

# CONCEPTION OBJET ET UML

---

# Sources

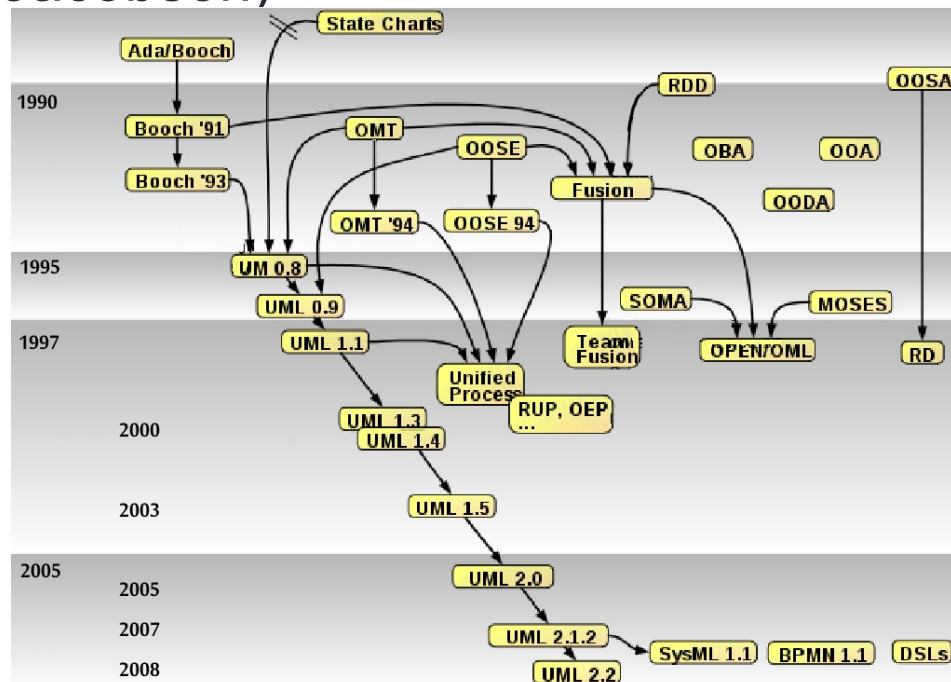
- Cours de Martine Gautier
- Cours de François Charoy
- Cours de Pascal Molli
- Slides de Lou Franco
- <http://www.extremeprogramming.org/>
- [UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition](#)
  - Martin Fowler
- <http://www.uml.org/>
  - Le site de référence
- De nombreuses figures sont issues de la spécification UML

# Objectifs

- Introduction à UML
  - Les usages
  - Les diagrammes
- A la fin du cours vous saurez (ou vous devriez savoir)
  - Identifier les types de diagramme
  - Interpréter un diagramme sans faire d'erreur trop grave
  - Dessiner des diagrammes sans trop d'erreurs d'intention
- Vous ne saurez pas mieux concevoir
  - Ni faire de Model Driven Engineering

# Avant UML

- Apparition dans les années 90 de plusieurs approches de modélisation objet
  - OMT (James Rumbaugh) - plutôt analyse
  - Booch (Grady Booch) – plutôt conception
  - OOSE (Ivar Jacobson) –



# Unified Modeling Language

- UML est né d'un besoin de standardisation des modèles de conception
- 1994 = Fin de la guerre des modèles
  - Rumbaugh et Booch décident d'unifier leurs travaux relativement proches (Rational).
  - Ils seront rejoints par Jacobson.
- Résultats satisfaisants très rapidement
- Evolutions prises en charge par l'OMG (Object Management Group)
  - [www.omg.org](http://www.omg.org)
- Version 2.2 en 2008

# Les niveaux de conformité des outils

- Niveau 0 – Fondation
  - Structure fondamentale et éléments comportementaux
- Niveau 1 – Basic
  - Diagrammes simples
- Niveau 2 – Intermédiaire
  - Diagrammes plus évolués
- Niveau 3 – Complet
  - Toutes les constructions sont disponibles
- Objectif : permettre une construction progressive des outils UML 2.0

# Définition

- UML se définit comme étant un langage de modélisation graphique et textuel, destiné à
  - comprendre et décrire des besoins,
  - communiquer
  - spécifier,
  - documenter des systèmes,
  - esquisser des architectures logicielles,
  - concevoir des solutions.
- Utilisé pour le développement de systèmes à forte composante logicielle
  - banques, télécommunications, transports, aérospatiale, commerce, sciences, ...

# 13 diagrammes

- Diagrammes structurels
  - Classes : Classes et relations
  - Objets : configuration d'instances
  - Composants : Structure et connections entre composants
  - Déploiement
  - Package : structure hiérarchique de compilation
  - Composite Structure : composition d'une classe à l'exécution
- Diagrammes comportementaux
  - Cas d'utilisation : interaction des utilisateurs
  - Etats : impact des événements sur les objets
  - Activités : Comportement procédural et parallèle
  - Diagrammes d'interactions
    - Séquence : interactions entre objets (séquentielles)
    - Communication : Interaction entre objets (collaboration)
    - Interaction : diagramme de séquence et d'activité
    - Timing : interactions entre objets (temporelles)



# DIAGRAMMES DE USE CASE

---

# Description des fonctionnalités d'un système

- Définir les fonctionnalités attendues du système.
- Définir les rôles et les utilisateurs des fonctionnalités
- Définir les dépendances entre utilisateurs et fonctionnalités

# Pourquoi décrire les cas d'utilisation?

- Ils représentent l'expression des besoins fonctionnels de l'application.
  - Partie intégrante du cahier des charges
  - Ecrits dans un formalisme basé sur le langage naturel,
  - Compréhensibles par les experts et les informaticiens.
- Utiles à la fois au dialogue développeurs-utilisateurs et aux développeurs eux-mêmes.
- Evitent une partie des oublis, imprécisions, dérives, ...
- Servent de base à tout le processus de développement, jusqu'aux tests.

# Les scénarios

- Commencer par décrire des scénarios
- Un scénario décrit les étapes d'interactions entre un utilisateur et le système
  - Le client parcourt le catalogue
  - Il ajoute les objets qui l'intéresse dans son caddy
  - Pour payer il donne ses informations de cartes bancaires et son adresse
  - Il confirme l'achat
  - Le système contrôle la validité du paiement et confirme la commande.

# Les acteurs

- Un acteur est l'abstraction d'un rôle joué par des entités extérieures au système.
  - utilisateur, dispositif matériel, autre système
- Un acteur peut consulter ou modifier l'état du système.
- Exemple : Guichet Automatique Bancaire (GAB)
  - porteur d'une carte bancaire
  - client de la banque
  - opérateur de maintenance
  - système d'information de la banque
  - système d'autorisation CB

# Les acteurs

- Ne pas confondre utilisateur et acteur
  - un utilisateur peut jouer plusieurs rôles
  - l'opérateur peut aussi retirer de l'argent
  - plusieurs utilisateurs peuvent jouer le même rôle
  - on peut trouver plus d'une personne pour se servir d'un GAB !
- S'assurer que les acteurs communiquent directement avec le système et non pas par un intermédiaire.
  - Exemple magasin : le caissier est un acteur, mais pas le client.
- Les échanges entre acteurs ne concernent pas la modélisation du système.

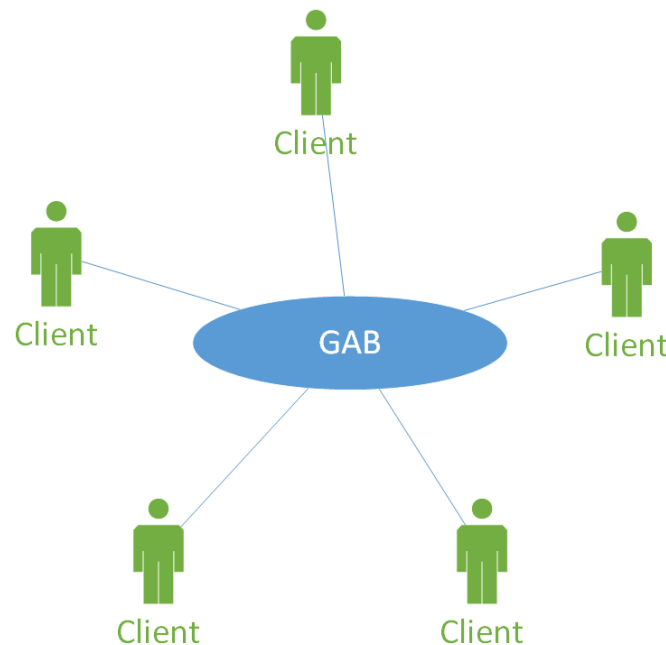
# Types d'acteurs

- On peut distinguer acteur principal et secondaire
  - l'acteur principal utilise/modifie le système
  - l'opérateur est un acteur principal
  - l'acteur secondaire est sollicité pour obtenir une information complémentaire
  - le système d'autorisation de CB est un acteur secondaire
- Représentation graphique



# Diagramme de contexte statique

- Diagramme de classes simplifié qui résume une liste exhaustive des acteurs, en précisant la multiplicité de chacun d'entre eux.





# Identification des cas d'utilisation

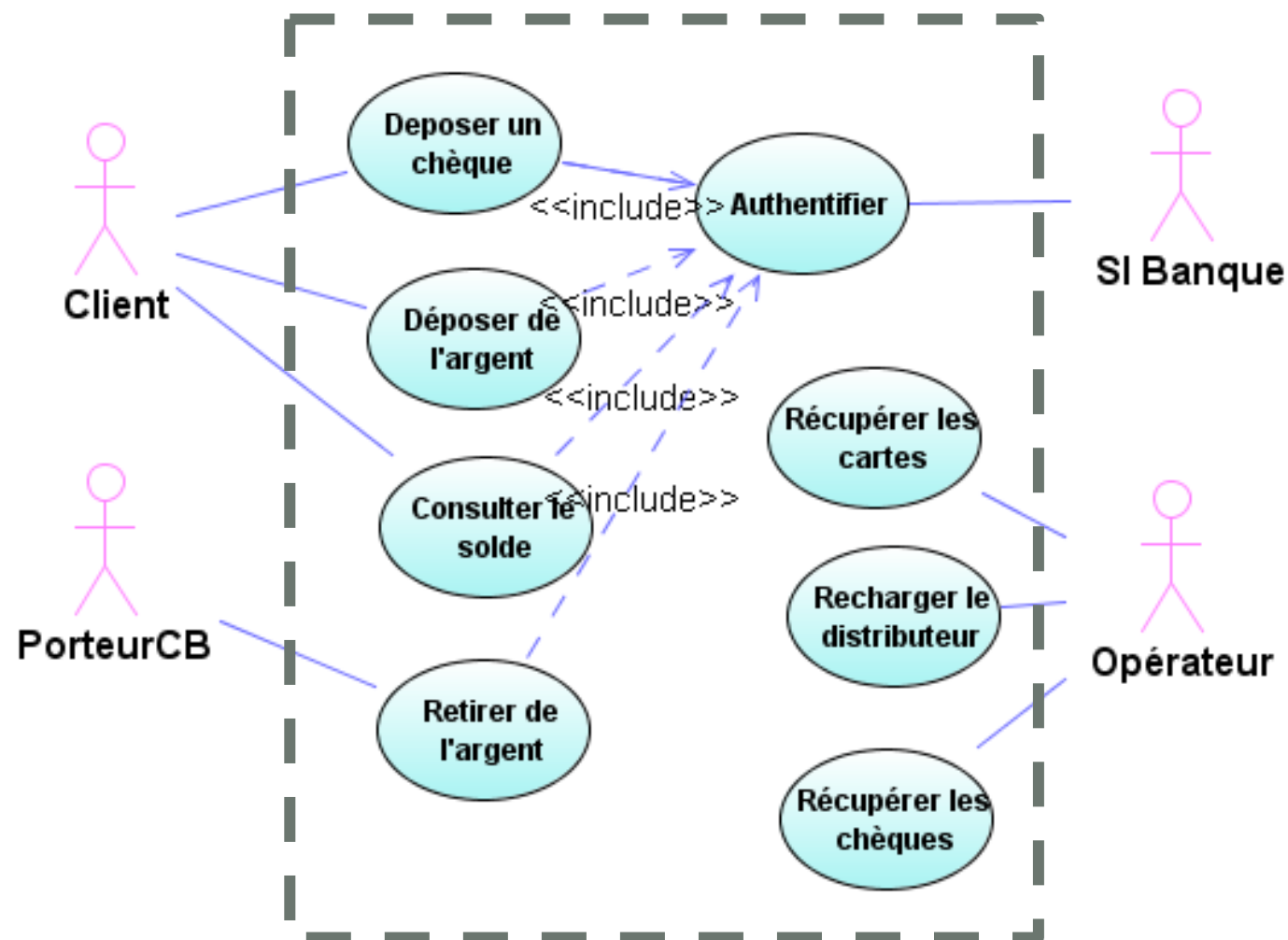
- Un cas d'utilisation modélise un service rendu par le système.
  - Il regroupe une famille de scénarios d'utilisation selon un critère fonctionnel.
- Comment construire les cas d'utilisation ?
  - Pour chaque acteur, identifier les intentions avec lesquelles il communique avec le système.
- Exemple GAB : client de la banque
  - retirer de l'argent
  - consulter le solde d'un compte
  - déposer de l'argent
  - déposer des chèques
- Représentation graphique



DéposerUnChèque

# Diagrammes de cas d'utilisation

- Un diagramme fait intervenir
  - un ou plusieurs acteurs,
  - les cas proprement dits,
  - des relations de communication entre les acteurs et les cas, portant le message facteur de déclenchement du cas.

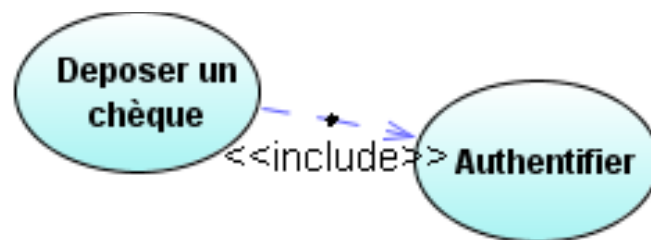


# Organisation des cas d'utilisation

- Problème : jusqu'où aller dans les détails des cas ?
  - Le nombre d'acteurs concernés par un cas doit être limité.
  - Un cas ne doit être ni trop simple, ni trop complexe
  - ...
- Il est possible (et souhaitable !) d'organiser les cas
  - en utilisant les relations d'inclusion, d'extension et de généralisation entre cas,
  - en regroupant les cas en packages, par acteur et/ou par cas.

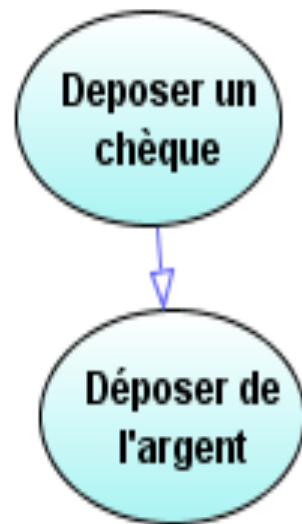
# Inclusion

- La relation d'inclusion indique qu'un cas inclut un autre cas d'utilisation.
  - On précise le point d'inclusion.
  - Cette relation évite de décrire plusieurs fois le même cas.
  - **Le cas inclus n'existe jamais seul.**



# Généralisation

- La relation de **généralisation** indique qu'un cas généralise un autre cas d'utilisation.
  - Le cas fils hérite du comportement et de la signification du cas père : il peut le compléter ou le remplacer.



# Relation d'extension

- Permet de modéliser une partie optionnelle d'un Use Case
- Permet de modéliser des cas conditionnels ou des variations
- Attention : ce n'est pas un moyen de décrire le flot de contrôle du cas !!!

# Extends vs use

- Intentions différentes
  - extends
    - Distinguer des variantes
    - L'acteur est lié au cas de base
  - uses/includes
    - Identification de comportements communs
    - Parfois aucun acteur directement associé au cas commun
- Ne permettent pas de décrire le comportement du Use Case !!!



# Détailler les cas d'utilisation

- Un diagramme de cas d'utilisation ne se suffit pas à lui-même.
  - Détailler la dynamique du cas
- Décrire le flot d'événements qui le constitue
  - En utilisant une description textuelle
  - En formalisant la description.

# La forme des use cases

- Pas de forme standard
- Plusieurs formes possibles
- Les principales sections s'y retrouvent
  - Le nom du cas
  - Iteration (pour un développement itératif)
  - Résumé
  - Préconditions
  - Déclencheurs
  - Le scénario de base
  - Les chemins alternatifs
  - Postconditions
  - Les règles métiers
  - Notes
  - Auteur and date

# User Stories (Mike Cohn)

ID	Theme	As a/an	I want to...	so that...	Notes	Priority	Status
2	Game	moderator	create a new game by entering a name and an optional description	I can start inviting estimators	If games cannot be saved and returned to, the description is unnecessary	Required	done
2	Game	moderator	invite estimators by giving them a url where they can access the game	we can start the game	The url should be formatted so that it's easy to give it by phone.		done
5	Game	estimator	join a game by entering my name on the page I received the url for	I can participate			done
6	Game	moderator	start a round by entering an item in a single multi-line text field	we can estimate it			done
8	Game	estimator	see the item we're estimating	I know what I'm giving an estimate for			done
<del>40</del>	<del>Game</del>	<del>participant</del>	<del>always have the cards in the same order across multiple draws</del>	<del>it's easy to compare estimates</del>	-	Replaced with A08 because I didn't want the story to talk about "the same order" as that might be a UI implementation detail	<del>todo</del>
35	Non-functional	user	have the application respond quickly to my actions	I don't get bored			done
36	Non-functional	user	have nice error pages when something goes wrong	I can trust the system and it's developers			done
A11	Non-functional	Researcher	results to be stored in a non-identifiable way	I can study the data to see things like whether estimates converged around the first opinion given by "estimator A" for example	No names or story text should be stored but we should store each card of each hand, know who played it, and know the final accepted estimate		
A05	Game	moderator	edit an item in the list of items to be estimated	so that I can make it better reflect the team's understanding of the item			
22	Archive	moderator	export a transcript of a game as a CSV file	I can further process the stories and estimates	Exported file should be directly importable back into the system.		done

# Avantages des use cases

- Faciles à comprendre par les clients
- Faciles à délimiter
- Traçables
- Bonne base d'évaluation de l'effort
- Évite de concevoir trop tôt
- Description sous forme d'histoires ou de scénarios

# Inconvénients

- Ne prennent pas en compte les besoins non fonctionnels
- Apprentissage nécessaire pour les lire
- ...

# Conclusion use case

- UML fait peu de choses pour la définition des besoins
  - acteurs
  - cas d'utilisation
  - relation entre cas
- Les diagrammes de cas UML ne représentent qu'une petite partie de la description des besoins.
- La description textuelle est absolument indispensable car le diagramme lui-même est peu instructif.

# LES DIAGRAMMES DE CLASSE

---

# Les Diagrammes de classes

- Diagramme de structure
- Permettent de présenter les classes d'un modèle
  - Leurs attributs
  - Leurs opérations
  - Leurs relations
- Utilisable pour l'analyse, la conception et l'implantation.

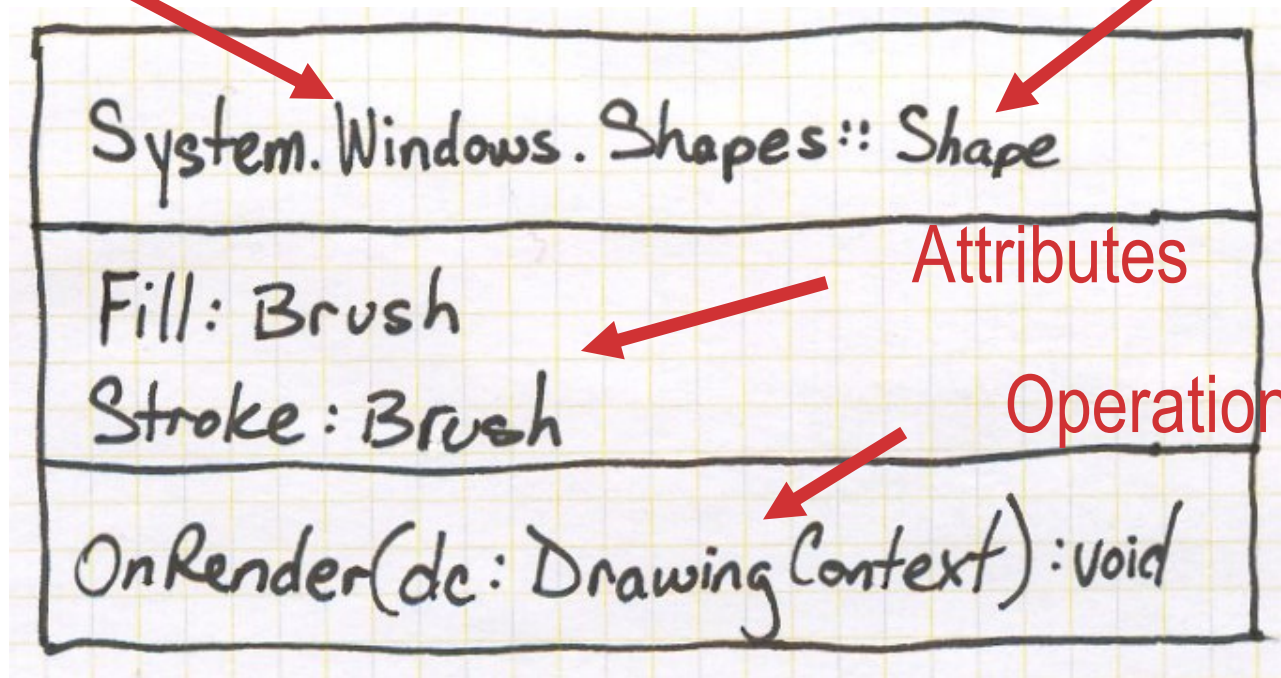


# Class Diagram Elements

Class

Package

Class



# Attributs

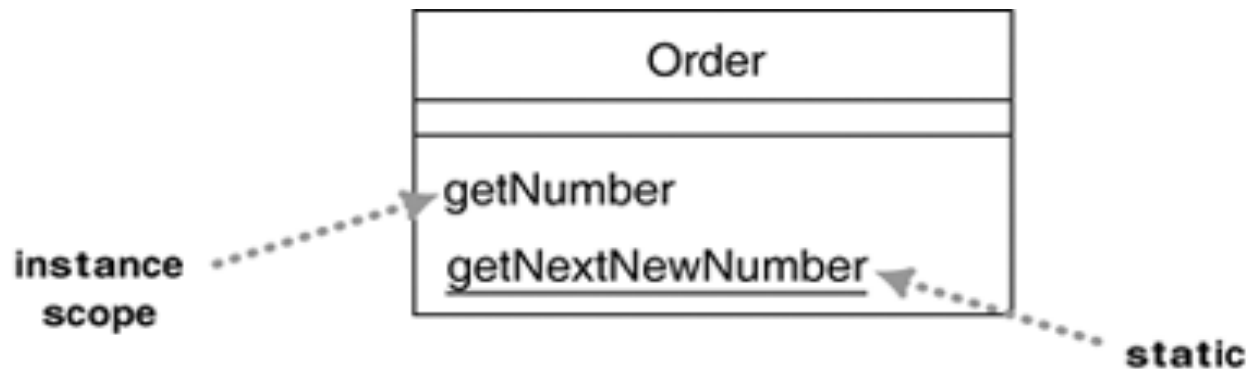
- Un attribut est une propriété d'une classe
- Description
  - Nom
  - Visibilité
  - Type
  - Cardinalité
  - Valeur par défaut
  - Autres propriétés (read-only, ordered....)

Order
+ dateReceived: Date [0..1] + isPrepaid: Boolean [1] + lineItems: OrderLine [*] {ordered}

[visibilité] nom [[multiplicité]] [:type][=valeur initiale][{propriété}]

# Attribut de classe

- **Attribut de classe** : attribut commun à toutes les instances de la classe



# Opérations

- Une opération représente un service pouvant être demandé à n'importe quelle instance de la classe.
  - Modifier l'état de l'objet
  - Modifier des liens avec d'autres objets
- Une opération est définie par un nom et une signature.

# Signatures des opérations

[visibilité] nom [(paramètres)][:type de retours][{propriétés}]

- Syntaxe des paramètres
  - [direction] nom : type [=valeur par défaut]
  - Direction = in, out ou inout
- Propriétés
  - {leaf} – opération concrète
  - {abstract} – ne peut être appelée directement
  - {isQuery} – ne change pas l'état
  - ...

# Visibilité des attributs et des opérations

- Trois niveaux de visibilité
  - + public : visible pour tous
  - # protégé : visible pour toutes les sous-classes
  - - privé : visible pour la classe.

exemple::Personne
-nom : string(idl)
+age : short(idl)
-prénom : string(idl)
+getNom() : string(idl)

# Notations possibles

<i>Window</i>
---------------

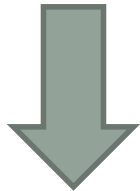
<i>Window</i>
size: Area visibility: Boolean
display() hide()

<i>Window</i>
+ size: Area = (100, 100) # visibility: Boolean = true + defaultSize: Rectangle - xWin: XWindow
display() hide() - attachX(xWin: XWindow)

<i>Window</i>
public size: Area = (100, 100) defaultSize: Rectangle protected visibility: Boolean = true private xWin: XWindow
public display() hide() private attachX(xWin: XWindow)

# Diagramme d'objet

exemple::Personne
-nom : string(idl)
+age : short(idl)
-prénom : string(idl)
+getNom() : string(idl)



<u>unePersonne : exemple::Personne</u>
nom : string(idl) = Kirk
age : short(idl) = 35
prénom : string(idl) = James

- Les instances des classes peuvent être modélisées




# Les relations

- Représente un modèle pour une connexion entre deux objets
- Organisées en hiérarchie
  - Dépendance : relation entre classes à l'exécution
  - Association : relation consistante
  - Composition : dépendance d'existence

# Les relations entre les classes

Dependency 

Aggregation 

Inheritance 

Composition 

Association 

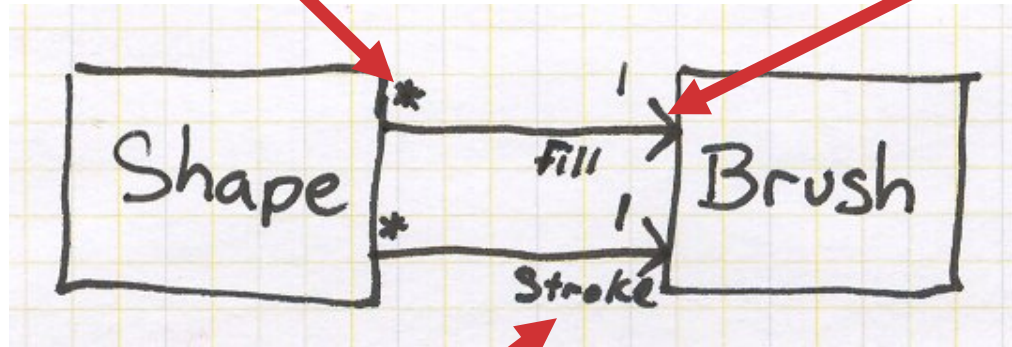
Directed Association 

Interface Type Implementation 

# Association

multiplicity

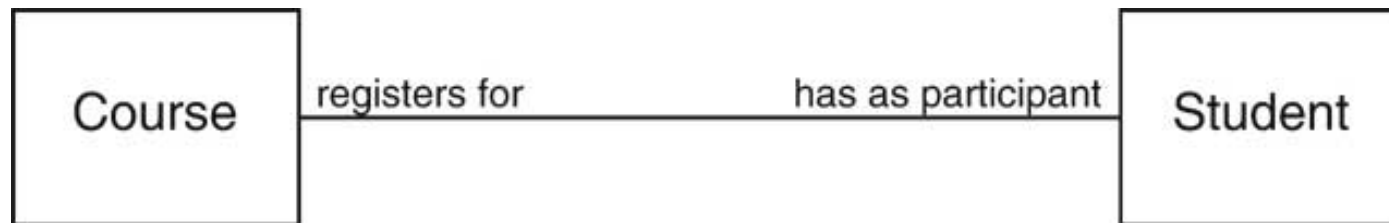
navigability



role

# Association

- Les associations ont des rôles
- Plus général que l'aggrégation



# Association et directionnalité

- Détermine les possibilités de navigation
- Toutes les associations doivent avoir une direction sauf si la bidirectionnalité est spécifiée.



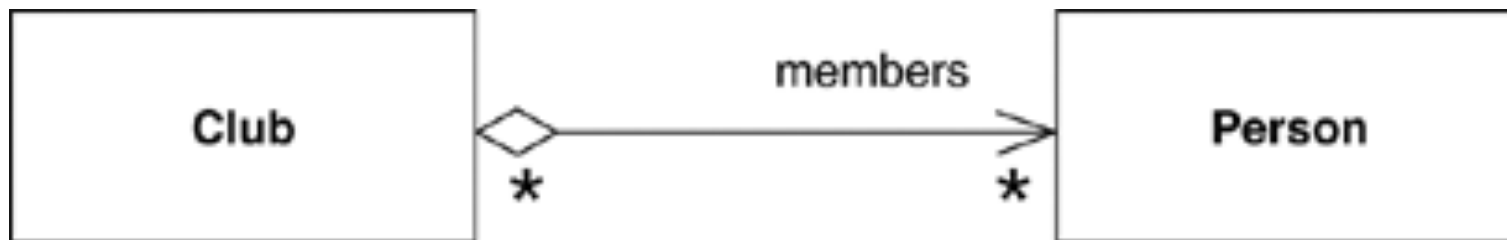
# Cardinalités

- Un nombre quelconque \*
- Un ou plusieurs: 1..\*
- Zéro ou un: 0..1
- Exactement un: 1



# Aggrégation

- Dans une agrégation, l'une des classes, le **conteneur**, est composé d'instances de l'autre classe.
  - Les propriétés de l'association subsistent.
  - Les parties peuvent être partagées par d'autres agrégats ; elles ont leur cycle de vie propre.



# Composition

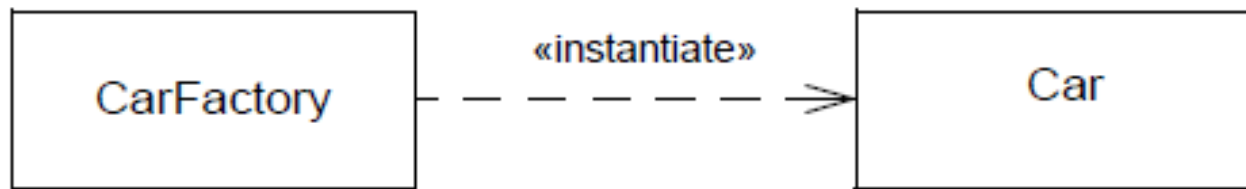
- Forme d'aggrégation spéciale
- Dépendance d'existence entre les objets



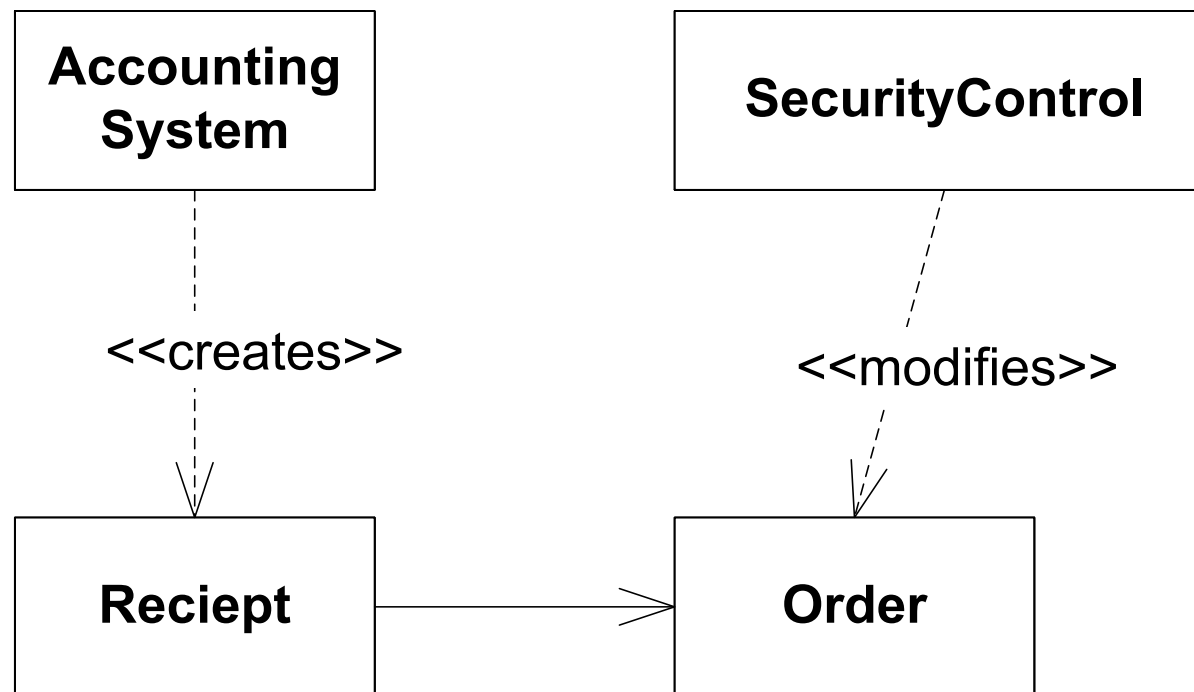


# Dépendance

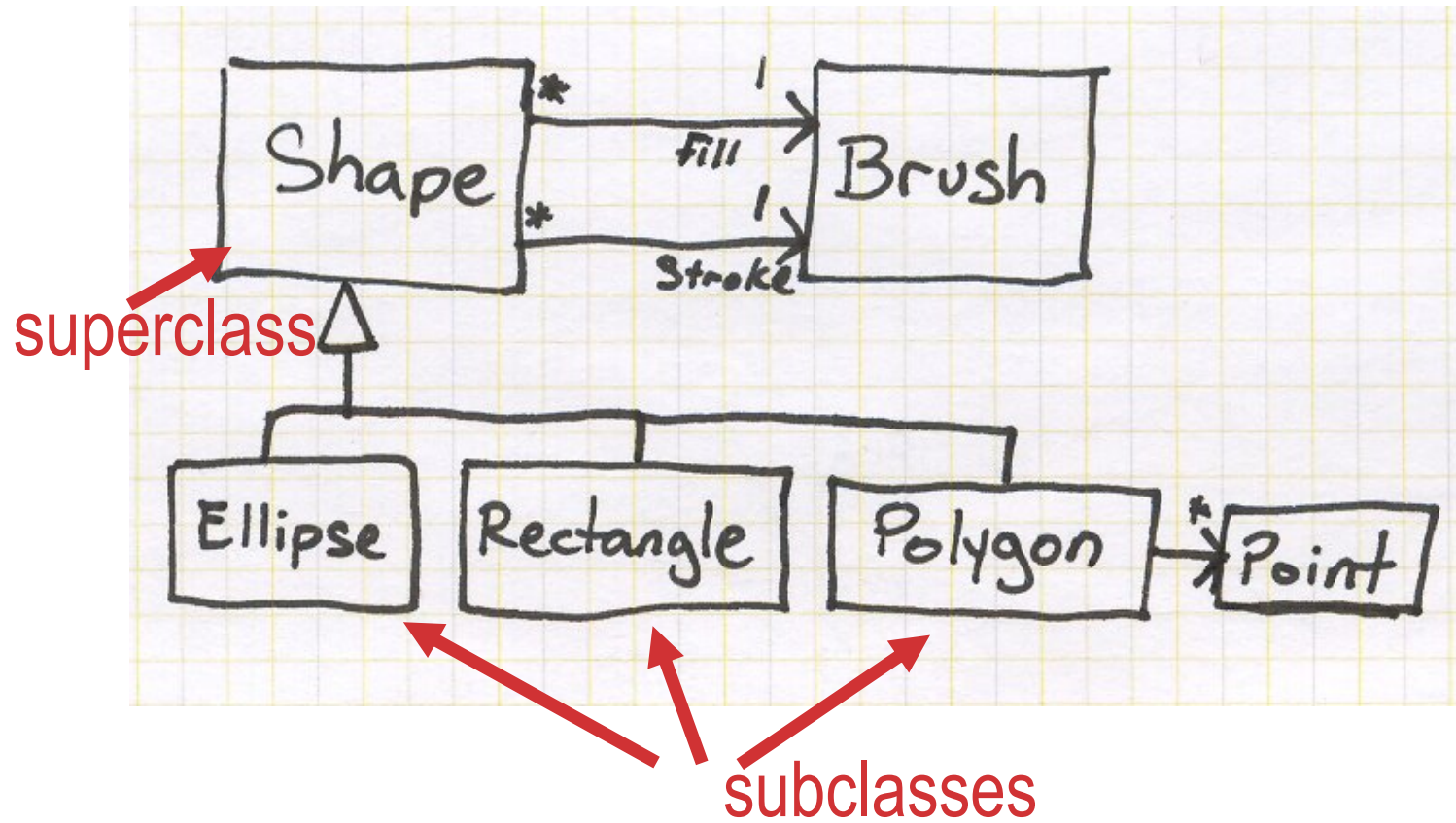
- Un objet affecte un autre objet (appelle, crée, ...)



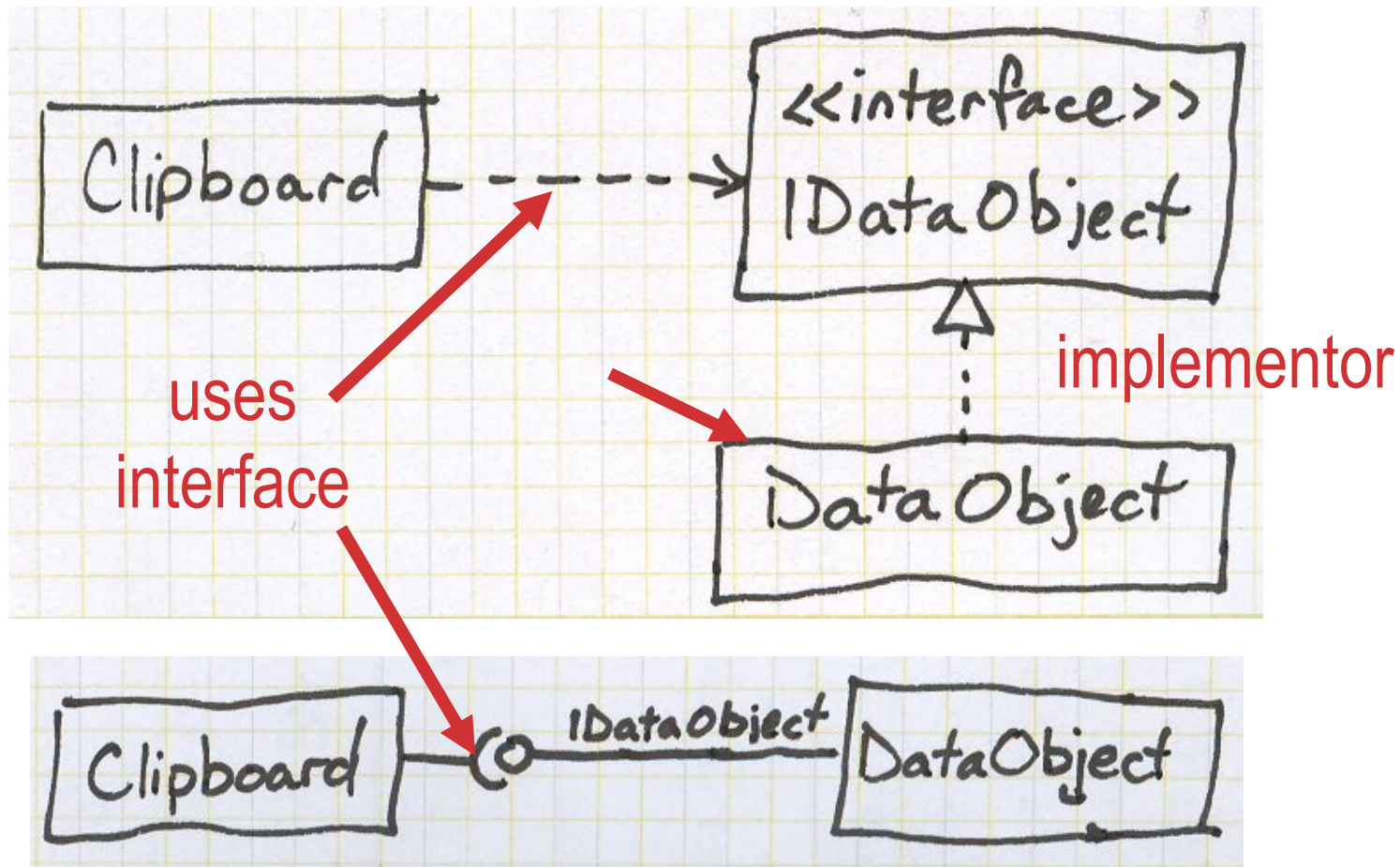
# Relations de dépendance



# Generalization



# Providing and Requiring Interfaces



# Les interfaces

- Une interface décrit un ensemble de méthodes (stéréotype)
- Une classe implante une interface

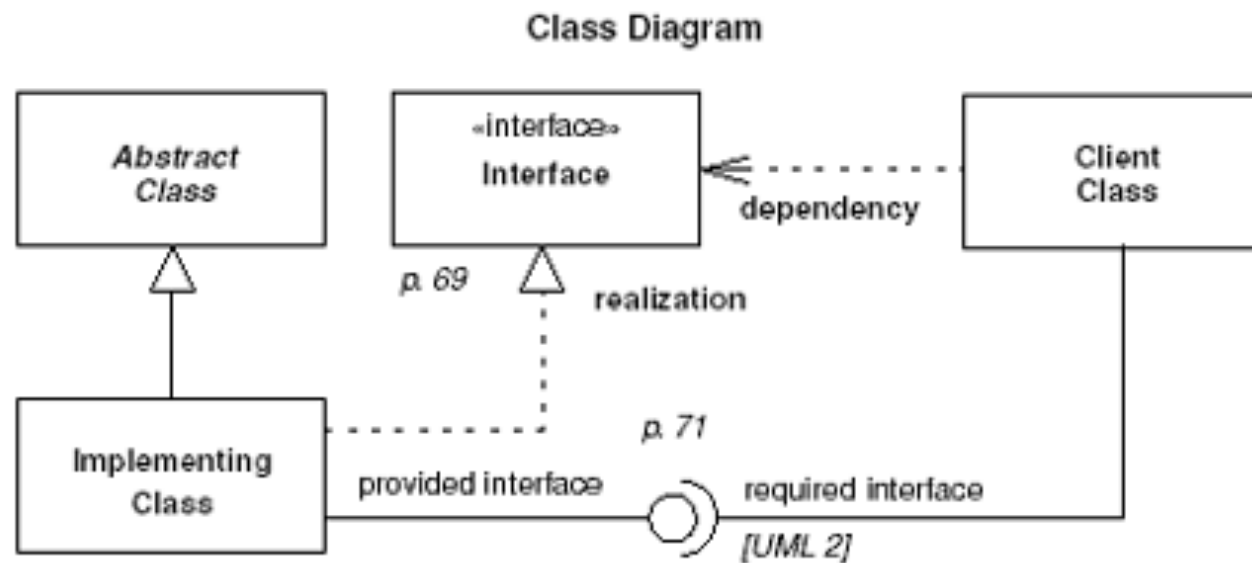


# Réalisation

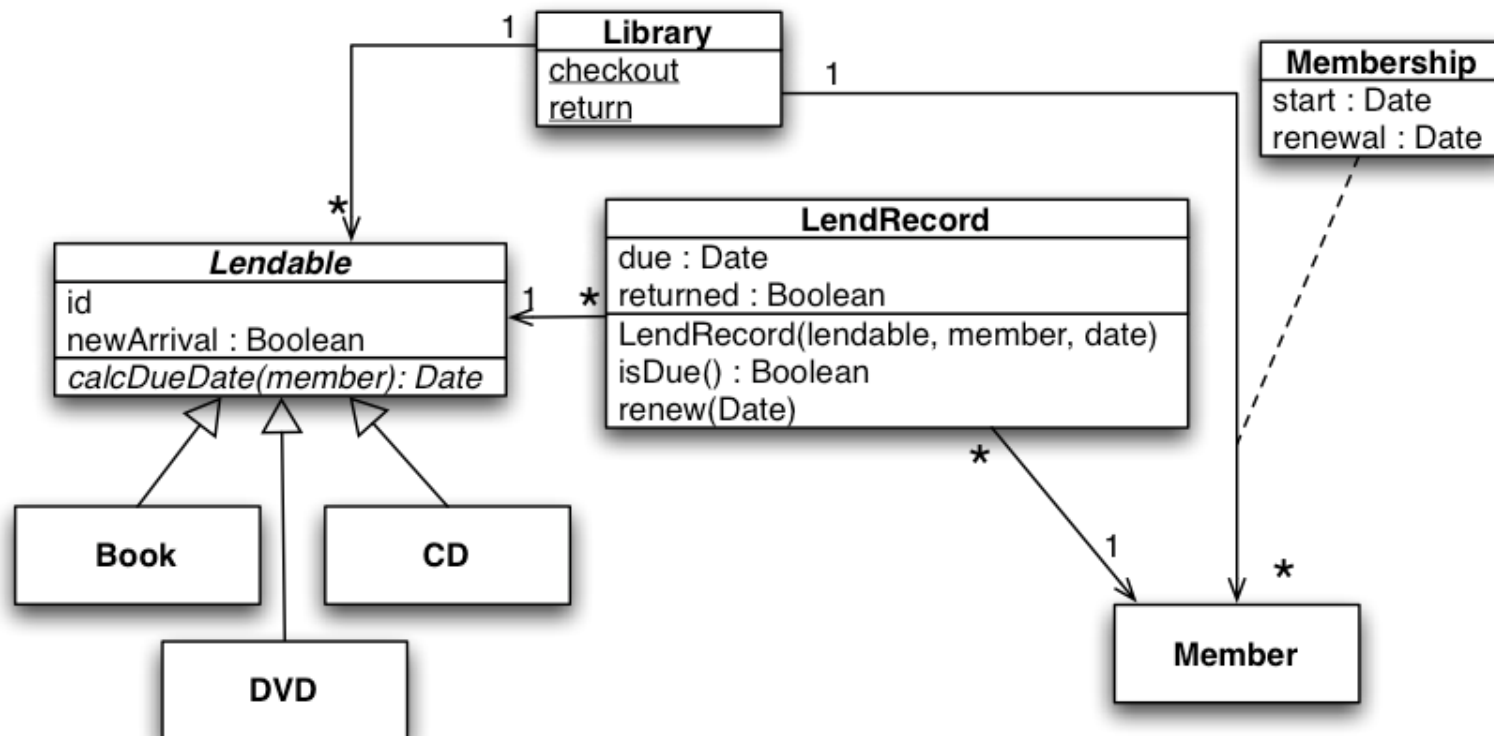
- Une classe peut réaliser plusieurs interfaces et une interface peut être réalisée par plusieurs classes.
- La relation de généralisation s'applique aux interfaces.
- La hiérarchie des interfaces peut être différente de celle des classes qui les réalisent



# Interfaces/Classes abstraites



# Example: Library Classes





# Conclusions

- Le diagramme de classe est une description statique d'une programme
- Un diagramme de classe n'est pas un simple modèle de données pour système d'information
- Il doit pouvoir ensuite être traduit dans un langage (ex: Java)

# DIAGRAMMES DE SÉQUENCE

---

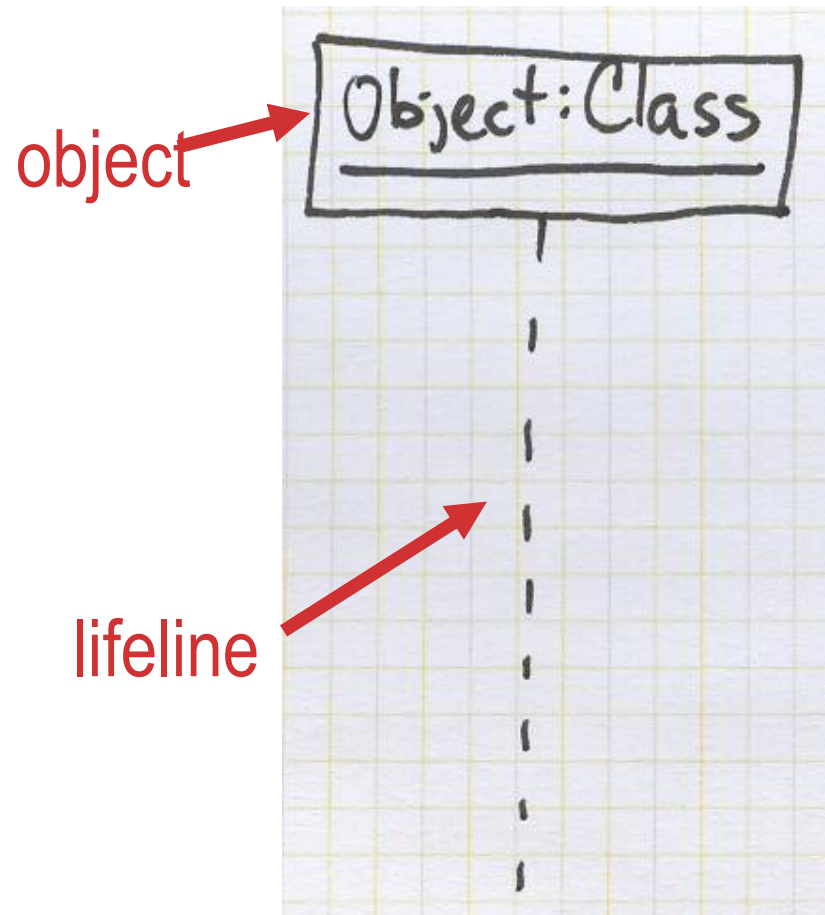
# Séquence/Communication

- Ces deux diagrammes contiennent la même information, présentée de manière différente.
  - Le diagramme de séquence met l'accent sur la chronologie des messages,
  - Le diagramme de communication met l'accent sur l'organisation structurelle des objets qui envoient et reçoivent des messages.

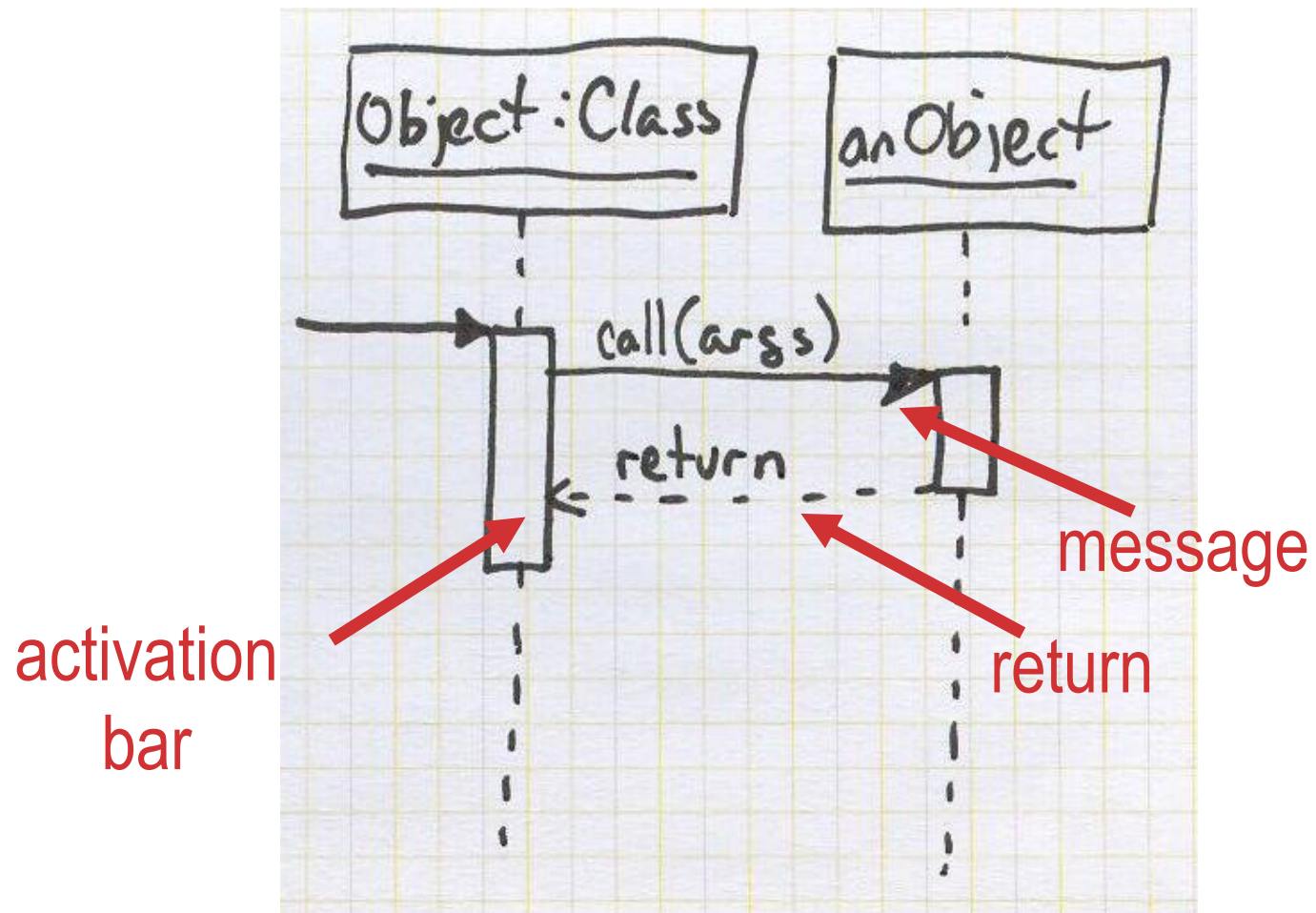
# Diagrammes de séquence

- Ce diagramme met l'accent sur la chronologie.
  - Modélisation des aspects dynamiques des systèmes temps réels et des scénarios mettant en œuvre peu d'objets.
- Un objet est matérialisé par un rectangle et une ligne de vie.
- Les messages sont matérialisées par des flèches allant de l'émetteur vers le récepteur.
- Ils s'utilisent de deux manières différentes, selon la phase de développement.
  - Documentation des cas d'utilisation
  - Représentation précise des interactions entre objets et répartition du flot de contrôle.

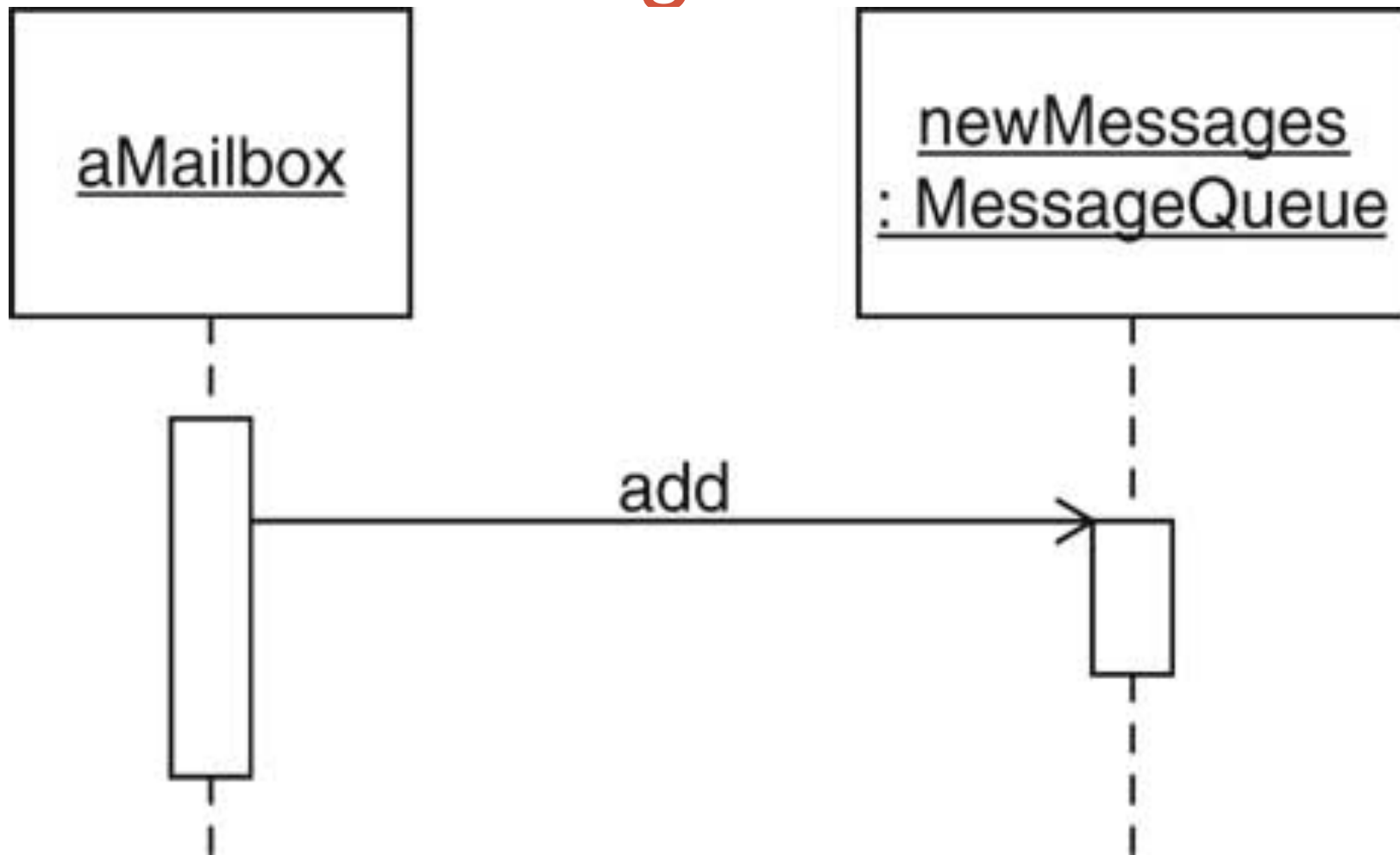
# Object lifeline



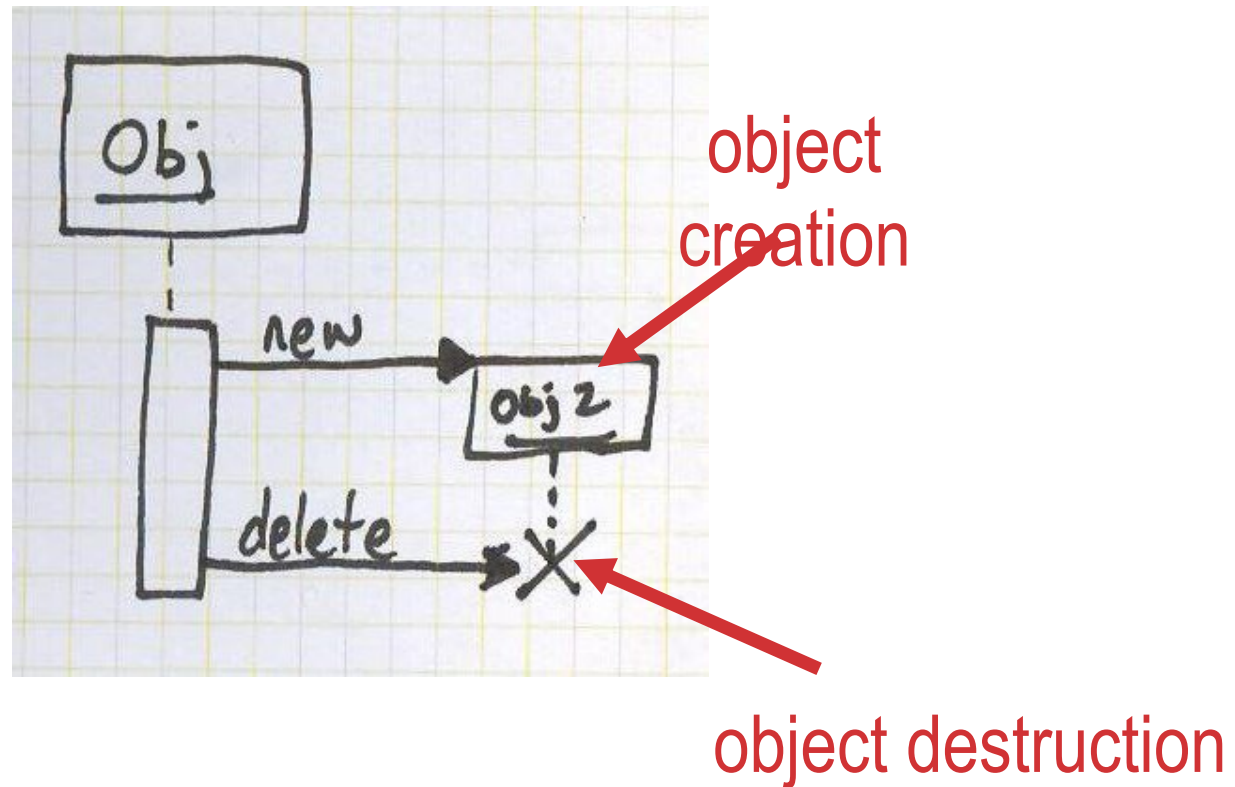
# Object activation



# Envoi de message

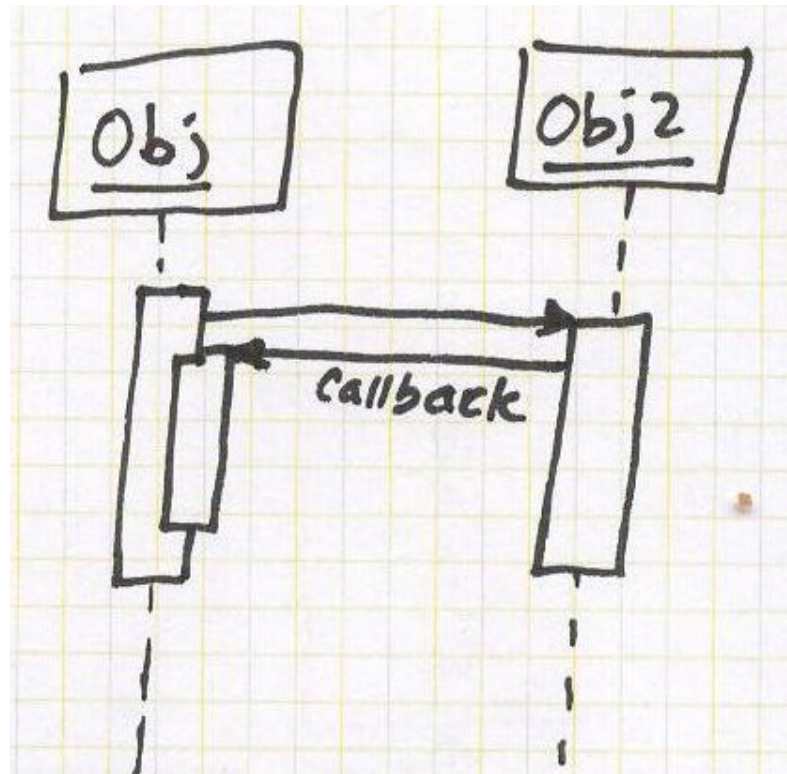


# Object creation and deletion

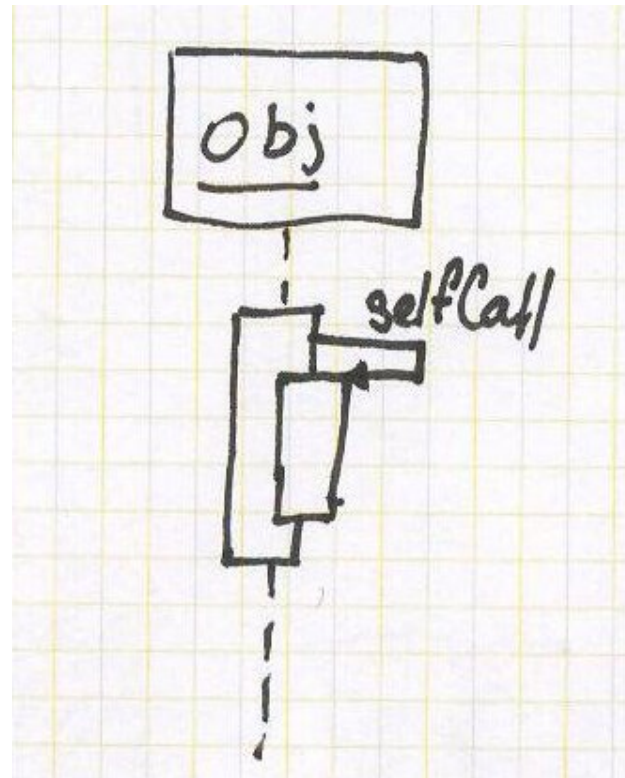




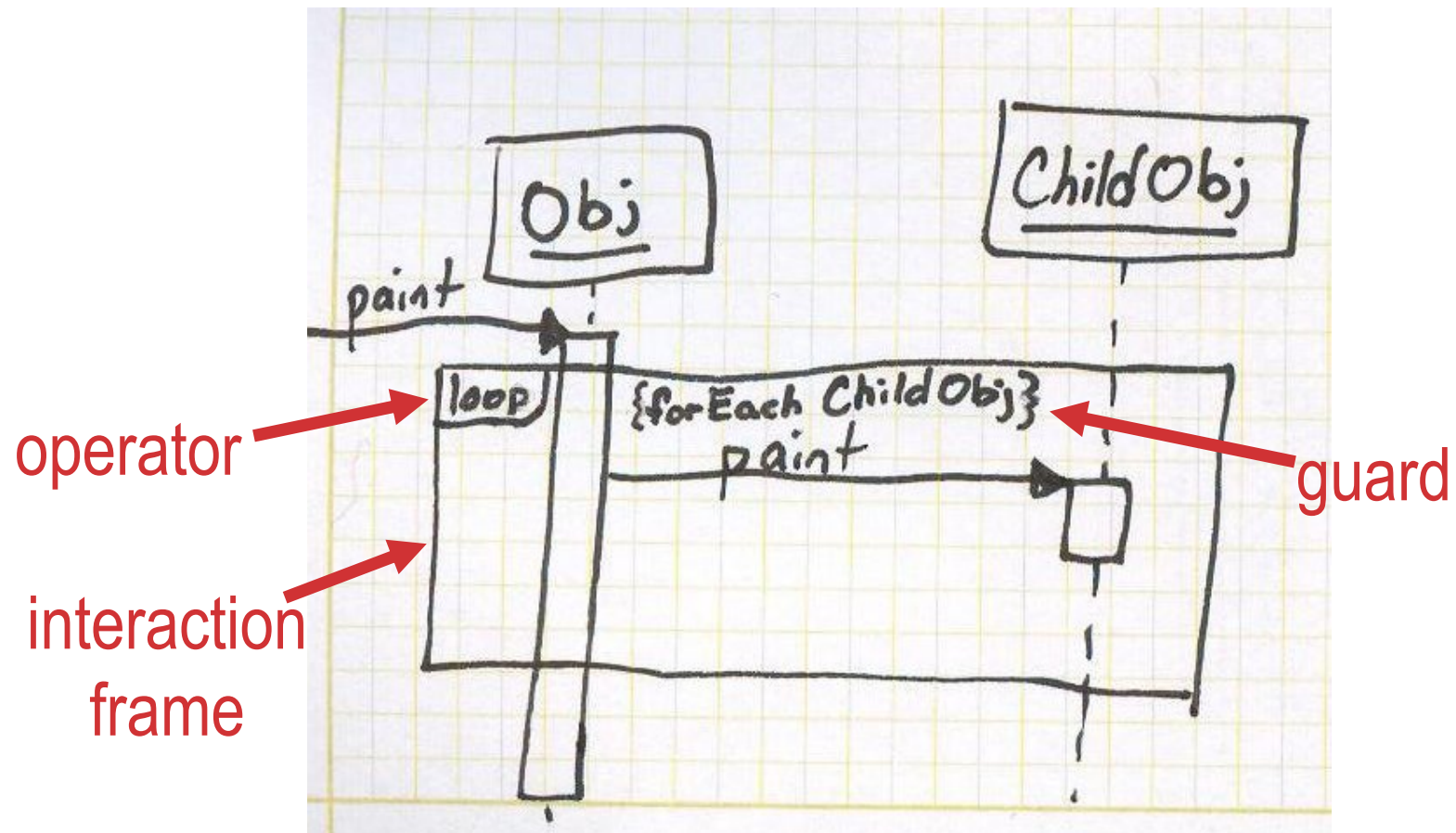
# Callbacks



# Object calling itself

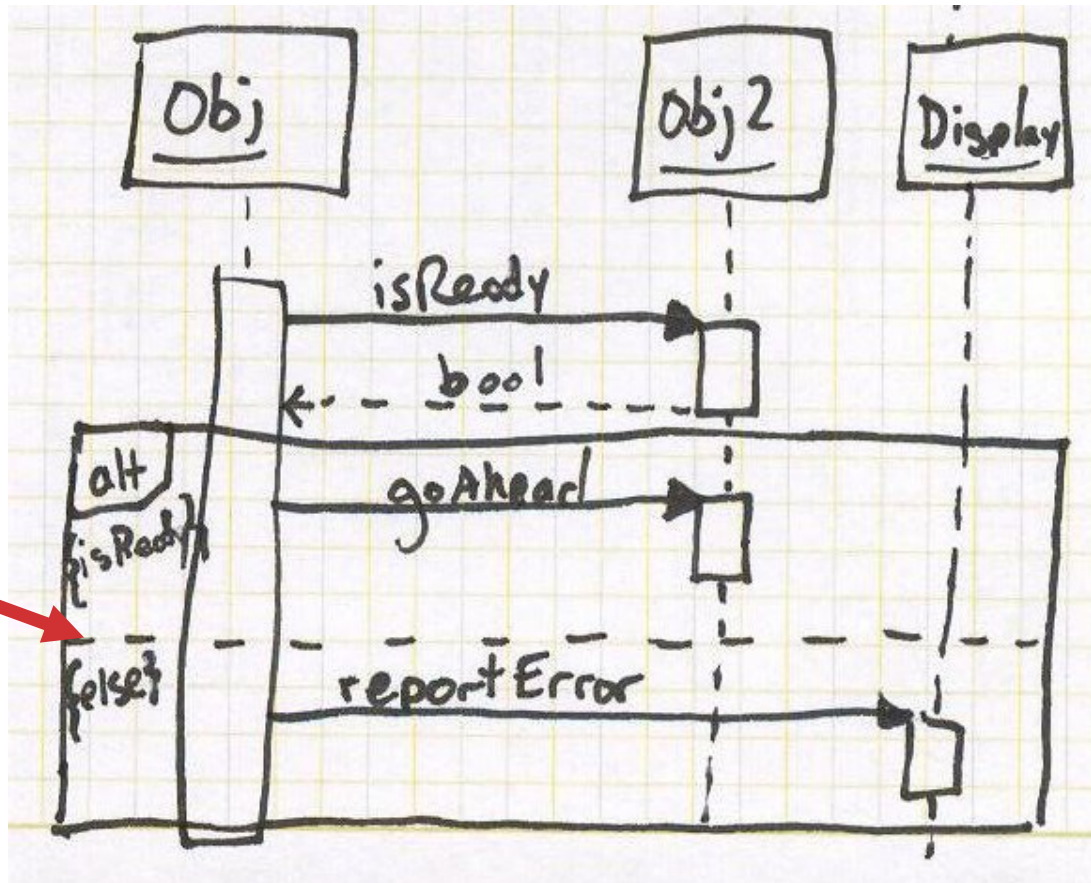


# Loops

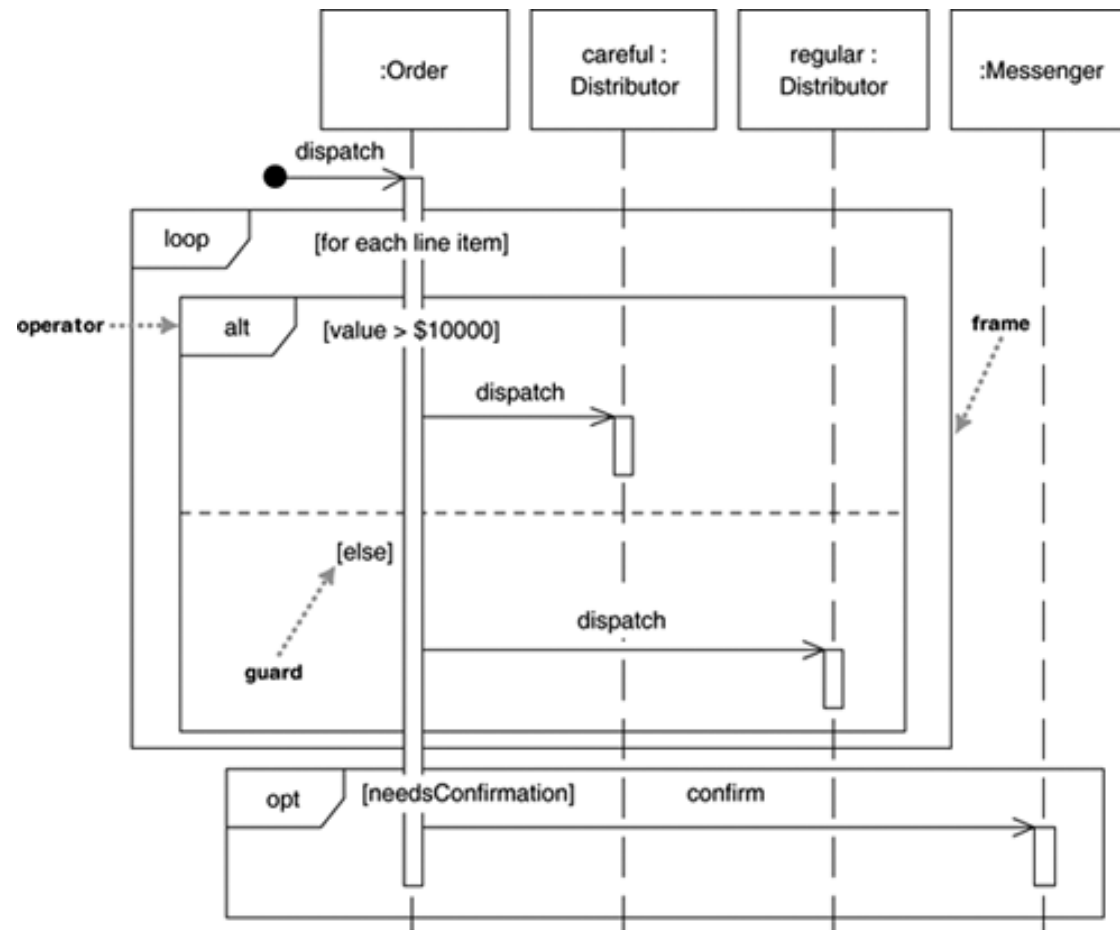


# Conditionals

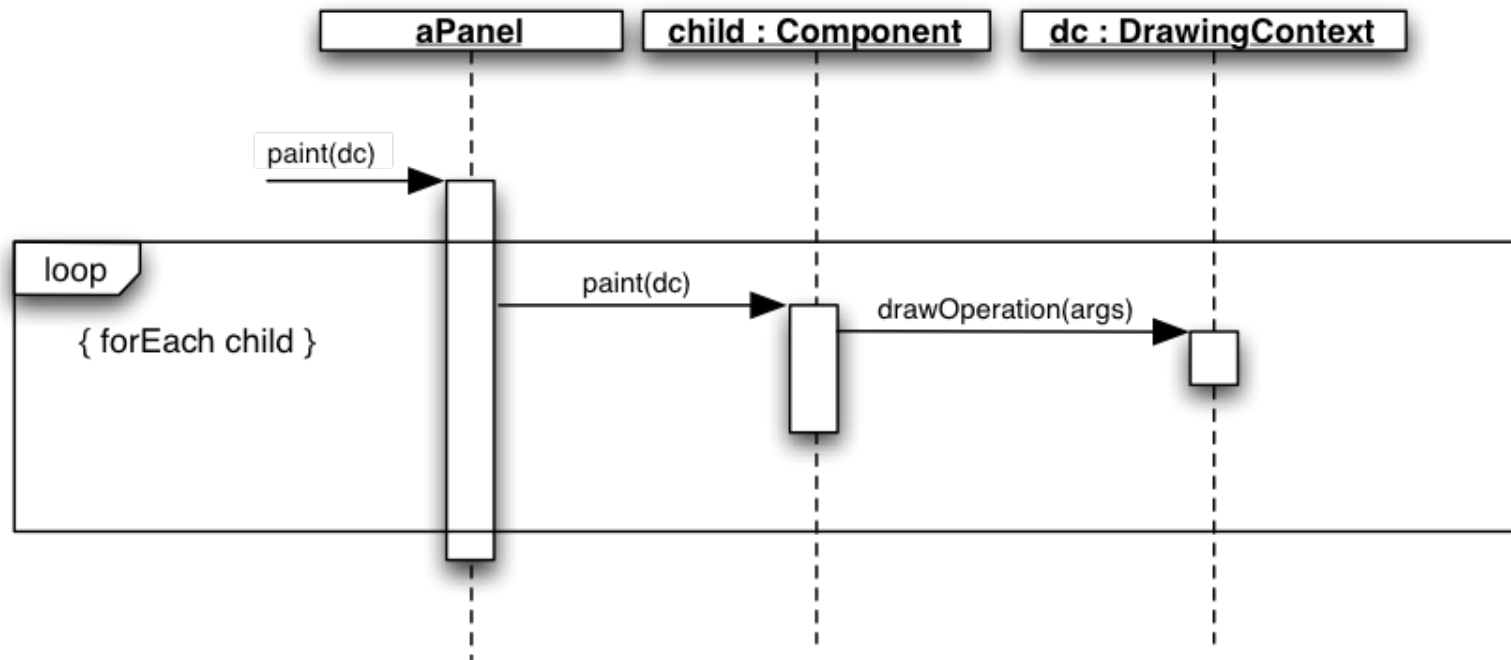
alternative  
separator



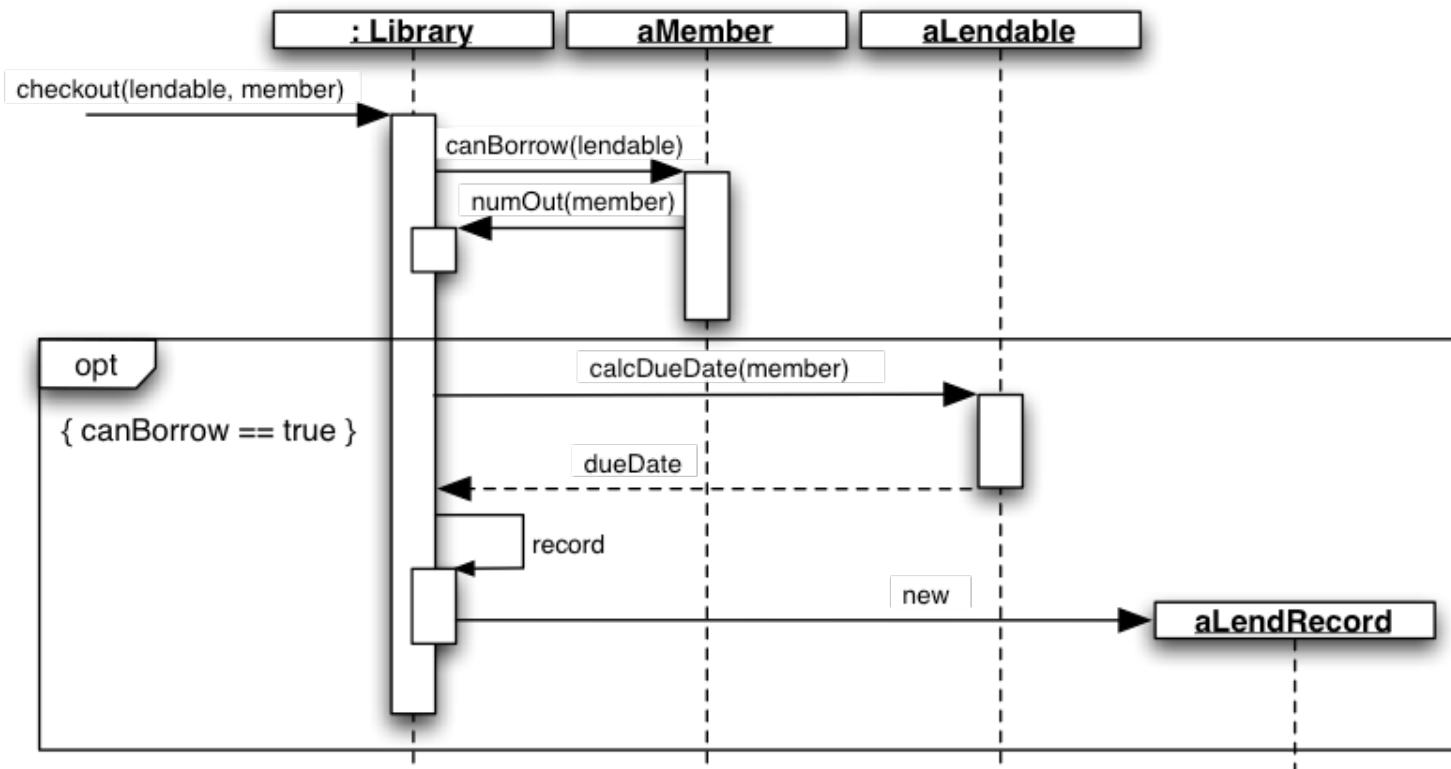
# Loops and conditionals



# Example: Panel painting



# Example: Library Checkout



# Les opérateurs

- Alt : ne s'exécute que si une condition est remplie
- Opt : ne s'exécute que si une condition est vraie
- Par: les fragments s'exécutent en parallèle
- Loop : le fragment peut s'exécuter plusieurs fois
- Region, neg,ref,sd



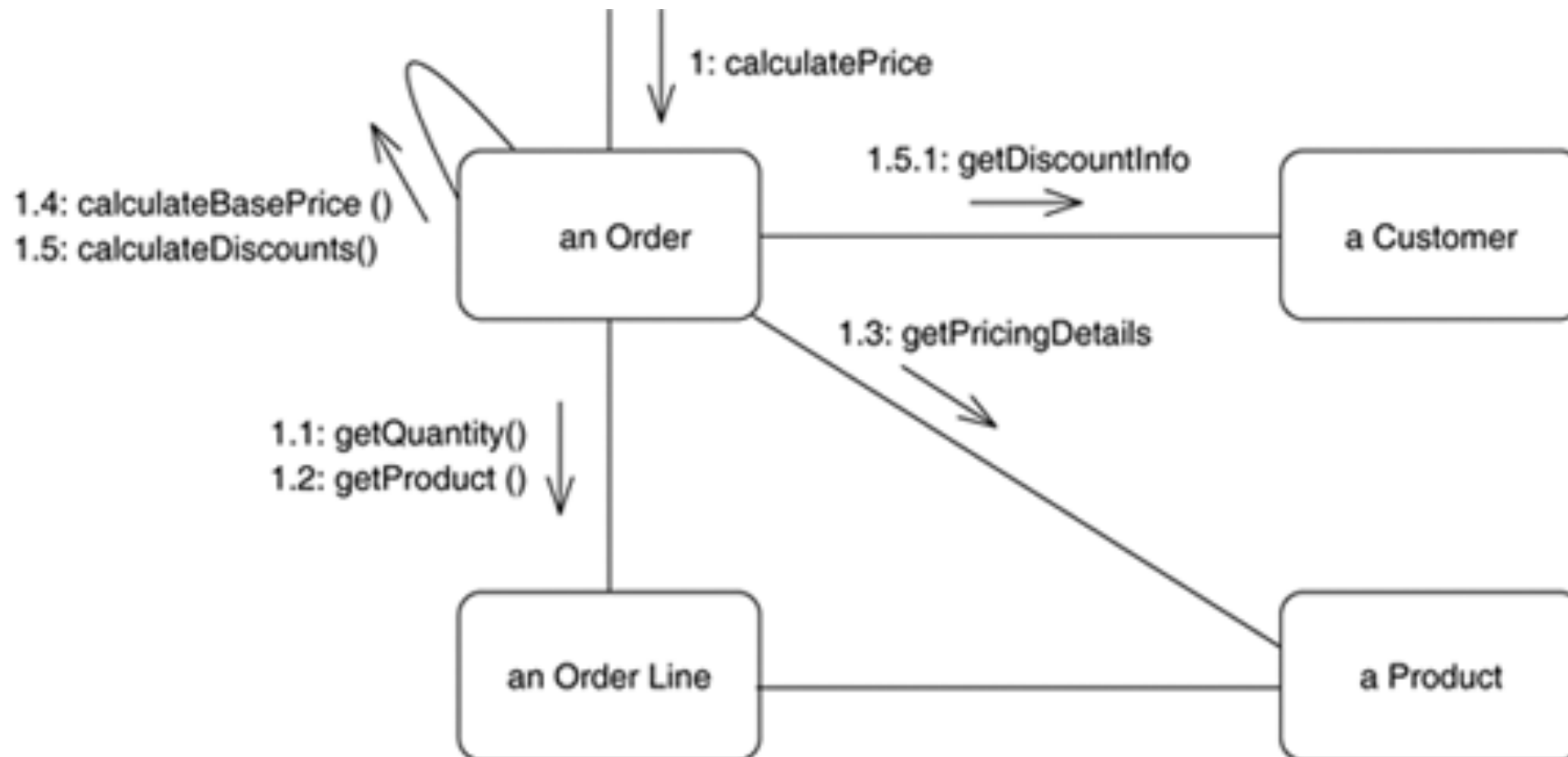
# Utilisation

- Les diagrammes de séquences permettent de mettre en évidence les collaborations entre objets
- Difficile de décrire des comportements précis
- La version UML 2.1 permet des spécifications plus pointues
  - Collaboration
  - Contraintes et invariants
  - ...

# Diagramme de communication

- Autre vue sur les interactions
- Met en évidence les collaborations entre les objets
- Graphe
  - les sommets sont des objets ;
  - les arcs mettent en évidence les relations entre les objets ;
  - les arcs portent des messages, associés à des numéros de séquence, définissant la chronologie.

# Example



# Les messages

- Quelques exemples d'envois de messages
  - 4 : Afficher(x, y)
  - 3.3.2 : Afficher(x,y)
  - 2.6 : age := soustraire (Aujourd'hui, dateDeNaissance)
  - [age >=16 ans] 6.2 : conduire()
  - 1 \* : noter ( )
- Il existe encore d'autres notations pour les flots parallèles.

# Conclusion

- Spécification du comportement des objets
- Permet de décrire une vue de la dynamique du modèle
  - Interactions entre les objets
  - Echanges de messages
  - Création/Suppression des objets
- Utilisable pour l'analyse, pour la description des cas d'utilisation et pour la conception

See also <http://www.agilemodeling.com/style/collaborationDiagram.htm>

# ACTIVITY DIAGRAM

---

# Diagrammes d'activité

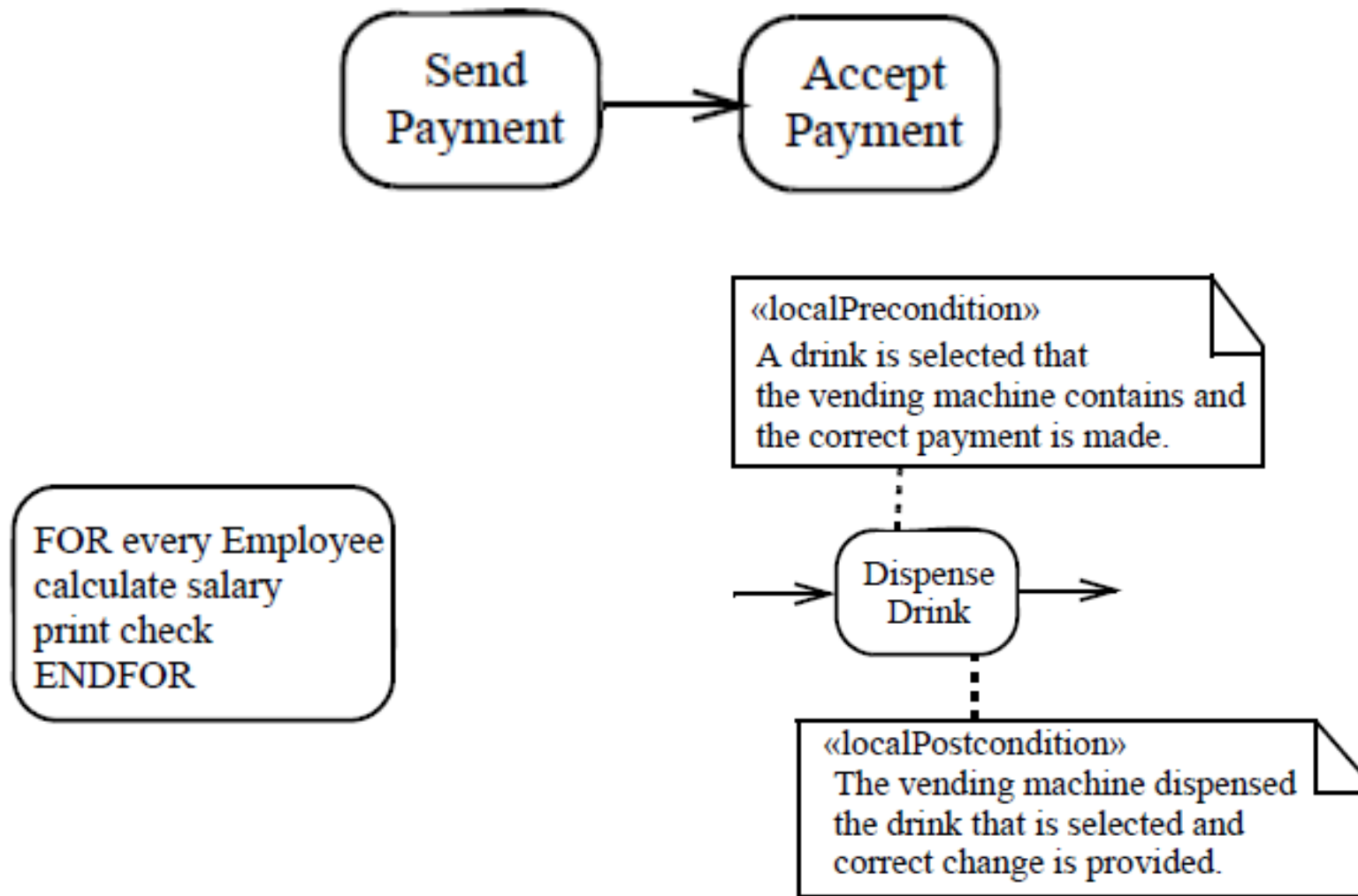
- Visualisation du flux des activités
- Usages
  - Description des cas d'utilisation (métier)
  - Description de la logique procédurale
  - Description des processus métiers
  - Description des workflows
  - Nouveauté de UML 2

# Actions et activité

- Action : étape à l'intérieur d'une activité
  - Non décomposable
- Activité : comportement composé d'actions
- Une action peut déclencher l'invocation d'une activité
- Une action a des liens entrants et sortants vers des activités ou des actions
  - Flot de contrôle
  - Flot de données
- Une action a des conditions d'activation.

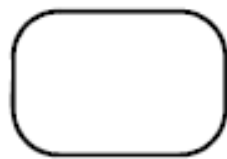


# Exemple d'actions



# Nœuds (Activity nodes)

- Niveau abstrait d'une étape d'une activité
  - Nœuds exécutable
  - Nœuds de contrôle
  - Nœud objet



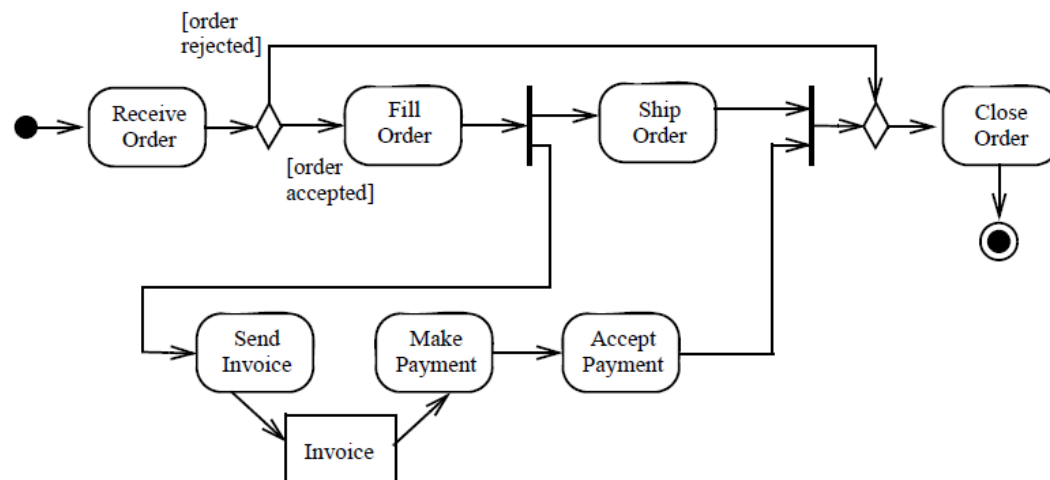
*Action node*



*Object node*

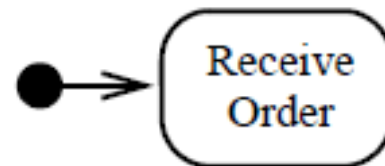


*Control nodes*

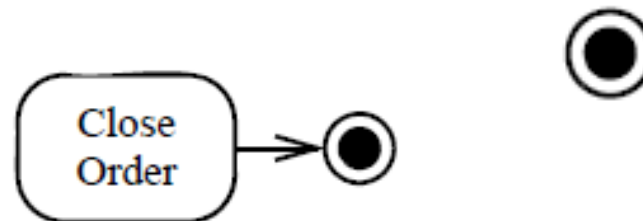


# Noeud initial, noeud final

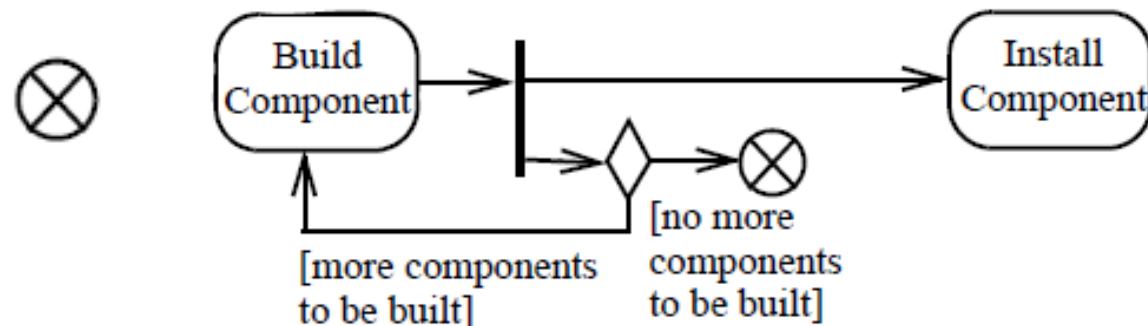
- Début du flot au démarrage de l'activité ●



- Fin de l'activité

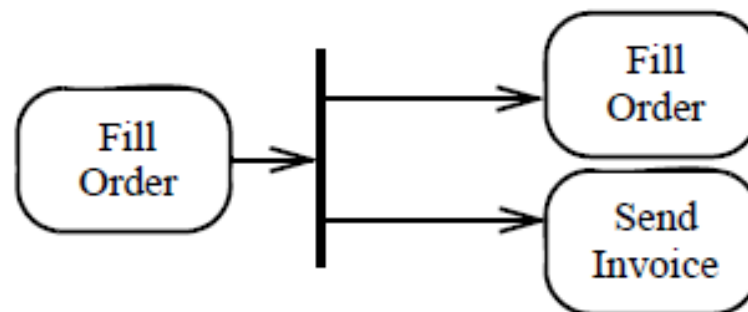
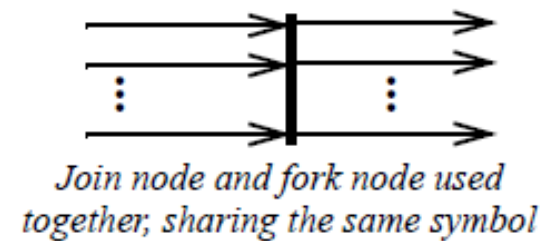
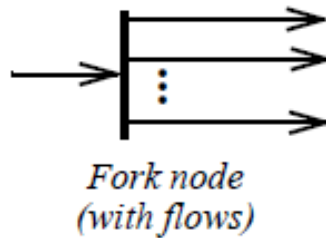
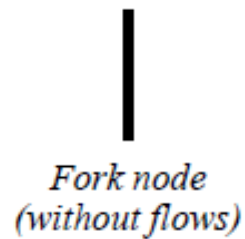


- Fin d'un flot



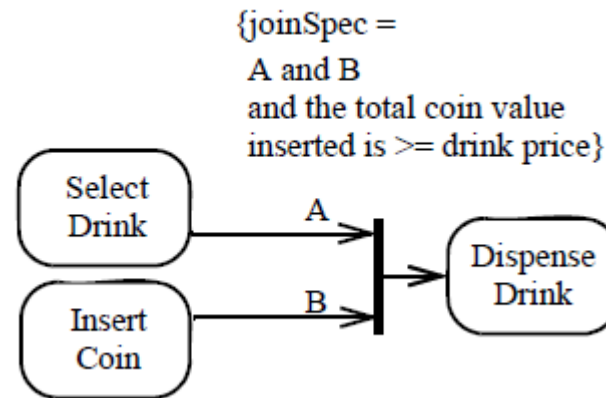
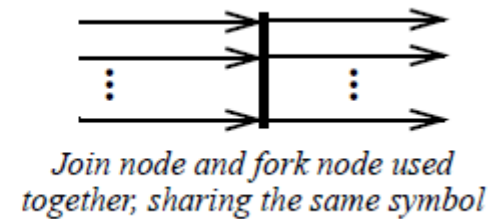
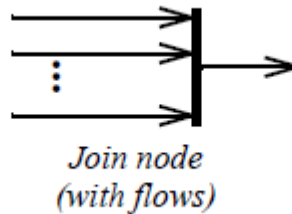
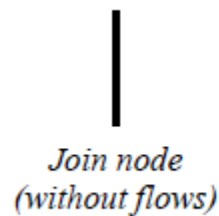
# Fork

- Permet de diviser un flot en plusieurs flots concurrents



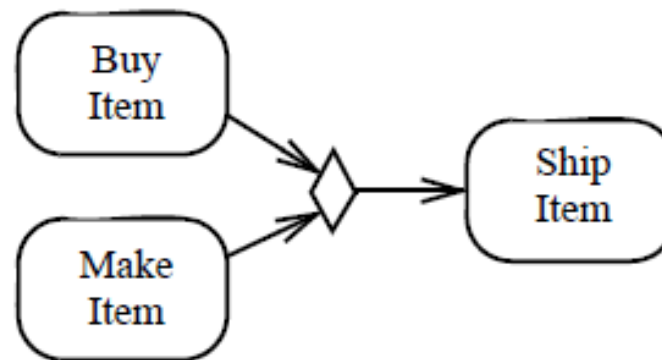
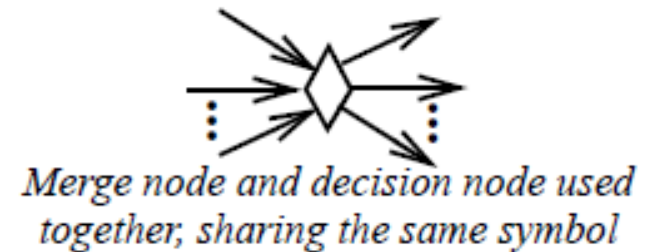
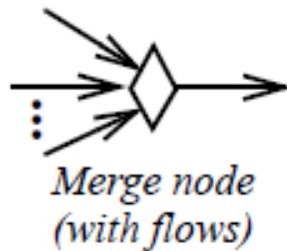
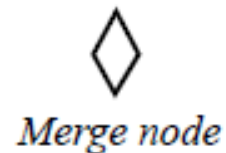
# Nœud join

- Nœud de synchronisation. Le comportement dépend de la notation



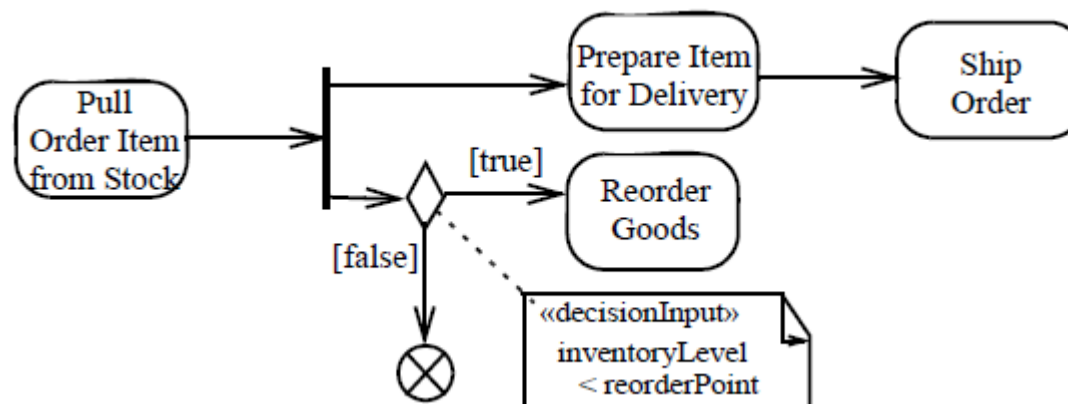
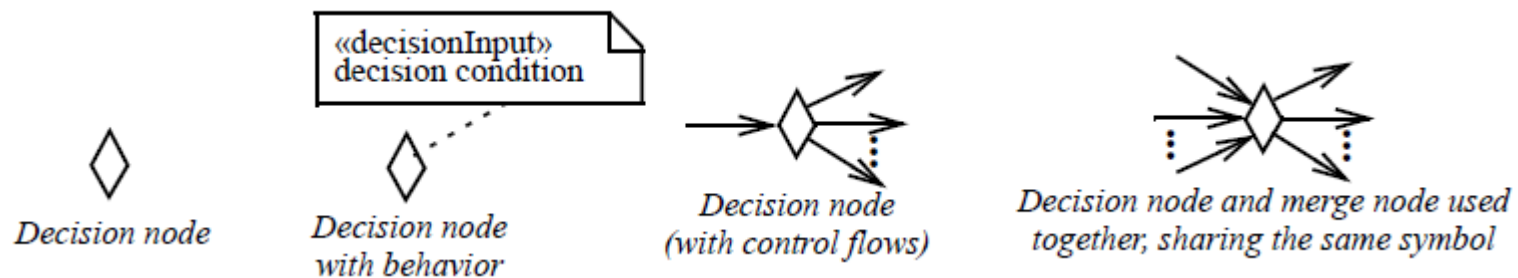
# Noeud merge

- Plusieurs arc entrants et un seul arc sortant
- Permet d'accepter un parmi plusieurs flots entrants
  - Ne sert pas à la synchronisation



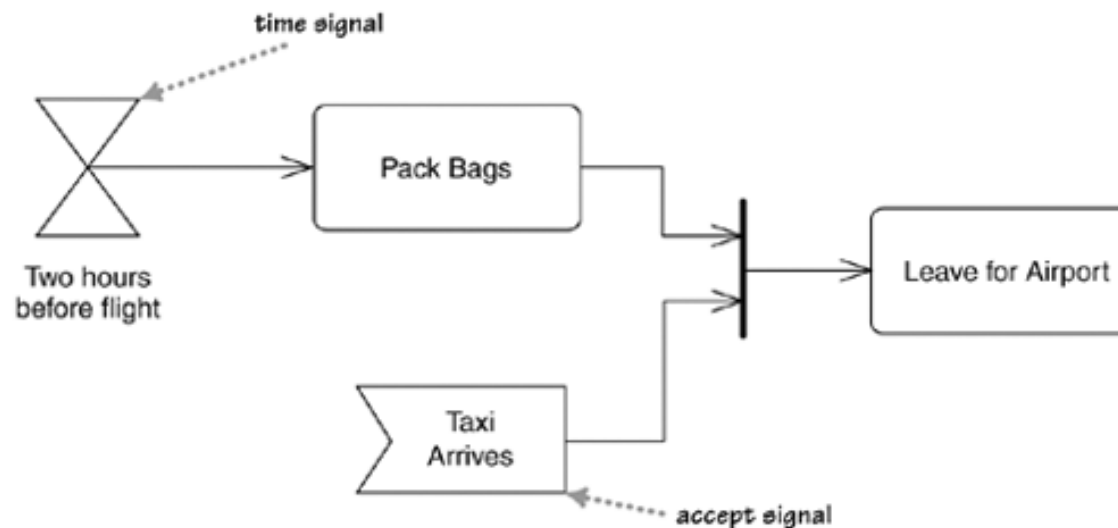
# Nœud décision

- Nœud de contrôle pour choisir entre plusieurs arcs sortants



# Les signaux

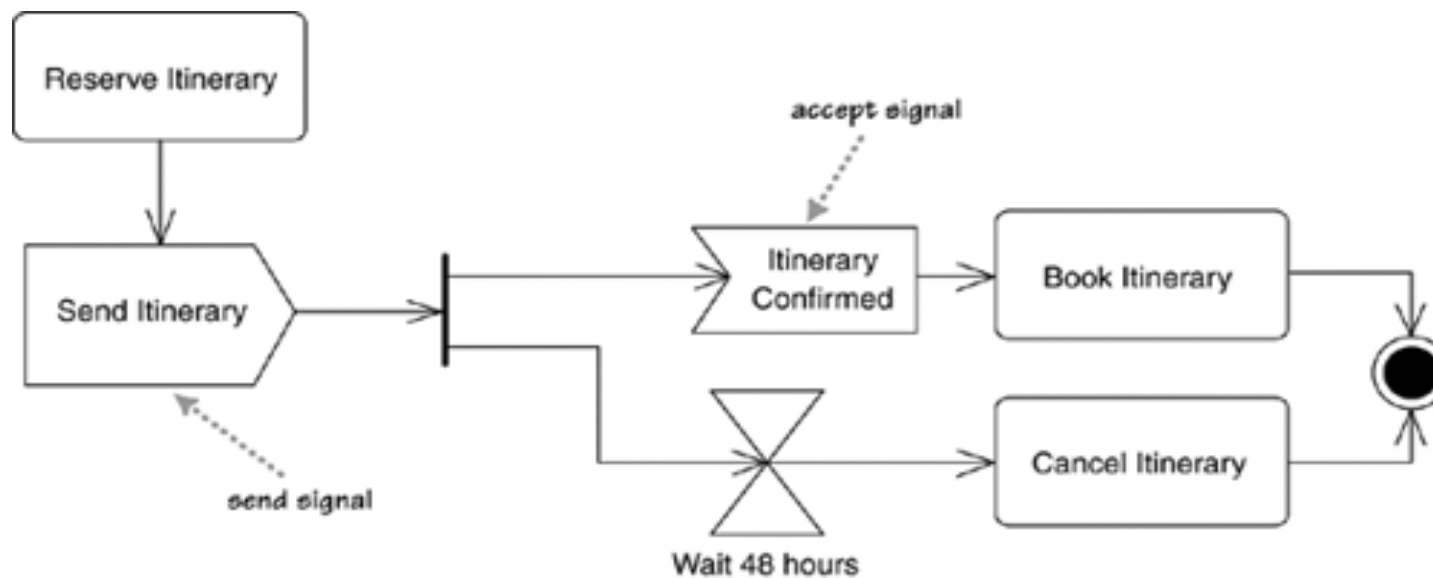
- Les signaux permettent d'indiquer des conditions d'invocation
- Un événement provient d'un processus externe



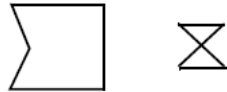
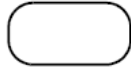

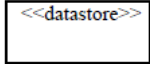




# Les signaux (suite)

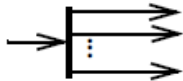

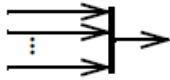
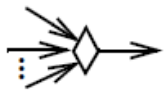
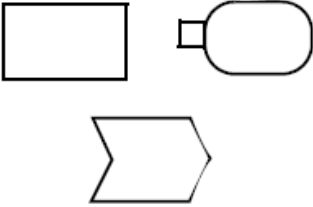
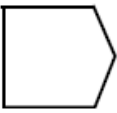
- On peut envoyer un signal

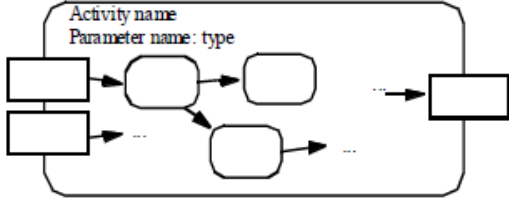
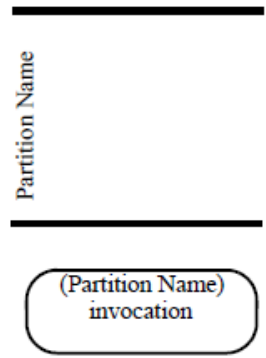
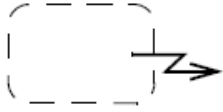


# Récapitulatif des notations

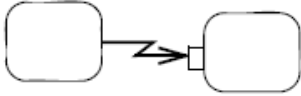
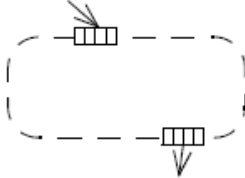
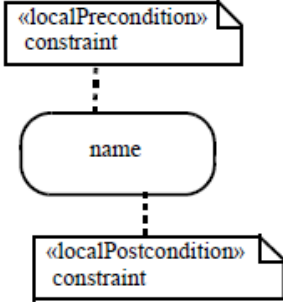
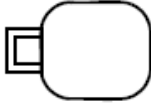
Node Type	Notation	Reference
AcceptEventAction		See “AcceptEventAction (as specialized)” on page 309.
Action		See “Action (from CompleteActivities, FundamentalActivities, StructuredActivities)” on page 311.
ActivityFinal		See “ActivityFinalNode (from BasicActivities, IntermediateActivities)” on page 330.
ActivityNode	<i>See ExecutableNode, ControlNode, and ObjectNode.</i>	See “ActivityNode (from BasicActivities, CompleteActivities, FundamentalActivities, IntermediateActivities, CompleteStructuredActivities)” on page 333.
ControlNode	<i>See DecisionNode, FinalNode, ForkNode, InitialNode, JoinNode, and MergeNode.</i>	See “ControlNode (from BasicActivities)” on page 356.
DataStore		See “DataStoreNode (from CompleteActivities)” on page 358.
DecisionNode		See “DecisionNode (from IntermediateActivities)” on page 359.
FinalNode	<i>See ActivityFinal and FlowFinal.</i>	See “FinalNode (from IntermediateActivities)” on page 371.
FlowFinal		See “FlowFinalNode (from IntermediateActivities)” on page 373.

# Récapitulatif des notations

Node Type	Notation	Reference
ForkNode		See “ForkNode (from IntermediateActivities)” on page 374.
InitialNode		See “InitialNode (from BasicActivities)” on page 376.
JoinNode		See “JoinNode (from CompleteActivities, IntermediateActivities)” on page 379.
MergeNode		See “MergeNode (from IntermediateActivities)” on page 385.
ObjectNode		See “ObjectNode (from BasicActivities, CompleteActivities)” on page 391 and its children.
SendSignalAction		See “SendSignalAction (as specialized)” on page 405.

Type	Notation	Reference
Activity		See “Activity (from BasicActivities, CompleteActivities, FundamentalActivities, StructuredActivities)” on page 315.
ActivityPartition		See “ActivityPartition (from IntermediateActivities)” on page 339.
InterruptibleActivityRegion		See “InterruptibleActivityRegion (from CompleteActivities)” on page 377.

# Récapitulatif des notations

Type	Notation	Reference
ExceptionHandler		See “ExceptionHandler (from ExtraStructuredActivities)” on page 361.
ExpansionRegion		“ExpansionRegion (from ExtraStructuredActivities)” on page 366
Local pre- and postconditions.		See “Action (from CompleteActivities, FundamentalActivities, StructuredActivities)” on page 311.
ParameterSet		See “ParameterSet (from CompleteActivities)” on page 397.

# Conclusion

- Permettent la description de la logique métier des applications
- Compréhensible par les analystes métiers.
- Description des enchainements et des activités
- Utilisable à toutes les étapes.

# DIAGRAMME D'ÉTATS

---

# State Diagrams (From Spec)

- The StateMachine package defines a set of concepts that can be used for modeling discrete behavior through finite state transition systems.
- In addition to expressing the behavior of a part of the system, state machines can also be used to express the usage protocol of part of a system. These two kinds of state machines are referred to here as
  - behavioral state machines and
  - protocol state machines.

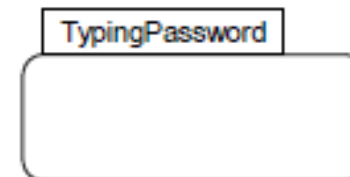
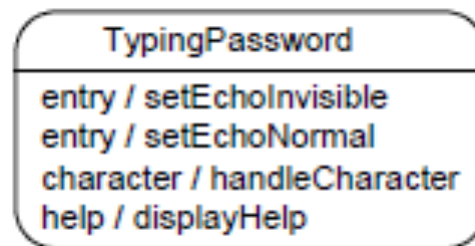
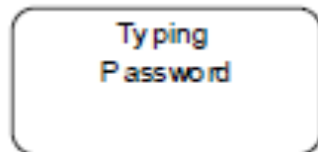


# Diagrammes d'état (comportement)



- Permettent de représenter le cycle de vie d'un objet
- Description des états et des transitions entre les états
- Représentation du comportement d'entités individuelles

# Etat

- Situation durant laquelle une invariant (implicite) est vrai
  - Attente d'un évènement
- Types d'états
  - Simple
  - Composite
  - Sous-Diagramme



# Etat initial et final

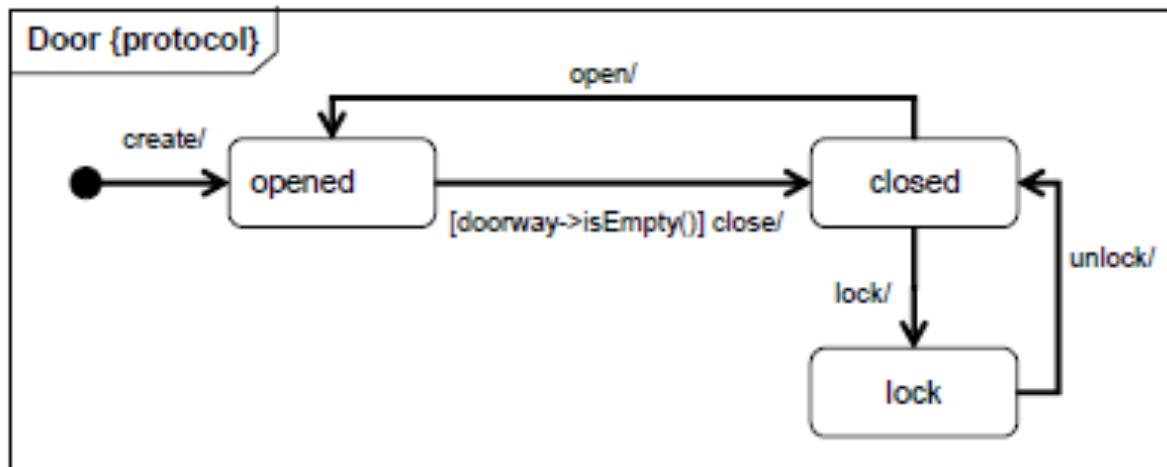
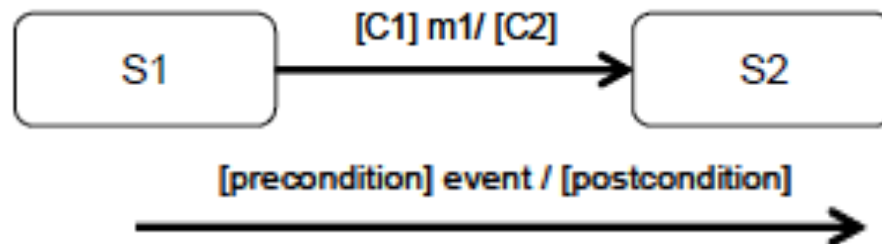
- Etat initial 
- Etat final : la région le contenant est terminée 
- Etat annulée 

# Transition

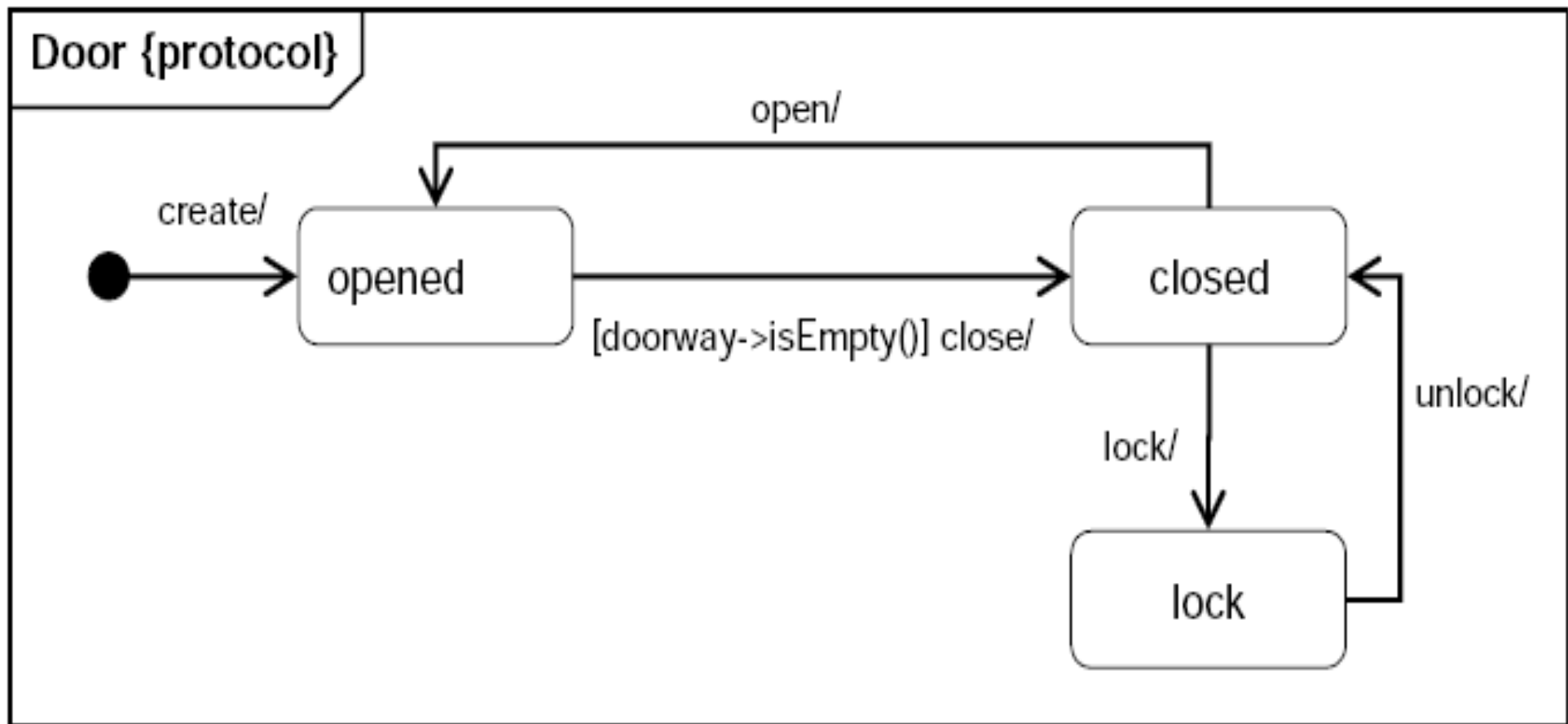
- Une transition est une relation entre deux états définie par :
  - l'état source,
  - l'événement déclencheur permettant le changement d'état,
  - la condition de garde qui n'autorise le changement d'état que si elle est vérifiée,
  - l'action, qui peut agir directement sur l'objet ou indirectement sur les autres,
  - l'état cible.

# Transitions

- m1 peut être appelée sur une instance quand elle est dans l'état S1 et que C1 est vraie.
- Ensuite C2 doit être vraie lorsqu'on est dans l'état S2.



# Transition : [Pre] Event /Post

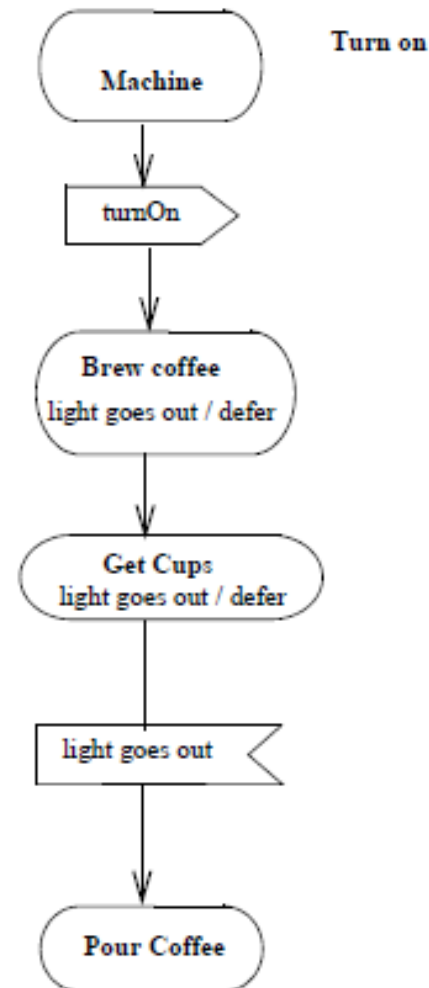
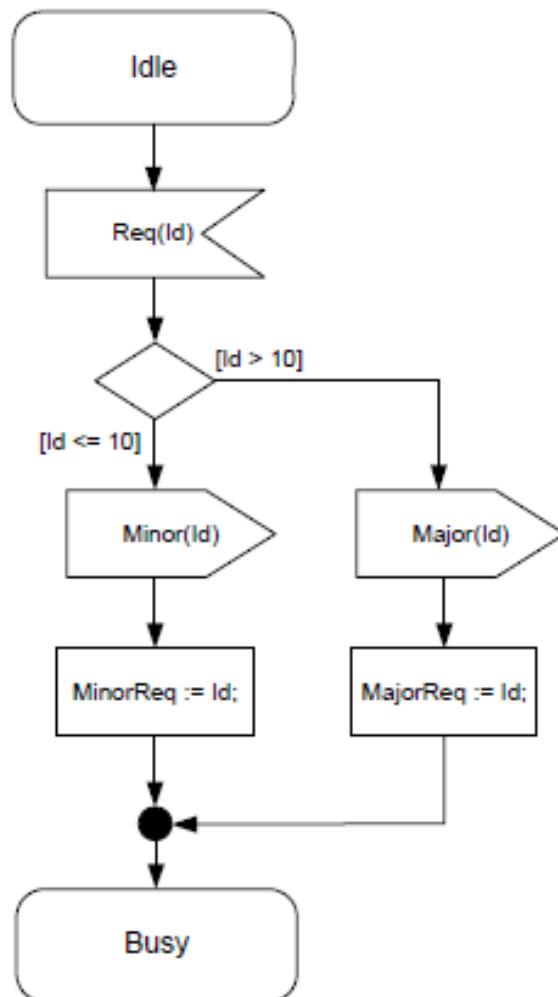


# Événement

- Précise qu'il s'est passé quelque chose de significatif.
  - réception d'un signal, envoyé par un autre objet ou un autre acteur (envoi asynchrone),
  - appel d'une opération sur l'objet récepteur (appel synchrone),
  - passage du temps,
  - changement dans la valeur d'une condition

# Time event

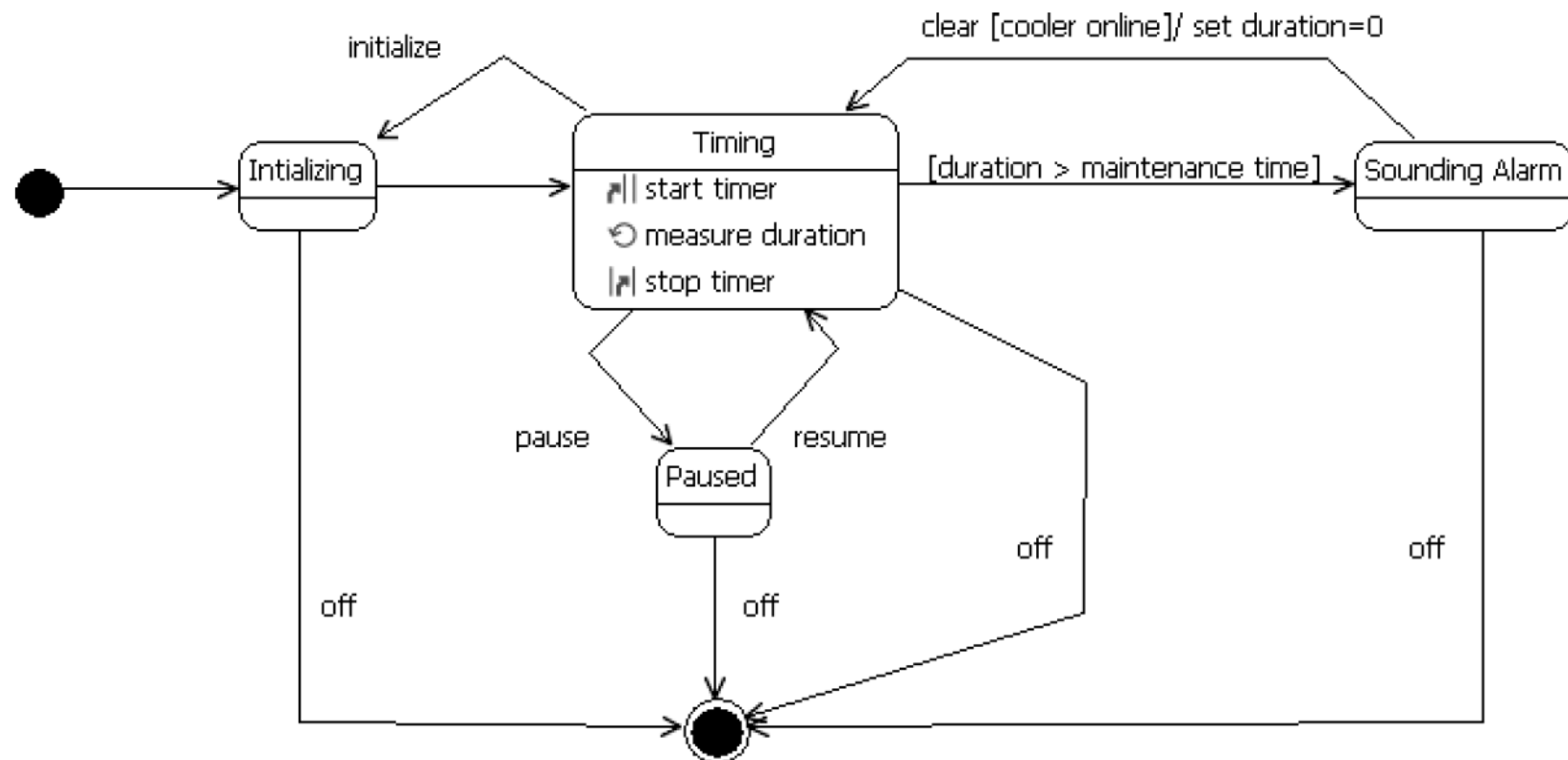
- Emission, réception de signal et exécution d'action sur des transiti





# Contrôle des transitions

- Des conditions peuvent être utilisées pour contrôler les transitions



# Faire un diagramme d'état

- Est-ce qu'il y a un état d'entrée et un état de fin ?
- Quels sont les états intermédiaires
- Quelles sont les transitions entre ces états
- Quels sont les méthodes qui provoquent les transitions (DdC)
- Quelles sont les transitions impossibles (à documenter)
- Pas nécessaire pour toutes les classes.

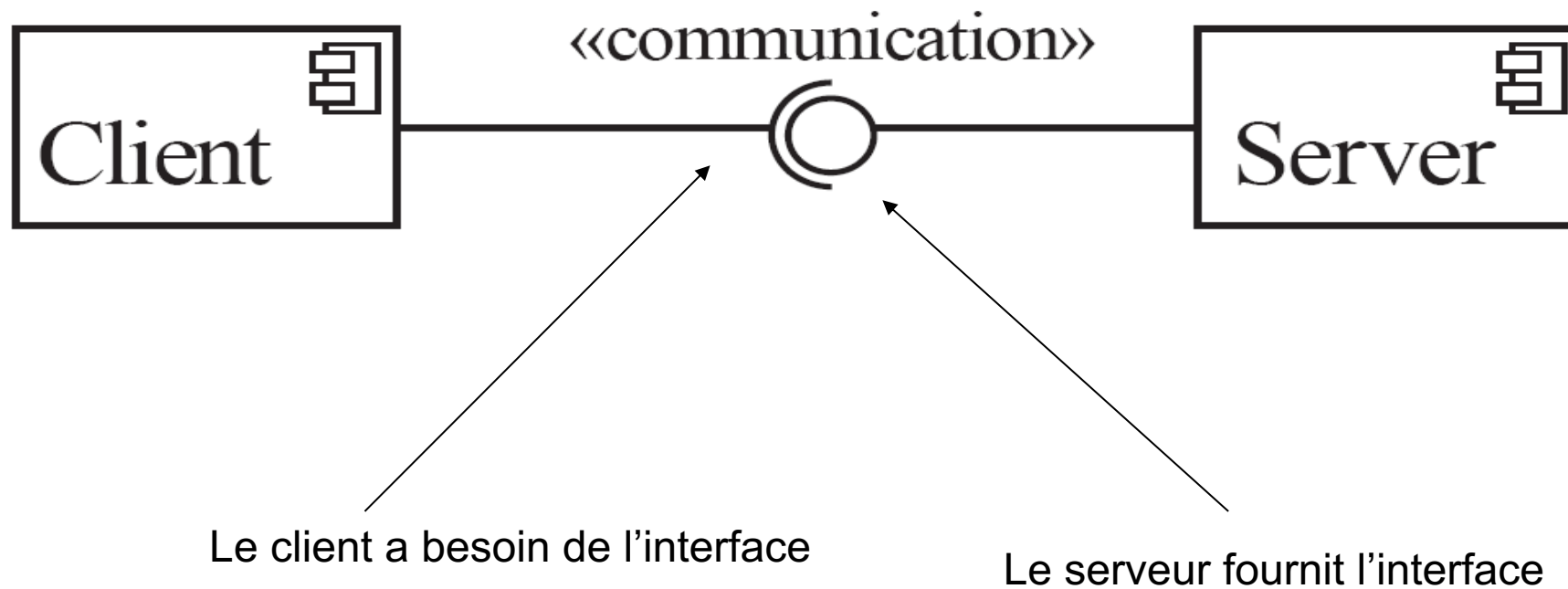
# COMPONENT DIAGRAM

---

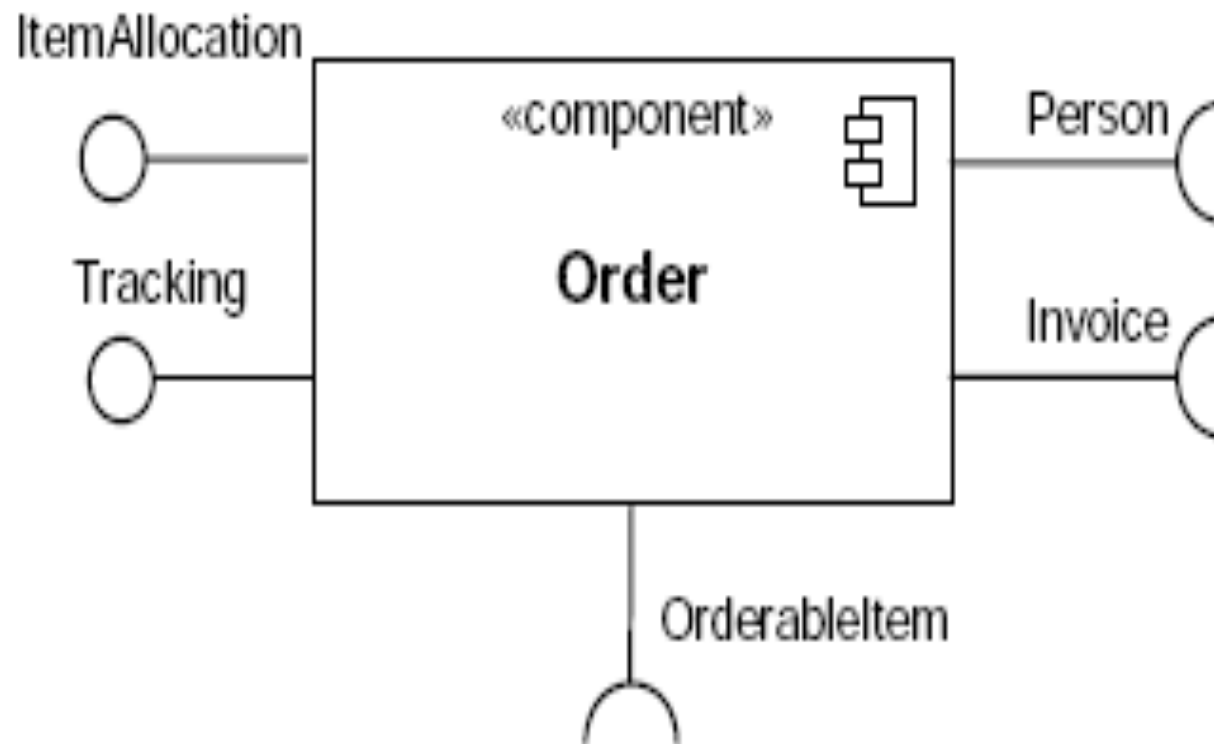
# Components diagram

- From the Spec:
- The Components package specifies a set of constructs that can be used to define software systems of arbitrary size and complexity.
- In particular, the package specifies a component as a modular unit with well-defined interfaces that is replaceable within its environment.

# Component diagrams

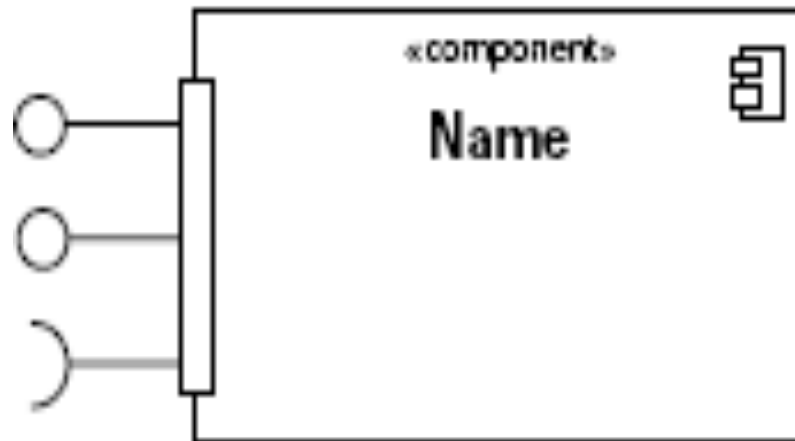


## 2 Provided/3 required



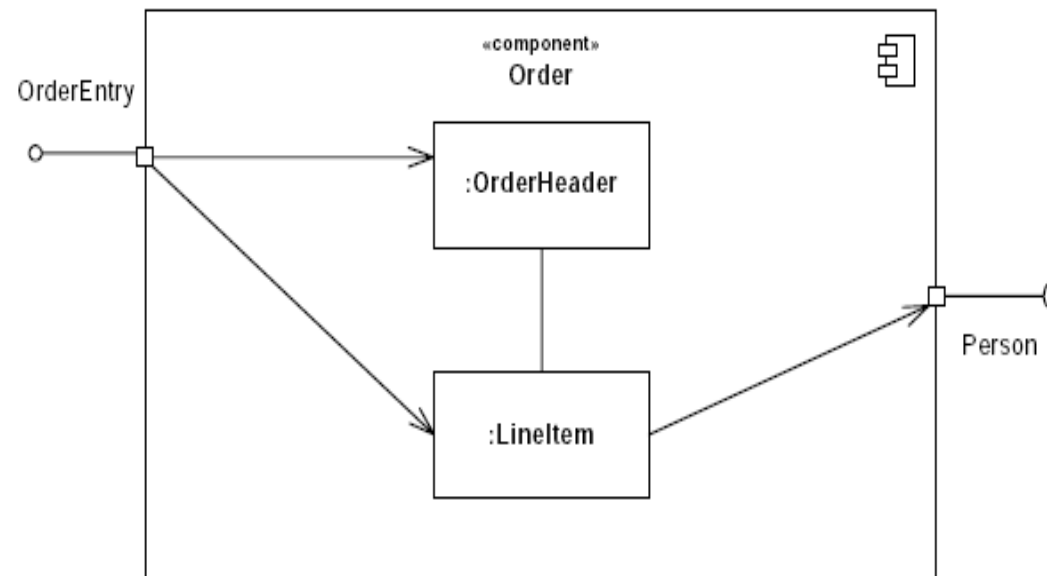
# Les ports

- Décomposition des interfaces fournies et nécessaires en ports.
- Organisation logique
- La liaison est en général (pas toujours) faite à l'exécution.



# Connecteur de délégation

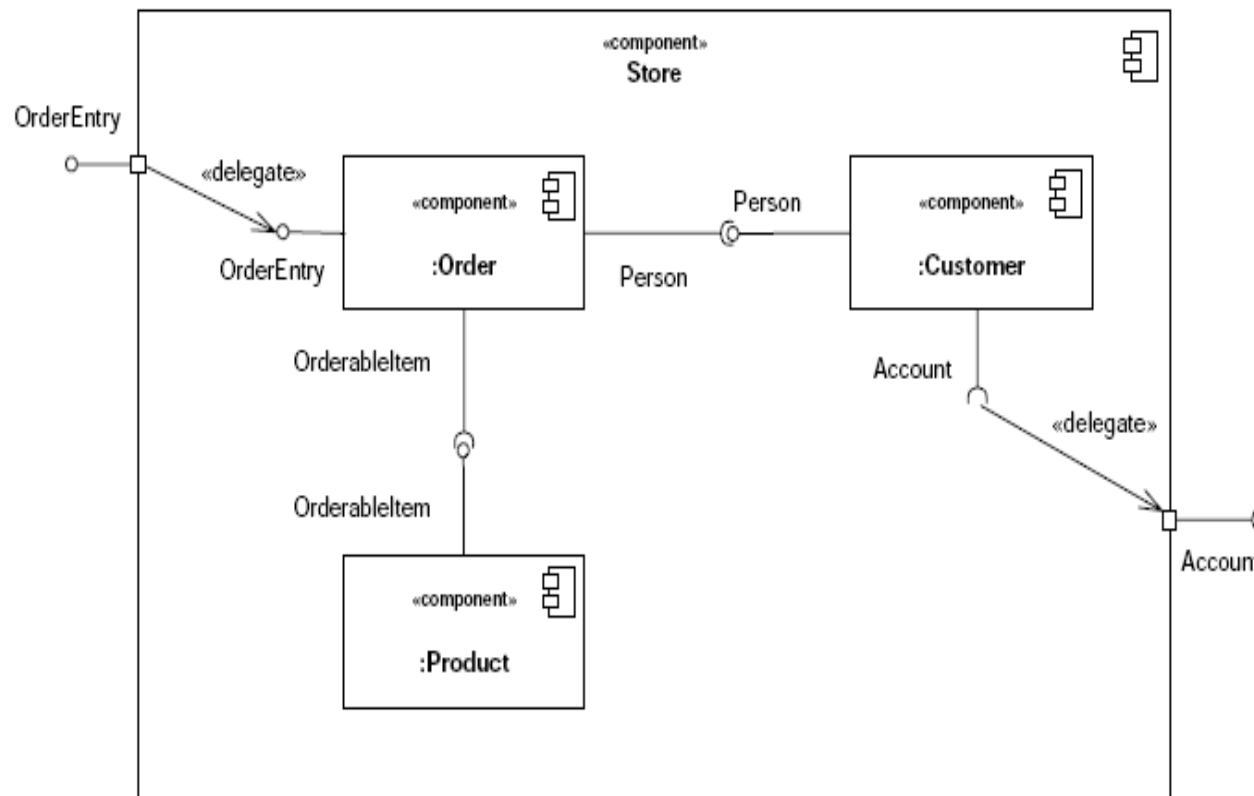
- Lie le contrat externe d'un composant à sa réalisation interne
- Transfert des signaux externes en signaux internes





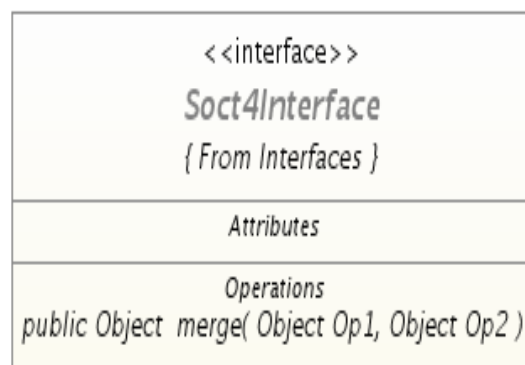
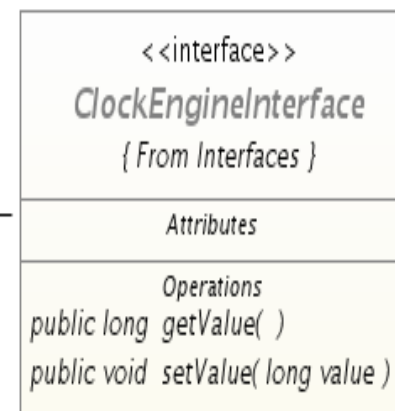
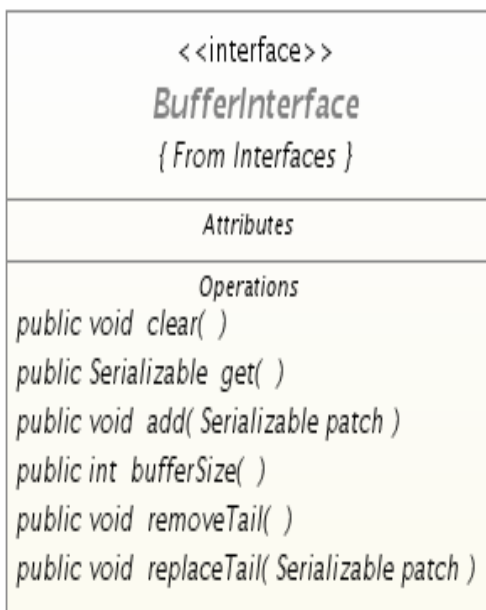
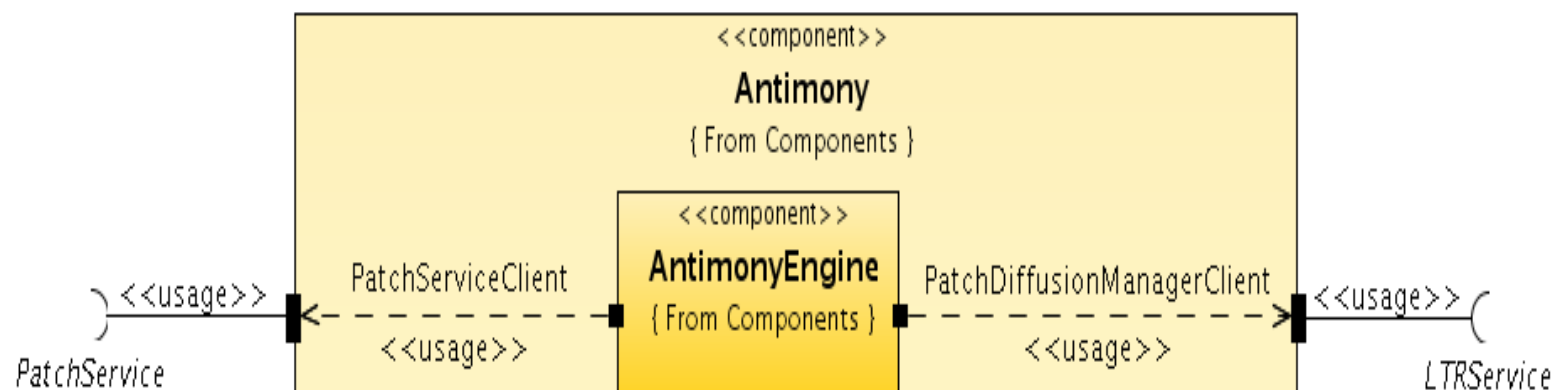
# Autre exemple

- Découplage interface/réalisation



# Sous-systèmes

- On décompose le système en sous-systèmes, en se basant sur
  - les paquetages d'analyse
  - les modèles d'architecture
- Un sous-système a des responsabilités clairement définies ; il communique avec les autres
  - un sous-système = un composant UML



# UML et la conception

- UML est un outil utilisable à plusieurs niveaux
  - Analyse, communication, conception
  - MDA, génération de code
- UML n'est pas une méthode
- Connaitre UML ne veut pas dire savoir concevoir !
- Connaitre UML permet de concevoir de manière moins ambiguë
- Ne pas connaître UML provoque des erreurs graves
- Le **but ultime** est de produire une application qui répond à un cahier des charges