MODELAÇÃO DO COMPORTAMENTO - diagramas de sequência

MODELAÇÃO E ANÁLISE DE SISTEMAS | TP

ILÍDIO OLIVEIRA ico@ua.pt
v2018-04-04

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

From analysis onward....

Analysis modeling answers the questions of who will use the system, what the system will do, and where and when it will be used.

During analysis, detailed requirements are identified and a system proposal is created.

The team produces the **functional model** (use-case diagram, activity diagrams, and use-case descriptions), **structural model** (CRC cards and class diagram, and object diagrams), and **behavioral models** (sequence diagrams, communication diagrams, behavioral state machines, CRUDE analysis).

Functional. What the system does? The external behavior of the system (back-box).

Structural. What are the parts ("things", objects) composing the system?

Behavior. How does the system do the operations? Explanation of the system interactions.

Behavioral modeling target

Behavioral models describe the dynamic aspects of an information system.

During analysis: behavioral models describe what the **internal logic of the processes** is without specifying how the processes are to be implemented.

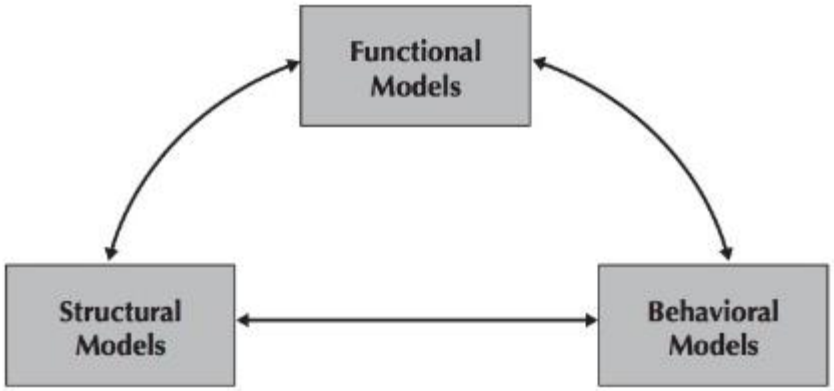
(in the design and implementation phases, the detailed design is fully specified)

Behavioral modeling is also **use-case driven**.

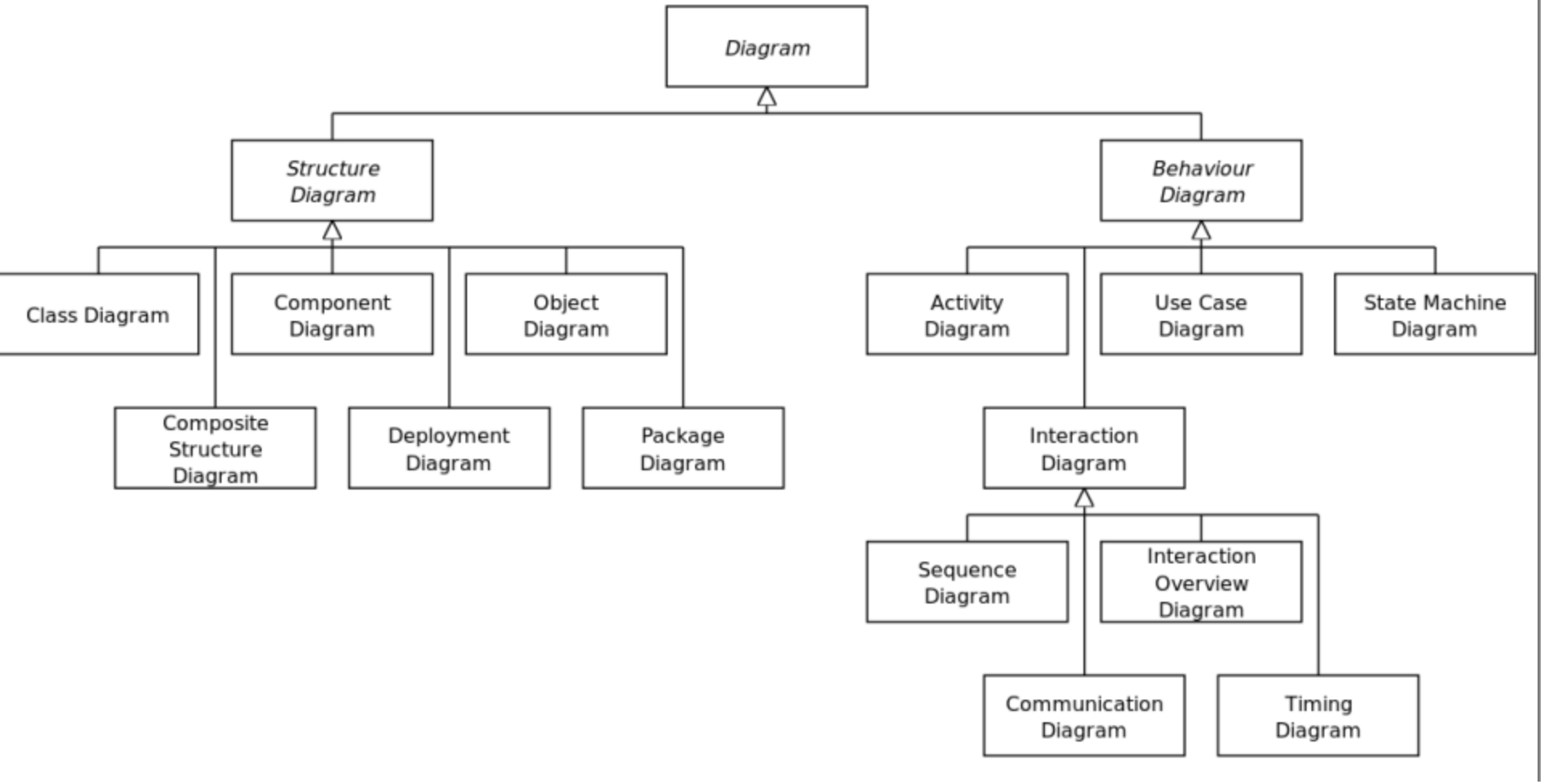
One of the primary purposes is to **show how the underlying objects in a problem domain will work together to form a collaboration** to support each of the use cases' scenarios.

Structural models → the objects and the relationships

Behavioral models → internal view of the process that a use case describes.



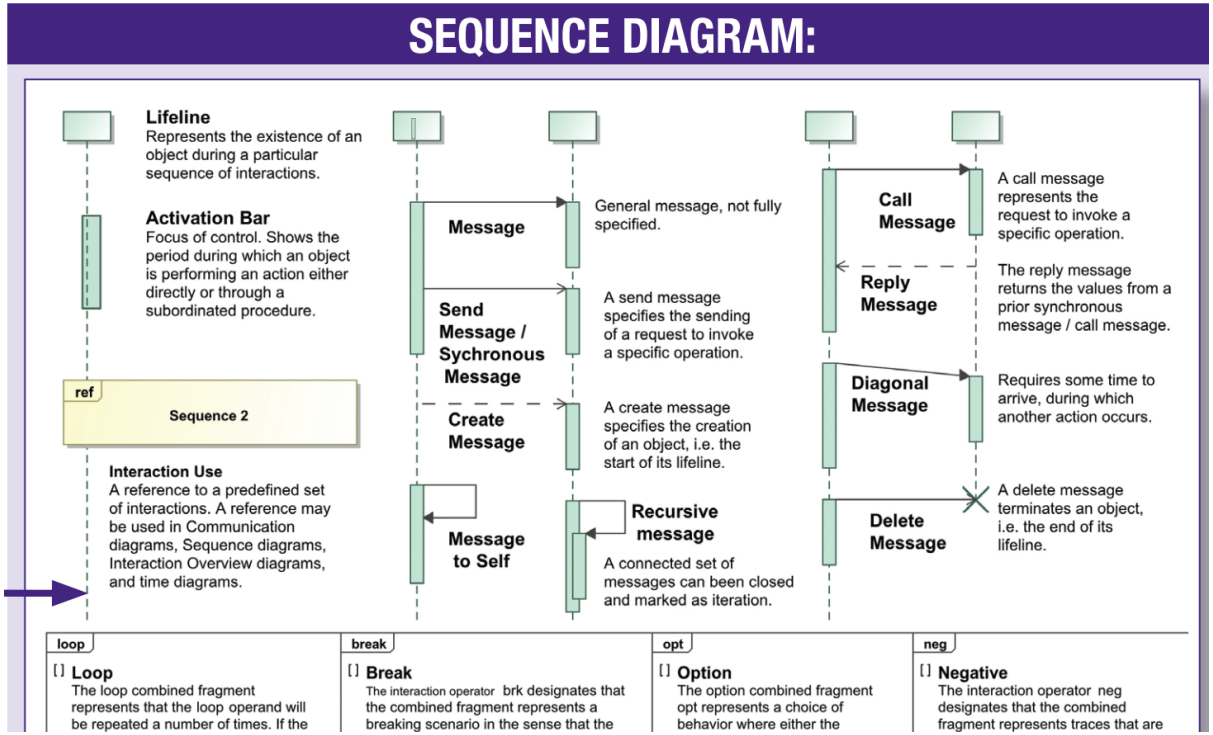
Diagramas da UML 2.x



Sequence diagram

Illustrate **the objects** that participate in a collaboration (e.g.: use case) and **the messages** that pass between them over **time**.

A sequence diagram is a dynamic model that shows the explicit sequence of messages that are passed between objects in a defined interaction.



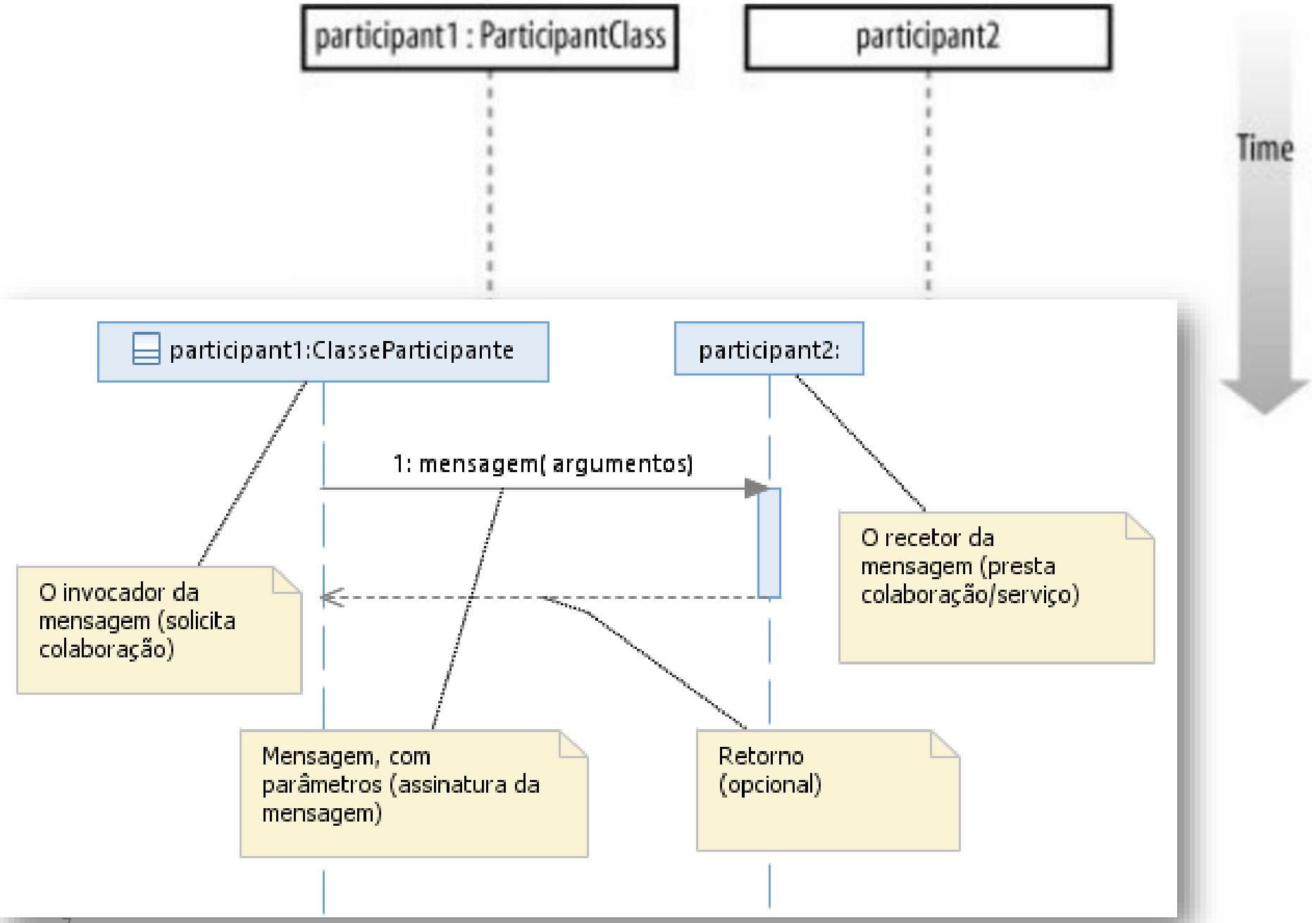
NoMagic's UML Reference card

Focus on **object-level collaboration**

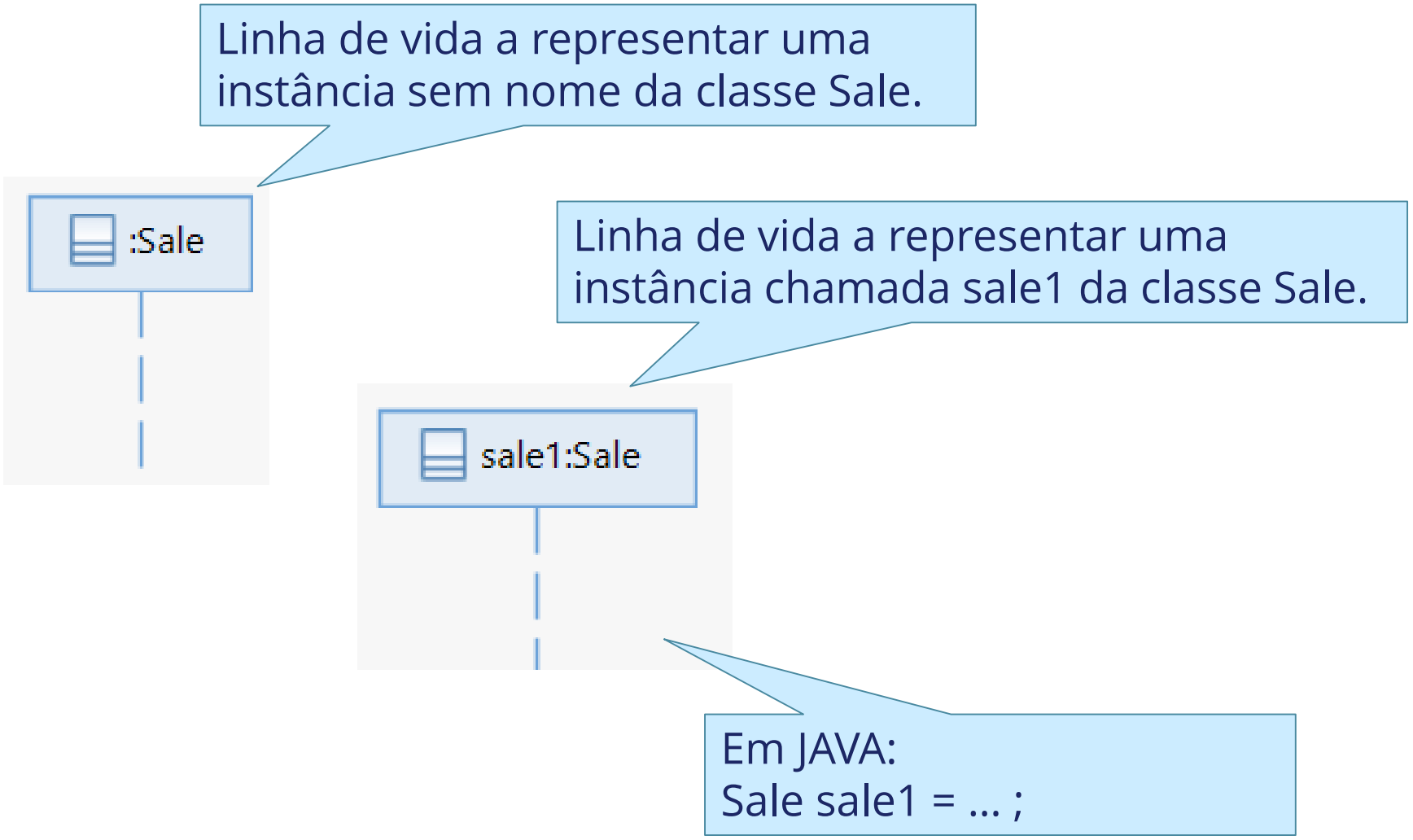
The modeling focus of class diagrams is at the class level, whereas the interaction diagrams focus on the **object level**.

Each object has attributes that describe information about the object. **Each object also has behaviors**. At this point in the development of the **evolving system**, the behaviors are described by operations. An operation is an action that an object can perform.

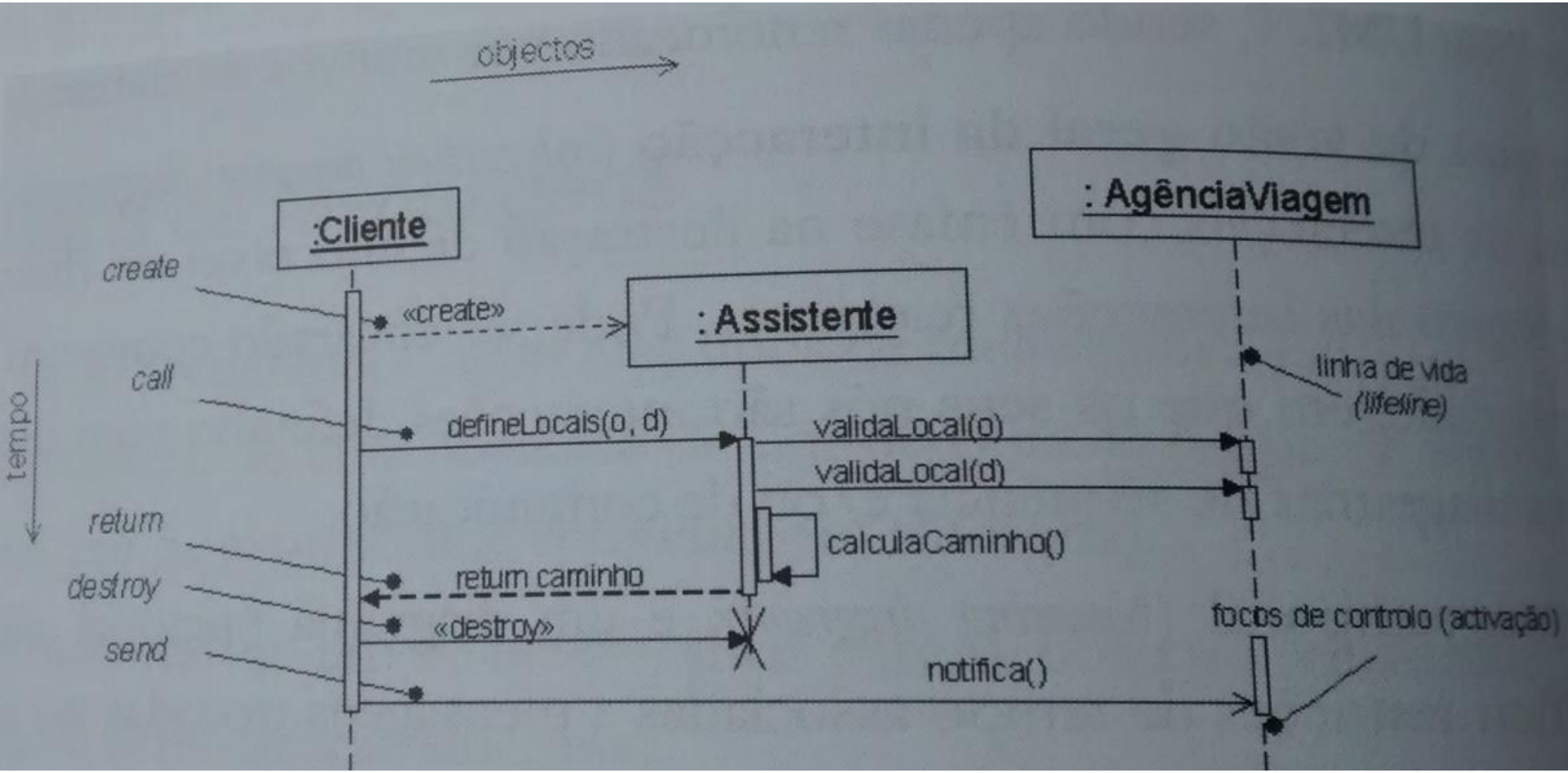
Each object also can **send and receive messages**. Messages are information sent to objects to tell an object to execute one of its behaviors. Essentially, a message is a function or a *call* from one object to another object.


participant1 : ParticipantClass
participant2
Time
participant1:ClasseParticipante
participant2:
1: mensagem(argumentos)
O invocador da mensagem (solicita colaboração)
O recetor da mensagem (presta colaboração/serviço)
Mensagem, com parâmetros (assinatura da mensagem)
Retorno (opcional)

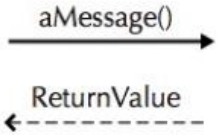
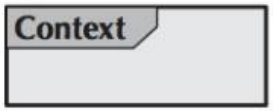
Aspetos notacionais dos DS



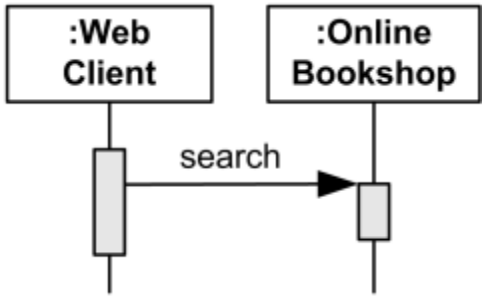
Colaboração entre objetos por mensagens (síncronas)



Term and Definition	Symbol
An actor: ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative).	 **anActor** **<<actor>>** **anActor**
An object: ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram.	**anObject : aClass**
A lifeline: ■ Denotes the life of an object during a sequence. ■ Contains an X at the point at which the class no longer interacts.	

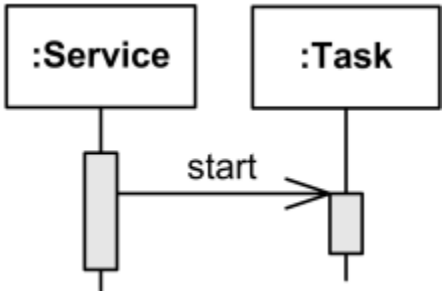
A message: ■ Conveys information from one object to another one. ■ A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow.	
A guard condition: ■ Represents a test that must be met for the message to be sent.	
For object destruction: ■ An X is placed at the end of an object's lifeline to show that it is going out of existence.	**X**
A frame: ■ Indicates the context of the sequence diagram.	

Semântica da invocação

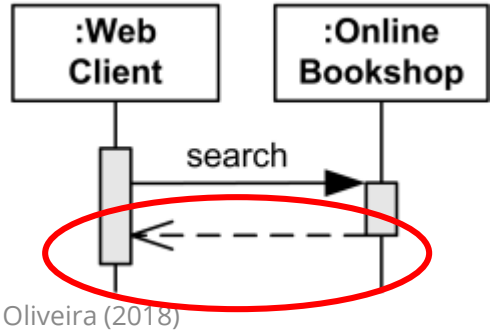


síncrona

```
...
result = B.search();
...
```



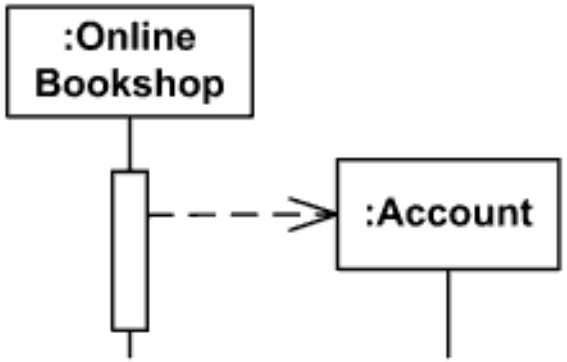
assíncrona



retorno

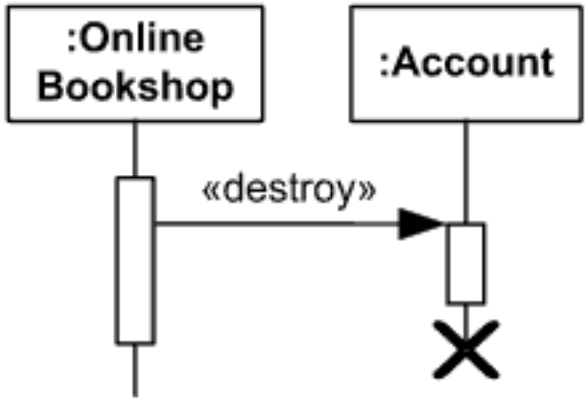
From http://www.uml-diagrams.org/sequence-diagrams.html

Modelar a criação/destruição do participante



create

```
...
Account a= new Account()
...
```

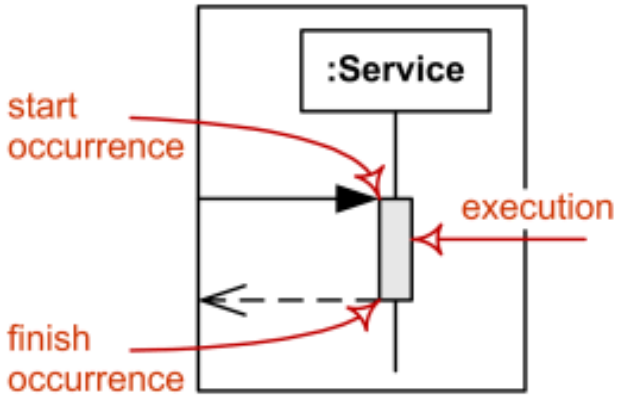


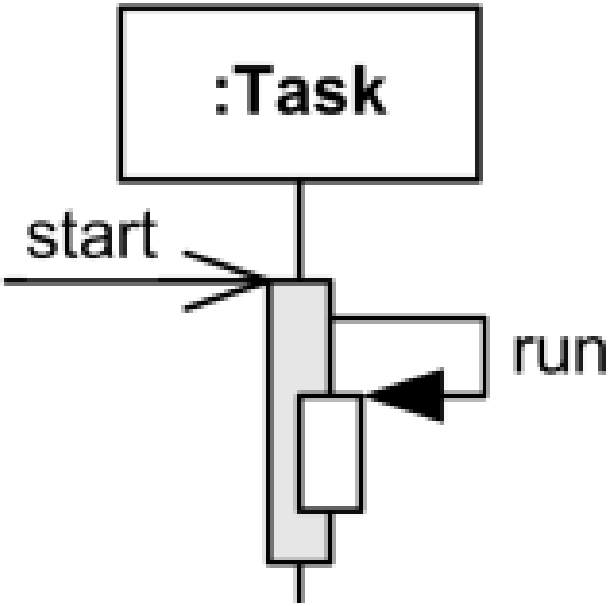
destroy

```
...
a=null;  // in java
[a release] // objective C
free a; // C
```

From http://www.uml-diagrams.org/sequence-diagrams.html

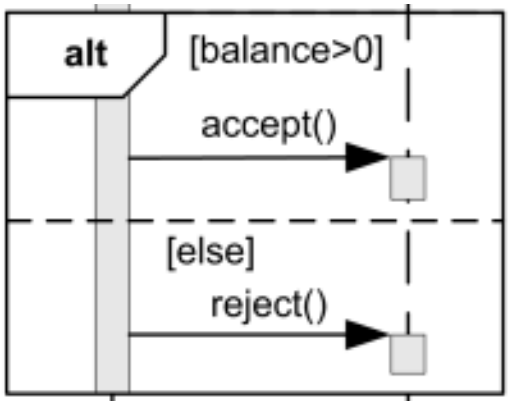
Ativação



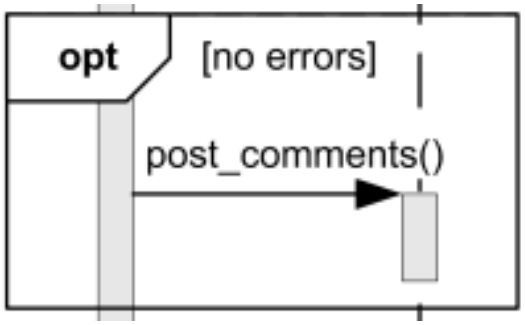


From http://www.uml-diagrams.org/sequence-diagrams.html

Fragmentos para mostrar alternativas



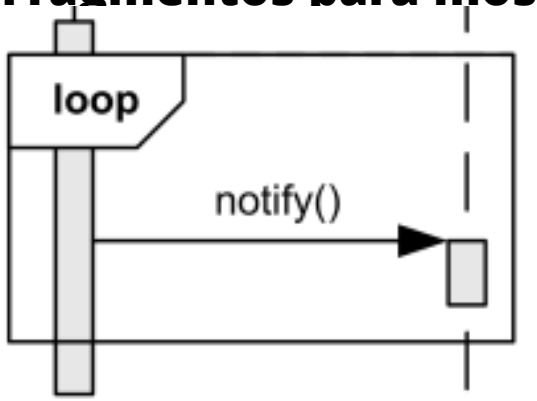

```
If (balance>0)
        otherObj.Accept()
Else
        otherObj.Reject()
```



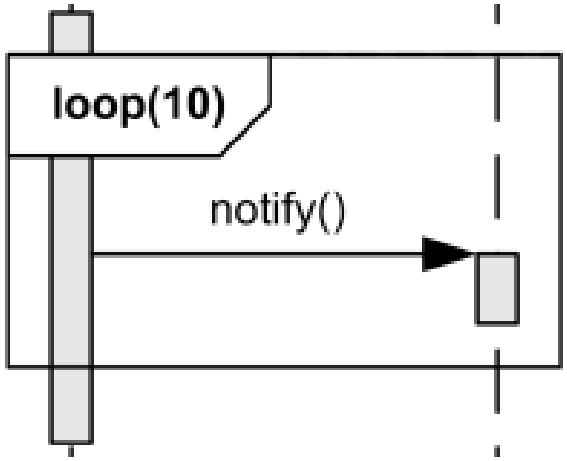

```
If ( no errors )
        otherObj.Post_comments()
```

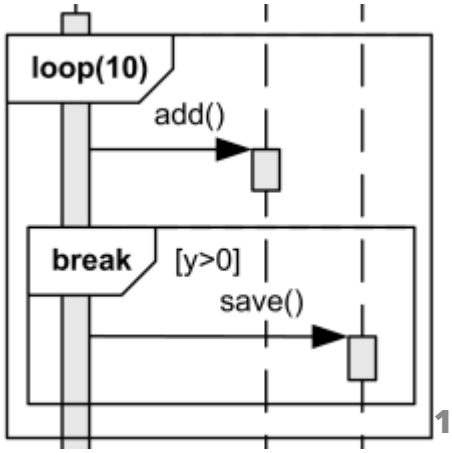

From http://www.uml-diagrams.org/sequence-diagrams.html

Fragmentos para mostrar loops



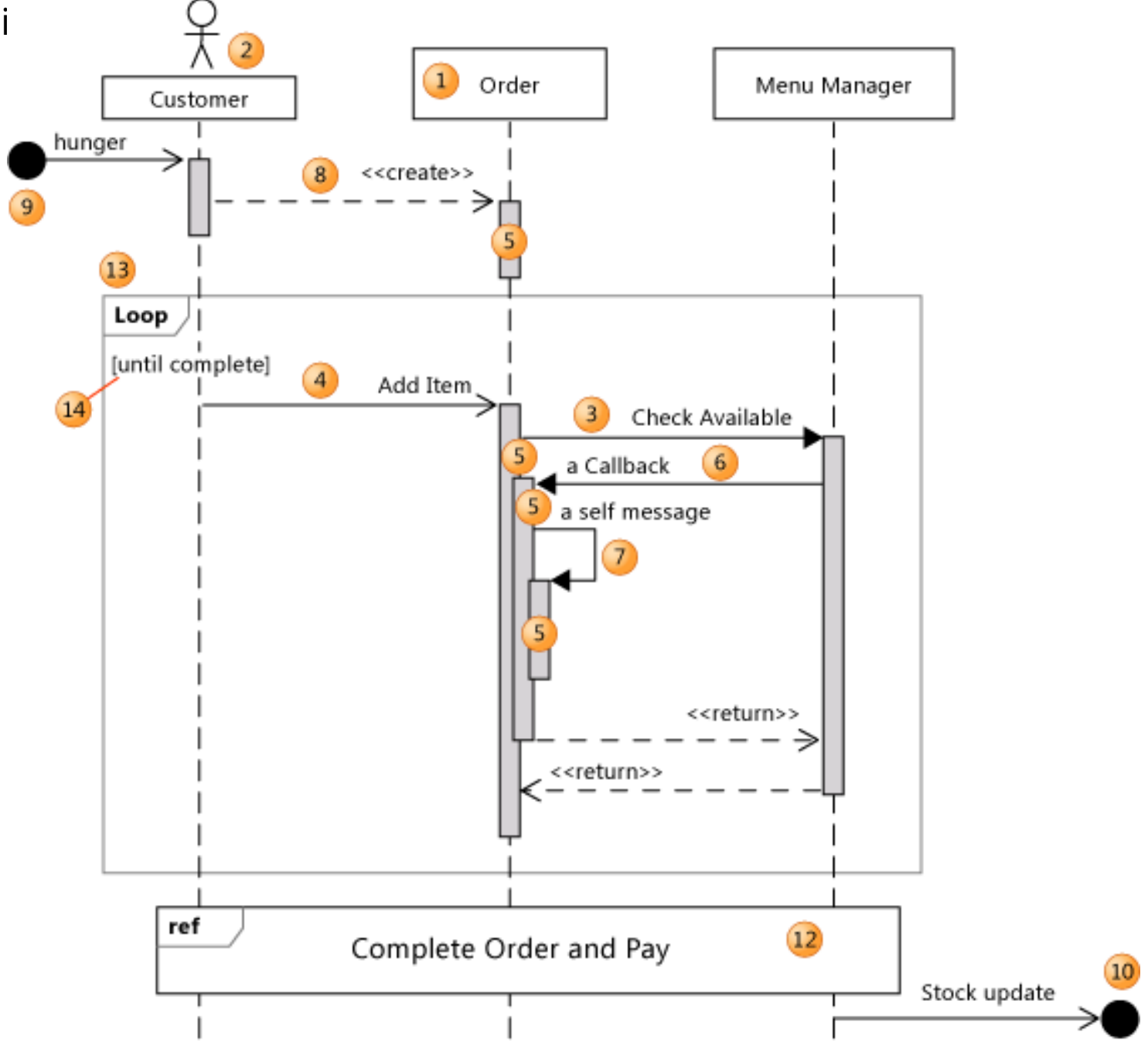

```
i=0;
While( i<10 )
{
 otherObj.Add()
 If ( y>0 ) break;
 i++
}
```





From http://www.uml-diagrams.org/sequence-diagrams.html

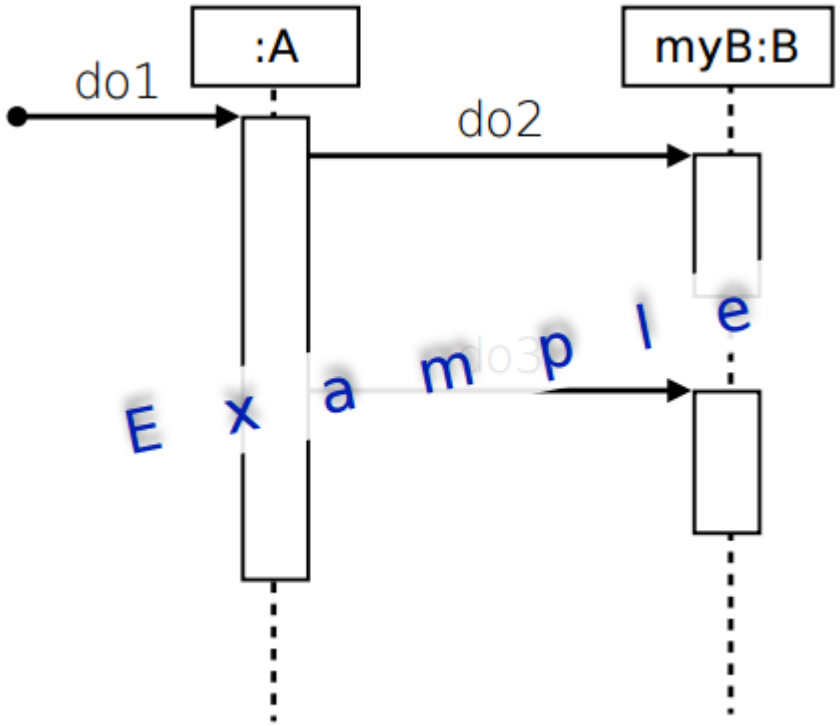
l Oli



https://msdn.microsoft.com/en-us/library/dd409377.aspx

Quatro tipos de diagramas de iteração disponíveis

- D. Sequência
  Formato alinhado
- D. Comunicação
  Formato grafo
- Diagrama temporal (*timming*)
- Diagrama de visão geral da interação (*interaction overview*)



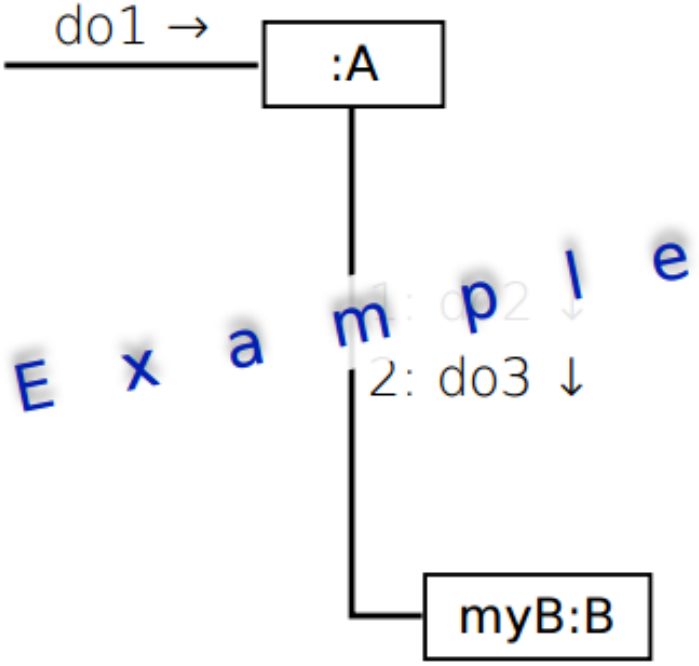
Quatro tipos de diagramas de iteração disponíveis

D. Sequência
Formato alinhado

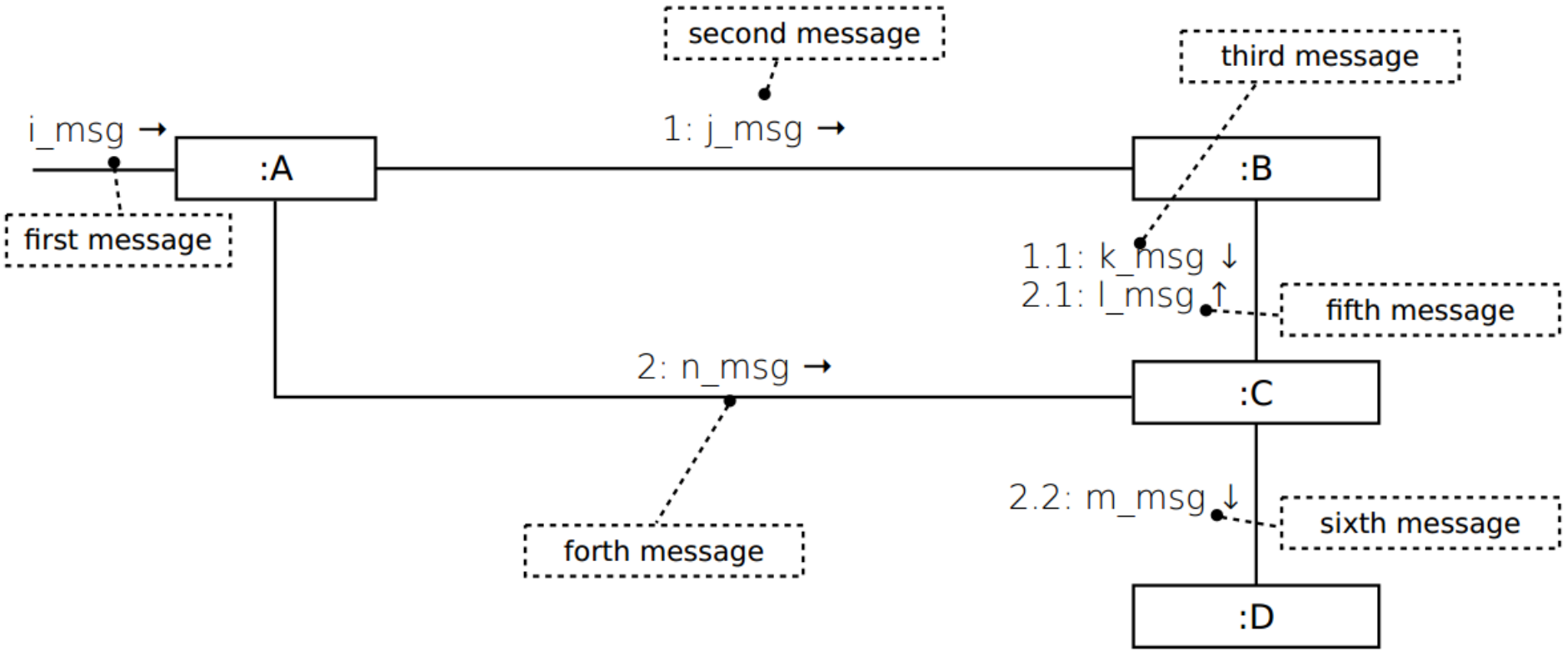
D. Comunicação
Formato grafo

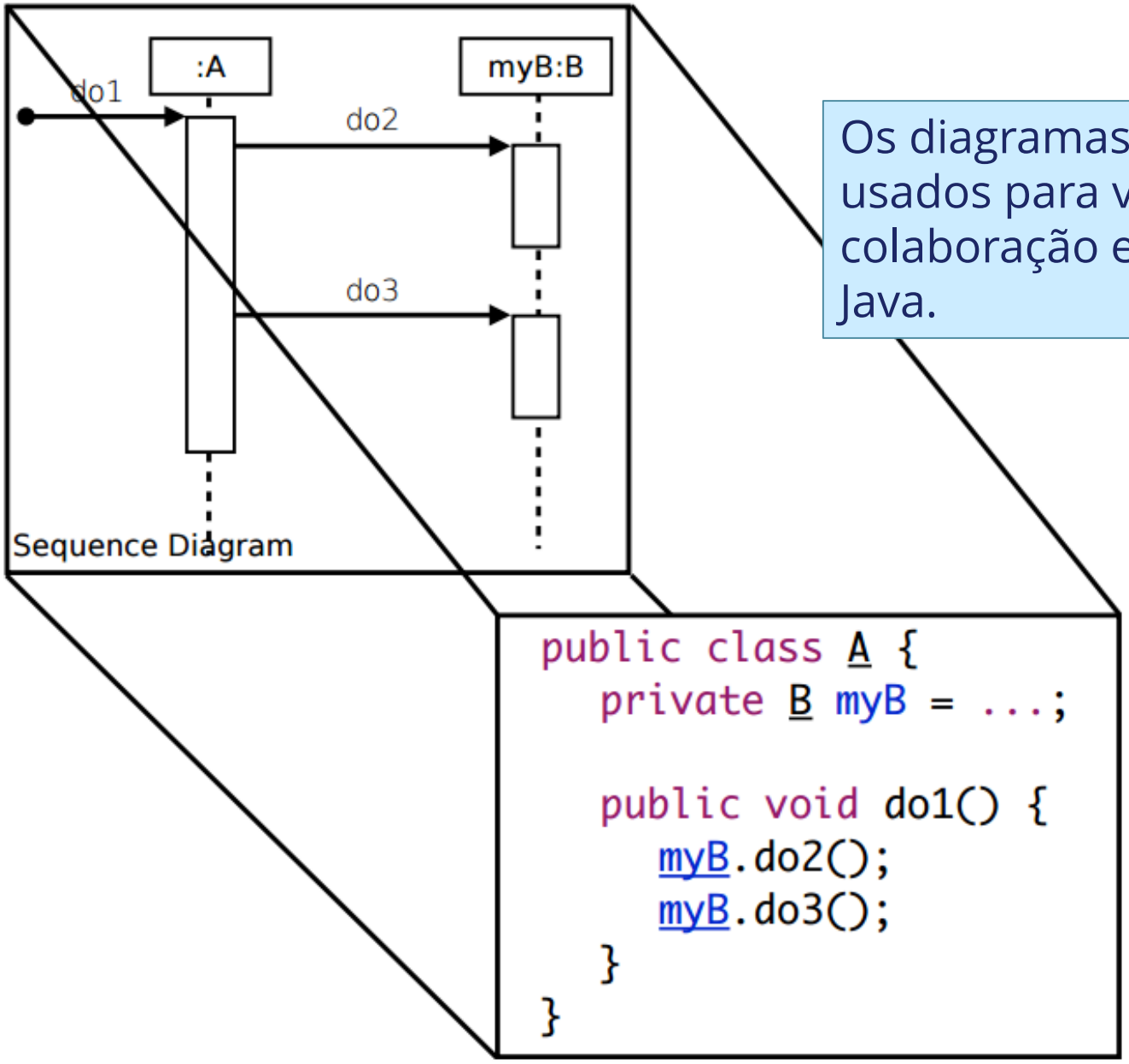
Diagrama temporal (*timming*)

Diagrama de visão geral da interação (*interaction overview*)



Diagramas de comunicação






```
public class A {
   private B myB = ...;

   public void do1() {
      myB.do2();
      myB.do3();
   }
}
```

Os diagramas de podem ser usados para visualizar a colaboração entre objetos em Java.

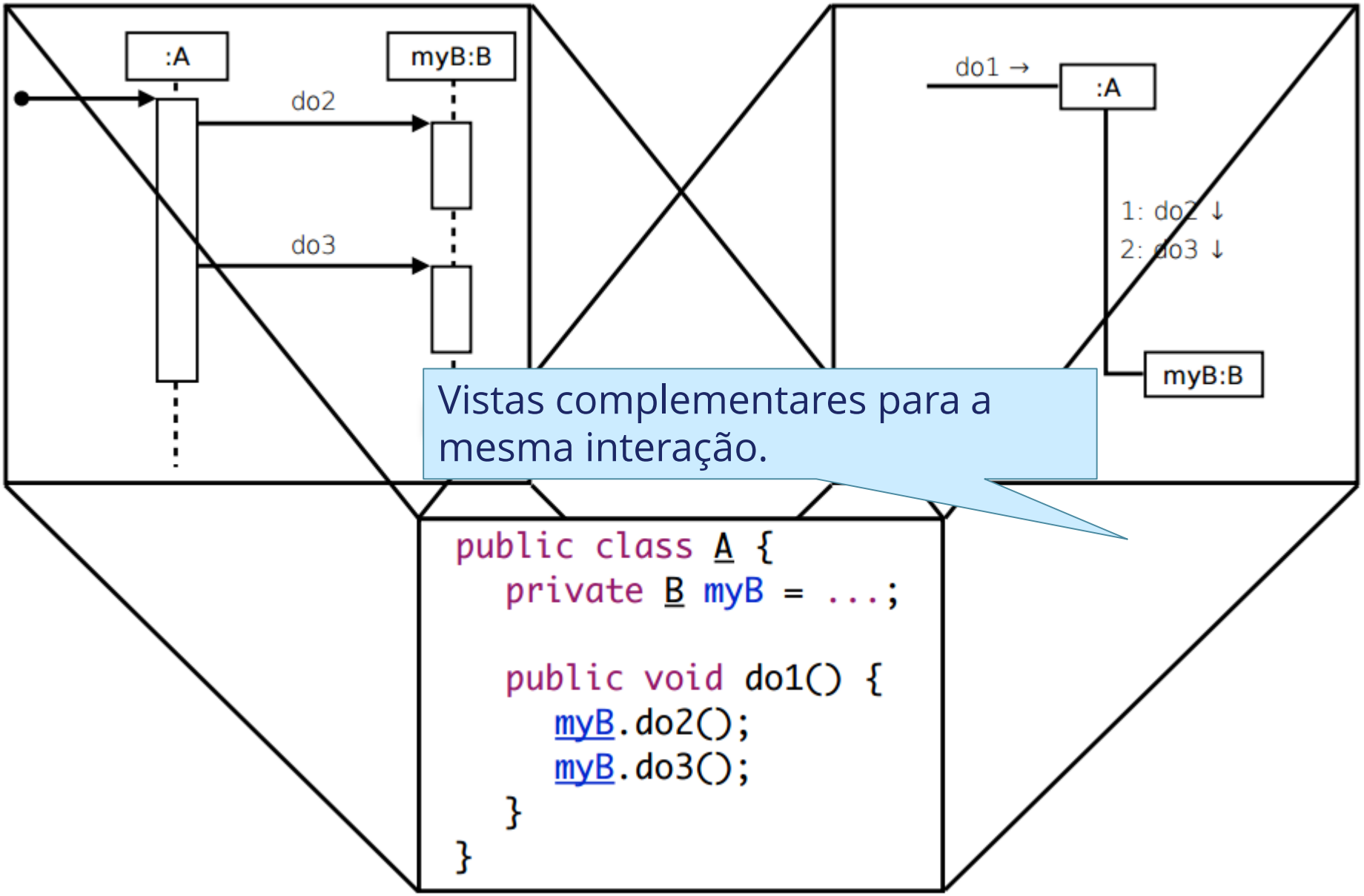

:A
myB:B
do2
do3
do1 →
:A
1: do2 ↓
2: do3 ↓
myB:B
Vistas complementares para a mesma interação.
public class A {
private B myB = ...;
public void do1() {
myB.do2();
myB.do3();
}
}

Figure Sequence Diagram for a Scenario of the Make Patient Appt Use Case

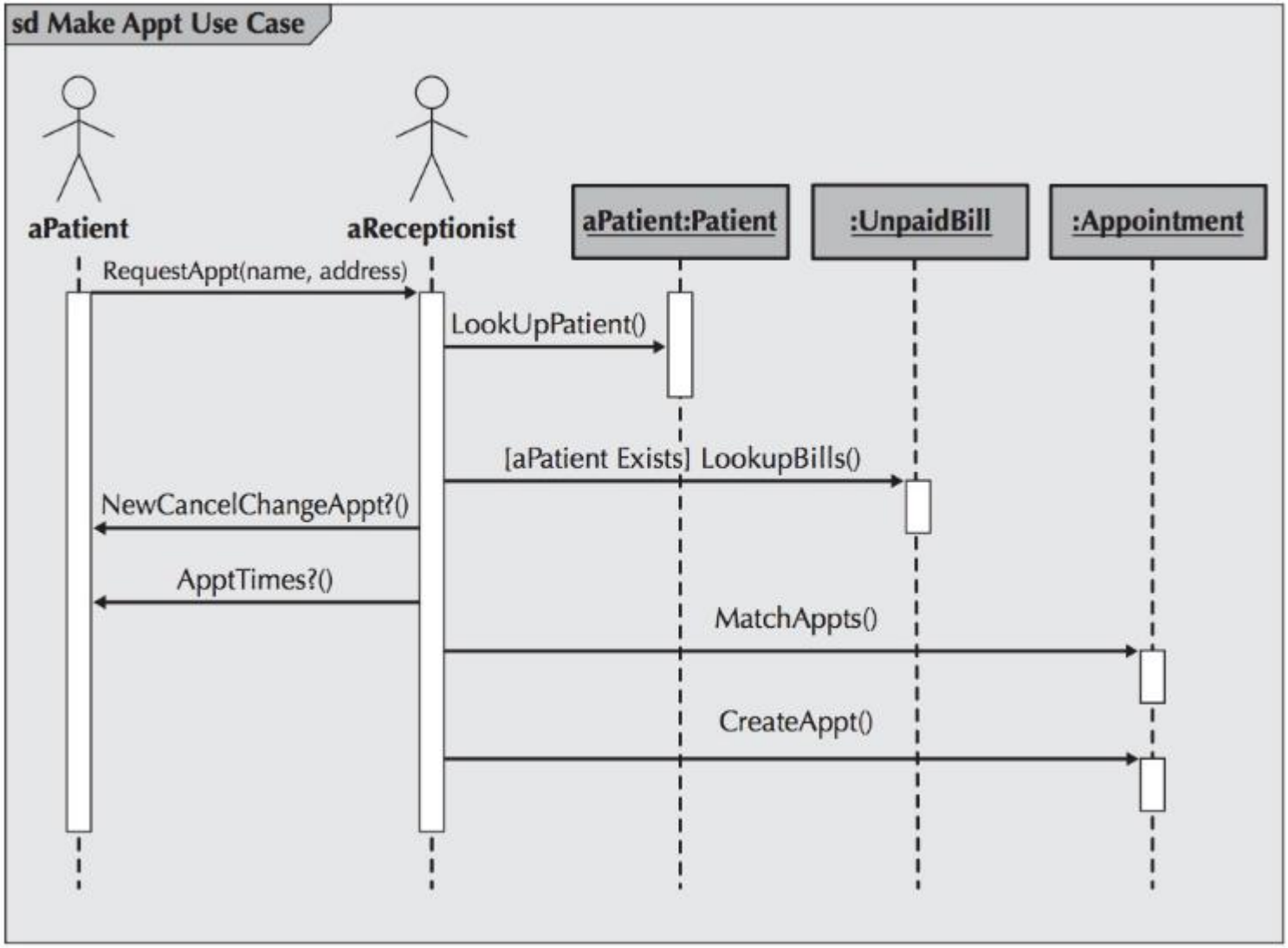
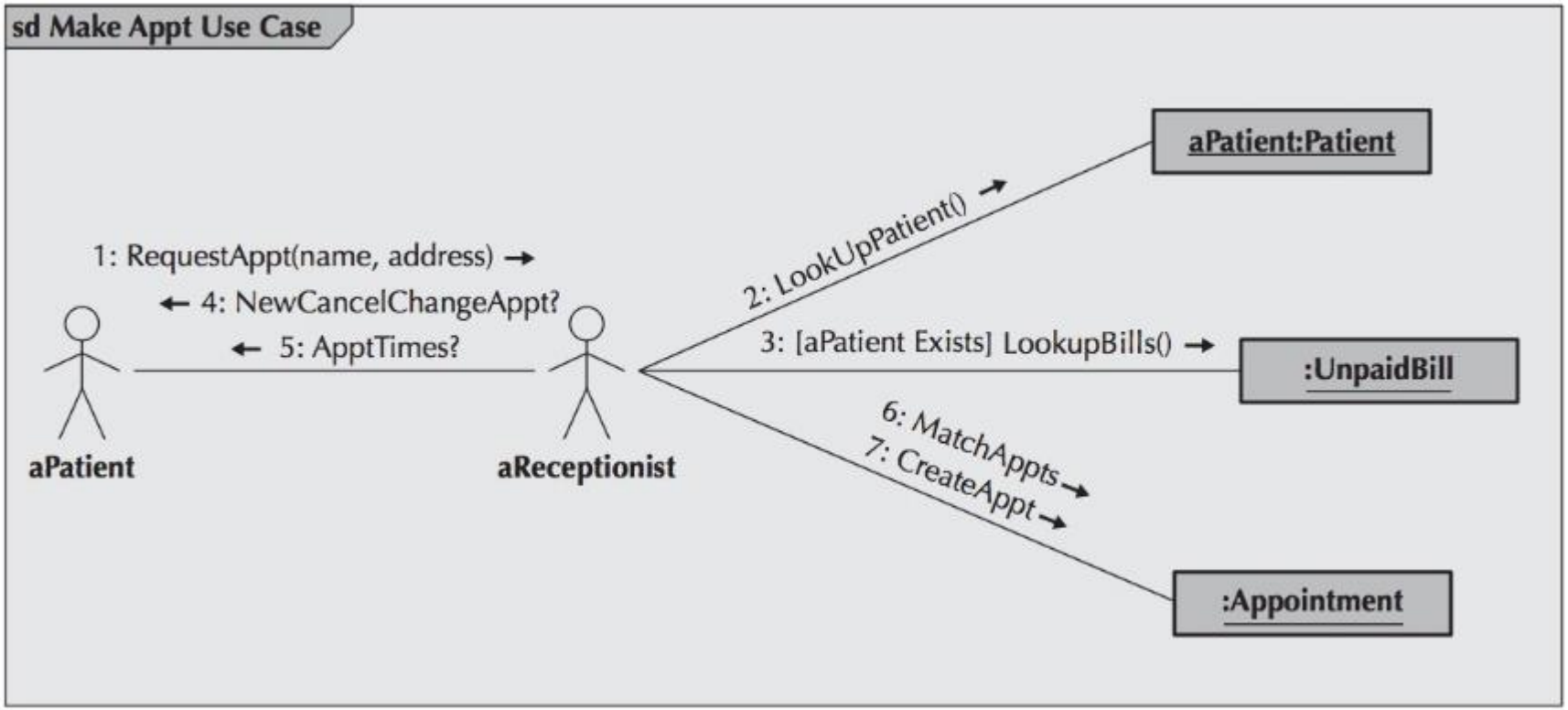


Figure Communication Diagram for a Scenario of the Make Patient Appt Use Case



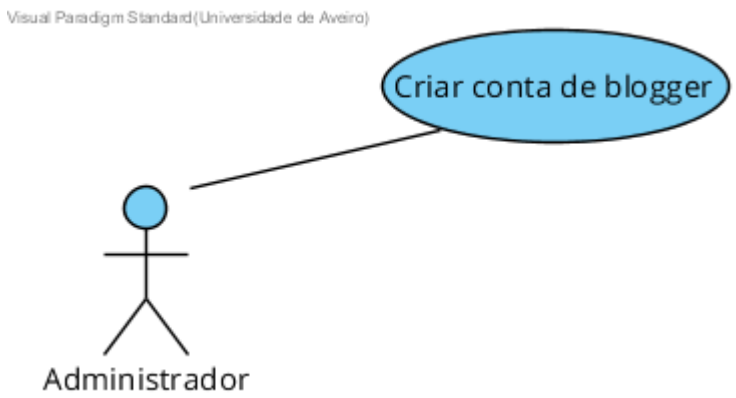
Diagramas de sequência *vs* diagramas de comunicação

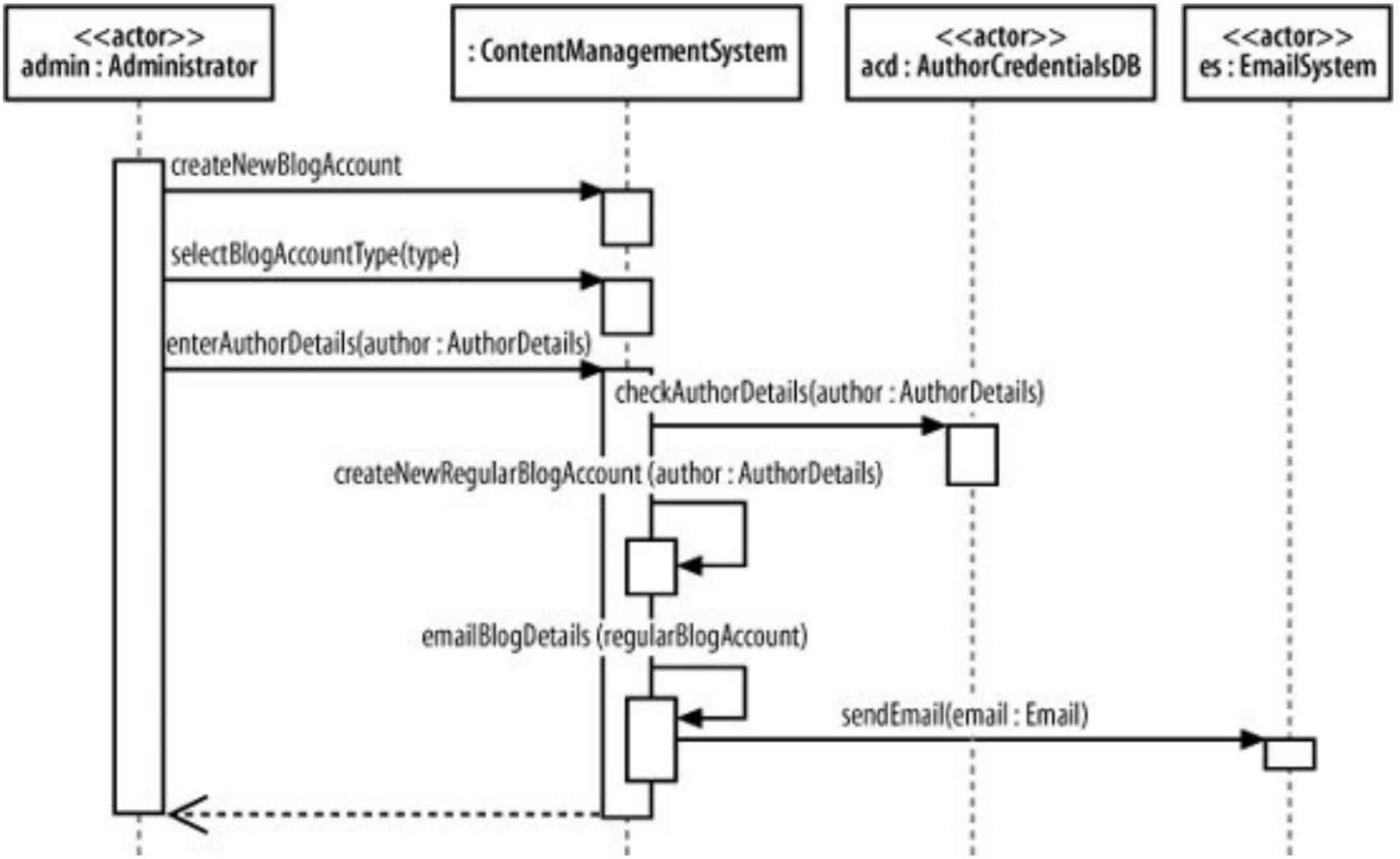
	Benefícios	Limitações
Diagrama de sequência	• Mostra claramente a sequência/ordem temporal das mensagens • Possibilidades de notação alargadas	• Cresce para a direita à medida que se acrescentam objetos
Diagrama de comunicação	• Mais fácil de desenhar (objetos podem ser adicionado em qq parte)	• Menos expressivo (ordem temporal) • Menos opções de notação • Pouco suportado nas ferramentas UML

Um caso de utilização é **realizado pela colaboração entre atores e sistema**

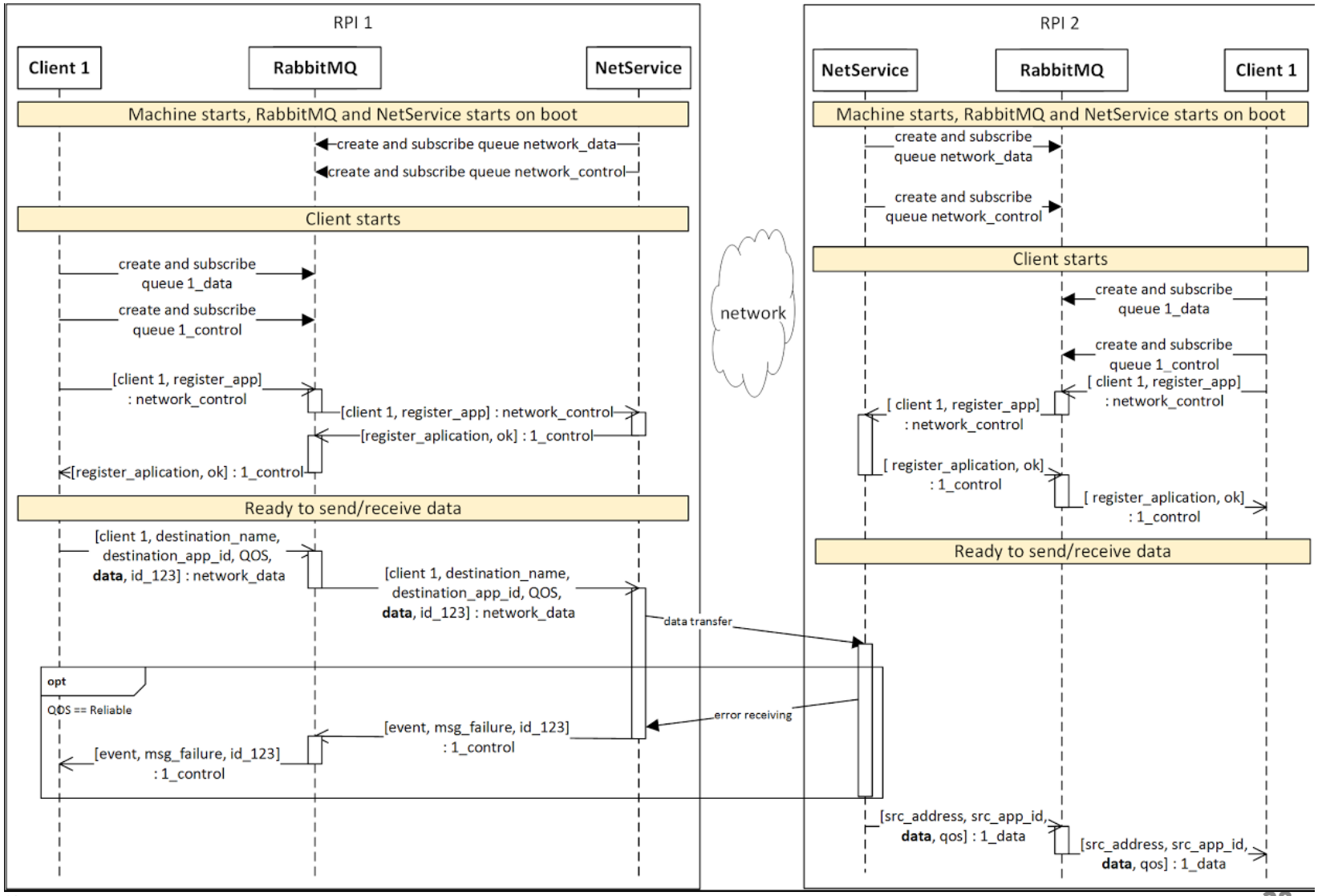
FLUXO TÍPICO

1. O Administrador pede ao sistema (CMS) para criar uma nova conta de *blogger*
2. O Administrador escolhe o tipo Normal
3. O Administrador fornece os detalhes do autor do blog
4. O CMS verifica os detalhes (se já existe) na base de Dados de Autores
5. O CMS cria a nova conta normal.
6. Um sumário dos detalhes da nova conta são enviados por email ao autor.




<<actor>>
admin : Administrator
: ContentManagementSystem
<<actor>>
acd : AuthorCredentialsDB
<<actor>>
es : EmailSystem
createNewBlogAccount
selectBlogAccountType(type)
enterAuthorDetails(author : AuthorDetails)
checkAuthorDetails(author : AuthorDetails)
createNewRegularBlogAccount (author : AuthorDetails)
emailBlogDetails (regularBlogAccount)
sendEmail(email : Email)

Outro exemplo de aplicação: **documentar um protocolo**



Diagramas de sequência de sistema (DSS)

Mostram, para um cenário de um CaU:

- os eventos que os atores externos geram,
- a sua ordem temporal,
- as necessidades de integração entre sistemas

Estilo para a construção

O sistema é tratado como uma caixa-fechada

Não mostra componentes internos

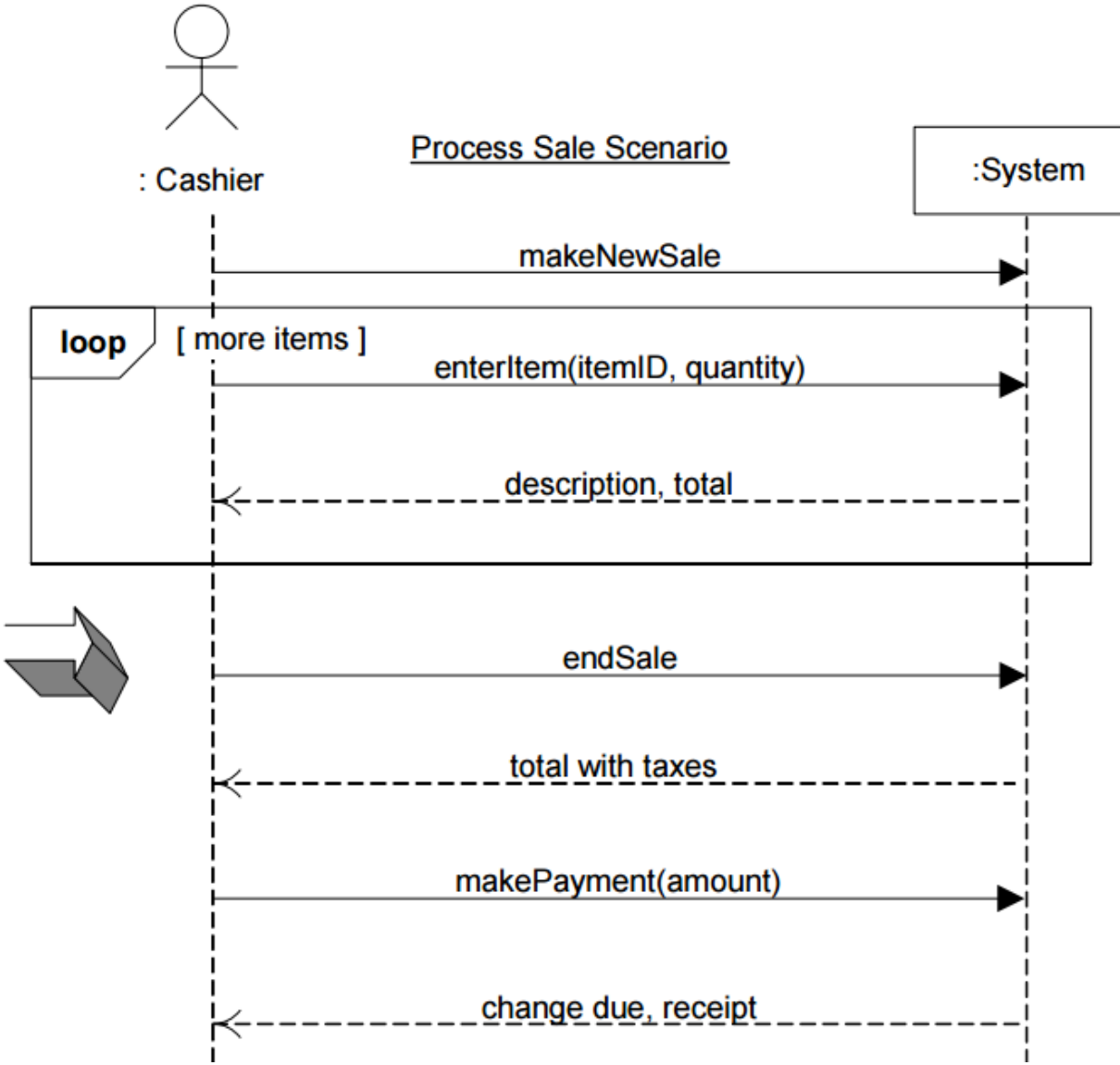
Mensagens que chegam ao Sistema são modeladas como operações

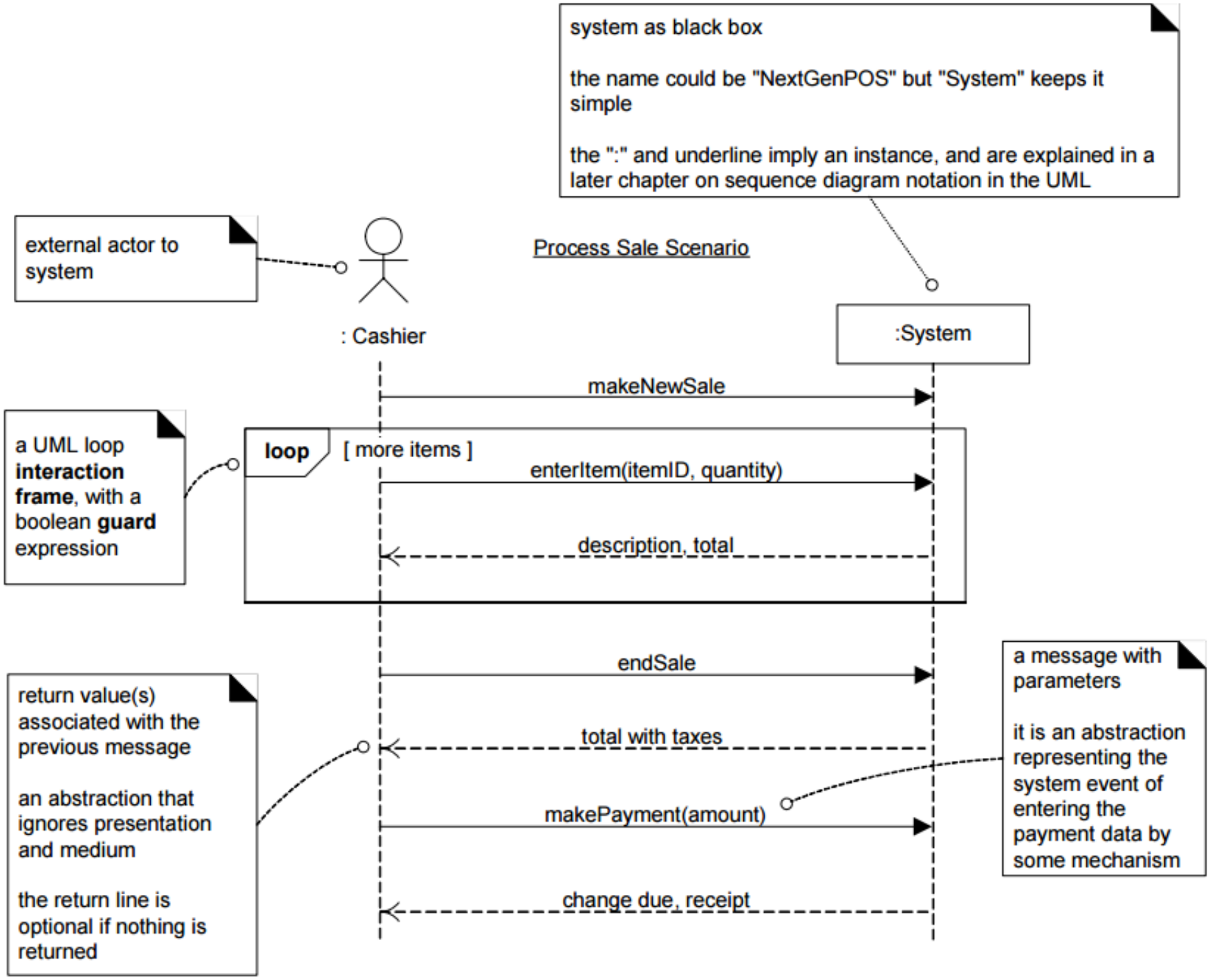
Os serviços solicitados/invocados pelos atores

Dos casos de utilização para o sistema

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.
...




system as black box
the name could be "NextGenPOS" but "System" keeps it simple
the ":" and underline imply an instance, and are explained in a later chapter on sequence diagram notation in the UML
external actor to system
Process Sale Scenario
: Cashier
:System
makeNewSale
a UML loop **interaction frame**, with a boolean **guard** expression
loop
[more items]
enterItem(itemID, quantity)
description, total
endSale
a message with parameters
it is an abstraction representing the system event of entering the payment data by some mechanism
return value(s) associated with the previous message
an abstraction that ignores presentation and medium
the return line is optional if nothing is returned
total with taxes
makePayment(amount)
change due, receipt

um novo diagrama de sequência de sistema (DSS) p/ cada CaU

uma *lifeline* para o actor primário (ou actores) e o sistema.

O sistema é modelado por uma classe que o representa globalmente

Opcionalmente, pode-se incluir o texto da descrição do CaU

Os DSS mostram operações de sistema

Uma **operação de sistema** (OpS) corresponde a um ponto de entrada no sistema

Encapsula um conjunto de interações subjacentes entre objetos, necessárias para realizar essa operação.

Etapa seguinte na construção do **modelo dinâmico:**

mostrar, para cada OpS, a rede de objetos que vai ser ativada.

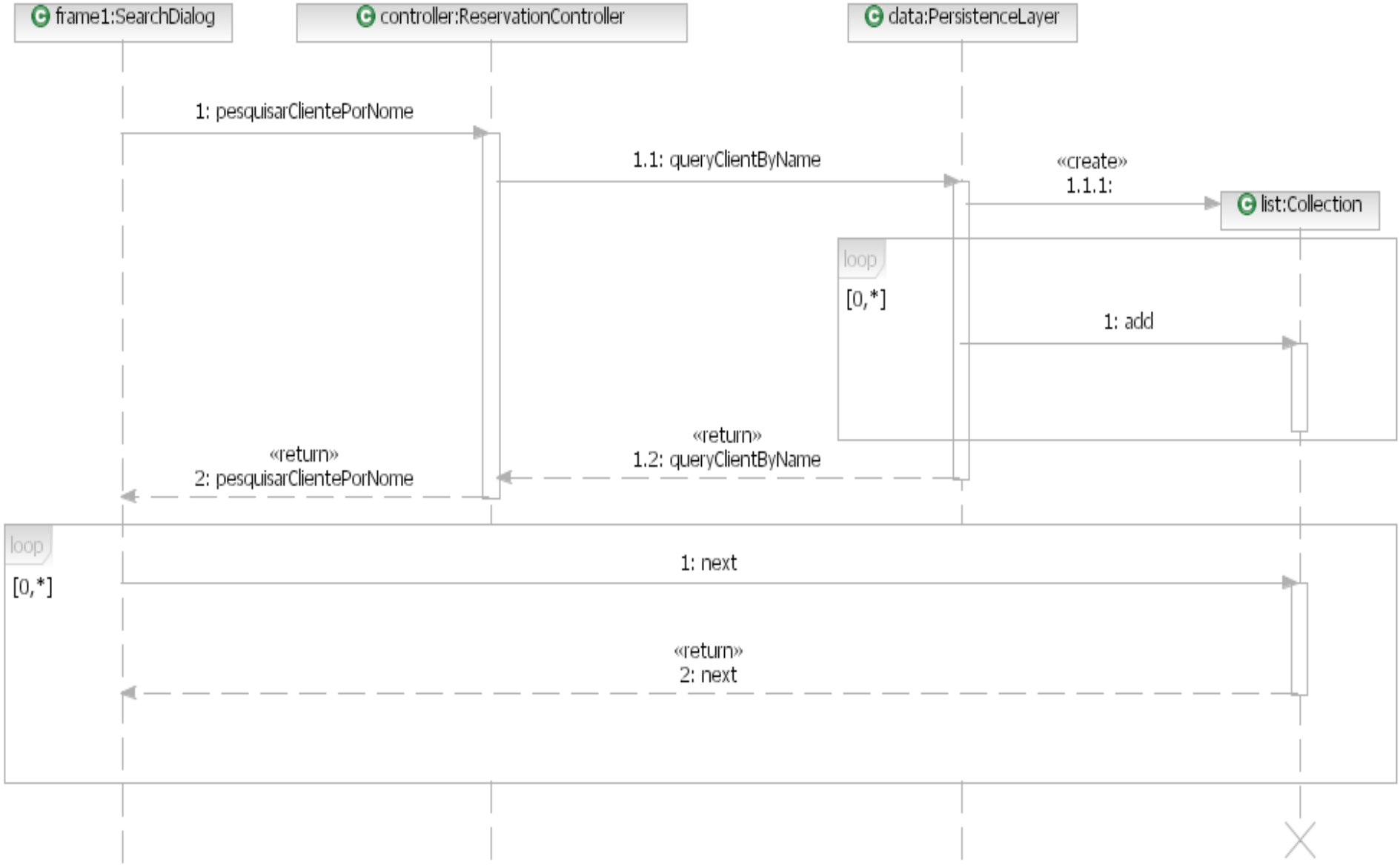
Estes objetos são instâncias de classes que devemos identificar

Como escolher as classes que participam na solução?

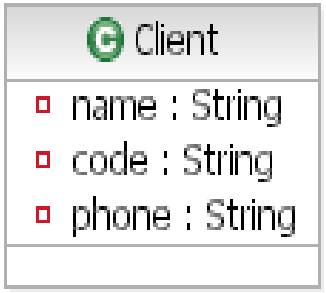
Essa atividade é o que designamos por desenho

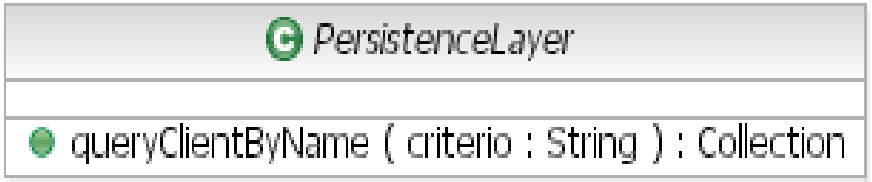
Ponto essencial: distribuir responsabilidades pelos objetos

seguindo os princípios do desenho por objectos (*Object-Oriented Design*).


frame1:SearchDialog
controller:ReservationController
data:PersistenceLayer
1: pesquisarClientePorNome
1.1: queryClientByName
«create»
1.1.1:
list:Collection
loop
[0,*]
1: add
«return»
1.2: queryClientByName
«return»
2: pesquisarClientePorNome
loop
[0,*]
1: next
«return»
2: next

Os diagramas de interacção ajudam a distribuir responsabilidades → encontrar os métodos





O objetos Java colaboram para realizar objetivos

```
public class Encomenda {
    private Cliente cliente;
    private ArrayList<ItemDaEncomenda> listaDeItens;
[…]
    public double getValorTotal() {
        double total = 0.0;

        Produto produto;
        for (ItemDaEncomenda umItem : listaDeItens ) {
            produto = umItem.getProduto();
            total += produto.getPrecoUnitario() * umItem.getQuantidade();
        }
        total = total * (1- cliente.getDescontoBase() );
        return total;
    }
}
```

Diagramas (gerados)

Diagramas demonstrados na TP:

- Classes, no VisualParadigm
- Sequência, no VisualParadigm
- Classes, no IntelliJ IDEA

Referências

[DEN'15] Dennis, A., Wixom, B. H., & Tegarden, D. (2015). *Systems analysis and design: An object-oriented approach with UML*. John Wiley & Sons.

→ chap. 6

[LAR'12] Larman, C. (2012). *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Interative Development*. Pearson Education.

→ chap. 10, chap. 15.