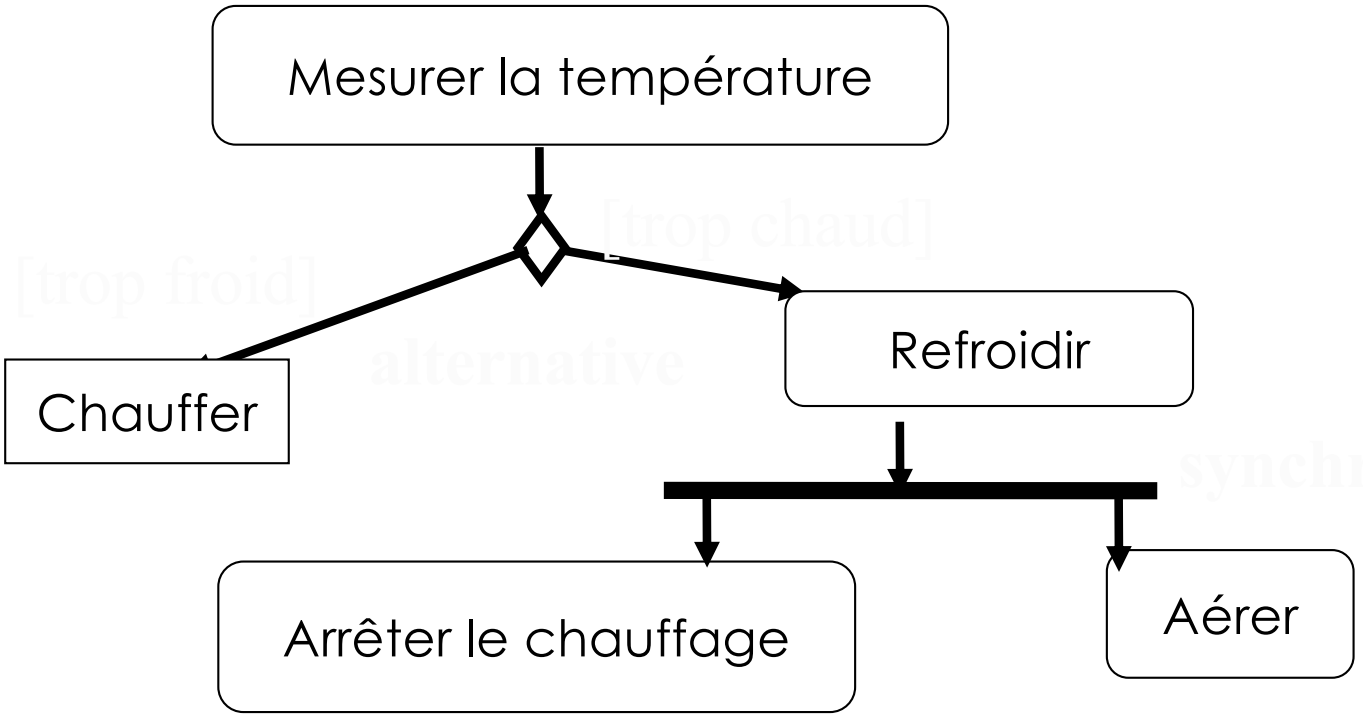
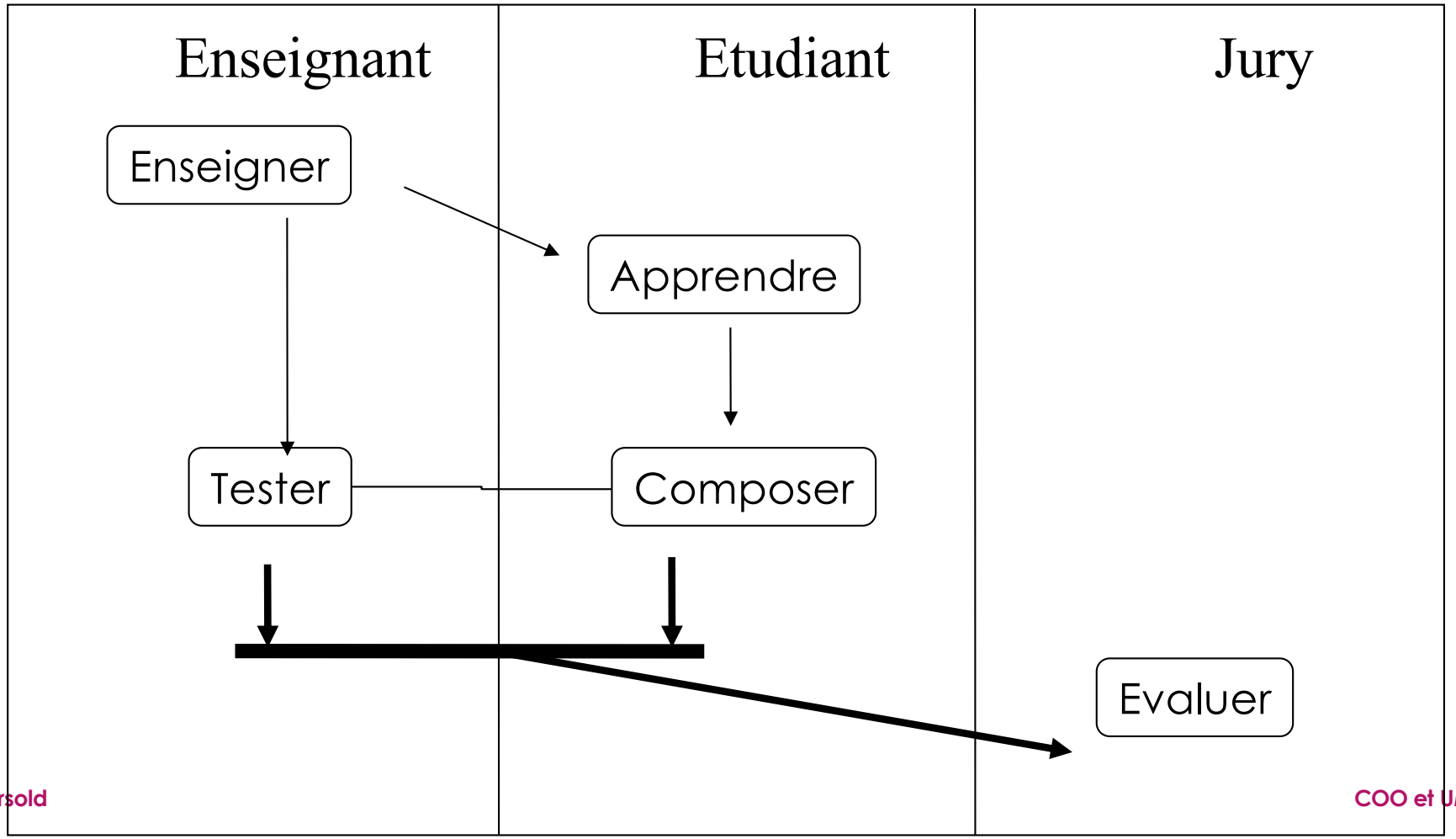
Les diagrammes UML : Diagrammes d'activités

- **Représentent les activités et les transitions entre activités**

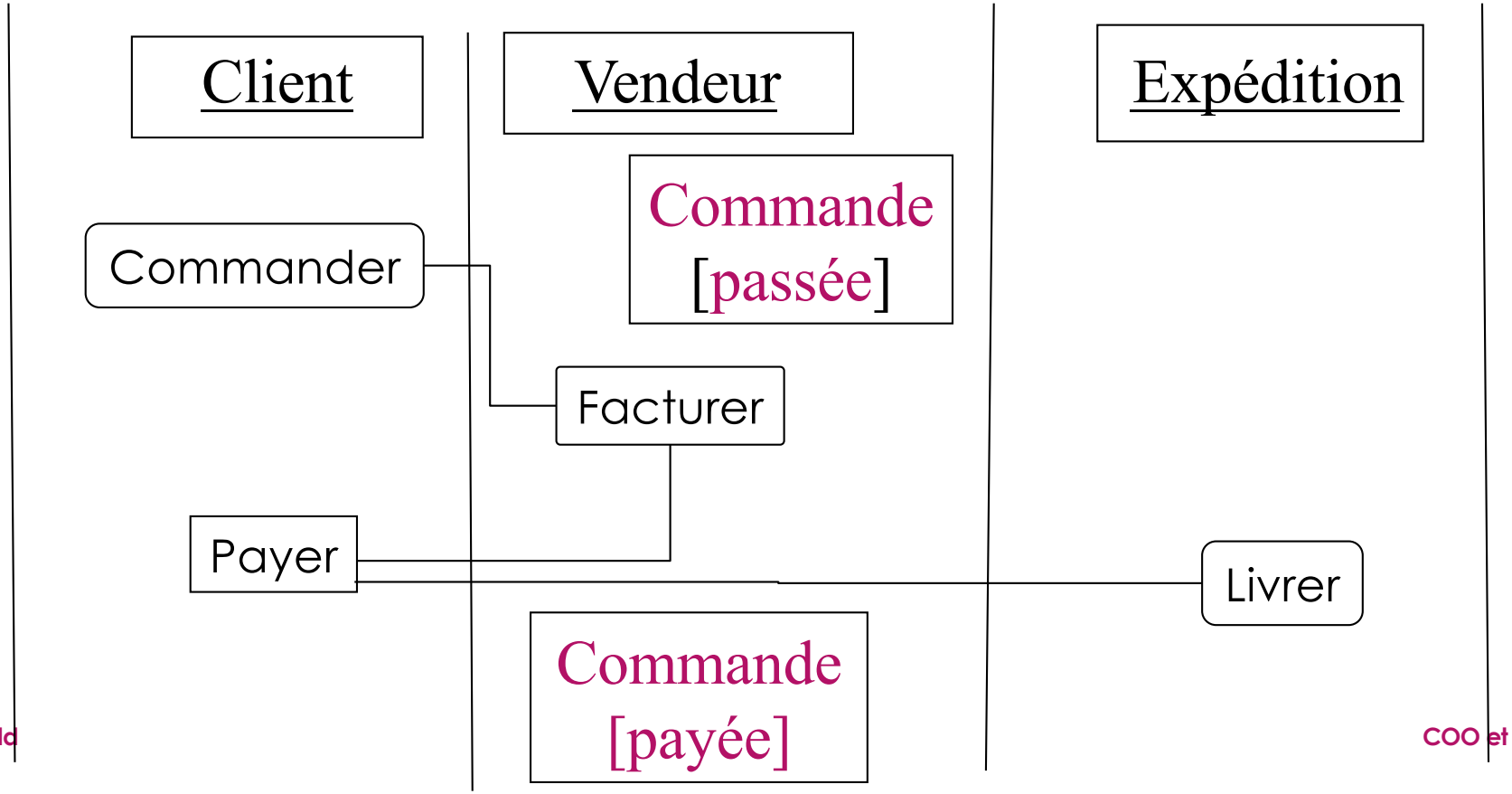


Les diagrammes UML : Diagrammes d'activités



Les diagrammes UML : Diagrammes d'activités

- Peuvent faire apparaître les objets, au sein des couloirs, ou dans des flots d'objets (spécifiant l'état)



Nœuds d'action

- Peuvent comporter des actions élémentaires

Action appeler (*call operation*) – L'action *call operation* correspond à l'invocation d'une opération sur un objet de manière synchrone ou asynchrone. Lorsque l'action est exécutée, les paramètres sont transmis à l'objet cible. Si l'appel est asynchrone, l'action est terminée et les éventuelles valeurs de retour seront ignorées. Si l'appel est synchrone, l'appelant est bloqué pendant l'exécution de l'opération et, le cas échéant, les valeurs de retour pourront être réceptionnées.

Action comportement (*call behavior*) – L'action *call behavior* est une variante de l'action *call operation* car elle invoque directement une activité plutôt qu'une opération.

Action envoyer (*send*) – Cette action crée un message et le transmet à un objet cible, où elle peut déclencher un comportement. Il s'agit d'un appel asynchrone (*i.e.* qui ne bloque pas l'objet appelant) bien adapté à l'envoi de signaux (*send signal*).

Action accepter événement (*accept event*) – L'exécution de cette action bloque l'exécution en cours jusqu'à la réception du type d'événement spécifié, qui généralement est un signal. Cette action est utilisée pour la réception de signaux asynchrones.

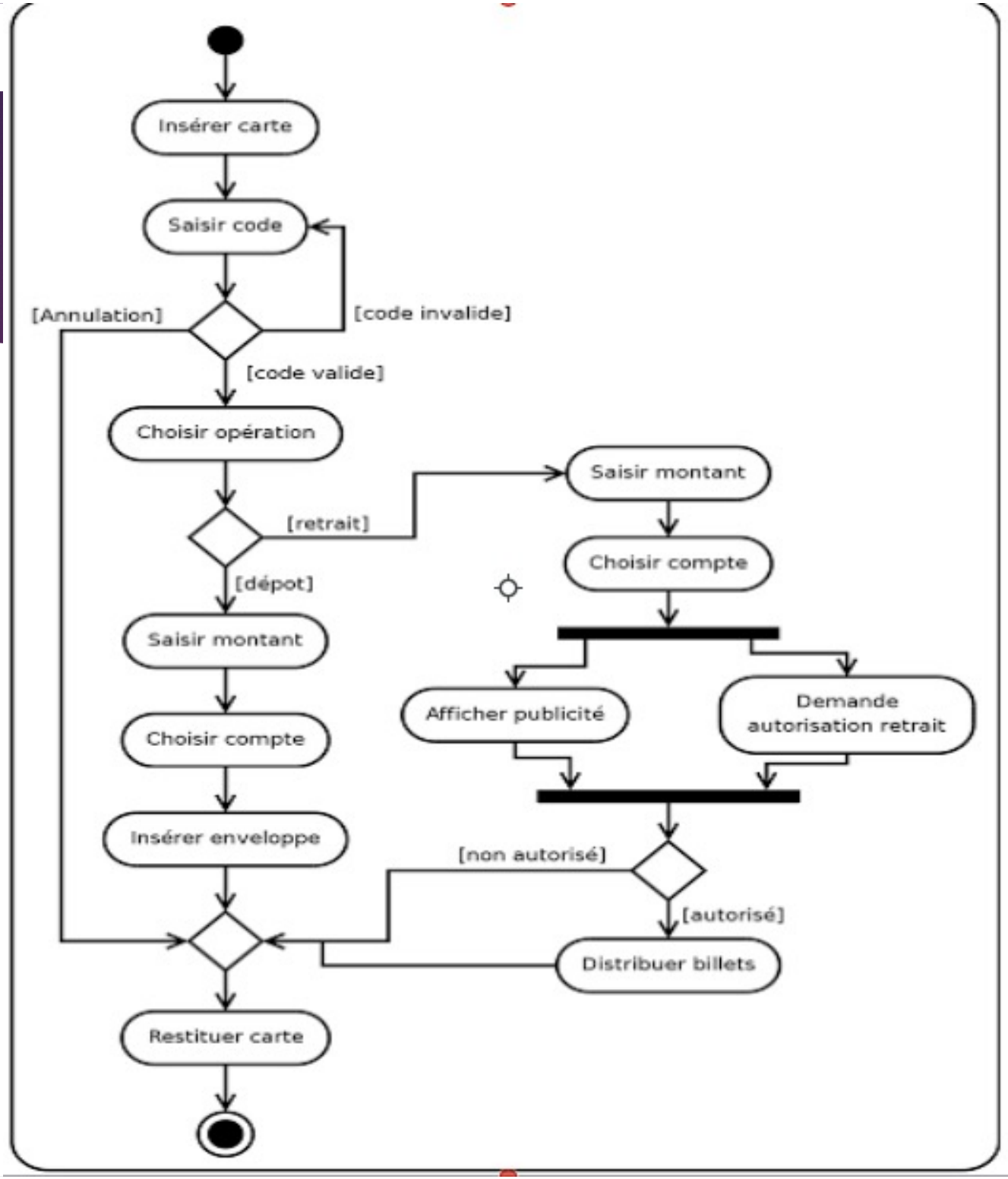
Action accepter appel (*accept call*) – Il s'agit d'une variante de l'action *accept event* pour les appels synchrones.

Action répondre (*reply*) – Cette action permet de transmettre un message en réponse à la réception d'une action de type *accept call*.

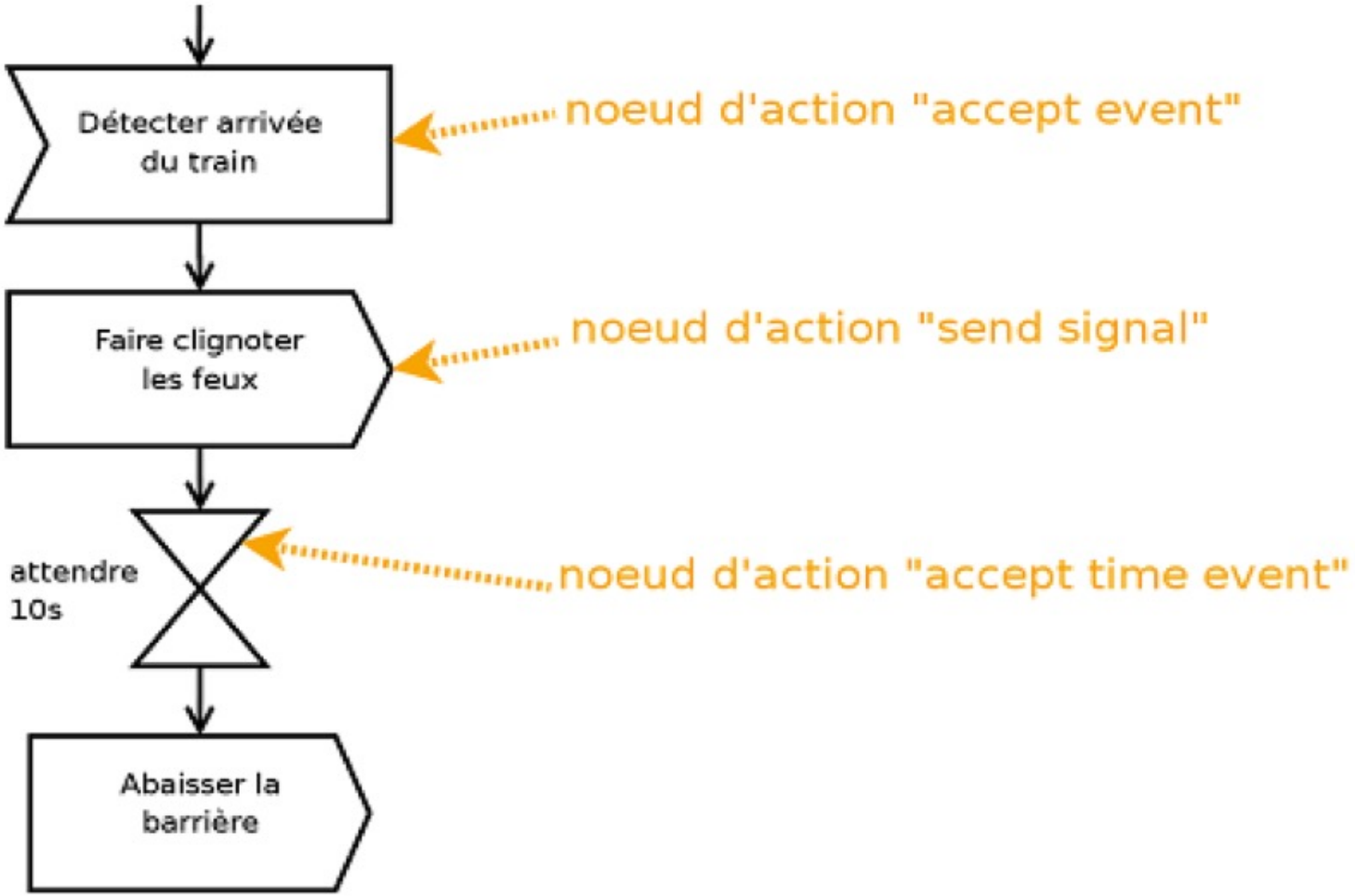
Action créer (*create*) – Cette action permet d'instancier un objet.

Action détruire (*destroy*) – Cette action permet de détruire un objet.

Action lever exception (*raise exception*) – Cette action permet de lever explicitement une exception.


Insérer carte
Saisir code
[Annulation]
[code invalide]
[code valide]
Choisir opération
[retrait]
Saisir montant
Choisir compte
[dépot]
Saisir montant
Choisir compte
Insérer enveloppe
Afficher publicité
Demande autorisation retrait
[non autorisé]
[autorisé]
Distribuer billets
Restituer carte

Formalisme : actions élémentaires

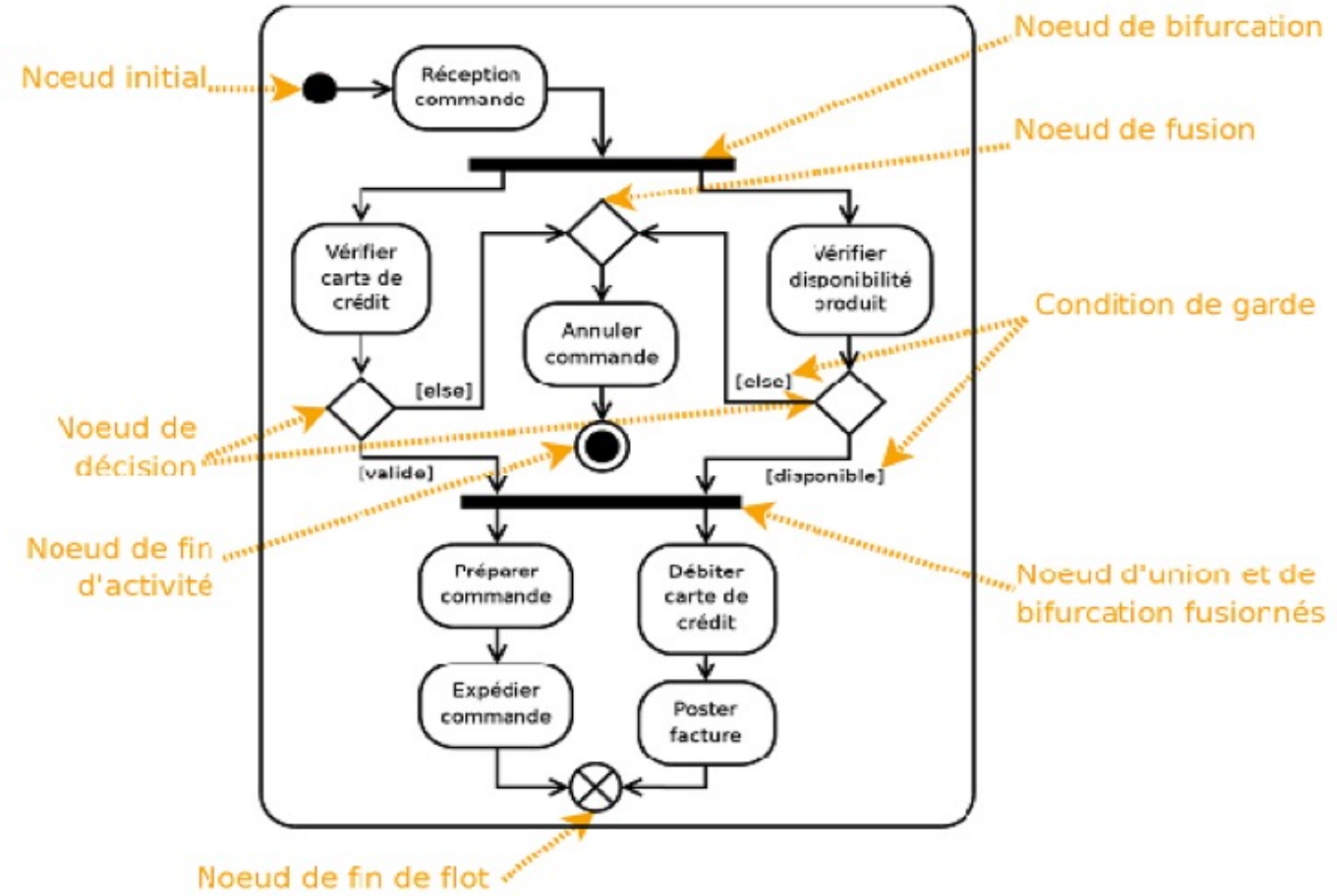


Nœuds de contrôles

- Nœud d'activité abstrait
  utilisé pour coordonner les activités

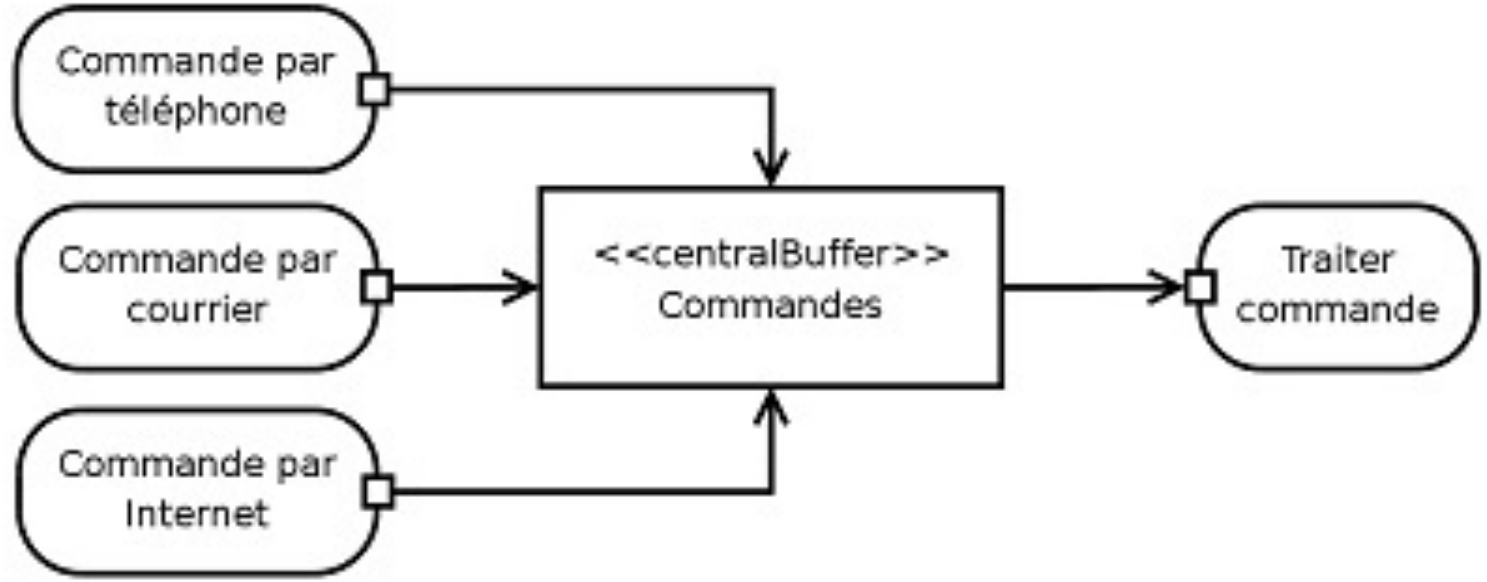
  - nœud initial (*initial node* en anglais) ;
  - nœud de fin d'activité (*final node* en anglais)
  - nœud de fin de flot (*flow final* en anglais) ;
  - nœud de décision (*decision node* en anglais) ;
  - nœud de fusion (*merge node* en anglais) ;
  - nœud de bifurcation (*fork node* en anglais) ;
  - nœud d'union (*join node* en anglais).

Nœuds de contrôles



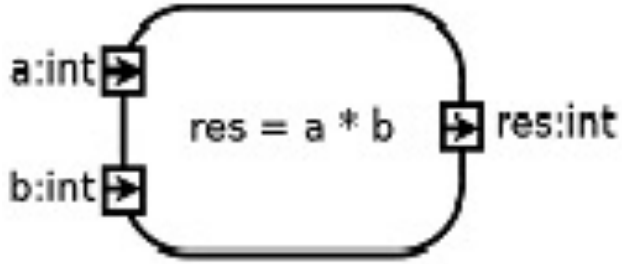
Nœuds de stockages de données

- Nœuds d'objets

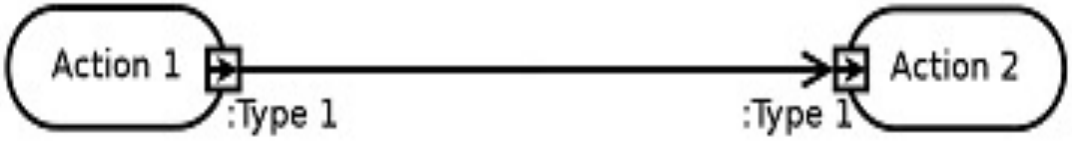


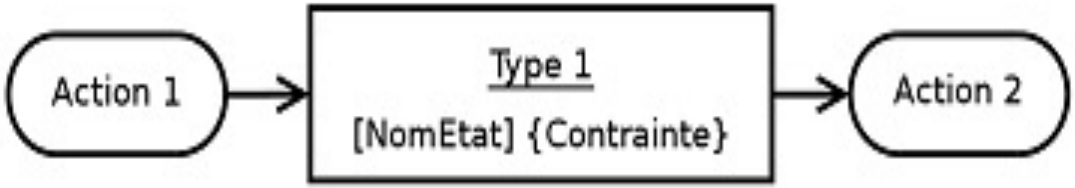
Flots d'objets

- Nœuds objets
  - Objet nommé
  - Etat

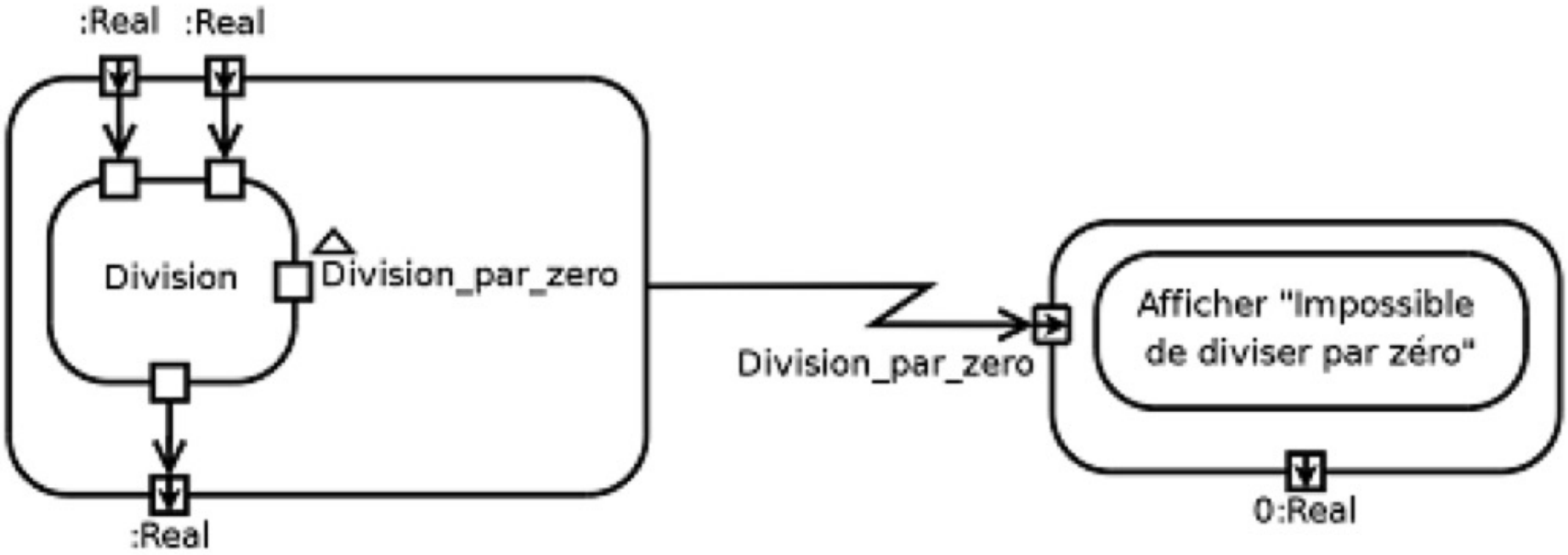


- Pin d'Entrée/Sortie
  - Valeur d'objet non générée par un flot
  - Doivent être valués pour que l'activité puisse s'exécuter
  - Idem en sortie





Exceptions


:Real
:Real
Division
Division_par_zero
:Real
Division_par_zero
Afficher "Impossible de diviser par zéro"
0:Real

UML : Diagramme d'activités : jouer

?

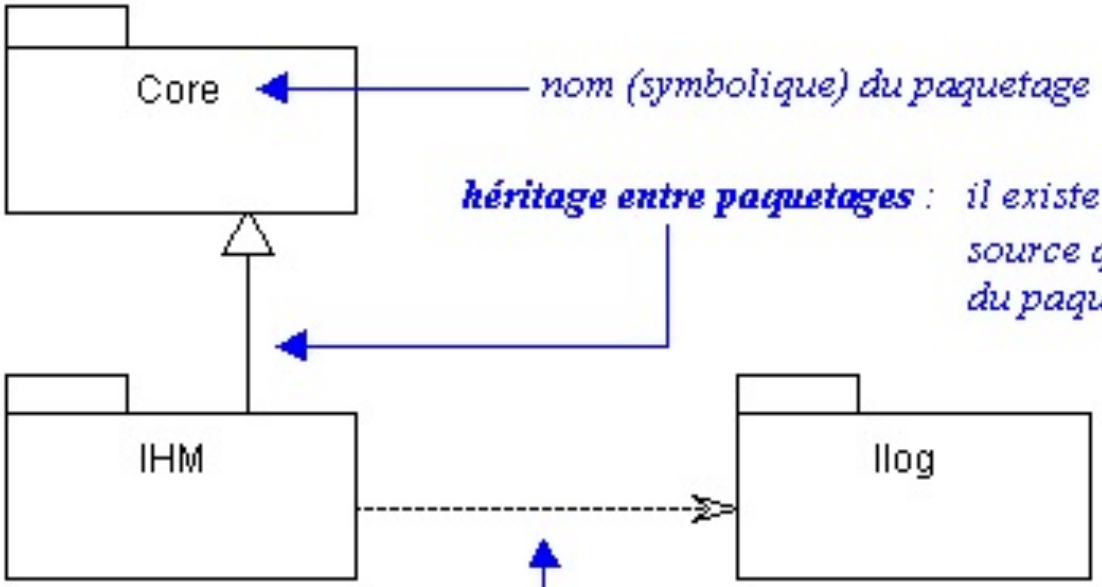
Conception Modulaire

- Diagrammes de paquetages
- Diagrammes de composants
- Diagrammes de déploiement
- Diagrammes des interactions

Diagrammes de Paquetages

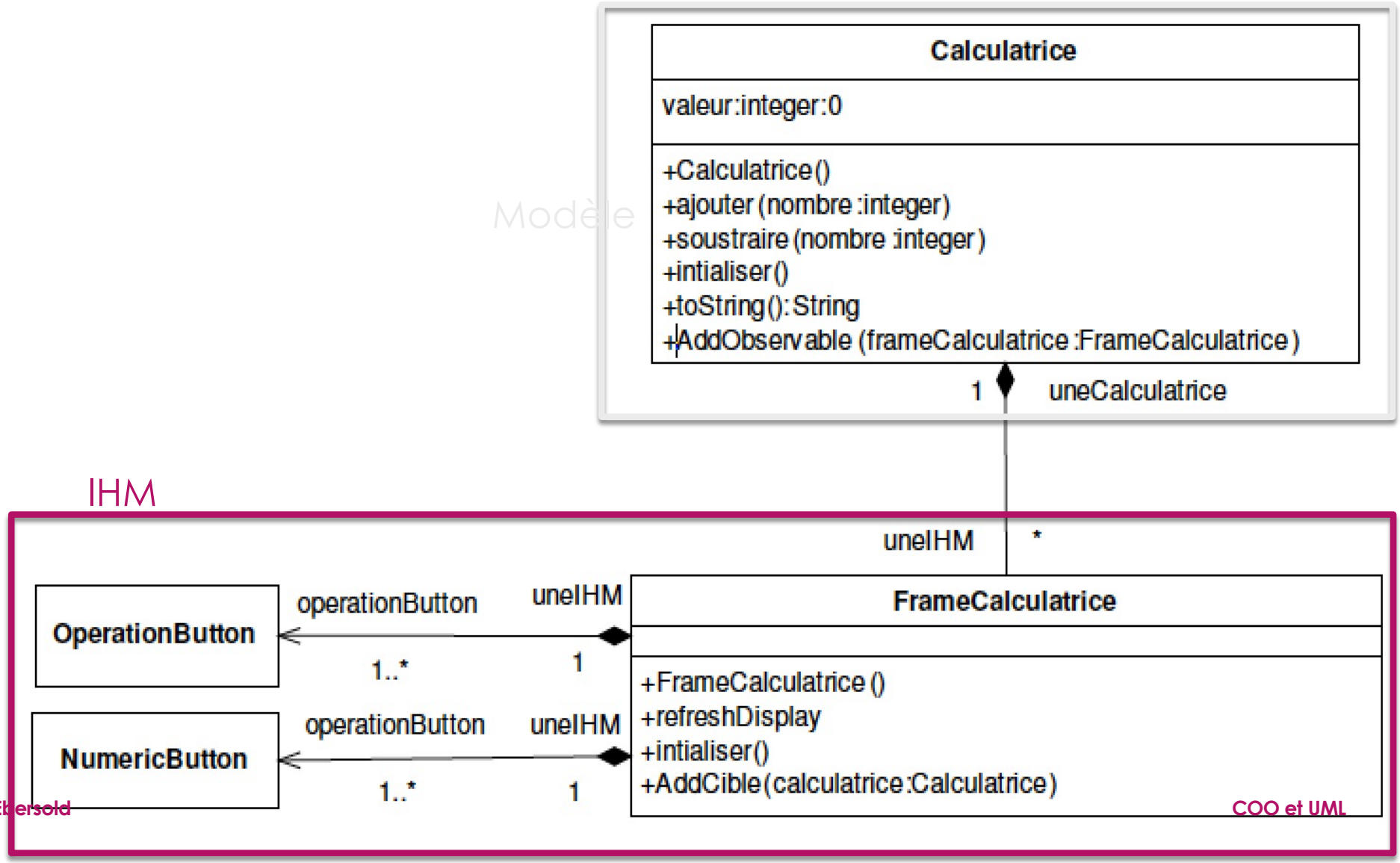
- Utilisés pour séparer le modèle en conteneurs logiques, et décrire leurs interactions à un haut niveau

Exemple de Paquetages



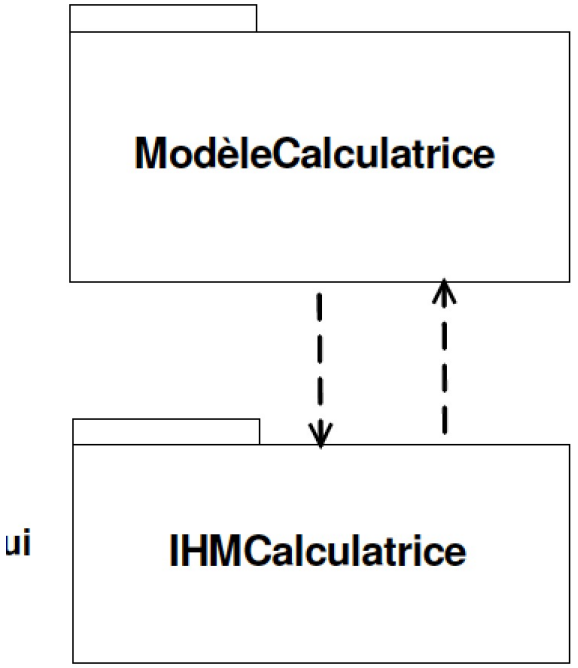
- *au moins un élément du paquetage source utilise les services d'au moins un élément du paquetage destination*
- *représente une relation d'obsolescence entre les éléments des deux paquetages.*

Exemple : paquetages Calculatrice



Exemple : Calculatrice

- Architecture de l'application en 2 couches
  - 2 paquetages
    - ModèleCalculatrice:
      Contient tous les éléments du modèle
    - IHMCalculatrice :
      Tous les éléments Calculatrice

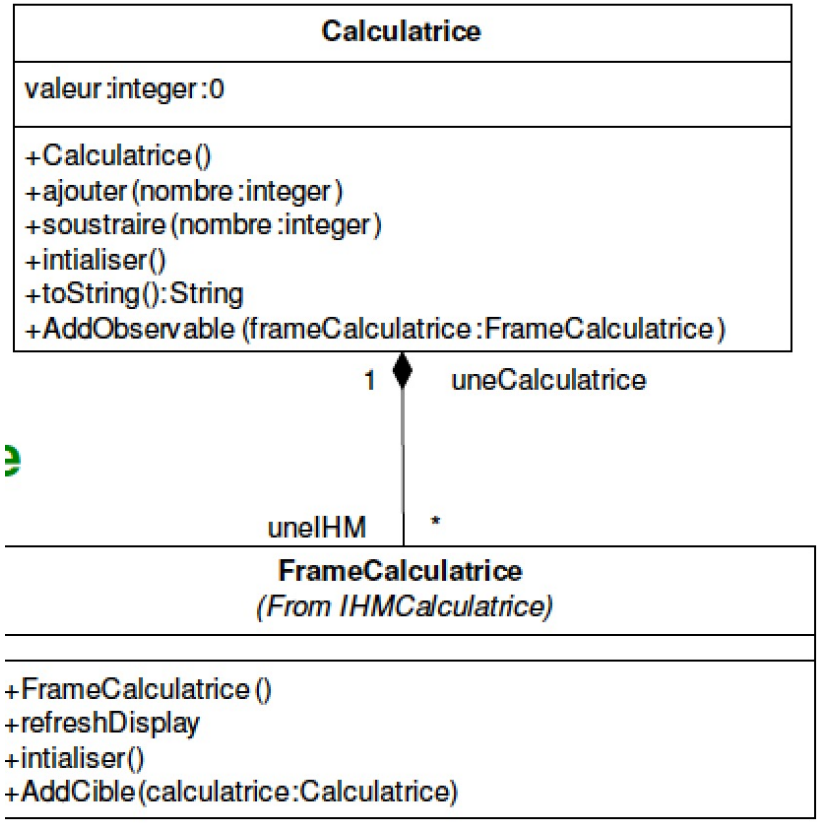


- Liés par des liens de dépendances :
  Tous les éléments du paquetage cible sont utilisables par le paquetage source

Exemple : paquetage ModèleCalculatrice

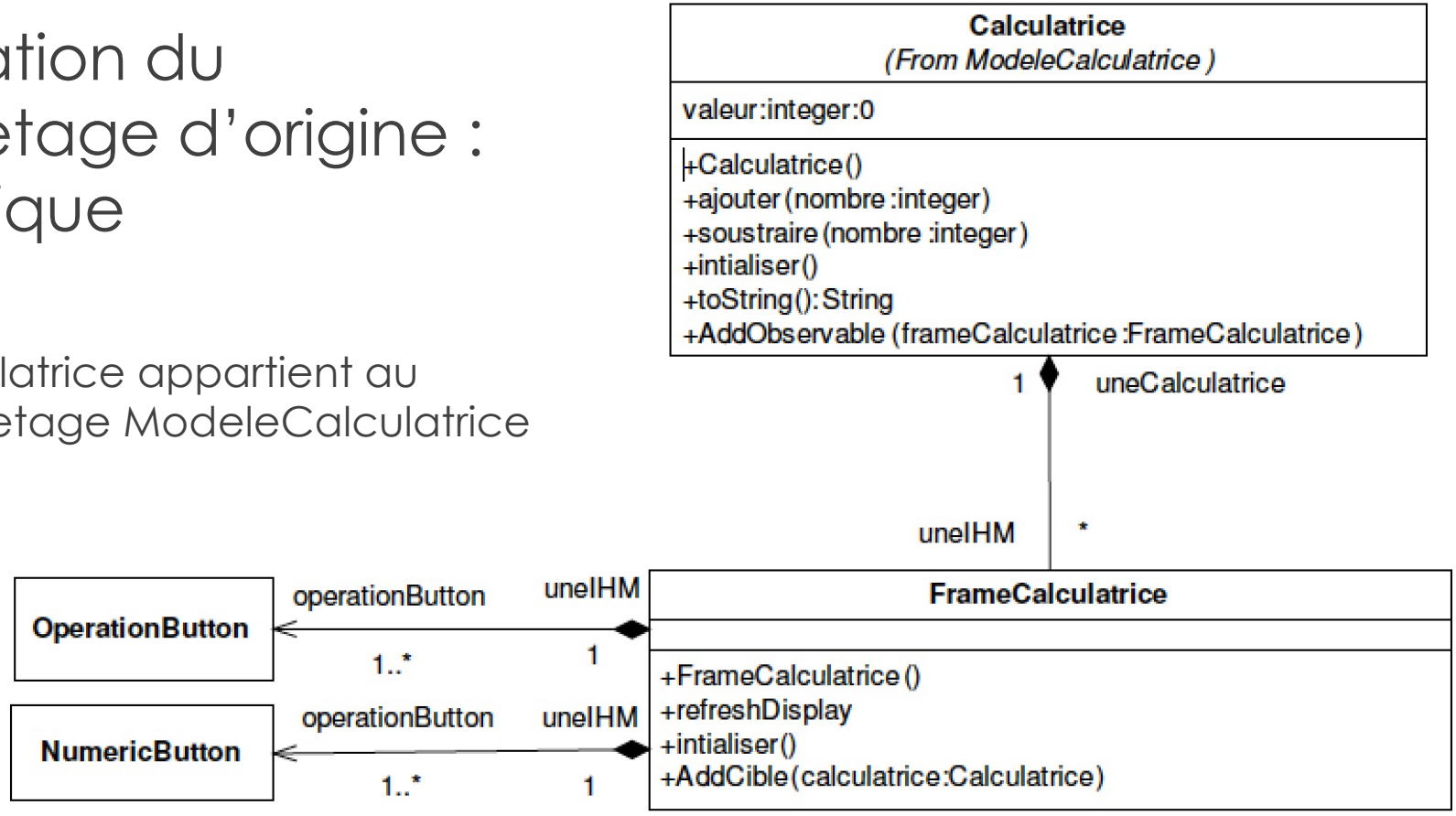
- Indication du paquetage d'origine : en italique

- FrameCalculatrice appartient au paquetage IHMCalculatrice



Exemple : paquetage IHMCalculatrice

- Indication du paquetage d'origine : en italique
  - Calculatrice appartient au paquetage ModeleCalculatrice



Diagrammes de Structure Composite ou diagrammes de composants

- Les diagrammes de composants montrent les relations entre les composants logiciels, leurs dépendances, communications, localisation, …
- Les diagrammes de composants donnent le moyen de stratifier la structure
- Les diagrammes de composants sont utilisés pour modéliser des structures à plus haut niveau, ou plus complexes, qui déclarent des interfaces précises.
- La plupart du temps, un composant fait intervenir plusieurs classes

Diagrammes de Structure Composite : Composant

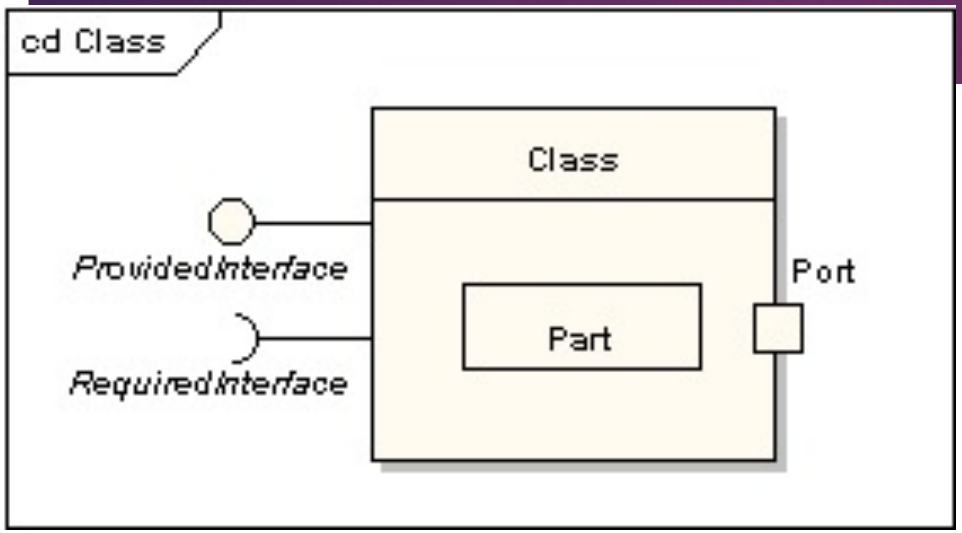
- UML1 : un composant est
  - une unité modulaire, « déployable » et interchangeable dans un système,
  - encapsulant l'implémentation et
  - n'exposant qu'un ensemble d'interfaces

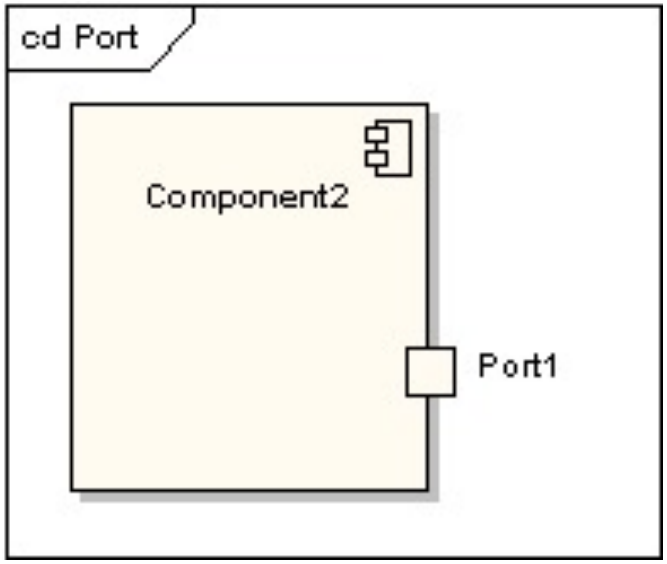
- UML 2.0 : les composants deviennent
  - des éléments structurant beaucoup plus abstraits,
  - qui représentent davantage des sous parties d'un système,
  - amenées à être modélisées selon différentes vues et
  - raffinées tout au long du cycle de développement

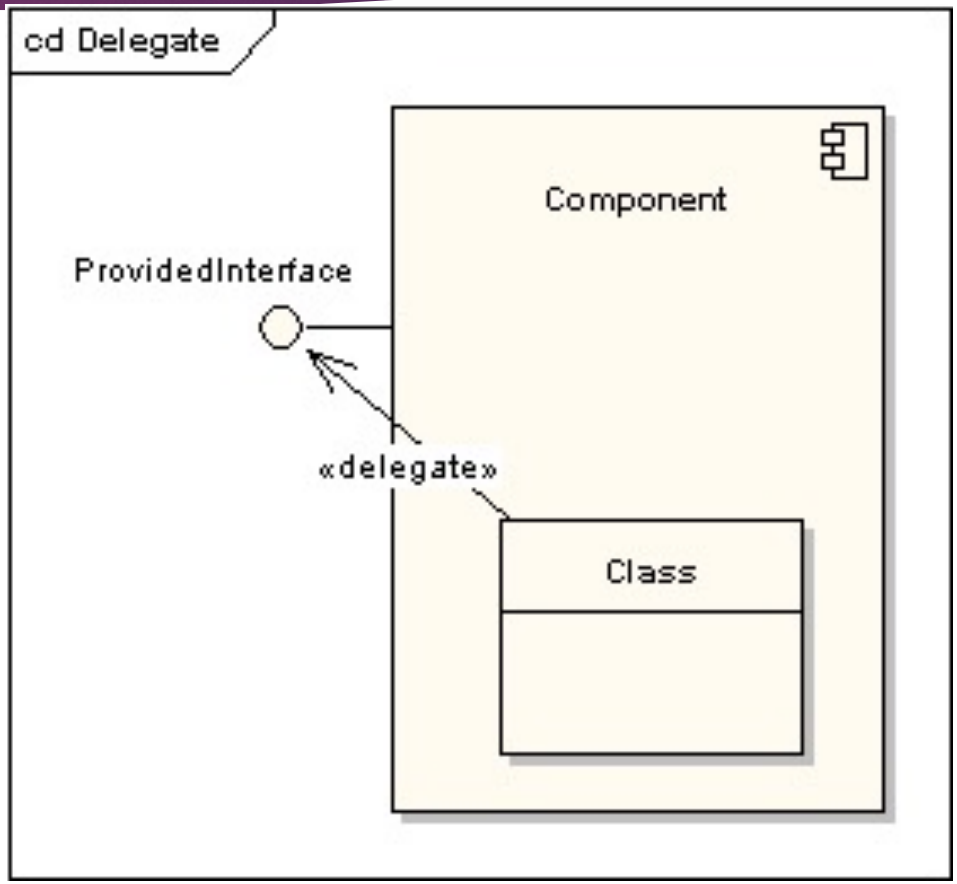
Diagrammes de Structure Composite : Composant

- Les composants sont des agrégats de haut niveau d'éléments logiciels plus petits :
  - classes, paquetages, autres composants
- Ils constituent des « boites noires » :
  - assemblées pour construire le logiciel.
- Les composants ont deux types d'interfaces, qui constituent leur partie visible :
  - Des services fournis aux autres composants
  - Des services requis des autres composants

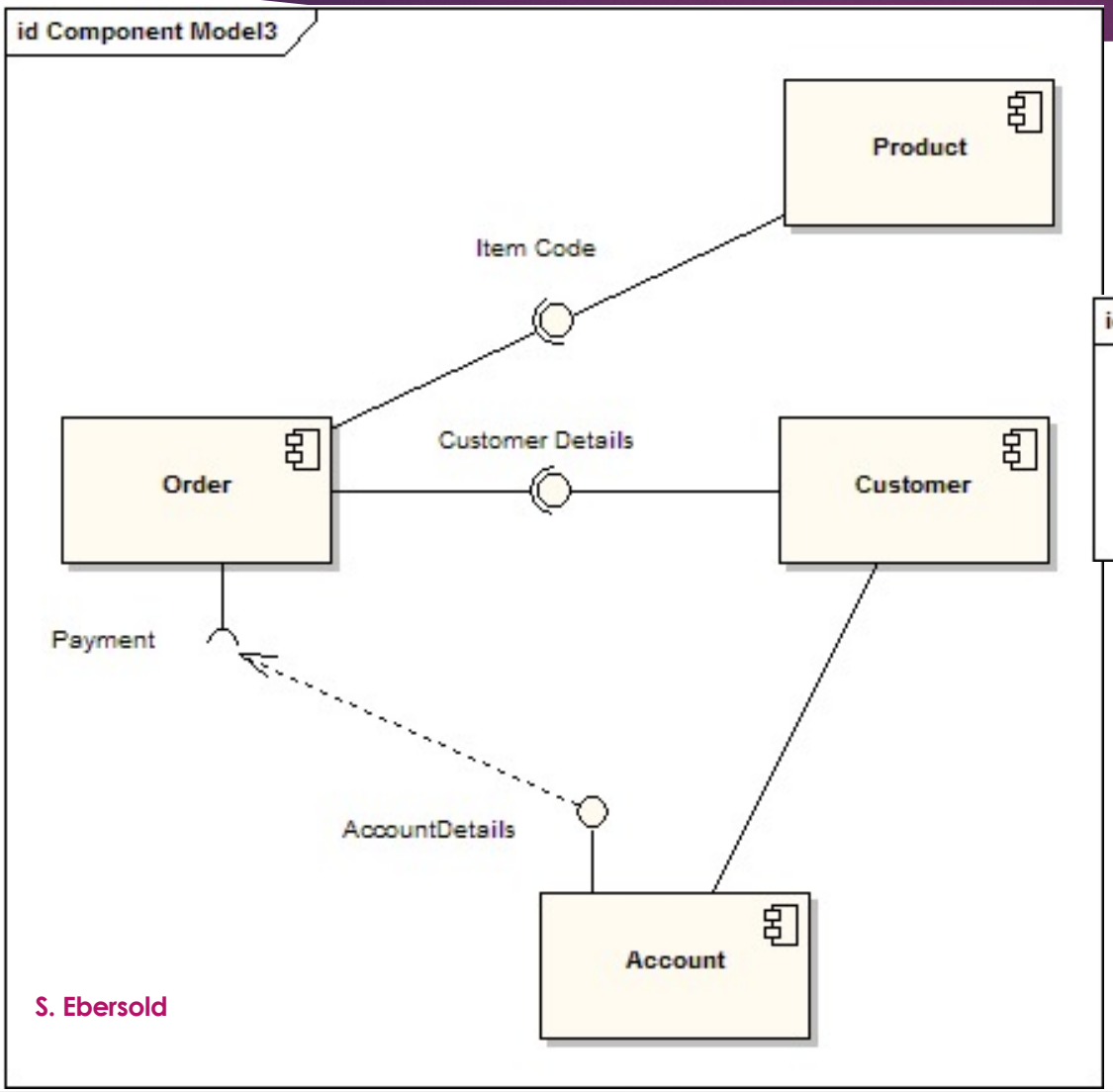
Exemples diagrammes de composants : Interfaces

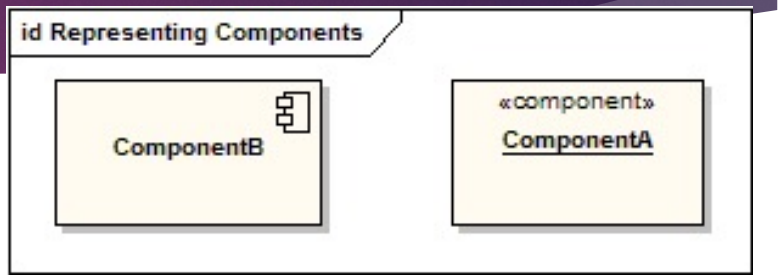


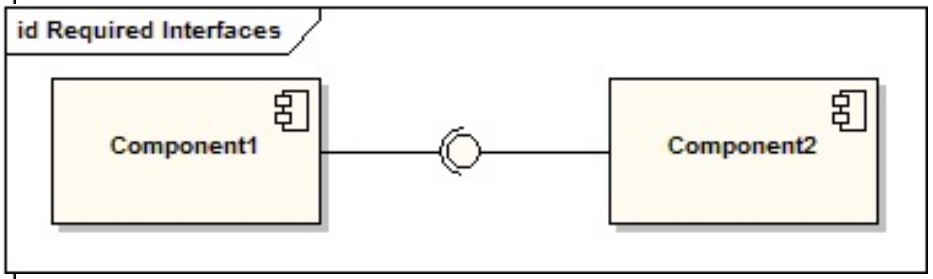


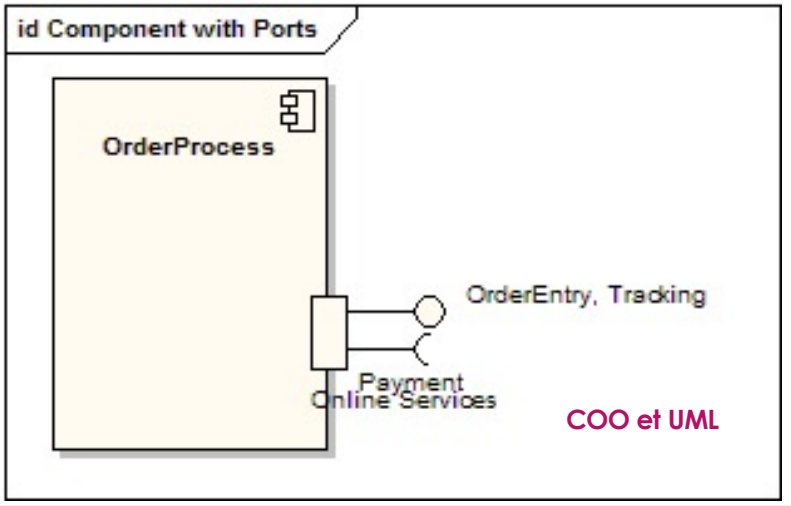


Exemples diagrammes de composants : Assemblage

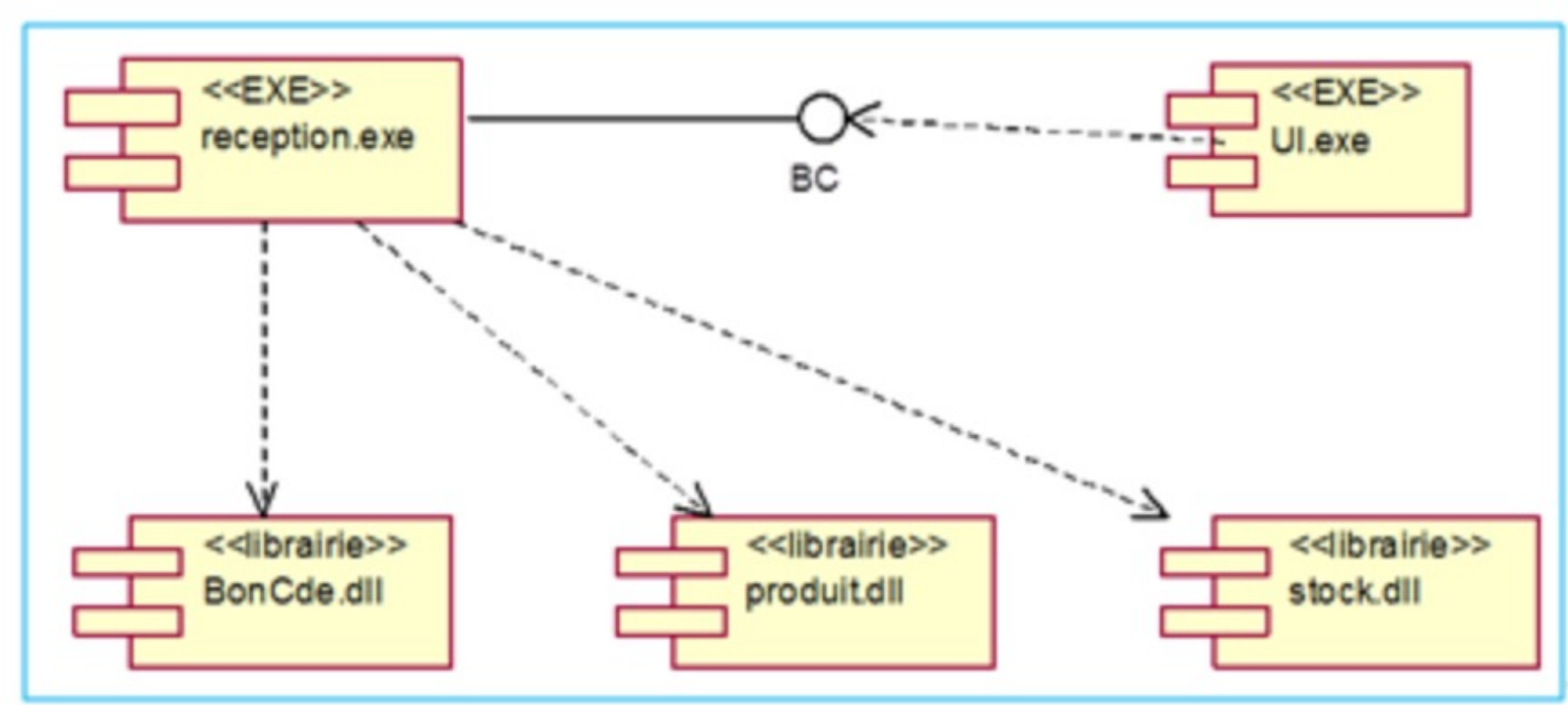








Exemples diagrammes de composants



Diagrammes de déploiement

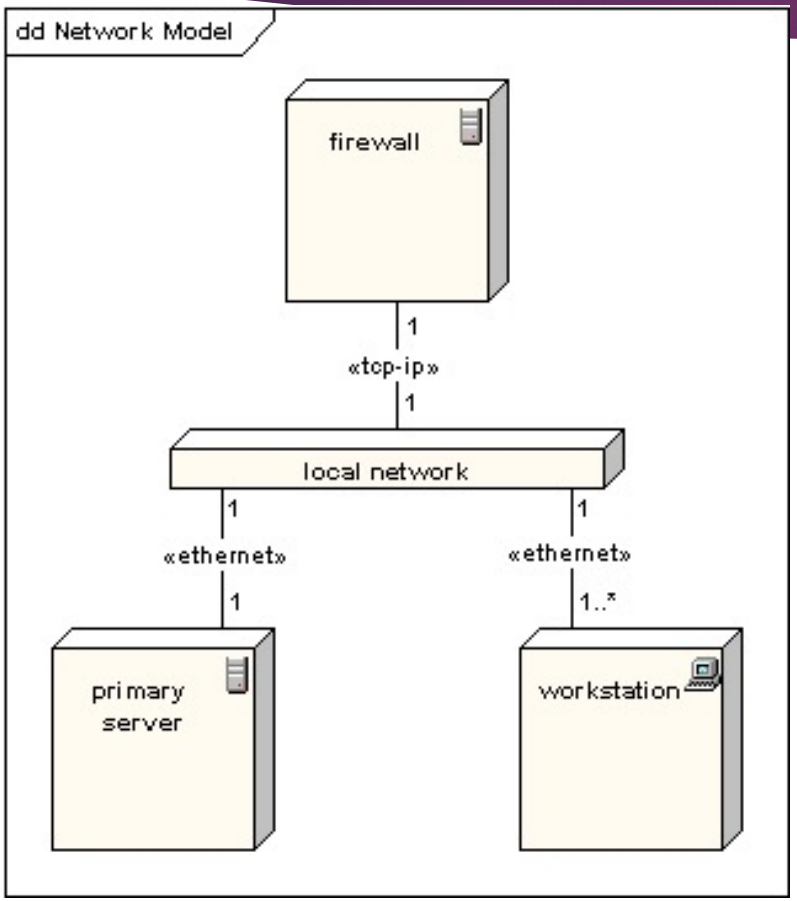
- Les diagrammes de déploiement décrivent la disposition concrète des éléments du modèle dans le monde physique

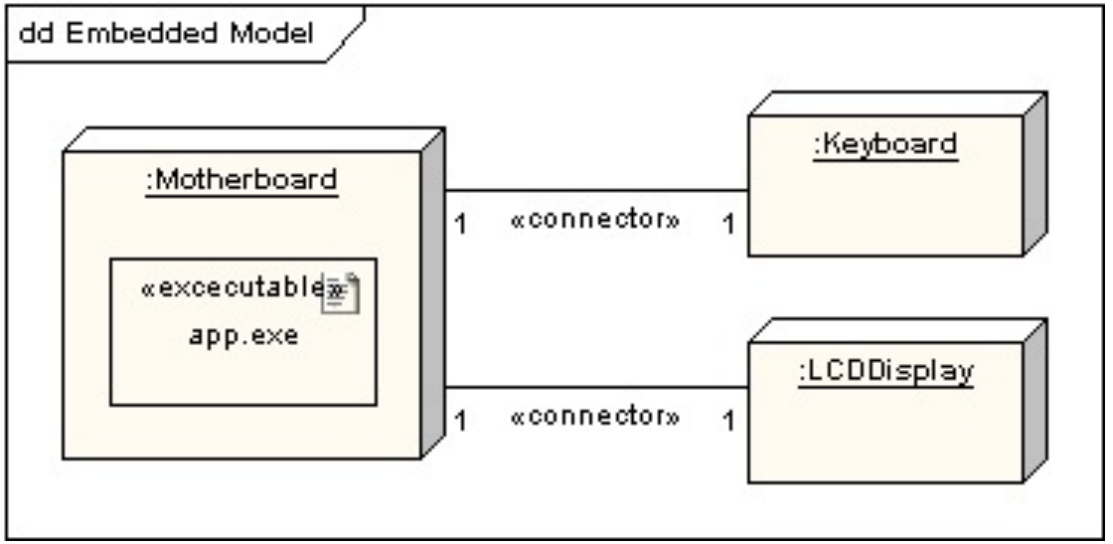
- Les diagrammes de déploiement permettent de montrer
  - la disposition physique des différent matériels – les noeuds – qui entrent dans la composition d'un système
  - la répartition des instances de composants, processus et objets qui sont alloués à ces matériels

Diagrammes de déploiement

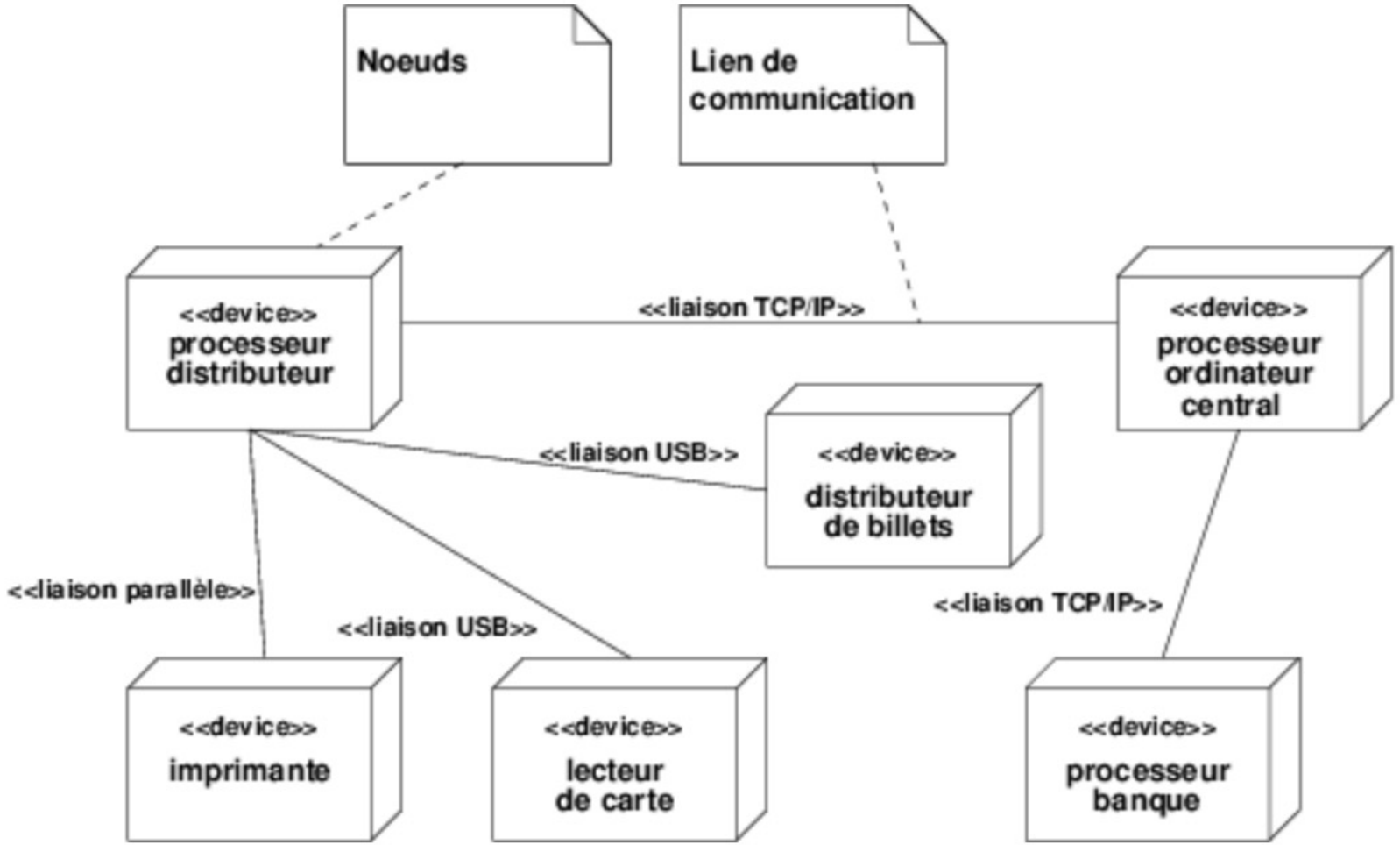
- Consiste à représenter le milieux physique dans lequel le composant est mis :
  - Précision du noeud (la machine)
    - **Chaque ressource matérielle est représentée par un nœud**
  - Précision du protocole (lien entre les noeuds)
    - Les différents nœuds sont connectés entre eux par des lignes qui symbolisent un support de communication, a priori, bidirectionnel

Exemples diagrammes de déploiement



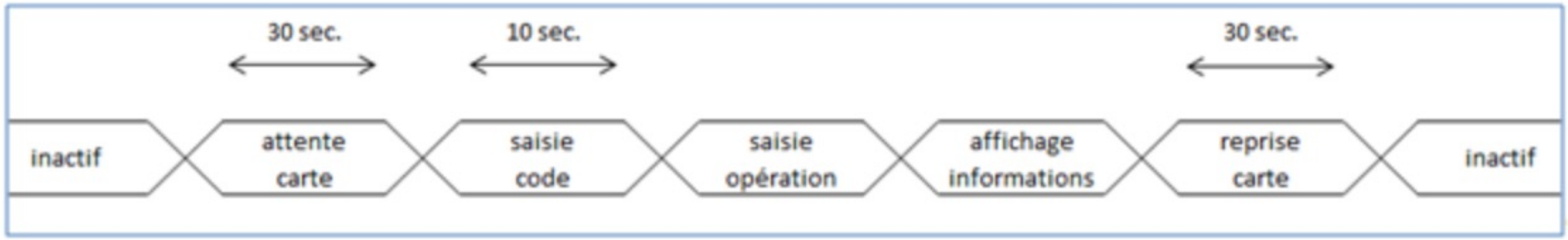


Exemples diagrammes de déploiement

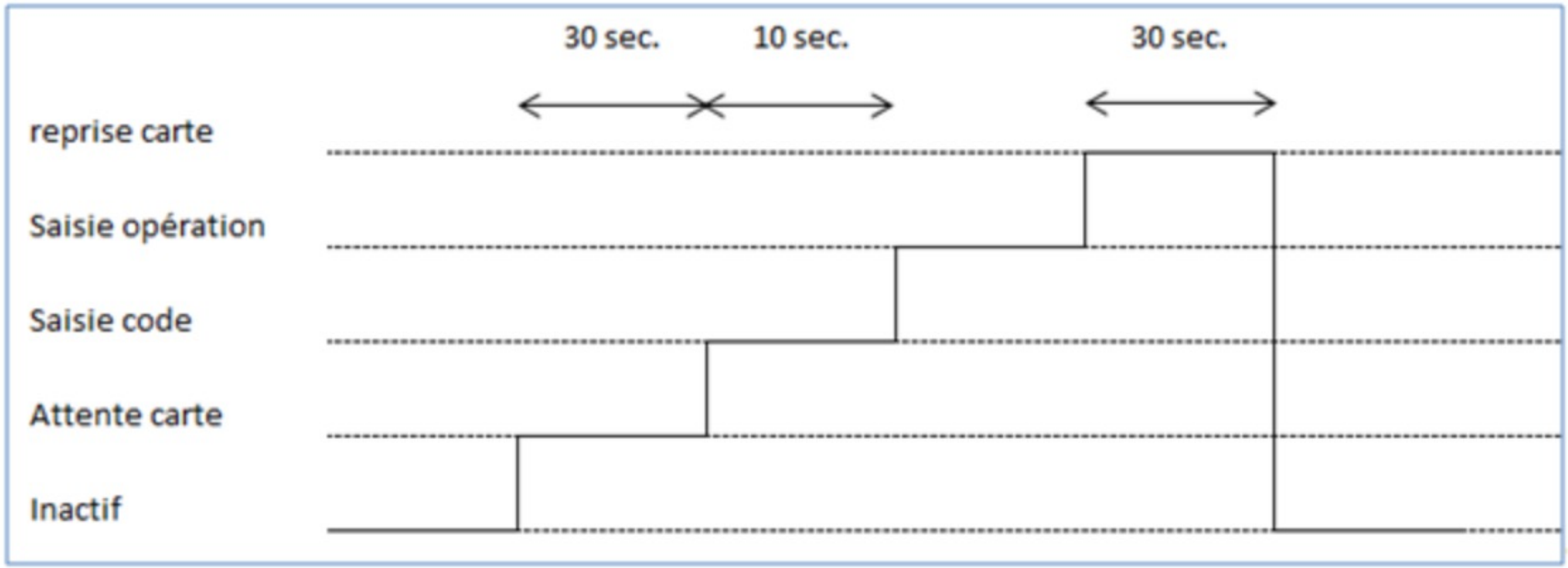


Timing Diagrams

- Ces diagrammes combinent les diagrammes de séquence et d'état pour proposer un point de vue sur l'évolution de l'état d'un objet au fil du temps, et sur les messages qui modifient cet état.

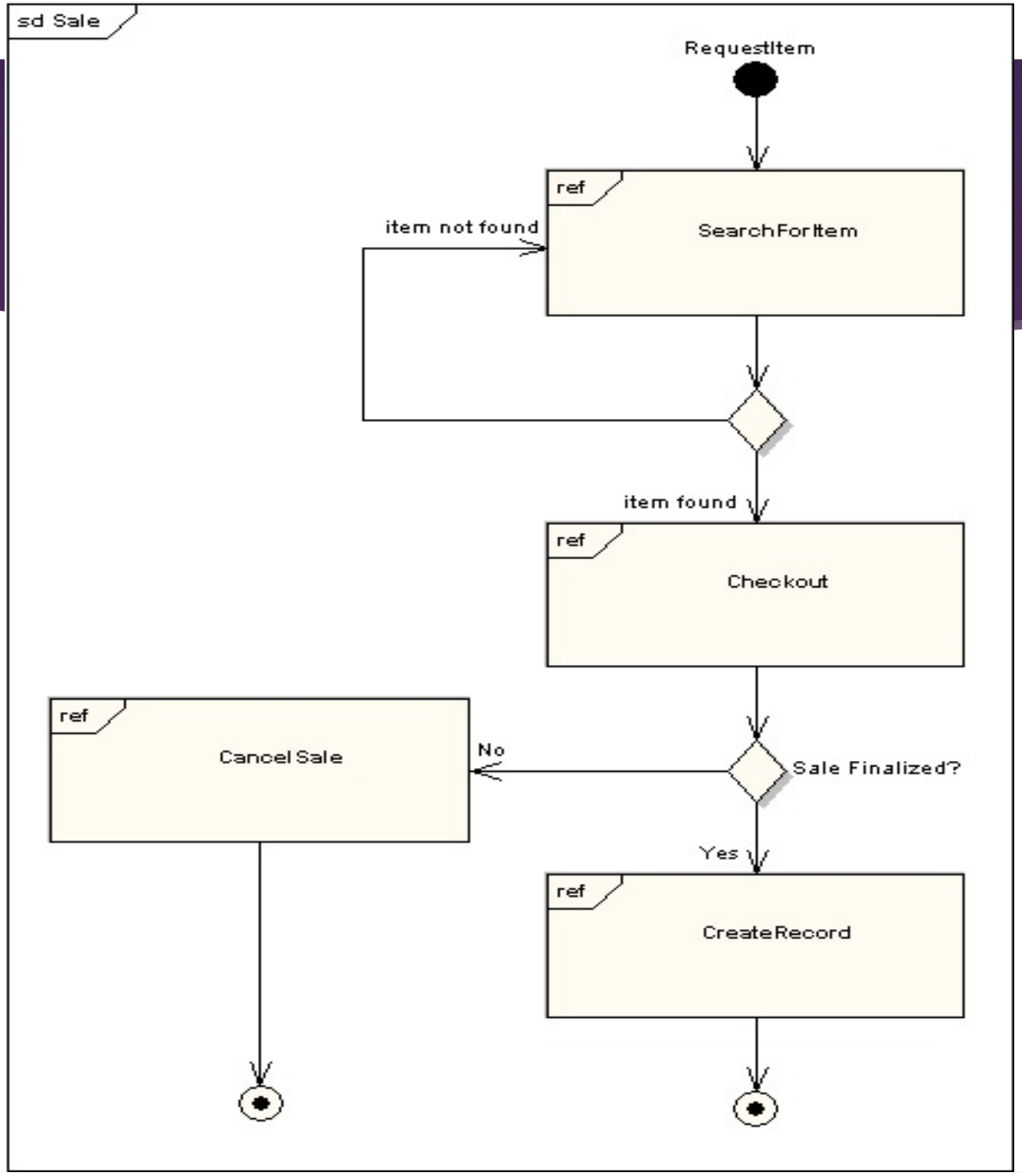


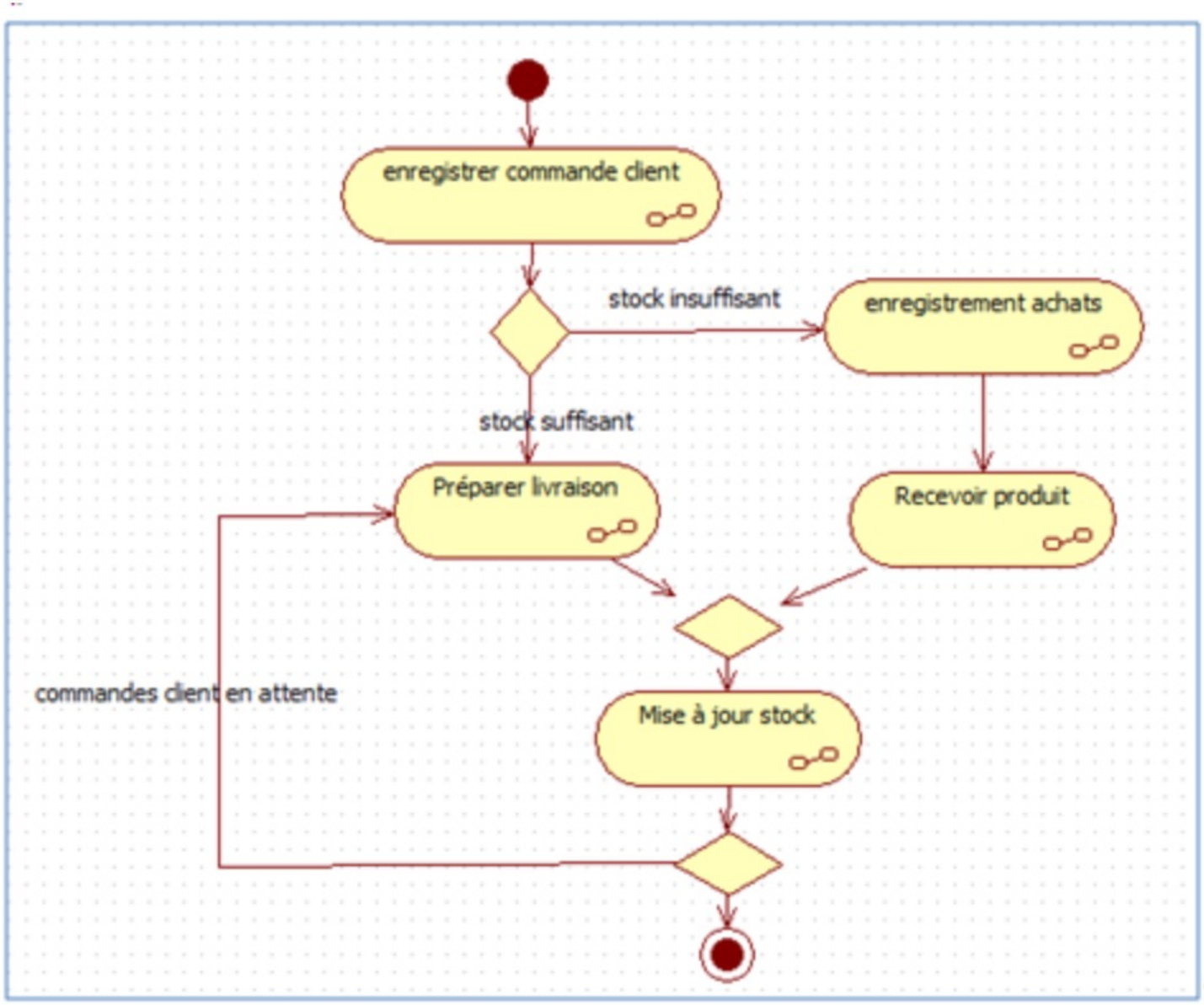
Un exemple de diagramme de temps avec un seul axe temporel



Interaction Overview Diagrams

- Ces diagrammes utilisent diagrammes d'activité et de séquence pour décrire comment des fragments d'interaction (décrits par des diagrammes de séquence) sont combinés par des points de décision et des flux

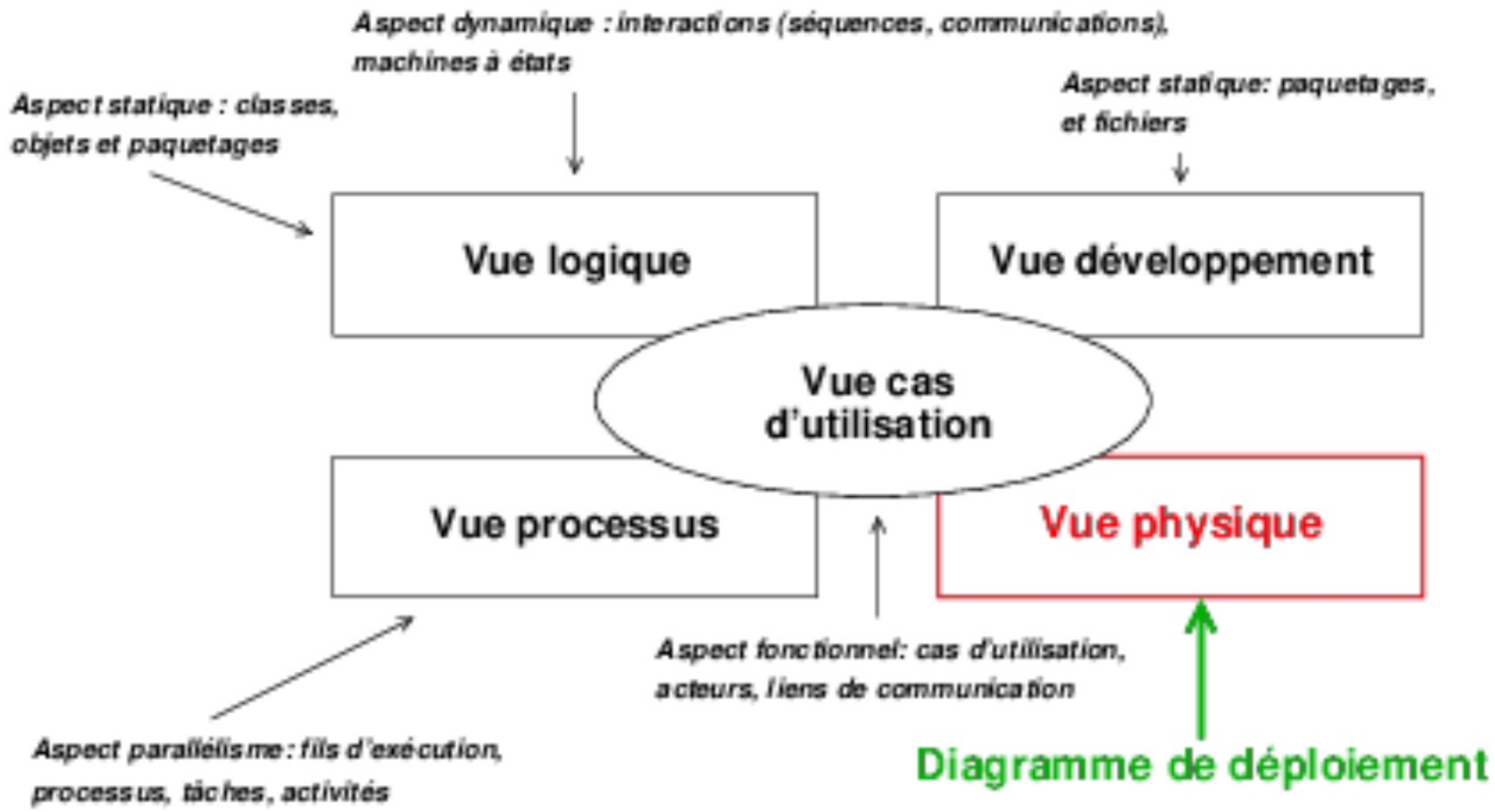

sd Sale
RequestItem
ref
SearchForItem
item not found
item found
ref
Checkout
ref
Cancel Sale
No
Sale Finalized?
Yes
ref
CreateRecord


enregistrer commande client
stock insuffisant
enregistrement achats
stock suffisant
Préparer livraison
Recevoir produit
commandes client en attente
Mise à jour stock

Conclusion : Les 4+1 Vues d'UML

- La vue logique concerne l'intégrité de conception.
- La vue de développement concerne l'intégrité de gestion.
- La vue des processus concerne l'intégrité d'exécution.
- La vue physique concerne l'intégrité de performance.
- La vue des cas d'utilisation guide et justifie les autres vues

Les 4+1 Vues d'UML



UML : Conclusion
Vues et diagrammes

- Vue des cas d'utilisation :

Fournit une vision orientée utilisation du système, permettant de le valider

- Diagramme des cas d'utilisation (acteurs et cas d'utilisation),
- diagramme d'objets (acteurs,objets, liens),
- diagramme de collaboration (acteurs, objets, liens, message),
- diagramme de séquences (acteurs, objets, message),
- diagramme d'états-transitions (états,transition)
- diagramme d'activités (activités et transitions).

La vue des besoins des utilisateurs

- Cette vue (dont le nom exact est "vue des cas d'utilisation"), guide toutes les autres :

- Dessiner le plan (l'architecture) d'un système informatique n'est pas suffisant, il faut le justifier !
- Cette vue définit les besoins des clients du système et centre la définition de l'architecture du système sur la satisfaction (la réalisation) de ces besoins.
- A l'aide de scénarii et de cas d'utilisation, cette vue conduit à la définition d'un modèle d'architecture pertinent et cohérent.
- Cette vue est la "colle" qui unifie les quatre autres vues de l'architecture.
- Elle motive les choix, permet d'identifier les interfaces critiques et force à se concentrer sur les problèmes importants.

UML : Conclusion
Vues et diagrammes

- Vue logique :

Fournit une vision des éléments du système, et de leurs relations et interactions

  - diagrammes d'objets (acteurs,objets, liens),
  - diagramme de collaboration (acteurs, classes, objet, liens),
  - diagramme de séquences (acteurs, objets, messages),
  - diagramme de classes (acteurs, classes, paquetages, relations),
  - diagramme d'états-transitions (états, transition)
  - diagramme d'activités (activités et transitions).

La vue logique

- **Vue de haut niveau**
  - **se concentre sur l'abstraction et l'encapsulation,**
  - **modélise les éléments et mécanismes principaux du système.**

- Elle identifie les éléments du domaine, ainsi que les relations et interactions entre ces éléments :
  - les éléments du domaine sont liés au(x) métier(s) de l'entreprise,
  - ils sont indispensables à la mission du système,
  - ils gagnent à être réutilisés (ils représentent un savoir-faire).

- Elle organise aussi (selon des critères purement logiques), les éléments du domaine en ***"catégories"*** :
  - pour répartir les tâches dans les équipes,
  - regrouper ce qui peut être générique,
  - isoler ce qui est propre à une version donnée, etc...

UML : Conclusion
Vues et diagrammes

- Vue des processus :

Fournit une vision de la dynamique du système, intervenant dans la mise en œuvre d'une certaine traçabilité

- diagramme de collaboration (classes, objet, liens),
- diagramme de séquences (objets, message),
- diagramme d'états-transitions (états, transitions)
- diagramme d'activités (activités et transitions)
- diagramme de composants (composants).

La vue des processus

- Cette vue est très importante dans les environnements multitâches ; elle montre :
  - La décomposition du système en terme de processus (tâches).
  - Les interactions entre les processus (leur communication).
  - La synchronisation et la communication des activités parallèles (threads).

UML : Conclusion
Vues et diagrammes

- Vue de développement (ou vue des composants):

Fournit une vision de la réalisation du système en termes d'architecture.

  - diagramme de paquetages
  - diagramme de composants (composants).

La vue de développement

- **Vue de bas niveau** (aussi appelée "vue de réalisation"), montre :
  - L'allocation des éléments de modélisation dans des modules (fichiers sources, bibliothèques dynamiques, bases de données, exécutables, etc...).
    - Identifie les modules qui réalisent (physiquement) les classes de la vue logique.
  - L'organisation des composants : la distribution du code en gestion de configuration, les dépendances entre les composants...Les contraintes de développement (bibliothèques externes...).
  - L'organisation des modules en "sous-systèmes", les interfaces des sous-systèmes et leurs dépendances (avec d'autres sous-systèmes ou modules).

UML : Conclusion
Vues et diagrammes

- Vue physique :

Montre le placement des logiciels sur les machines pour l'exécution du système.

  - diagramme de déploiement

La vue physique

- Cette vue très importante dans les environnements distribués :
  - décrit les ressources matérielles et la répartition du logiciel dans ces ressources :
    - La disposition et nature physique des matériels, ainsi que leurs performances.
    - L'implantation des modules principaux sur les noeuds du réseau.
    - Les exigences en terme de performances (temps de réponse, tolérance aux fautes et pannes...).