Alternativamente
  - *Data tadpoles*
    - *Setas com uma bola* 
  - Indicam o movimento dos dados



## Mensagens síncronas e assíncronas

- Síncronas

- Chamador da mensagem espera que a operação associada à termine
- Necessita de esperar pela mensagem
- Notação
  - Seta “fechada”:



- Assíncrona

- Chamador da mensagem prossegue imediatamente após enviar a mensagem
- Não precisa de esperar por uma resposta
- Usadas tipicamente em sistemas multi-thread
  - Várias mensagens em simultâneo
- Notação
  - Seta “aberta”:





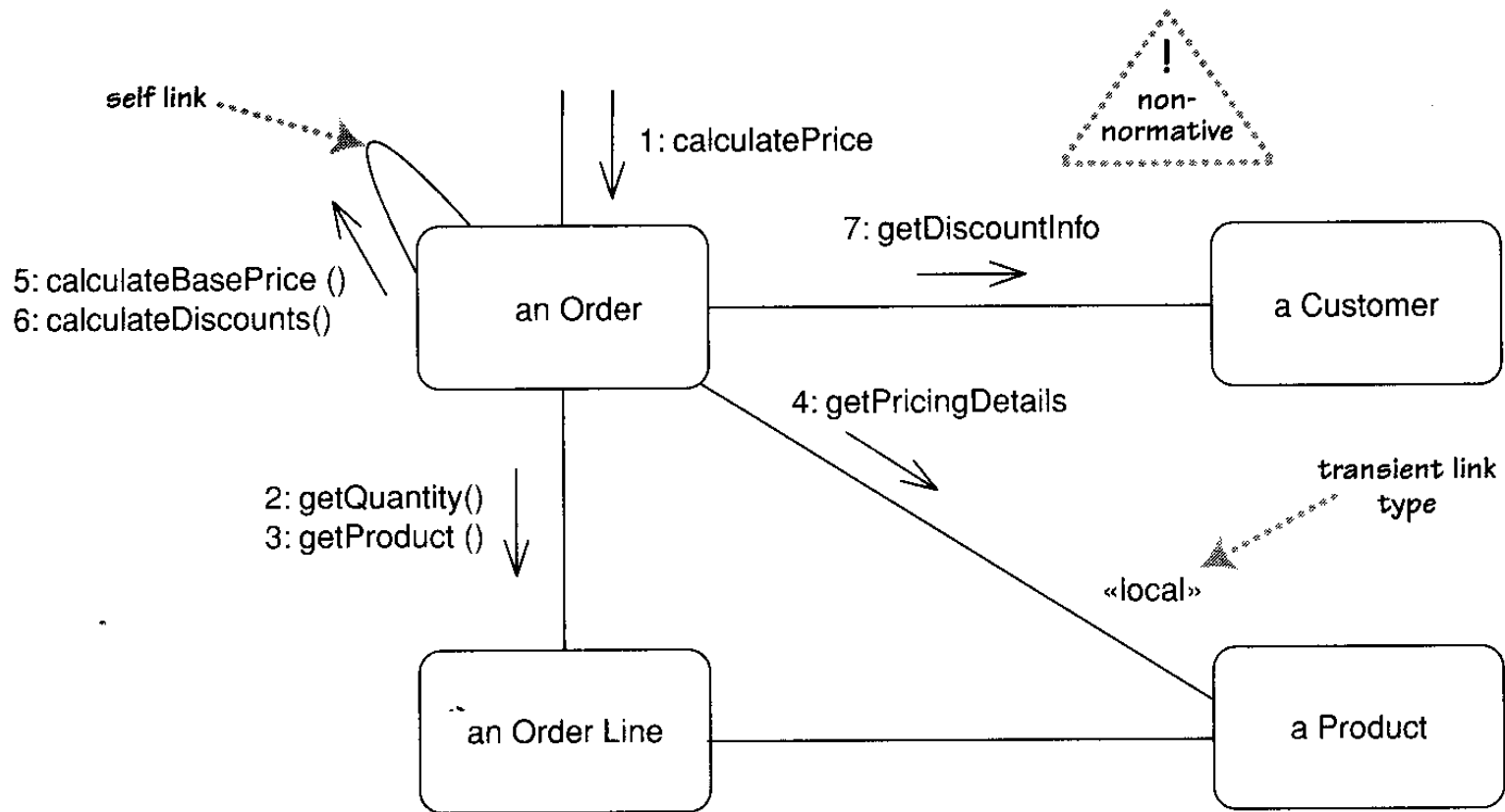
## Diagramas de comunicação

- Foco nos *links* de dados entre os participantes
- Participantes
  - Colocação “livre” no diagrama
- Links
  - Entre os participantes
  - Mostram como cada participante se liga a outro
  - Podem representar
    - Instâncias de associações entre classes
    - Links transientes
      - Ligações que só existem no contexto desta interação
      - Locais
      - anotação «local»
- Mensagens entre os participantes
  - Usando os links
  - Numerar as mensagens, para saber a sequência correta



## Diagramas de comunicação

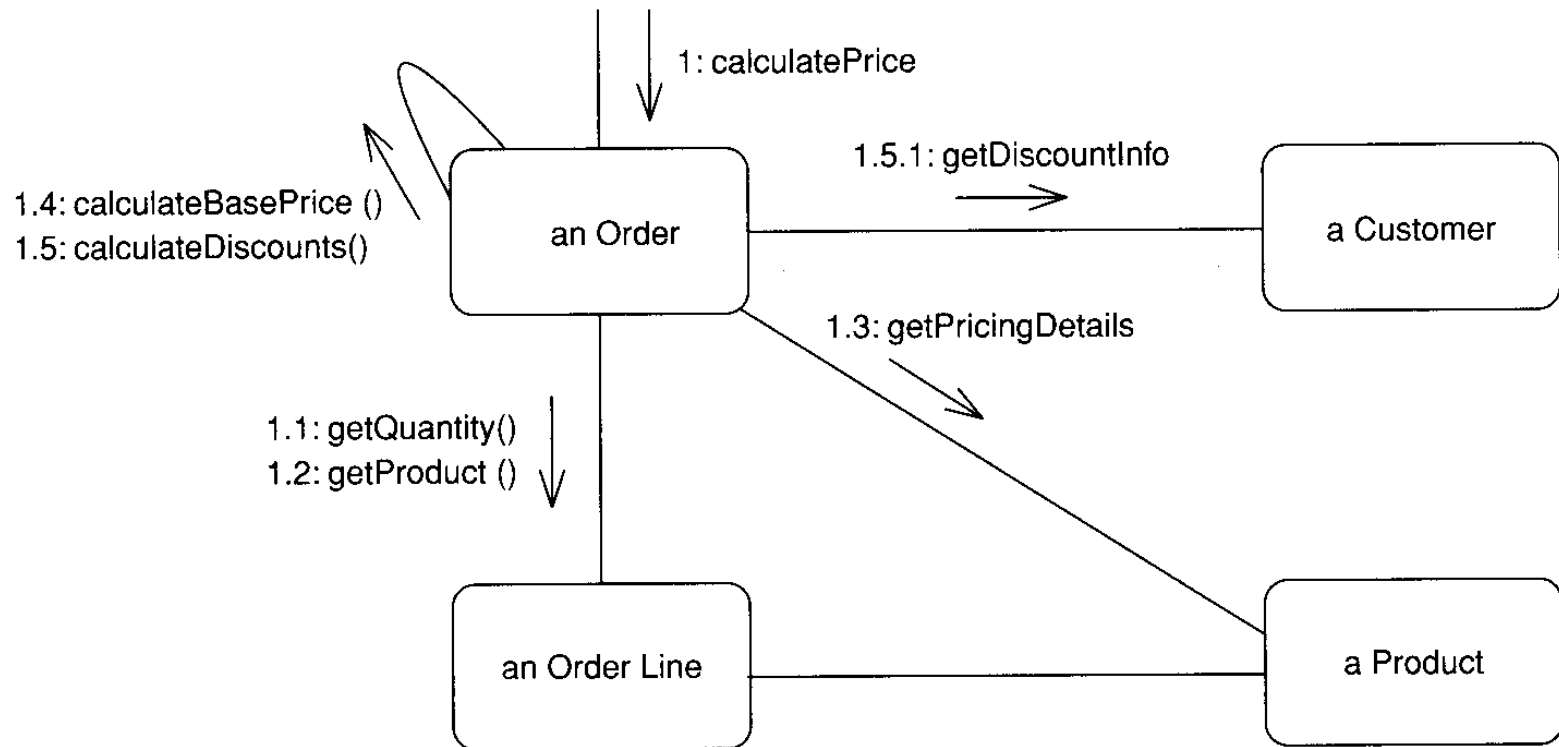
- Não têm notação precisa para especificar lógica de controlo
- Guardas e marcadores de iteração
  - Marcadores de iteração (iteration markers)
    - \* adicionado à mensagem
    - texto entre [ ] para indicar a base da iteração
  - Guardas
    - expressões condicionais entre [ ]
    - mensagem apenas é enviada se a guarda for verdadeira





## Numeração das mensagens

- Esquema sequencial
  - 1, 2, 3, 4, . . .
  - Não permite saber qual o “scope” ou o âmbito da invocação das mensagens
- *Nested decimal*
  - 1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.5.1





## Sequência vs Comunicação

- “Equivalentes entre si”
  - A partir de um, consegue-se chegar ao outro
  - Diagramas de comunicação não têm notação para lógica de controlo
- Sequência
  - Usar quando se quer focar a sequência de chamadas
- Comunicação
  - Quando se quer focar a ligação entre os participantes
  - Bom para explorar diferentes alternativas
    - Mais fácil de fazer alterações





- UML Distilled A Brief Guide to the Standard Object Modeling Language. Martin Fowler. 3rd edition. Addison-Wesley Professional. 2003. Capítulos 4 e 12.
- Software Engineering. Ian Sommerville. 10th Edition. Addison-Wesley. 2016. Capítulo 5.