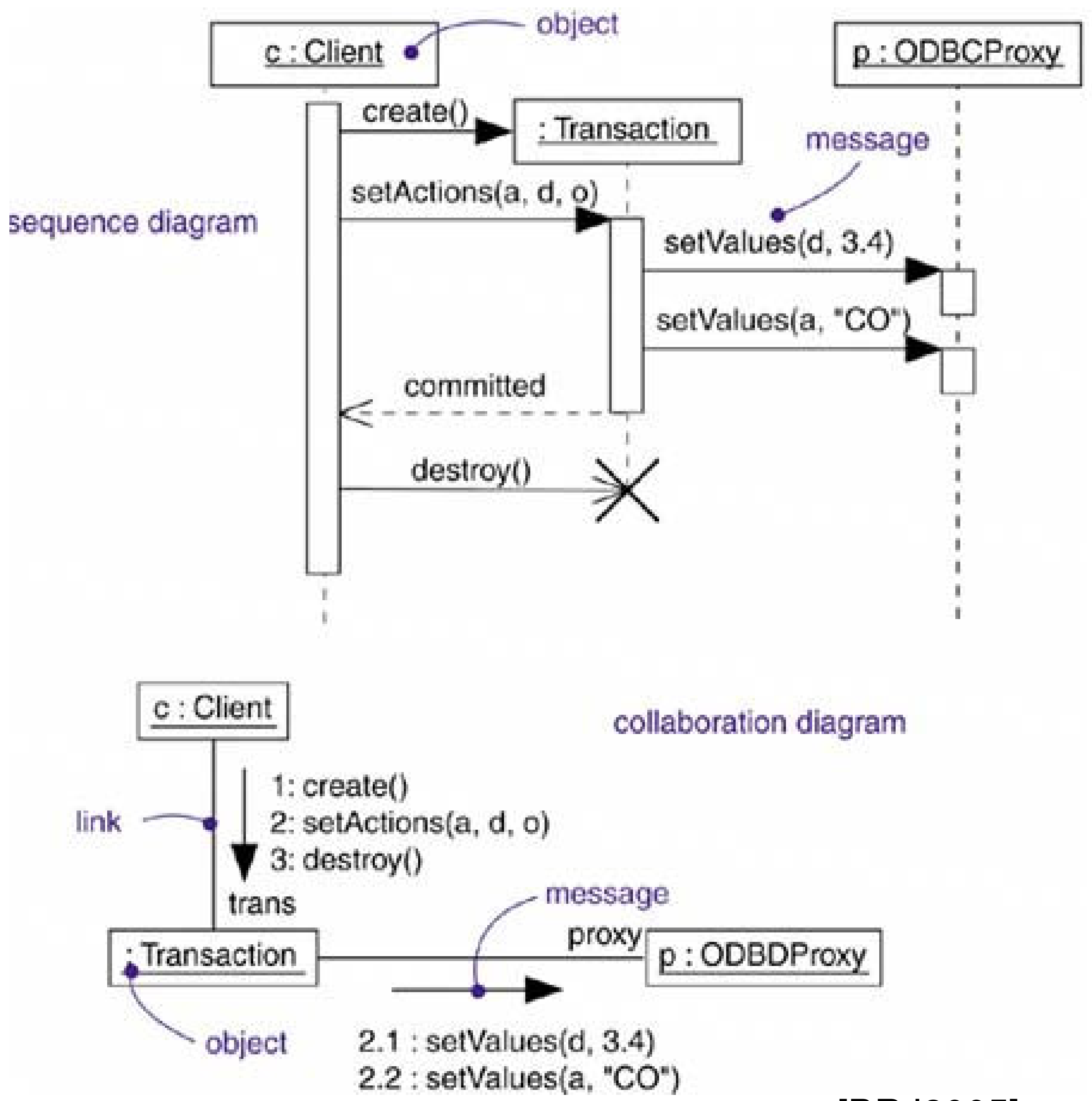
Analyse et conception

Cours de 1e année ingénieur

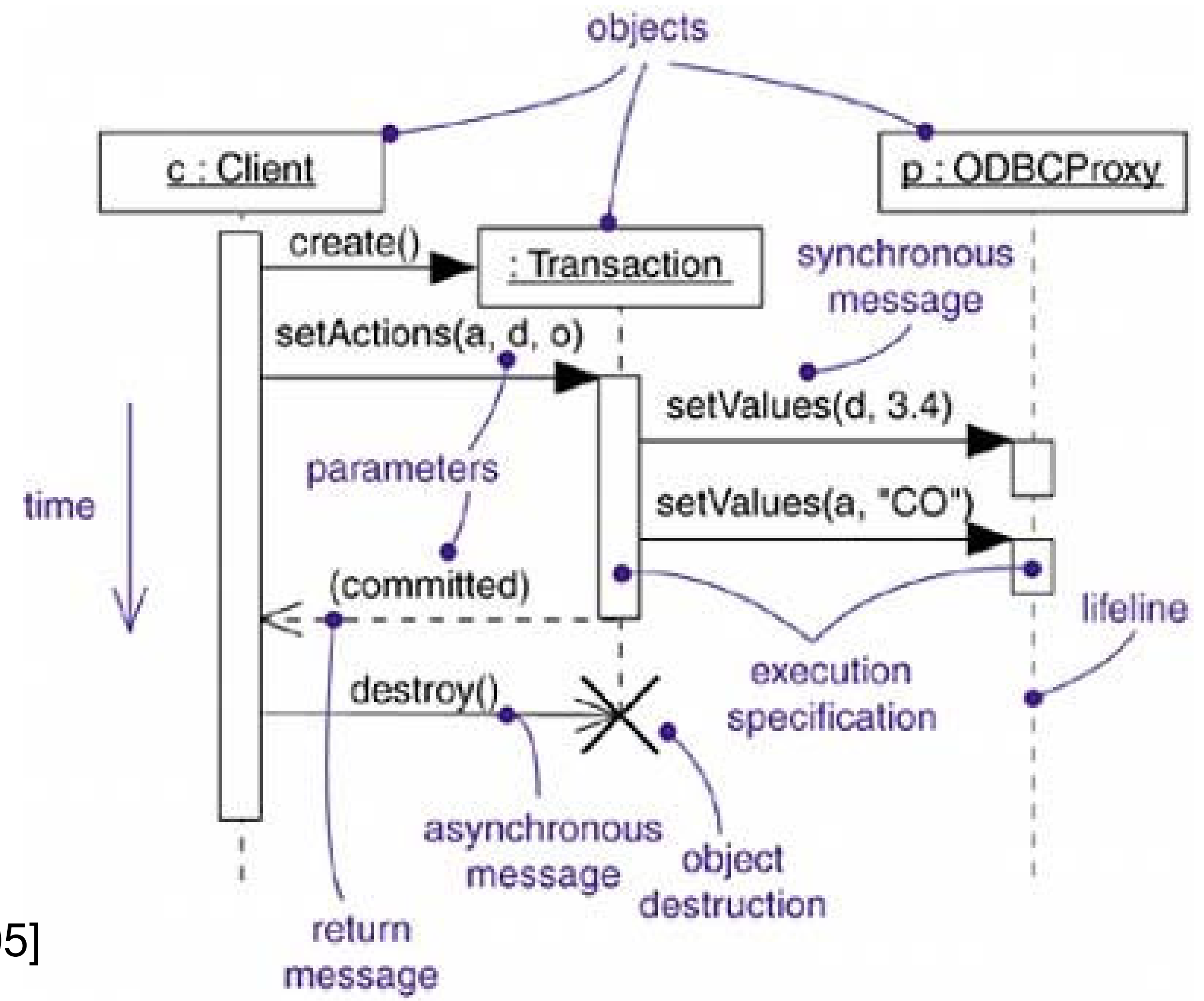
fabien.romeo@fromeo.fr
http://www.fromeo.fr

Diagrammes d’intéraction : 2 points de vues (temps vs. espace)



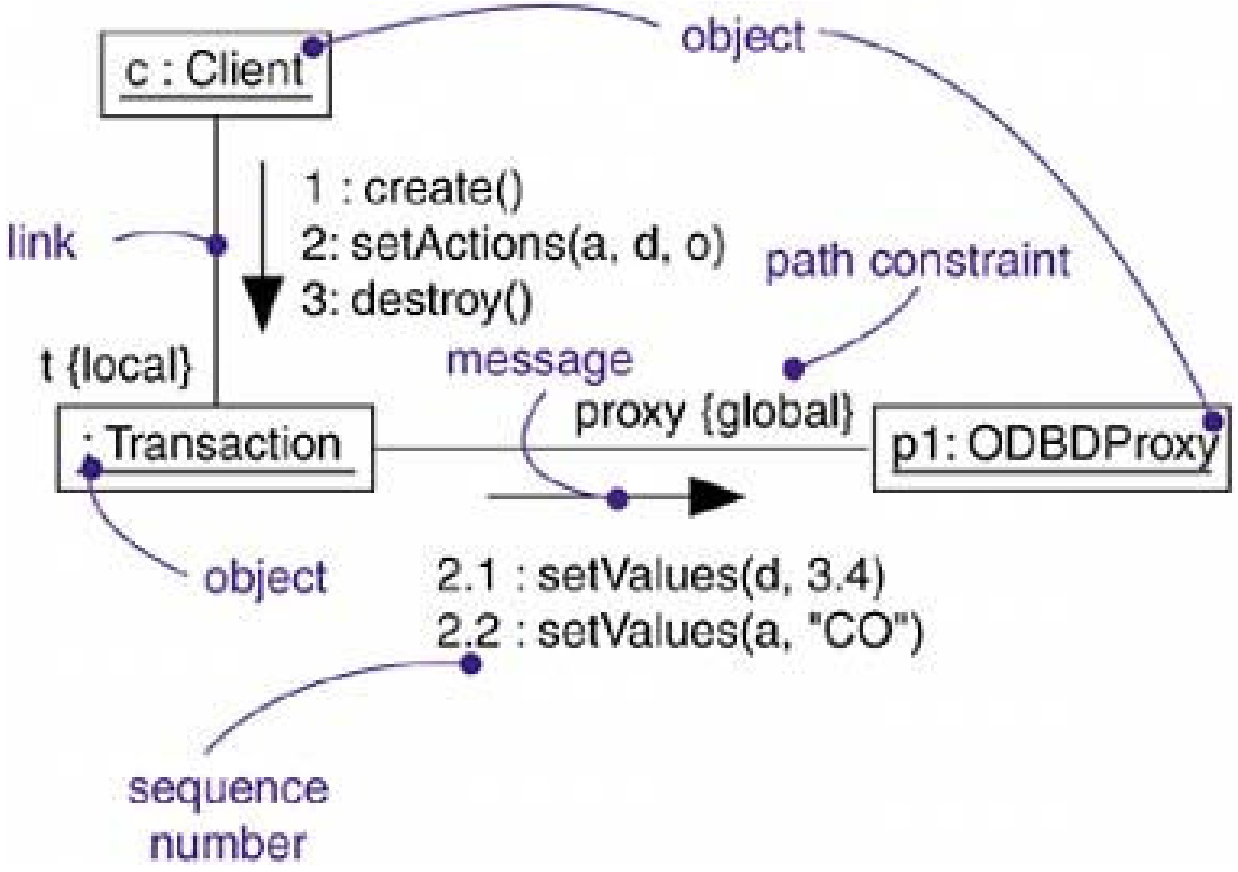
[BRJ2005]

Diagramme de séquence : notation UML 2



[BRJ2005]

Diagramme de communication : notation UML 2

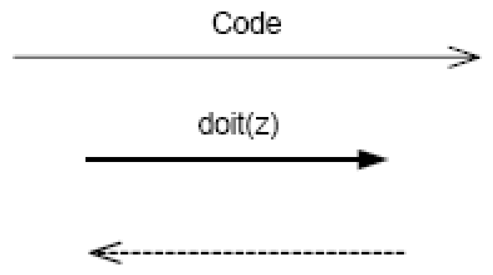


[BRJ2005]

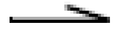
Messages

- A message is shown as a line from the sender message end to the receiver message end. The line must be such that every line fragment is either horizontal or downwards when traversed from send event to receive event. The send and receive events may both be on the same lifeline. The form of the line or arrowhead reflects properties of the message:
  - Asynchronous Messages have an open arrow head.
  - Synchronous Messages typically represent operation calls and are shown with a filled arrow head.
  - The reply message from a method has a dashed line.
  - Object creation Message has a dashed line with an open arrow.
  - Lost Messages are described as a small black circle at the arrow end of the Message.
  - Found Messages are described as a small black circle at the starting end of the Message.
  - On Communication Diagrams, the Messages are decorated by a small arrow in the direction of the Message close to the Message name and sequence number along the line between the lifelines
- Syntax for the Message name is the following:
  - *<messageident> ::= ([<attribute> ‘=’] <signal-or-operation-name> [‘(‘ [<argument> [‘,’<argument>]* ‘)’] [‘:’ <return-value>]) | ‘*’*
  - *<argument> ::= ([<parameter-name> '='] <argument-value>) | (<attribute> '=' <out-parameter-name> [':' <argument-value>]) | ' -'*

Messages : notation UML 2

Node Type	Notation	Reference
Message		Messages come in different variants depending on what kind of Message they convey. Here we show an asynchronous message, a call and a reply. These are all *complete* messages. See "Message (from BasicInteractions)" on page 491.

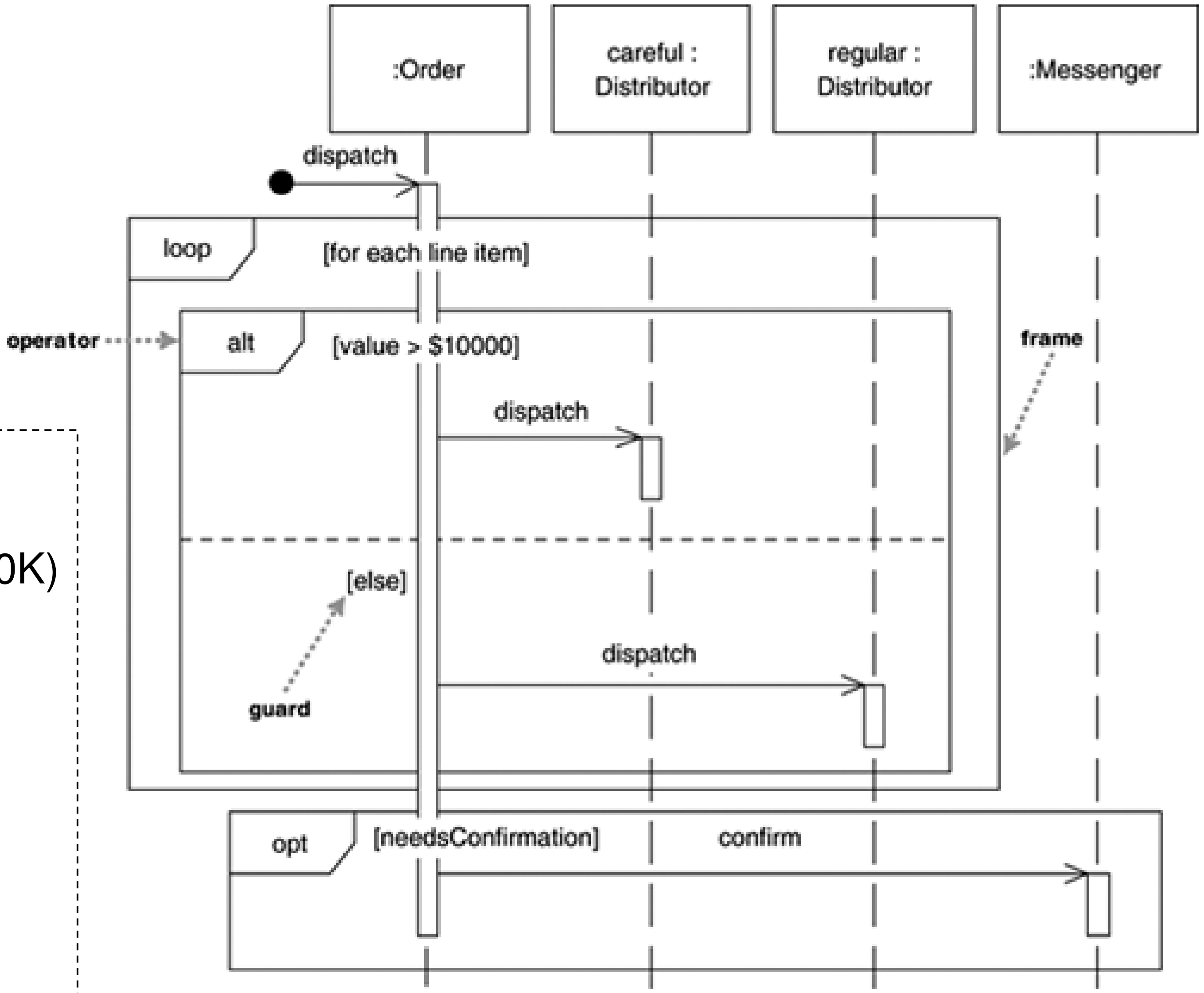
Messages : notation UML 1.3

half stick arrowhead 

Asynchronous flow of control. Used instead of the stick arrowhead to explicitly show an asynchronous communication between two Objects in a procedural sequence.

Interaction Frames

[Fowler2003]



```
procedure dispatch
 foreach (lineitem)
  if (product.value > $10K)
    careful.dispatch
  else
    regular.dispatch
  end if
 end for
 if (needsConfirmation)
   messenger.confirm
end procedure y
```

Interaction Frames - Opérateurs

- **alt** Alternative multiple fragments; only the one whose condition is true will execute.
- **opt** Optional; the fragment executes only if the supplied condition is true. Equivalent to an alt with only one trace.
- **par** Parallel; each fragment is run in parallel.
- **loop** Loop; the fragment may execute multiple times, and the guard indicates the basis of iteration.
- **region** Critical region; the fragment can have only one thread executing it at once.
- **neg** Negative; the fragment shows an invalid interaction.
- **ref** Reference; refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
- **sd** Sequence diagram; used to surround an entire sequence diagram, if you wish.
- **...**

Utilisation sur un exemple des 4 diagrammes UML étudiés jusqu’ici

- Un logiciel permettant à un commercial de calculer le prix d’une commande donnée
- Une commande est un ensemble de lignes de commande et est associé à un client
- Un ligne de commande est définie par
  - Une quantité
  - Un produit
- Un produit contient un ensemble de détails pour sa tarification
- Un client peut avoir des remises

(les diagrammes qui suivent sont réalisés avec starUML :
http://staruml.sourceforge.net)

Diagramme de use cases : calcul du prix d’une commande

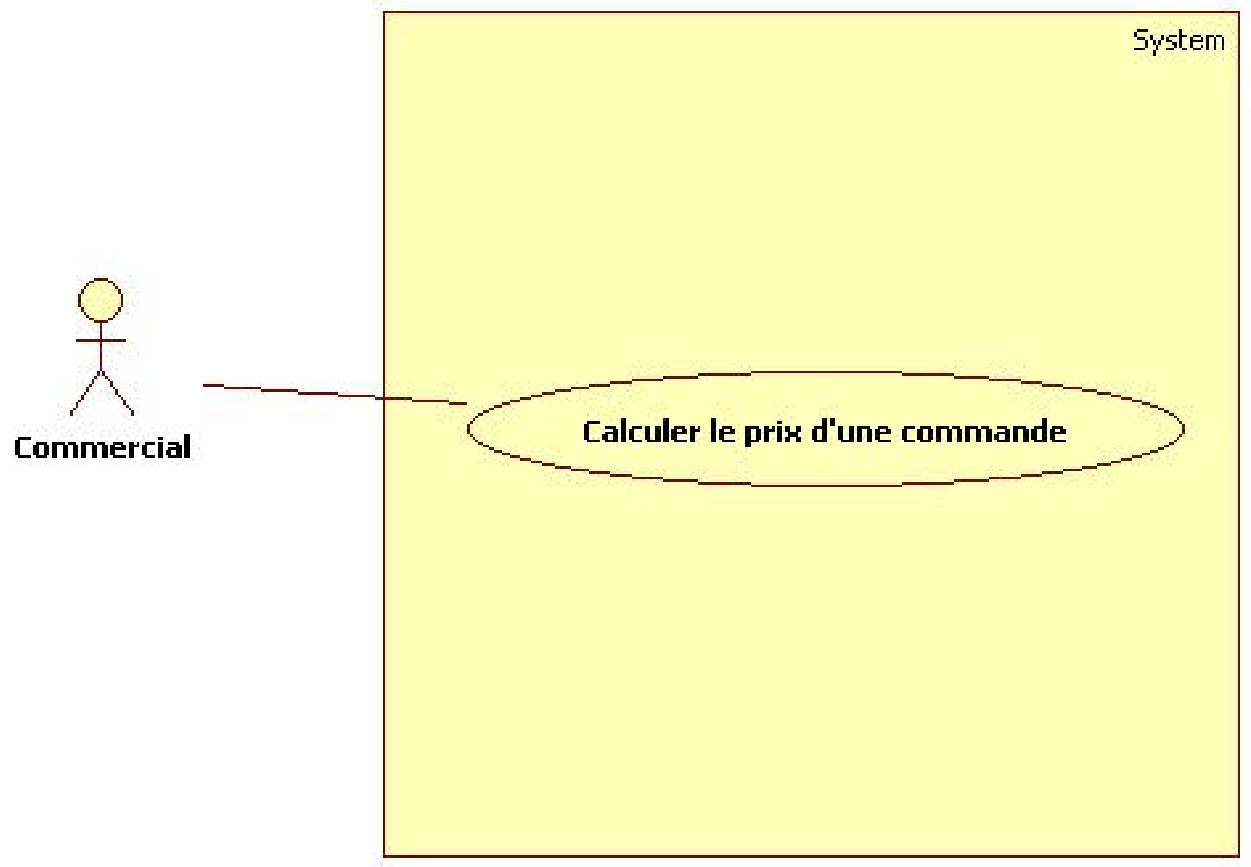


Diagramme de classes : calcul du prix d’une commande

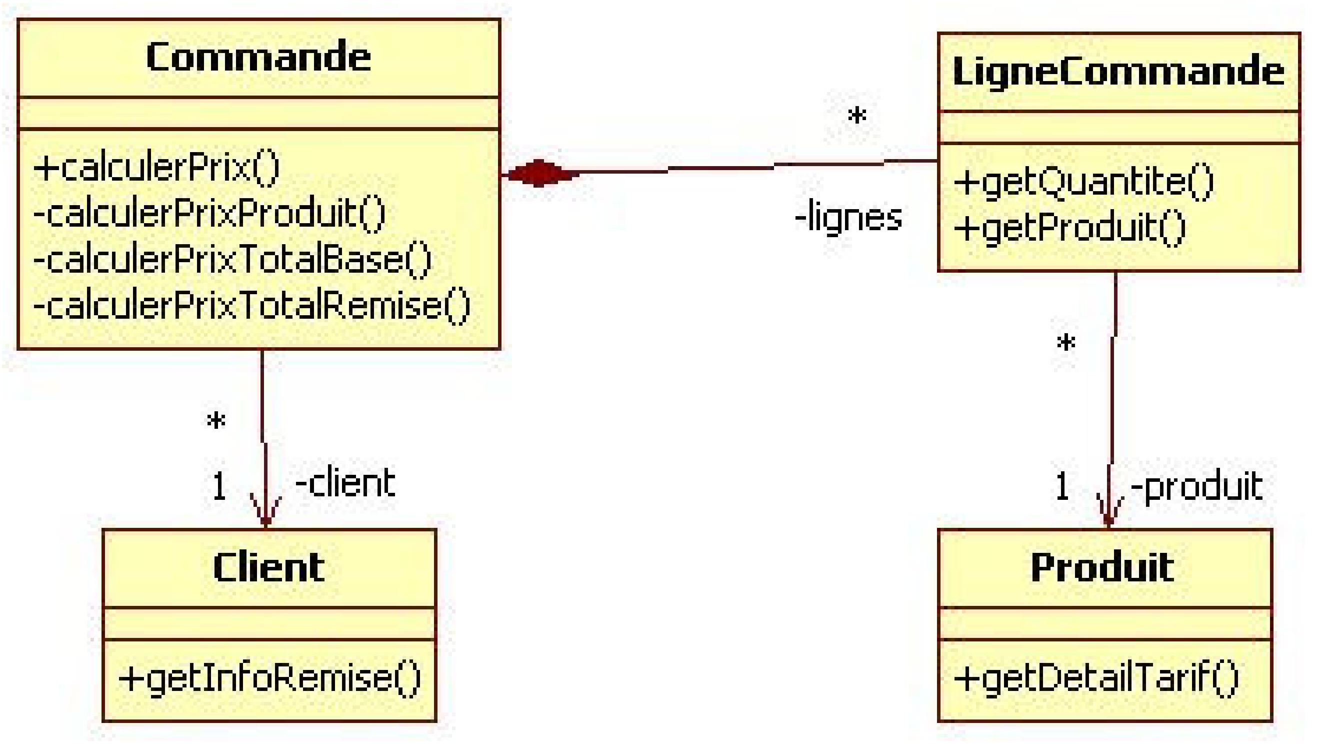


Diagramme de séquence : Calcul du prix d'une commande

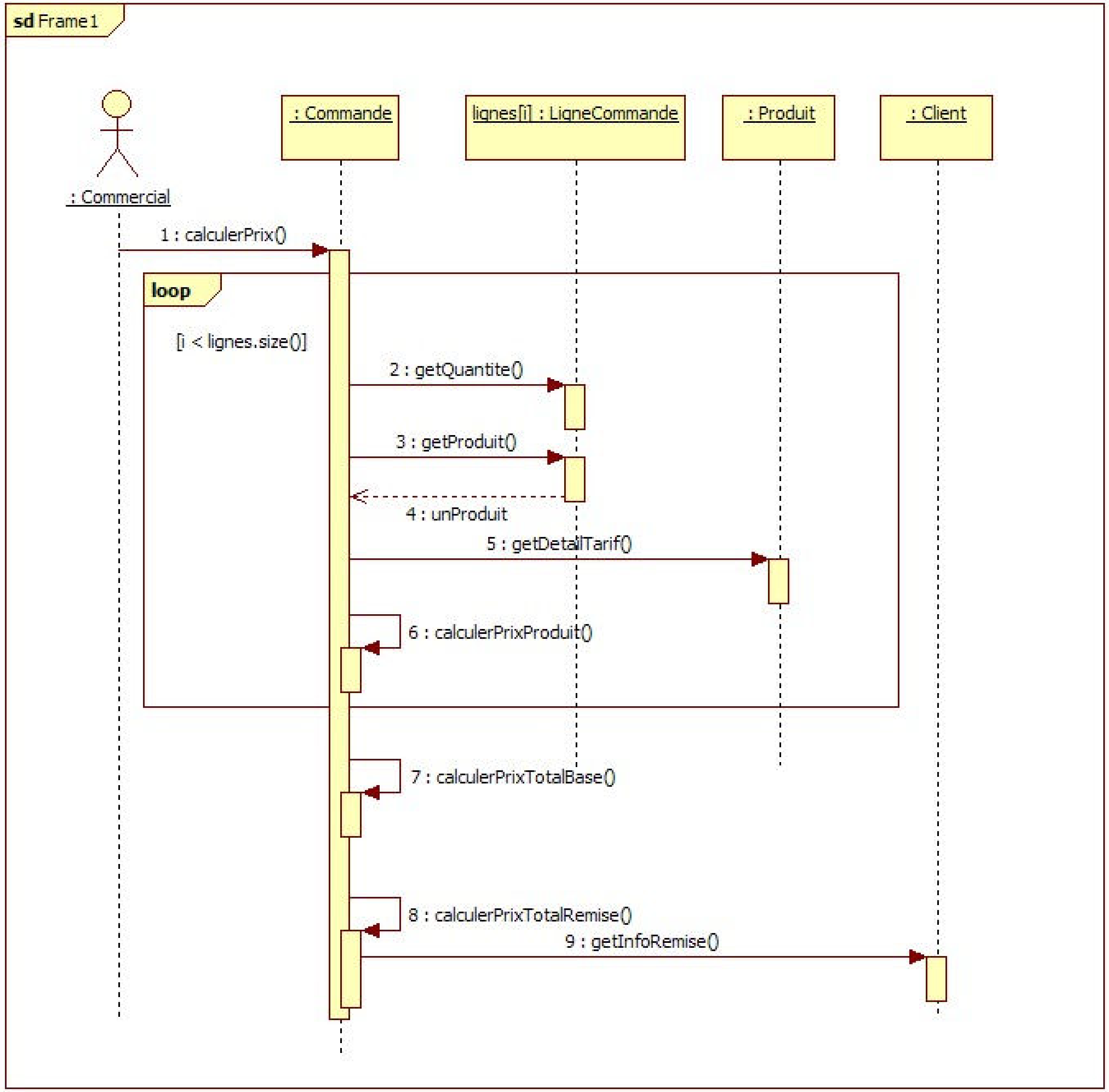
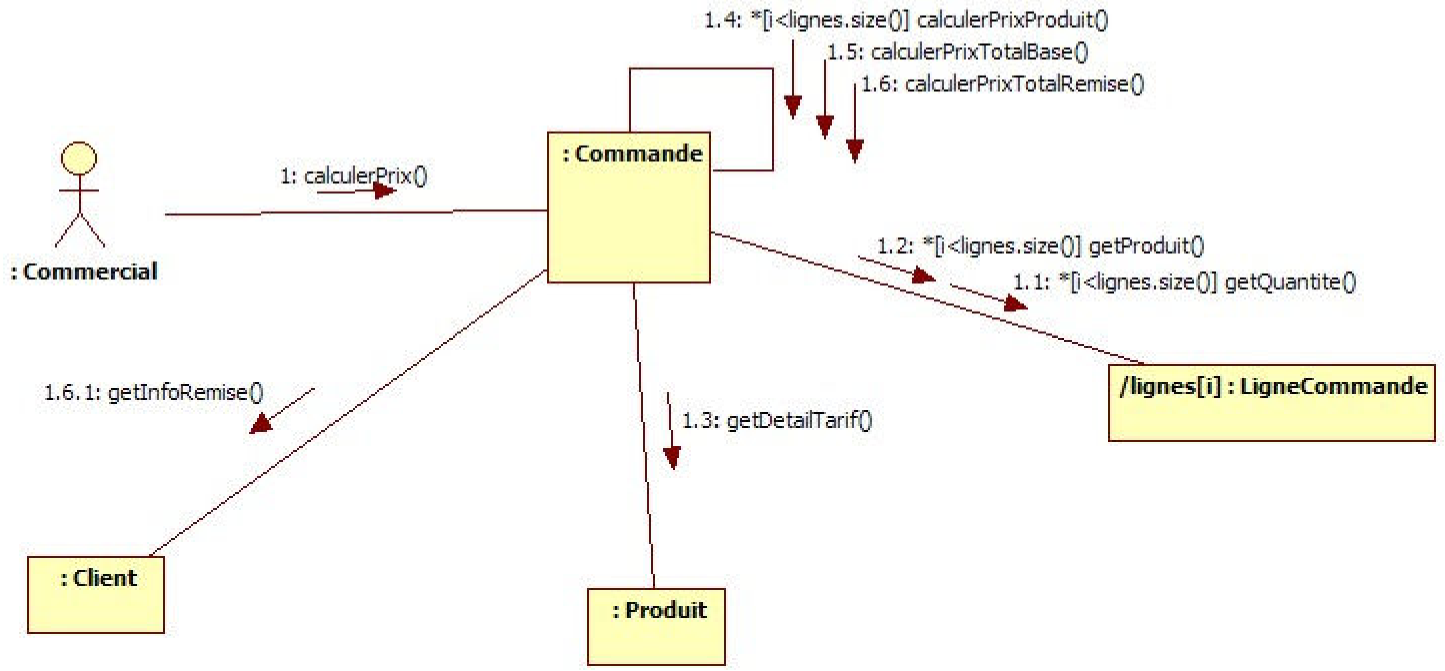


Diagramme de communication : calcul du prix d'une commande


1.4: *[i<lignes.size()] calculerPrixProduit()
1.5: calculerPrixTotalBase()
1.6: calculerPrixTotalRemise()
: Commande
1: calculerPrix()
: Commercial
1.2: *[i<lignes.size()] getProduit()
1.1: *[i<lignes.size()] getQuantite()
/lignes[i] : LigneCommande
1.6.1: getInfoRemise()
1.3: getDetailTarif()
: Client
: Produit