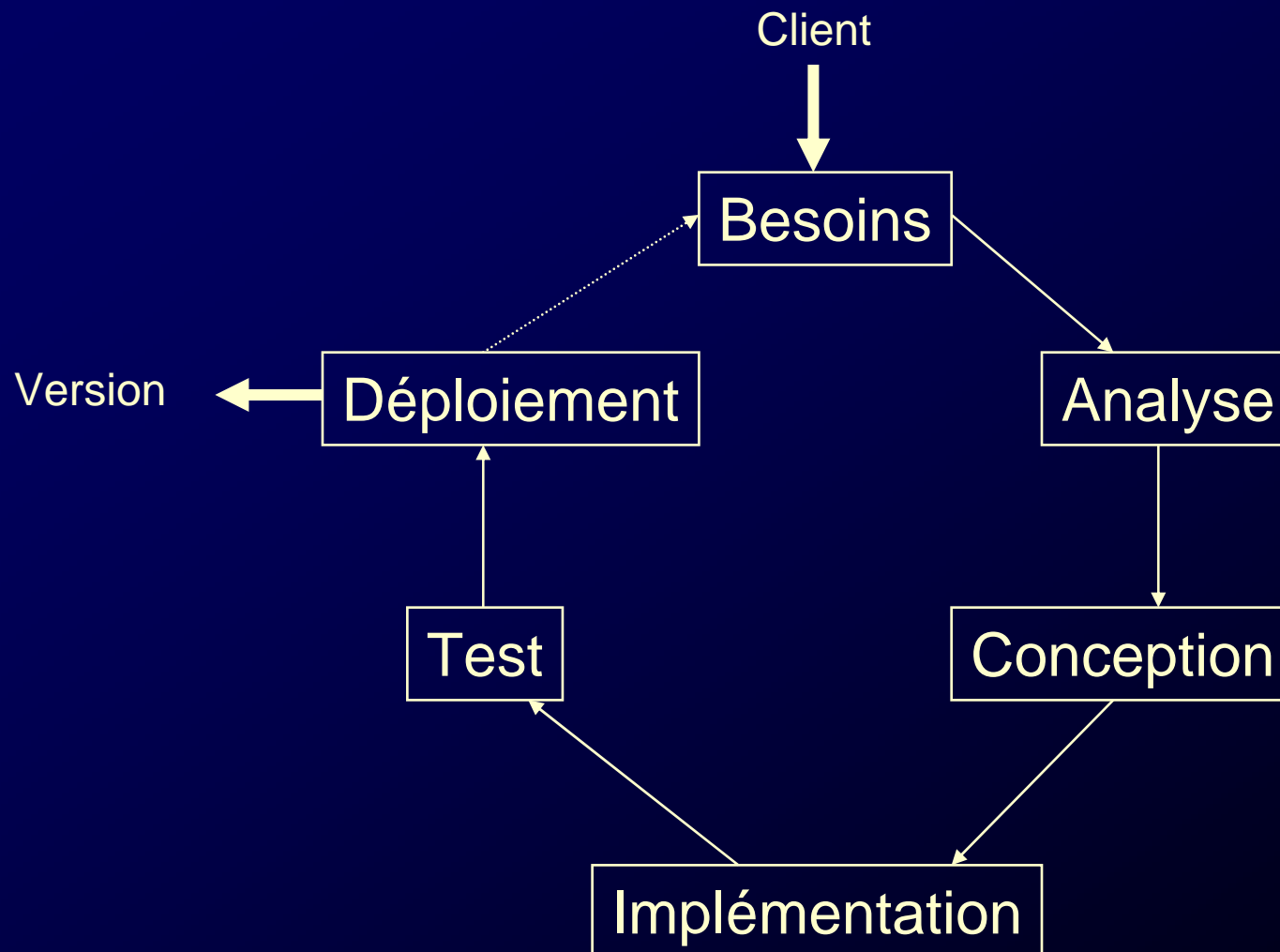


# *Unified Modeling Language*

---

# *Cycle de vie du logiciel*

---



# Développement de logiciel

---

## ◆ A faire ....

- *comprendre et conceptualiser* le problème (besoins, analyse)
- *résoudre* le problème (conception)
- donner une *solution* (implémentation)
- *documenter*

## ◆ UML: langage pour ...

- *spécifier, visualiser et comprendre* le problème
- *capturer, communiquer et utiliser* des connaissances pour la résolution du problème
- *spécifier, visualiser, et construire* la solution
- *documenter* la solution

# *UML: définition*

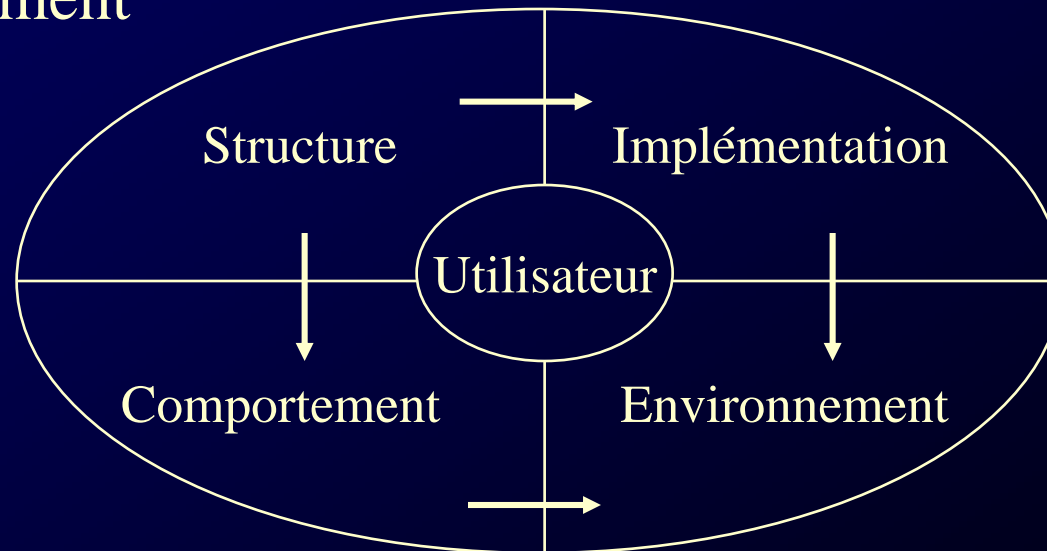
---

- ◆ UML: Unified **Modeling** Language
  - langage servant à **décrire des modèles** d'un système (réel ou logiciel) basé sur des concepts orienté-objets
  - système de notations pour modéliser les systèmes en utilisant des concepts orienté-objets
  - Langage de modélisation visuelle qui permet de:
    - spécifier le problème et la solution
    - visualiser le problème et la solution sous différents angles
    - construire la solution
    - documenter

# Modèles et Vues

---

- ◆ Les modèles du système sont visualisés par des vues
  - Modéliser: créer différentes vues du système
  - Chaque participant au développement voit le système différemment



# *UML et Vues*

---

- ◆ Vue Utilisateur
  - buts et objectifs des clients du système
  - besoins requis par la solution
- ◆ Vue Structurelle
  - aspects statiques, représentant la structure du problème
- ◆ Vue Comportementale
  - aspects dynamique, du comportement du problème et de sa solution
  - interactions et collaborations entre éléments de la solutions

# *UML et Vues*

---

- ◆ Vue Implémentation
  - aspects de structure et de comportement de la solution
- ◆ Vue Environnementale
  - aspects de structure et de comportement du domaine dans lequel la solution est réalisée

# *Vues*

---

- ◆ Les vues doivent être consistantes entre elles
- ◆ Accompanyer le cycle de vie du logiciel
- ◆ Modifier une vue implique une modification des autres vues



# Diagrammes UML

---

## ◆ Diagramme

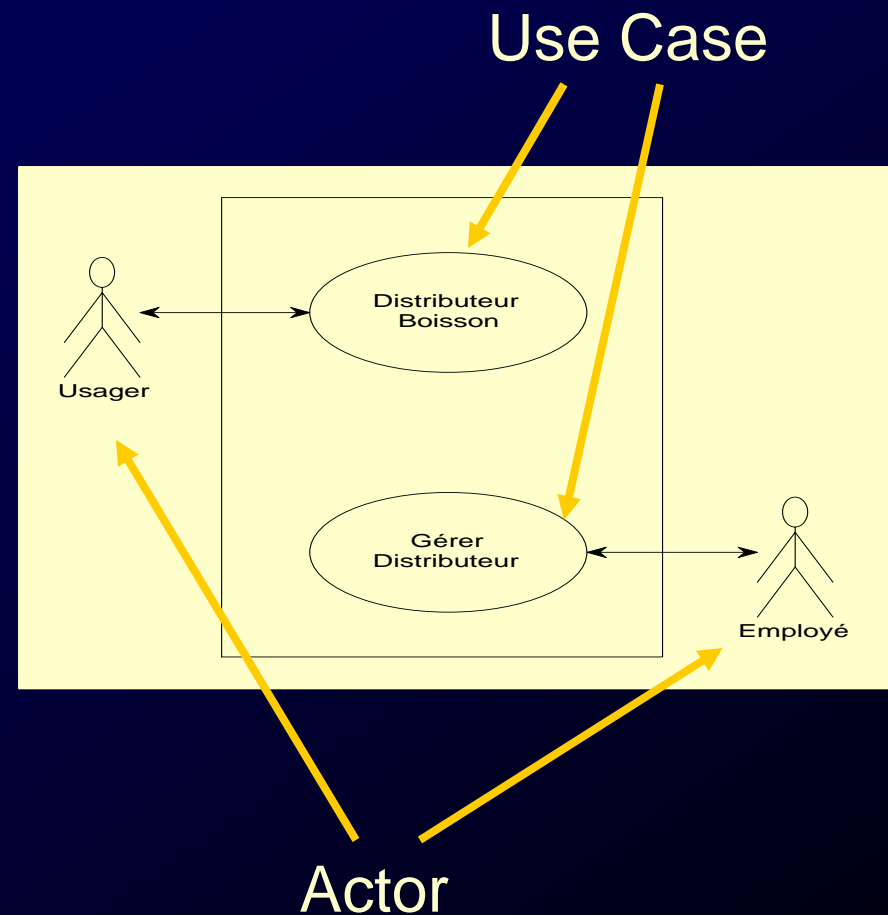
- Les diagrammes fournissent les informations sur le problème et sa solution
- Les diagrammes UML forment des **modèles** du système (spécifier, visualiser)
- Les combinaisons de diagrammes représentent les **vues** du système
- Diagramme: **Graphe** ayant des Nœuds et des Arcs

## ◆ Deux Types de diagrammes:

- Aspects dynamiques: comportement, collaborations, responsabilités
- Aspects statiques et structurels: relations

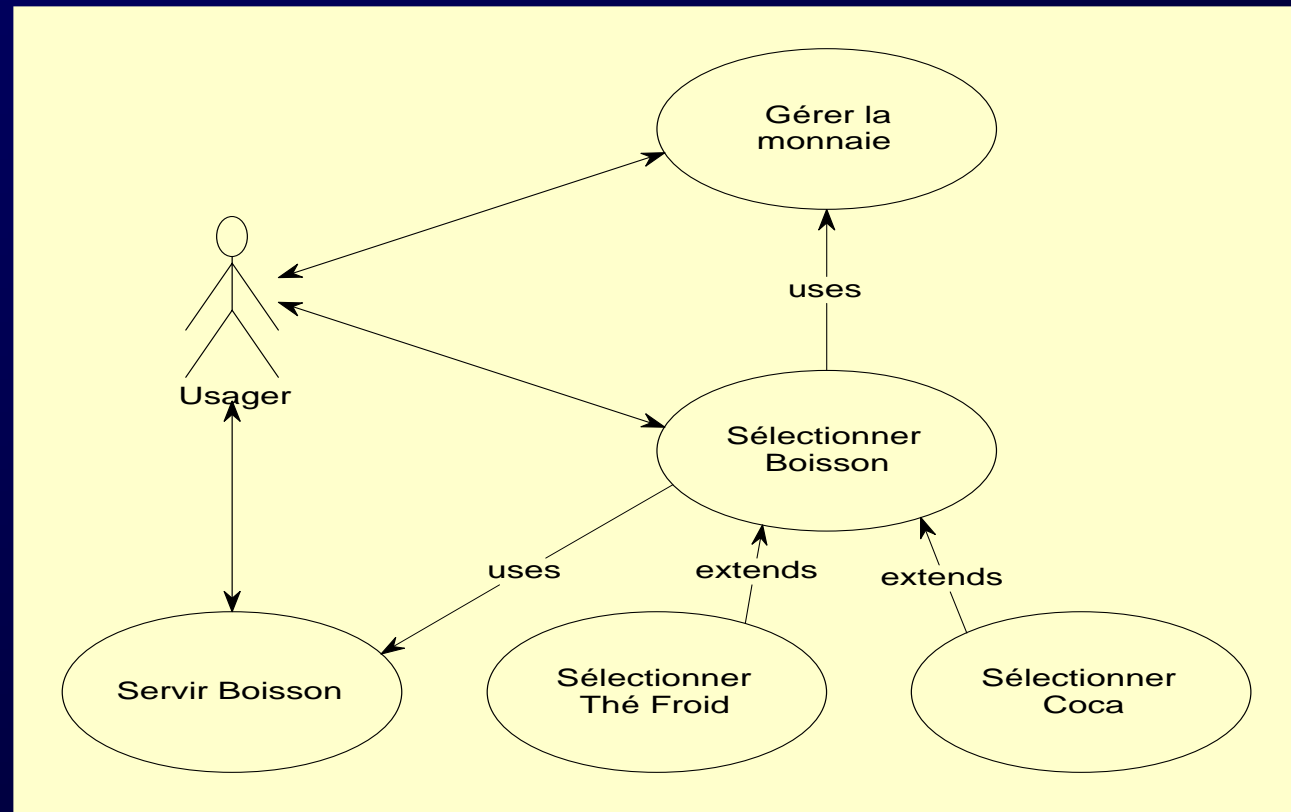
# Use Cases Diagrams

- ◆ Décrit la **fonctionnalité** que le système délivre à ses **utilisateurs** (humains ou système), et les **liens** entre eux
- ◆ **Actor**: rôle qu'un utilisateur joue dans le système
- ◆ **Use Case**: séquence d'actions que le logiciel garantit. Il définit les *besoins* du client.
- ◆ **Arcs**: use/include, extends (options)



# Use Case Diagrams

## Distributeur Boisson



# Class Diagrams

---

- ◆ Décrivent la **structure statique** du système à l'aide de classes, packages, et relations

- ◆ **Nœuds:**

- Packages

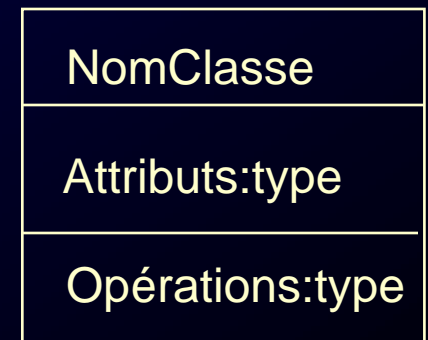


- Interfaces



- Classes: Attributs, Opérations (visibilité et paramètres)




- + public
    - - privé
    - # protégé
    - package



# *Class Diagrams*




---

## ◆ Arcs:

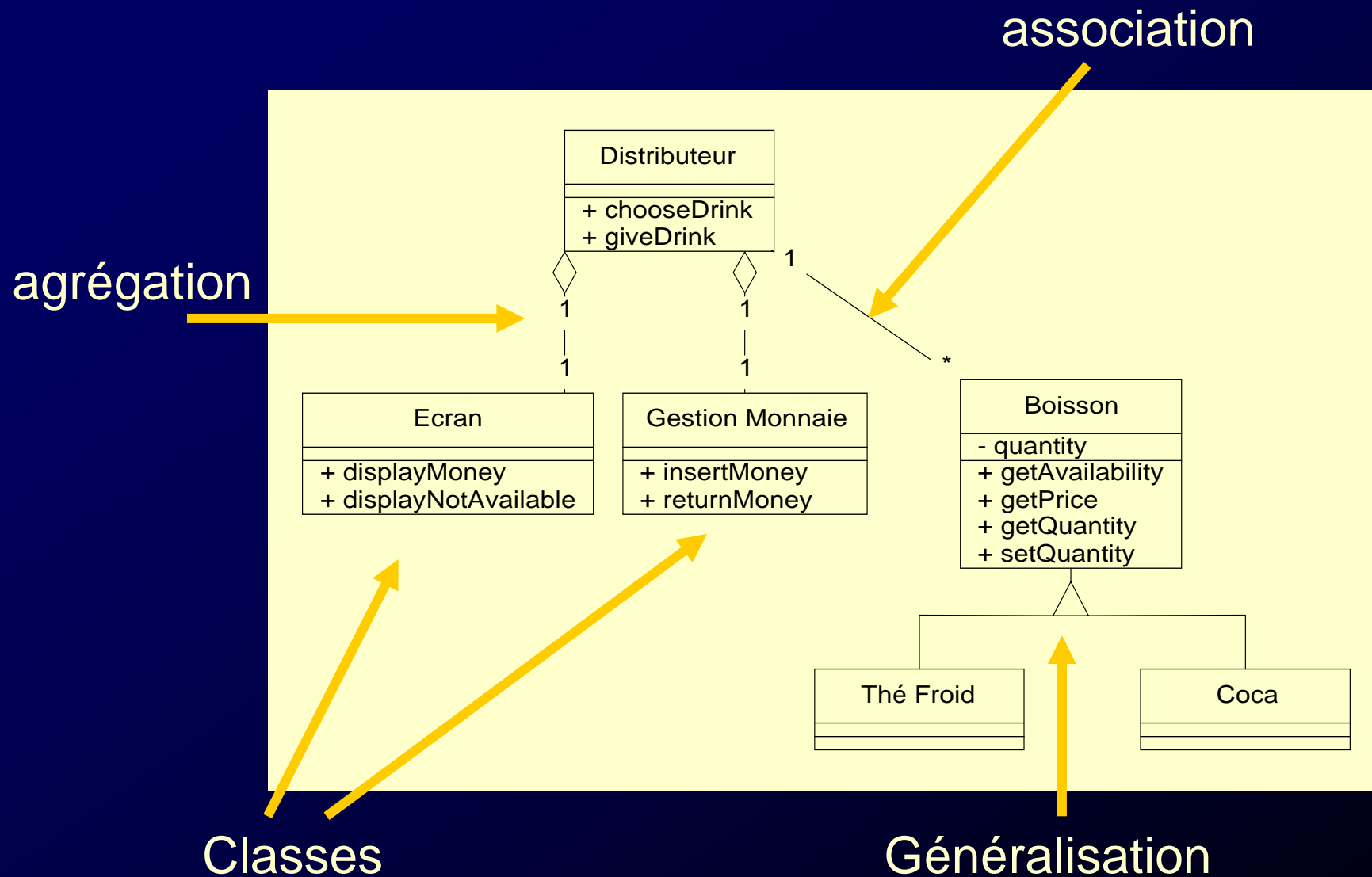
- dépendance →
  - référence à des classes ou objets passés en paramètres ou statiques (« use »)
  - relations entre package
- association →
  - navigabilité entre objets, message entre objets
  - flèche est optionnelle
- agrégation →
  - « has-a »

# *Class Diagrams*

---

- composition 
    - « has-a » avec responsabilité sur durée de vie
  - généralisation 
    - héritage
  - réalisation 
    - implements
    - réalisation d'un use case
- ◆ Les liens contiennent la multiplicité d'objets associés

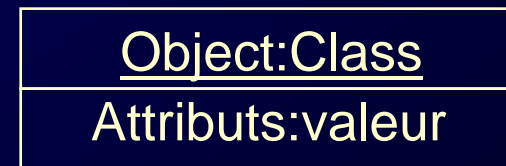
# Class Diagrams



# Object Diagrams

---

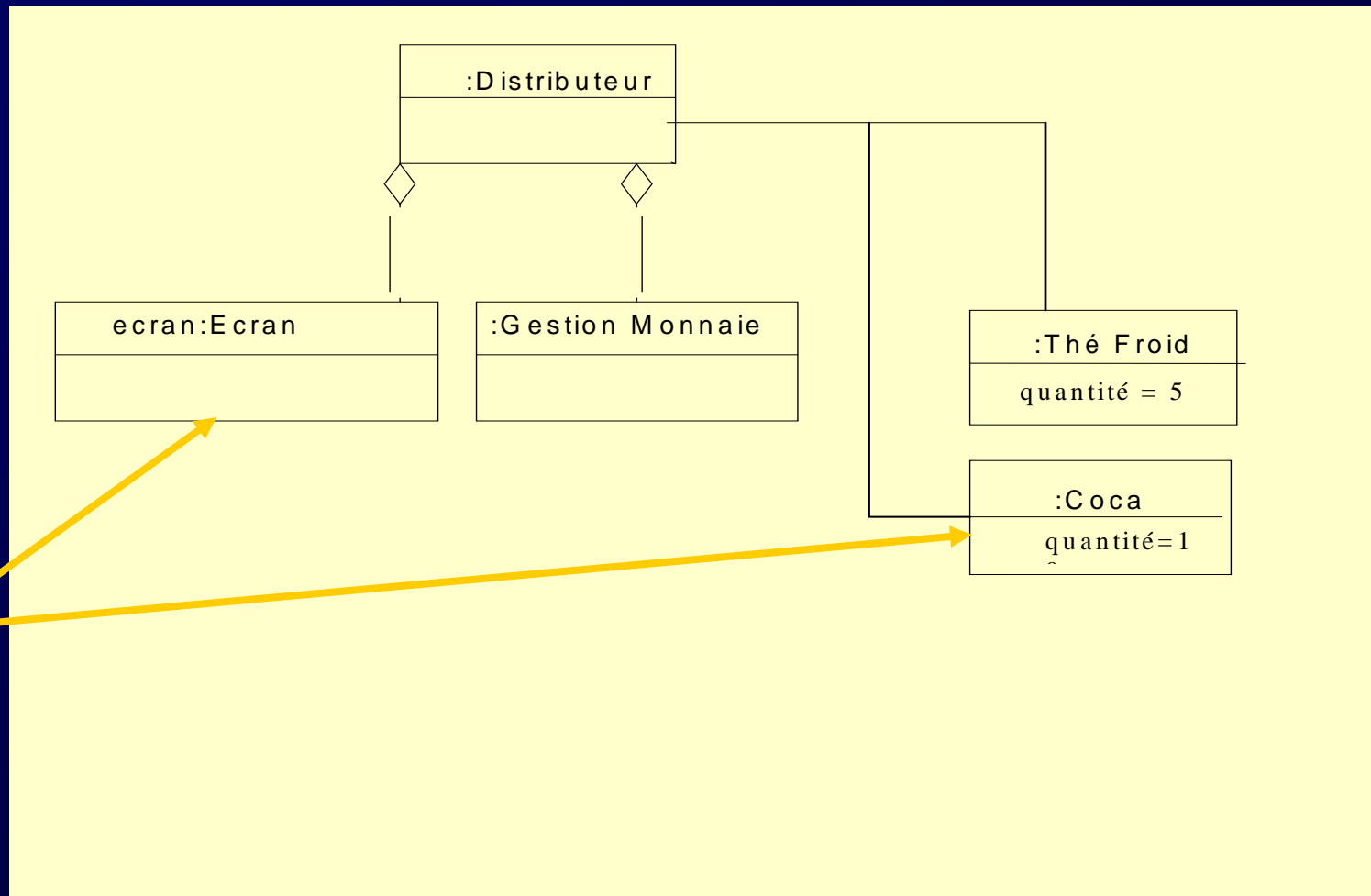
- ◆ Exemples de class diagrams avec des instances d'objets
- ◆ **Nœuds**: objets



- ◆ **Arcs**: relations entre objets, instances d'associations
- ◆ Respecter les multiplicités d'objets associés aux arcs (dans le class diagram)



# Object Diagrams



Objets

# Sequence Diagrams

---

- ◆ Etablissent le lien entre *Use Case* et *Class Diagrams*
- ◆ Décrivent l'échange de messages entre classes


- ◆ **Class rôles**

- objets participant à l'interaction

:Class

Object

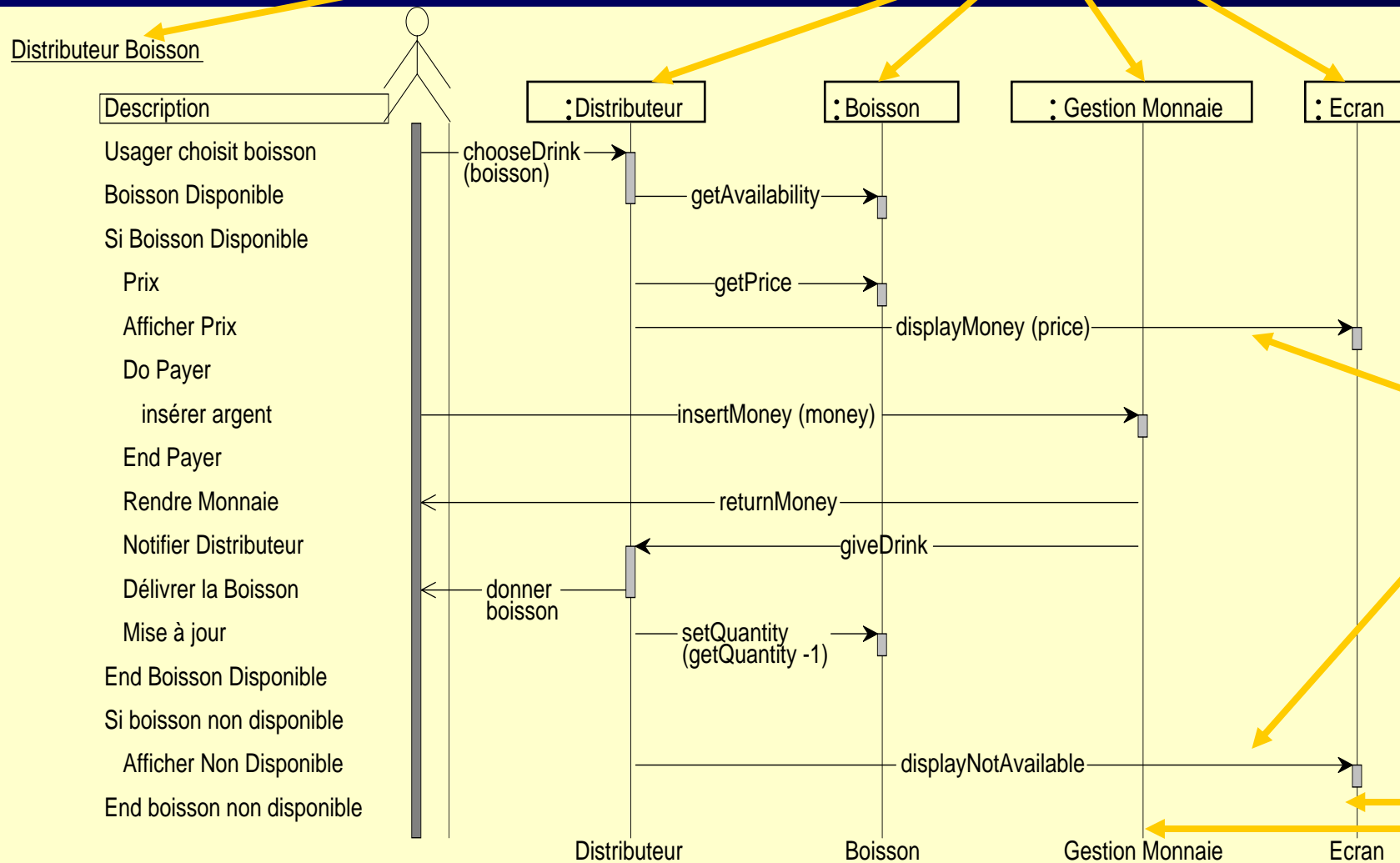
Object:Class

- ◆ **Lignes de temps**
- ◆ **Messages** activant des opérations chez l'objet receveur
  - représente la communication entre objets 
- ◆ Scénario: cas particulier de séquence

# Sequence Diagrams

Use Case

Objets



Messages

Temps

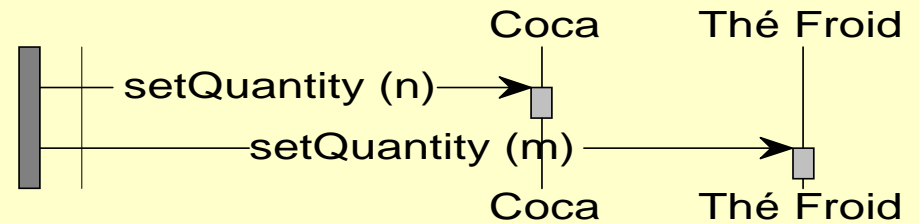
# Sequence Diagrams

## Gérer Distributeur

### Description

Ajouter Coca

Ajouter Thé Froid

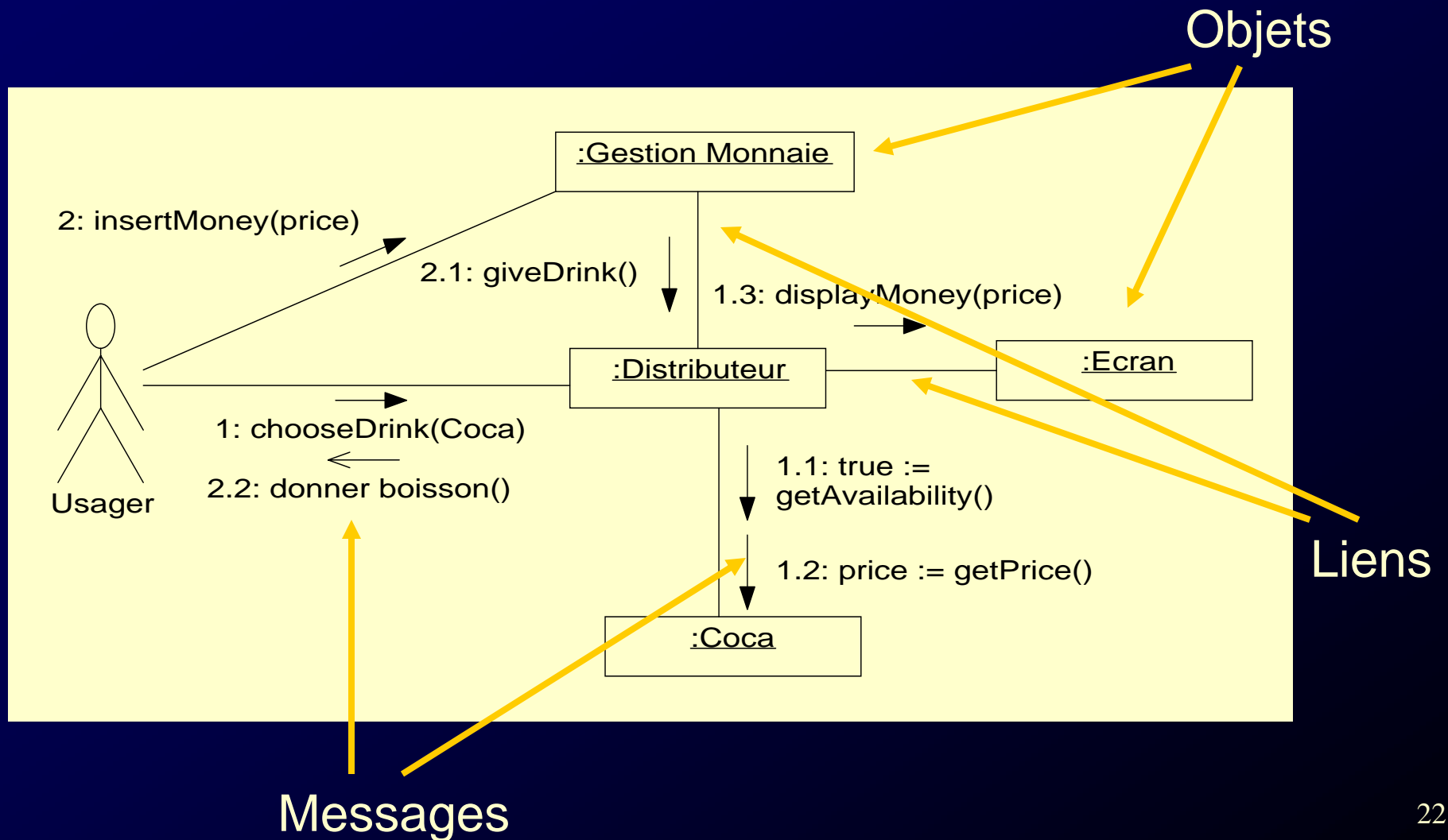


# Collaboration Diagrams

---

- ◆ Décrivent les échanges de *messages* entre classes, et définissent les associations
  - sémantiquement équivalents aux sequence diagrams, mais ...
    - sequence diagrams illustrent l'ordre des événements
    - collaboration diagrams représentent les interconnexions entre objets et sont visuellement différents
- ◆ **Class roles**: objets participant à l'interaction
- ◆ **Liens**: instances d'associations
- ◆ **Messages**: envoyés le long des liens
- ◆ Scénarios: cas particulier

# Collaboration Diagrams

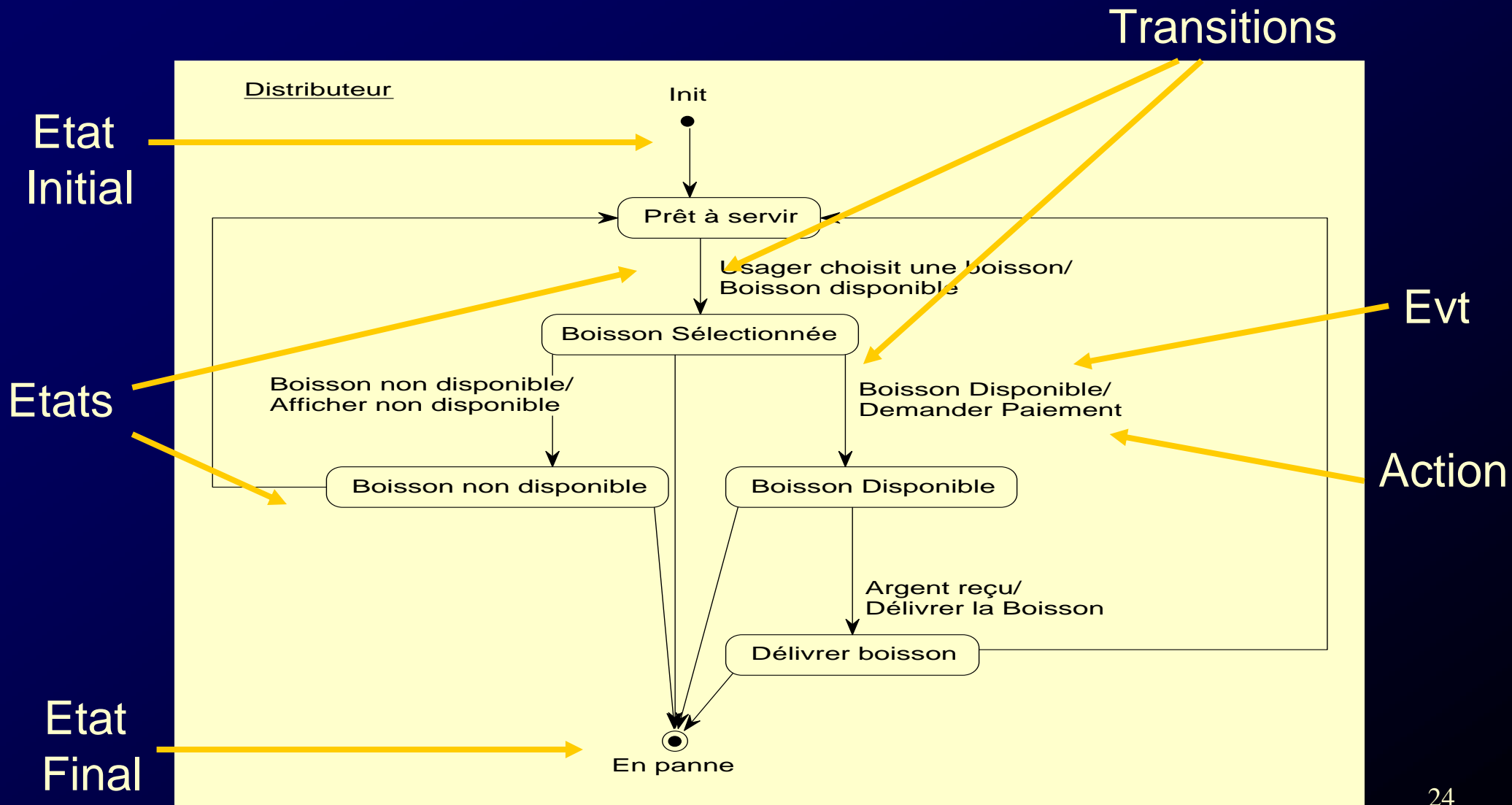


# Statecharts Diagrams

---

- ◆ Décrivent les états et le comportement d'une classe en réponse à des événements *extérieurs*
- ◆ Etats
  - initial, final, intermédiaire
  - sous-états
- ◆ Transitions
  - nom d'événement / action

# Statecharts Diagrams





# Activity Diagrams

---

- ◆ Décrivent le *comportement* d'une *classe* en réponse à des calculs *internes*
- ◆ Similaire aux statecharts, mais pour les événements internes (et non extérieurs)

## ... *et encore*

---

- ◆ Component Diagrams

- Décrivent *l'organisation* des *composants* et les *dépendances* qui les lient

- ◆ Deployment Diagrams

- Décrivent les *ressources de calcul*, leurs *configurations* et le lien entre les *exécutables* et les ressources de calcul

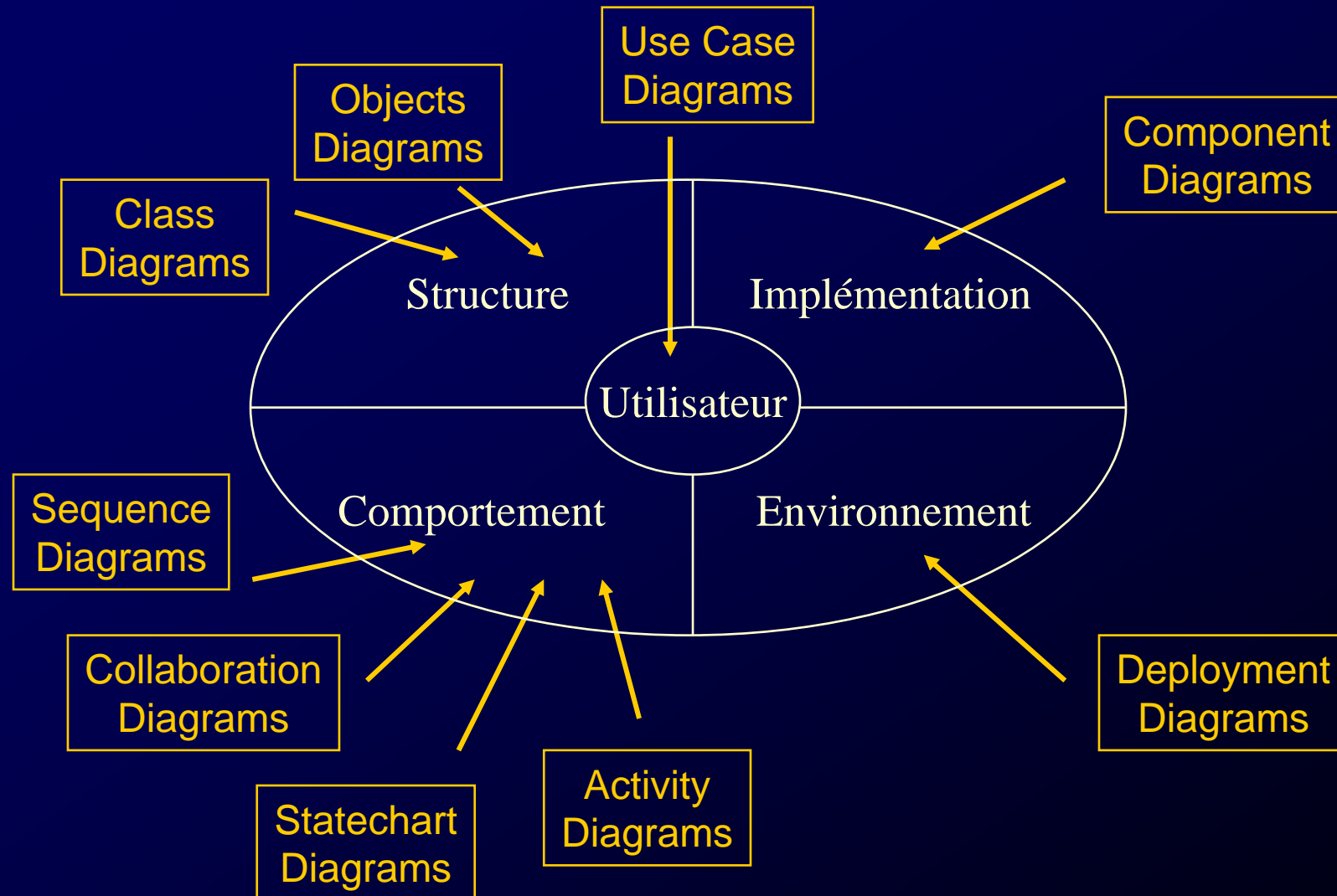
- ◆ OCL: Object Constraint Language

- Langage permettant d'exprimer des conditions attachées à des éléments d'un modèle

- ◆ Stéréotypes

- nouveaux éléments peuvent être définis et introduits dans un modèle

# Diagrammes et Vues



# *Diagrammes*

---

- ◆ Deux Types de diagrammes:
  - Aspects dynamiques: comportement, collaborations, responsabilités
    - use case, activity, statechart, sequence, collaboration diagrams
  - Aspects statiques et structurels: relations
    - class, object, component, deployment diagrams

# *UML et Méthodologies*

---

- ◆ UML est une notation
- ◆ UML n'est pas une méthodologie
- ◆ Les méthodologies peuvent utiliser UML comme notation
  - RUP (Rational Unified Process)
  - OOSE (Object-Oriented Software Engineering, Jacobson)
  - OMT (Object Modeling Technique, Rumbaugh)

# *Bibliographie*

---

- ♦ « UML in a nutshell », Sinan Si Alhir. O'Reilly, 1998.
- ♦ « Java Design - Objects, UML, and Process », Kirk Knoernschild. Addison-Wesley, 2002.