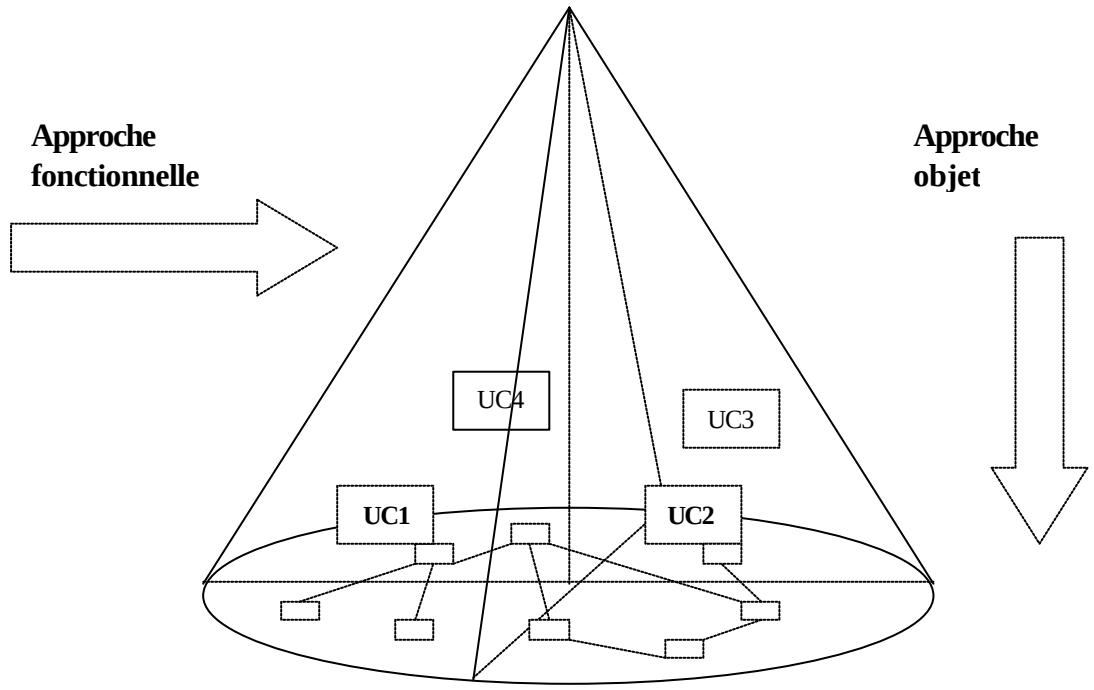
Modèles d'usage

Use cases
Le client présente son système d'un point de vue fonctionnel.

Les *Use Cases* vont permettrent de reformuler les besoins et constituent un moyen de communication très efficace entre utilisateurs et équipes de développement.

Un Use Case décrit le système du point de vue de son utilisation, c'est-à-dire :
? Les interactions entre le système et les acteurs
? Les réactions du système aux événements externes
? Il permet de développer un système "orienté utilisateur".



A partir du cahier des charges, la première étape de conceptualisation consiste à:
✍ identifier les acteurs
✍ identifier les événements
✍ identifier les Use Cases.

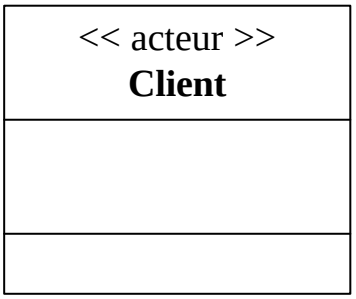
Acteurs
Un acteur est une entité externe agissant sur le système.
Il peut s'agir :
✍ d'un utilisateur humain
✍ d'une machine
✍ d'un autre système ou sous-système

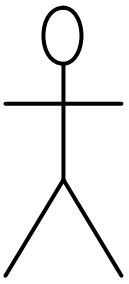
L'acteur peut consulter et/ou modifier l'état du système.

Le système répond à l'acteur en lui fournissant des informations, et le prévient éventuellement des changements d'état.

L'acteur doit être défini à travers le **rôle** qu'il joue par rapport au système.
✍ Différentes entités peuvent utiliser le système de la même façon (un rôle unique).
✍ Une même entité peut utiliser le système de diverses façons (plusieurs rôles).

Notation :





Client

Typologie des acteurs

Du point de vue du système, il existe deux types d'acteurs :

✍ les acteurs primaires, qui utilisent le système

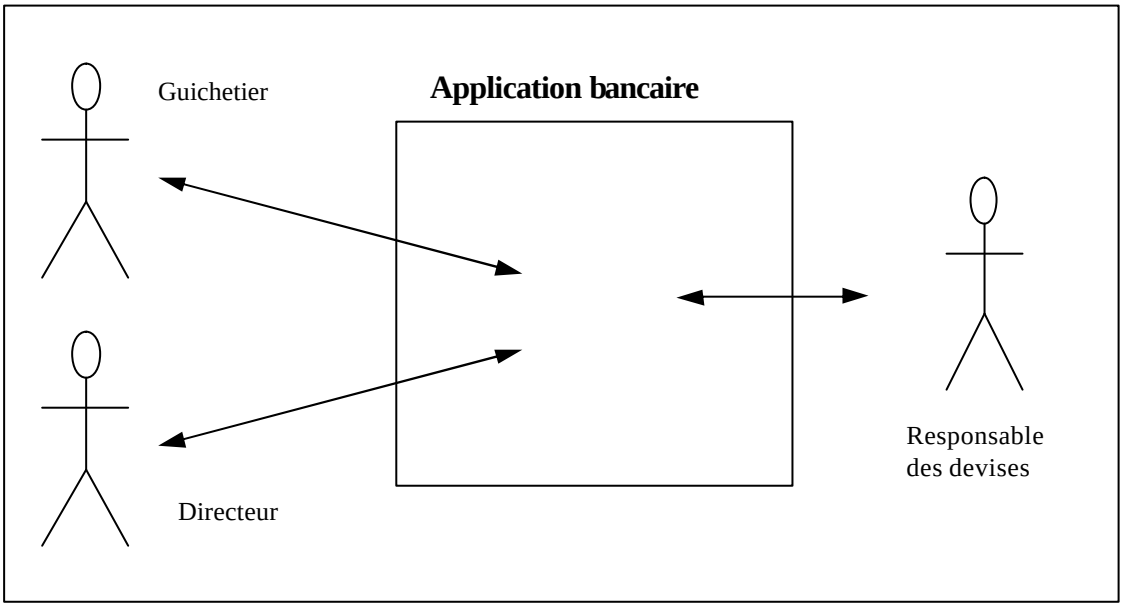
✍ les acteurs secondaires, qui administrent le système.

Par exemple, pour une application bancaire, on peut imaginer les guichetiers qui enregistrent les opérations courantes et le directeur de l'agence qui établit le bilan de l'agence.

Pour effectuer un change de devise, il faut connaître le cours de la devise.

Il faut donc qu'un acteur secondaire soit capable de fournir ces informations.

Cet acteur peut être représenté par une personne chargée de la saisie des cours journaliers ou par une application du système central qui fournit les informations nécessaires.

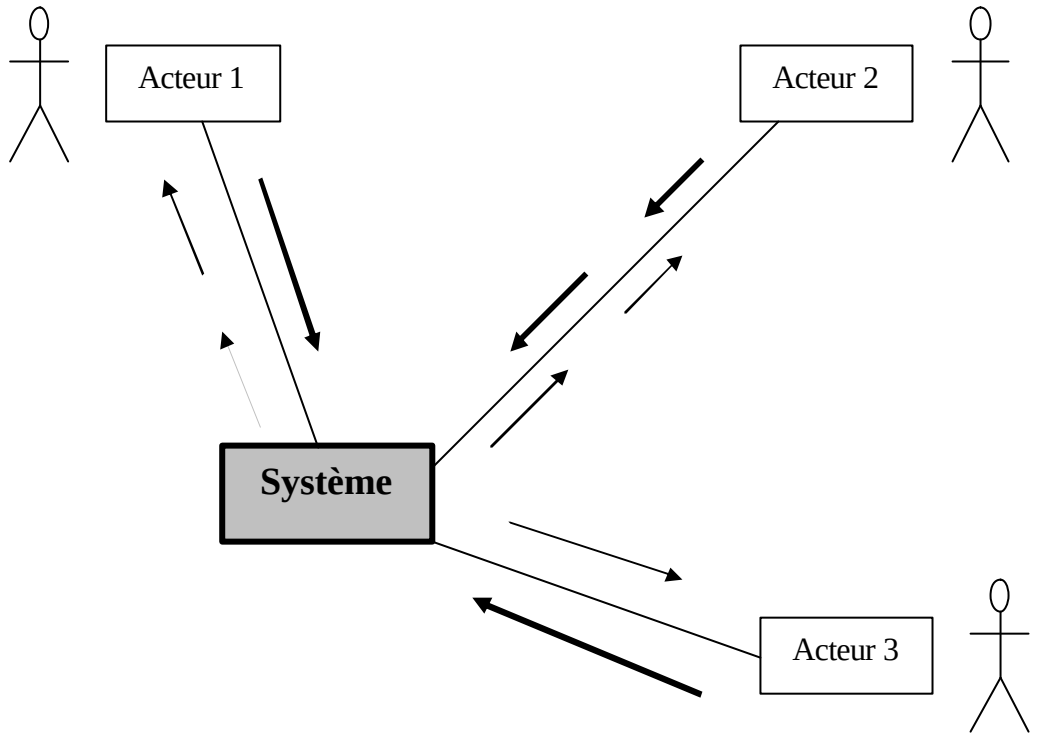


Selon la taille de l'agence, un individu (salarié d'une banque) peut jouer plusieurs rôles (guichetier, responsable de devises) et plusieurs individus peuvent jouer le même rôle.

En conclusion, un acteur correspond à un rôle vis-à-vis du système.

Evénement – diagramme de contexte
Pour identifier précisément les événements, il faut en premier lieu déterminer la frontière du système.

Le diagramme de contexte montre les échanges entre le système et les acteurs.



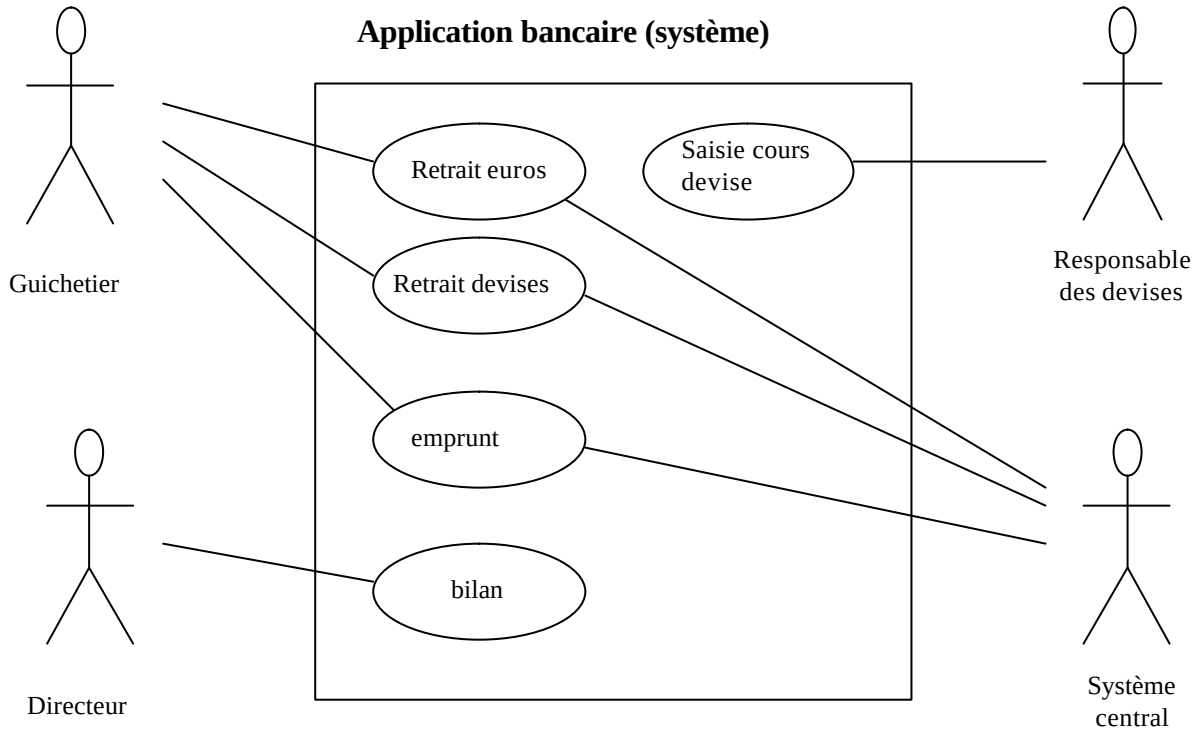
Les événements externes (⟵) sont envoyés des acteurs vers le système.
Ils vont permettre d'identifier les use cases.

Le diagramme de contexte n'existe pas dans UML bien qu'il soit très utile.
Il emprunte le même formalisme qu'un diagramme de collaboration.

Le diagramme de Use Cases

L'exécution du Use case est contrôlée par des événements externes envoyés au système par les acteurs.

L'ensemble des Use Cases décrit les exigences fonctionnelles du système.



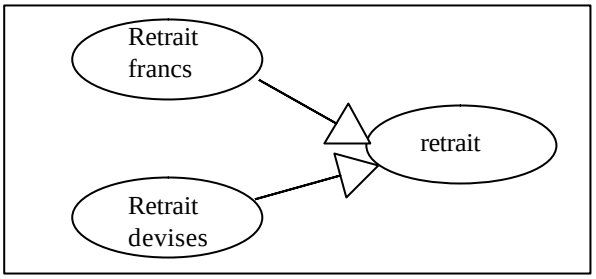
Cette représentation permet de voir de façon simple :

- ✍ les différents acteurs
- ✍ comment est délimité le système
- ✍ les fonctionnalités demandées au système
- ✍ les rôles des différents acteurs vis-à-vis du système.

Organiser les uses cases au sein d'un système

La généralisation

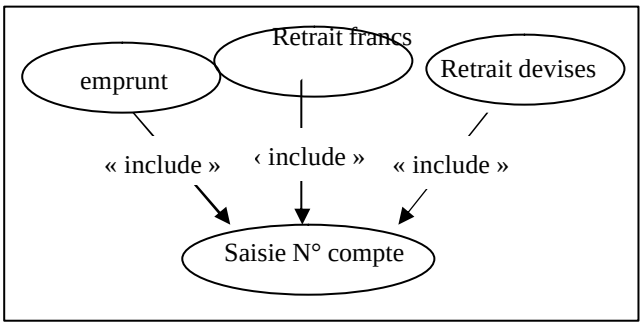
Considérons les use cases « retrait francs » et retrait devises ».
Nous pouvons imaginer un use case « retrait » qui décrit les fonctions communes dont héritent les use cases « retrait francs » et « retrait devises ».



Elle indique que le use case fils : « retrait devises » hérite de toutes les caractéristiques du use case père : « retrait ».
Le use case « retrait devises » est une spécialisation du use case « retrait ».

La relation « include »

Elle indique que le use case qui est pointé par la flèche est une sous-partie de l'autre.

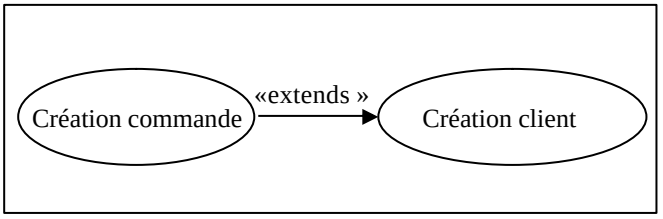


La relation « include » permet de :

- ✍ factoriser des use cases correspondant à des fonctionnalités importantes qui servent fréquemment
- ✍ expliciter la constitution d'un use case complexe en le décomposant en plusieurs use cases.

La relation « extends »

Permet de faire l'insertion optionnelle d'un comportement dans un use case étranger. Est souvent utilisé pour décrire une alternative dans un scénario.



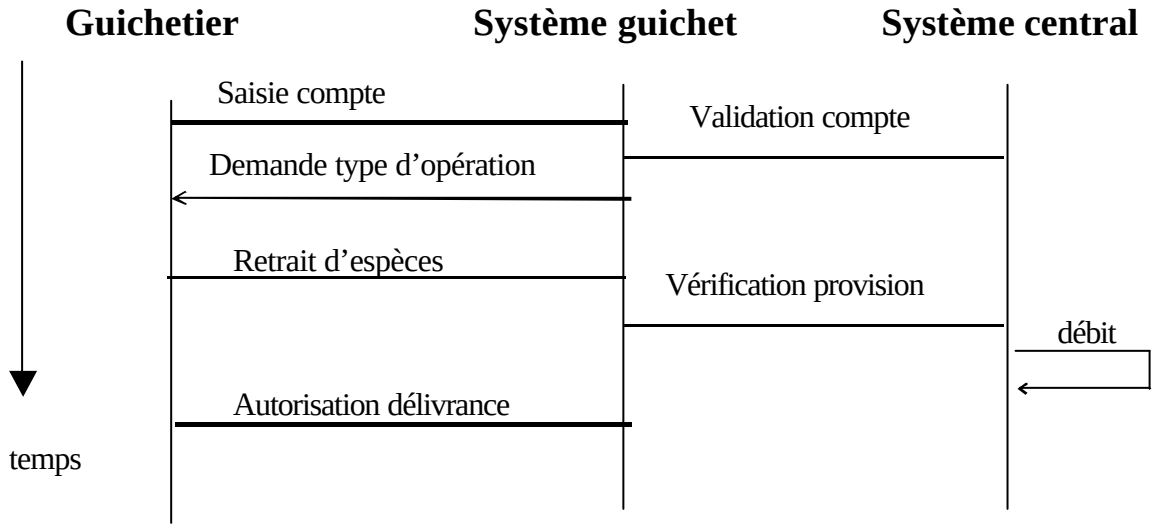
Description d'un Use Case
Un use case est une séquence d'actions réalisées par le système produisant un résultat observable à un acteur particulier.
Sa description doit être synthétique et facilement compréhensible.

Un use case peut être décrit de différentes façons :

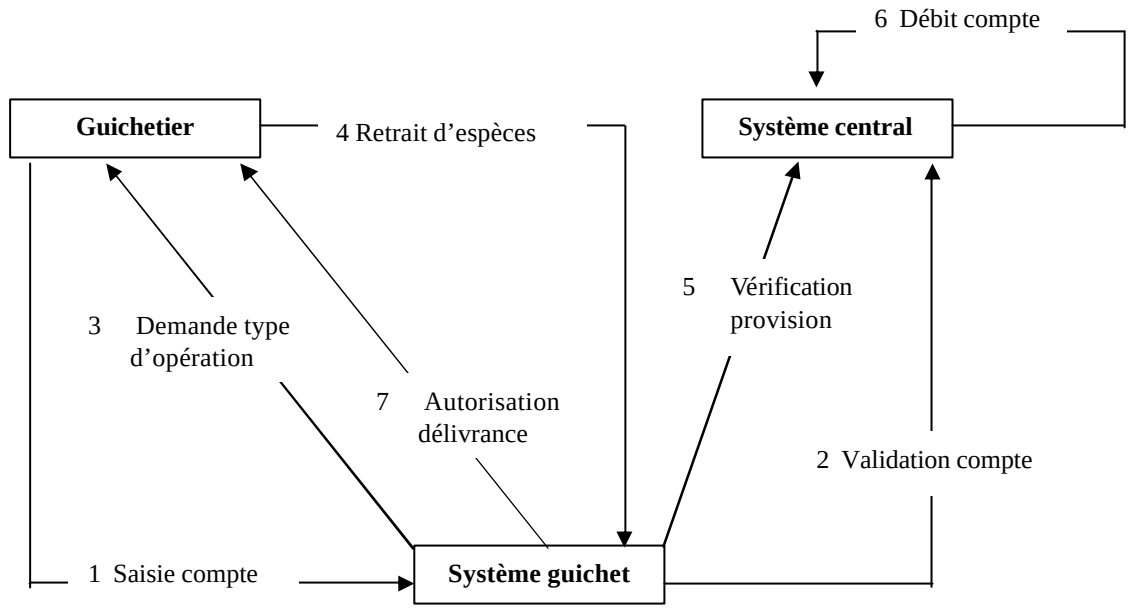
Une description textuelle

Use case « retrait »
Le guichetier saisit le numéro de compte du client L'application valide le compte auprès du système central L'application demande le type d'opération au guichetier Le guichetier sélectionne un retrait d'espèces Le système « guichet » interroge le système central pour s'assurer que le compte est suffisamment approvisionné Le système central effectue le débit du compte Le système notifie au guichetier qu'il peut délivrer le montant demandé.

Un diagramme de séquence



Un diagramme de collaboration



Là, les interactions sont représentées par des flèches, mais la chronologie par des numéros. Cette notation devient vite difficile à lire pour les use cases qui nécessitent beaucoup d'interactions. Par contre, elle met en évidence les acteurs dont le rôle est important.

Les use cases entre la machine et les acteurs humains utilisent un interface graphique. Ceux-ci peuvent être décrit assez finement par la succession d'écrans annotés qui indiquent les options et les informations que doit renseigner l'utilisateur pour un cas d'utilisation donné.

Exercice

La médiathèque – use cases

Voir énoncé.

1. **Déterminer les acteurs**
2. **Déterminer les use cases**
3. **Elaborer un diagramme de use cases en essayant de trouver un exemple de relation « uses » et « extends ».**

Modèles dynamiques

- ✍ Permettent de comprendre et de décrire le comportement des objets et leurs interactions.
- ✍ Servent à définir ou à préciser les opérations.

Deux types de représentations :

dynamique entre objets avec les deux diagrammes d'interaction :

diagramme de séquence

diagramme de collaboration

dynamique interne à un objet avec le diagramme d'états-transitions.

Diagramme de séquence

Met en avant l'aspect *temporel* des interactions. Le temps s'incrémente du haut vers le bas de la figure, mais les espaces ne sont pas significatifs; seules les séquences d'événement sont représentées, non le temps qui les sépare.
Chaque objet concerné est représenté par une ligne verticale.

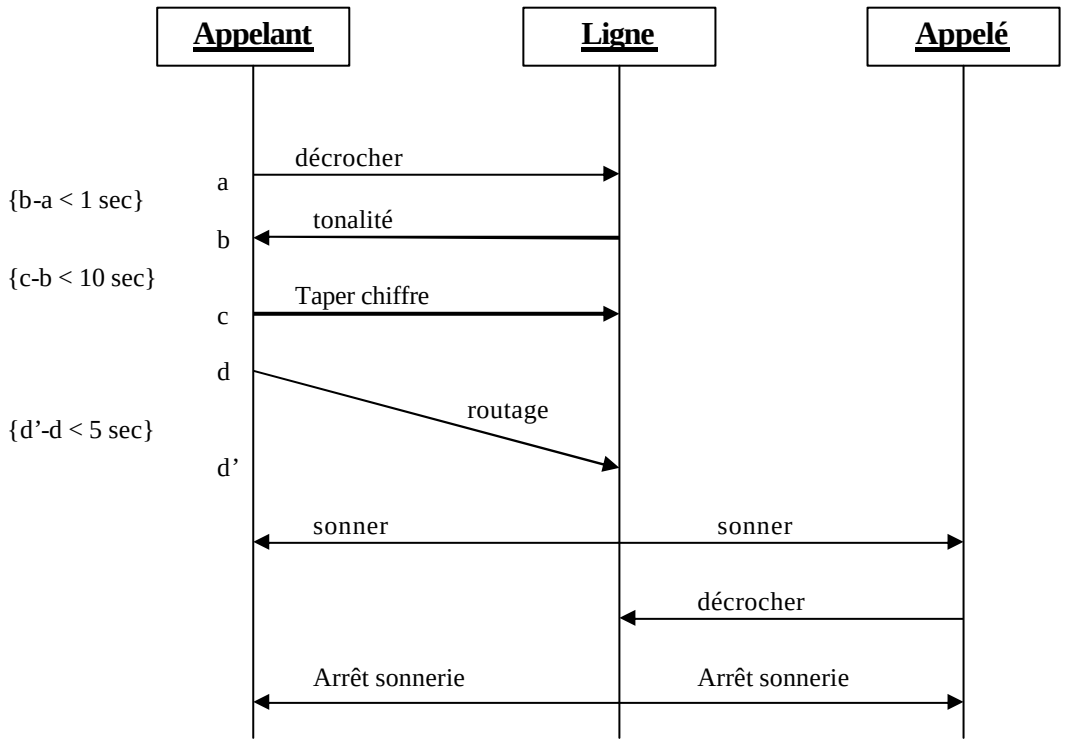
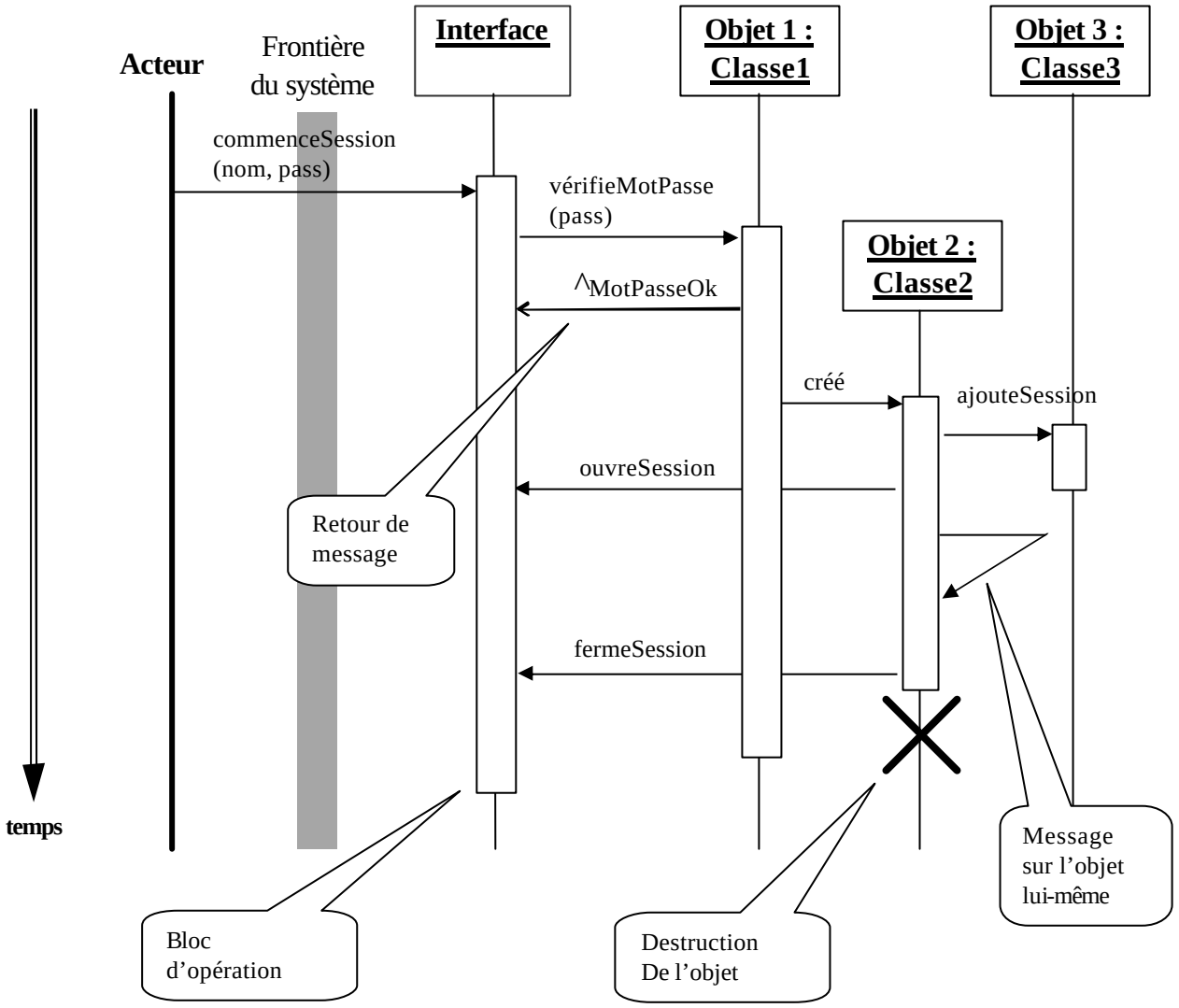


Diagramme de séquence – notations complémentaires:

Les stimulus inter-processus, ceux qui franchissent la frontière du système, sont plutôt appelés des **signaux**.
Les autres stimulus intra-processus sont appelés les **messages**.



Les **retours de messages** (^) ne sont indiqués que s'ils augmentent la compréhension du modèle.
Dans UML version 1 la tête de flèche est différente (⟵).

Les rectangles sur les barres verticales sont des **blocs d'opérations** qui montrent les périodes d'activité des objets.

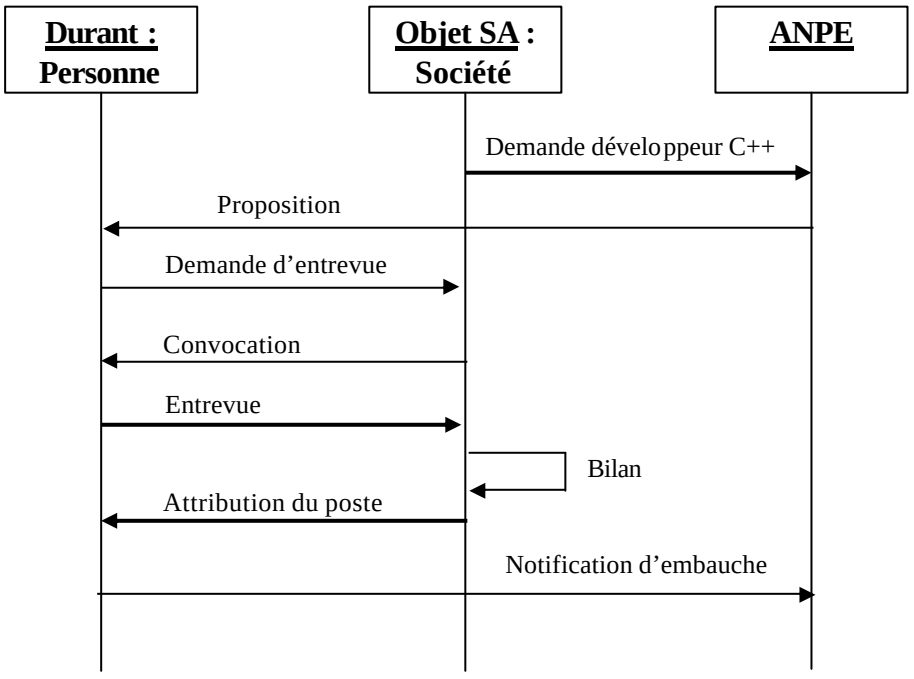
Utilisation des diagrammes de séquence - les scénarios - démarche
Dans UML, l'élaboration de la liste des scénarios est une activité importante de l'analyse.

On peut proposer un début de démarche :

1. définir et décrire les cas d'utilisation : forme textuelle
2. pour chaque cas d'utilisation, choisir les scénarios
3. construire les diagrammes d'interaction pour chaque scénario.

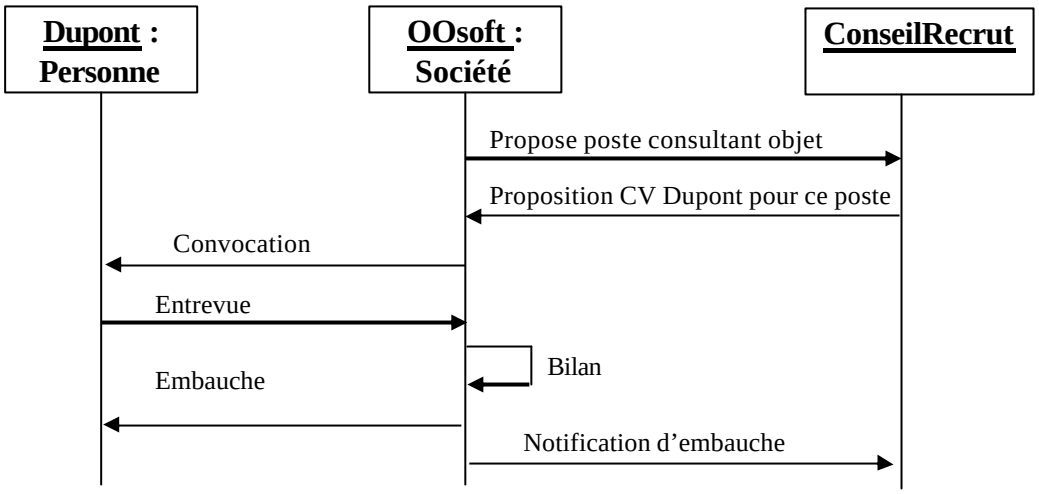
Un *scénario* est une série d'événements ordonnés dans le temps, simulant une exécution particulière du système.

Exemples de scénarios
Ce premier scénario décrit le recrutement d'un développeur C++ par la société Objet SA qui s'adresse à l'ANPE.

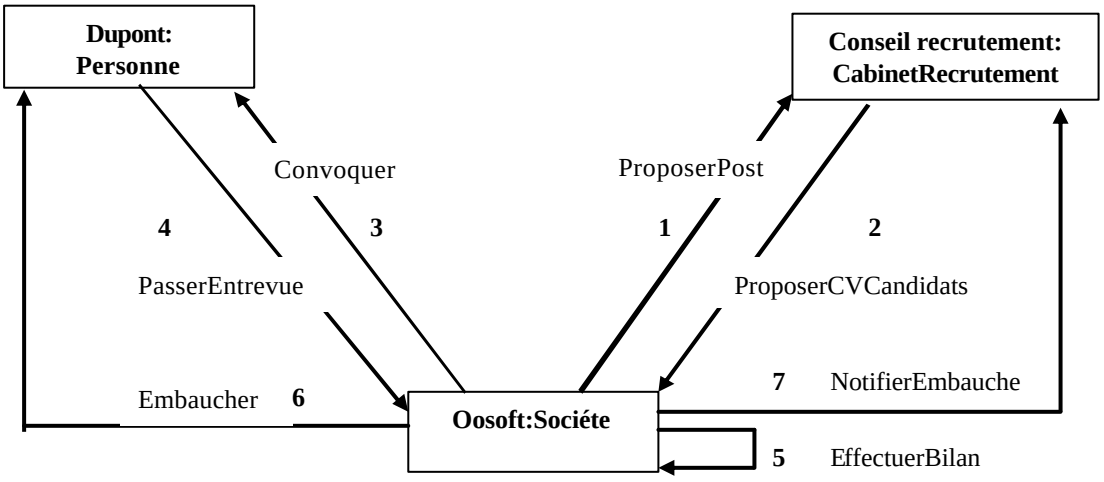


Remarque : A priori on construit le *scénario de base* sans tenir compte des exceptions (erreurs, absence de réponse quand le temps prévu est dépassé).

Ce deuxième scénario décrit le recrutement d'un consultant objet par la société OOsoft qui s'adresse au cabinet « Conseil Recrutement ».



Ce scénario peut aussi se représenter sous la forme d'un diagramme de collaboration.



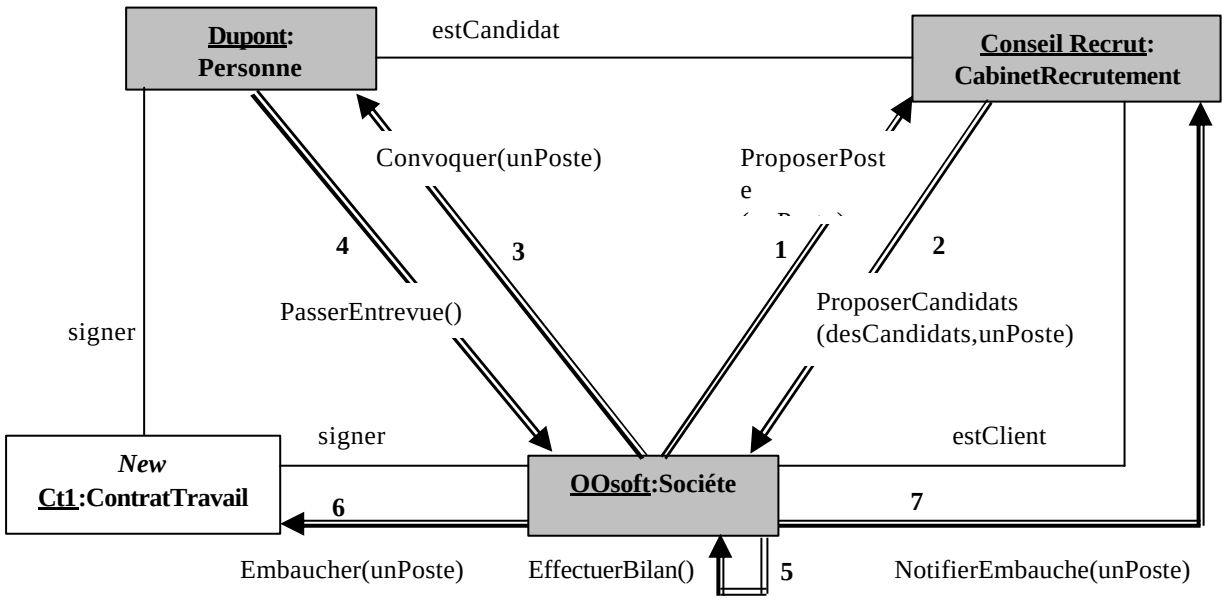
Les messages qui se déroulent de façon séquentielle sont numérotés.

Diagramme de collaboration
Il représente du point de vue statique et dynamique les objets impliqués dans la mise en place d'une fonction applicative.

Sa granularité est plus ou moins fine, selon qu'il décrit un ensemble d'opérations utilisées dans l'exécution d'une fonction ou une opération plus simple.

Deux notions fondamentales sont utilisées dans un diagramme de collaboration.

- **Le contexte** : c'est une vue statique partielle des objets qui collaborent pour réaliser une fonction.. Il correspond donc à une partie du modèle objet.
- **Les interactions** : ce sont des séquences de messages échangés par les objets dans le cadre de la réalisation d'une opération.



Ce diagramme est souvent « chargé » en notations et parfois moins lisible que les scénarios.
Pourtant, s'il n'est guère utile durant la phase d'analyse, il le devient particulièrement lors de la phase de conception. car il permet de faire le lien entre le modèle objet et le modèle dynamique.

Exercice

Paiement au restaurant

Elaborer un diagramme de séquence décrivant le paiement d'un repas au restaurant par carte de crédit.

Les objets considérés seront le client, le serveur et un terminal portable.

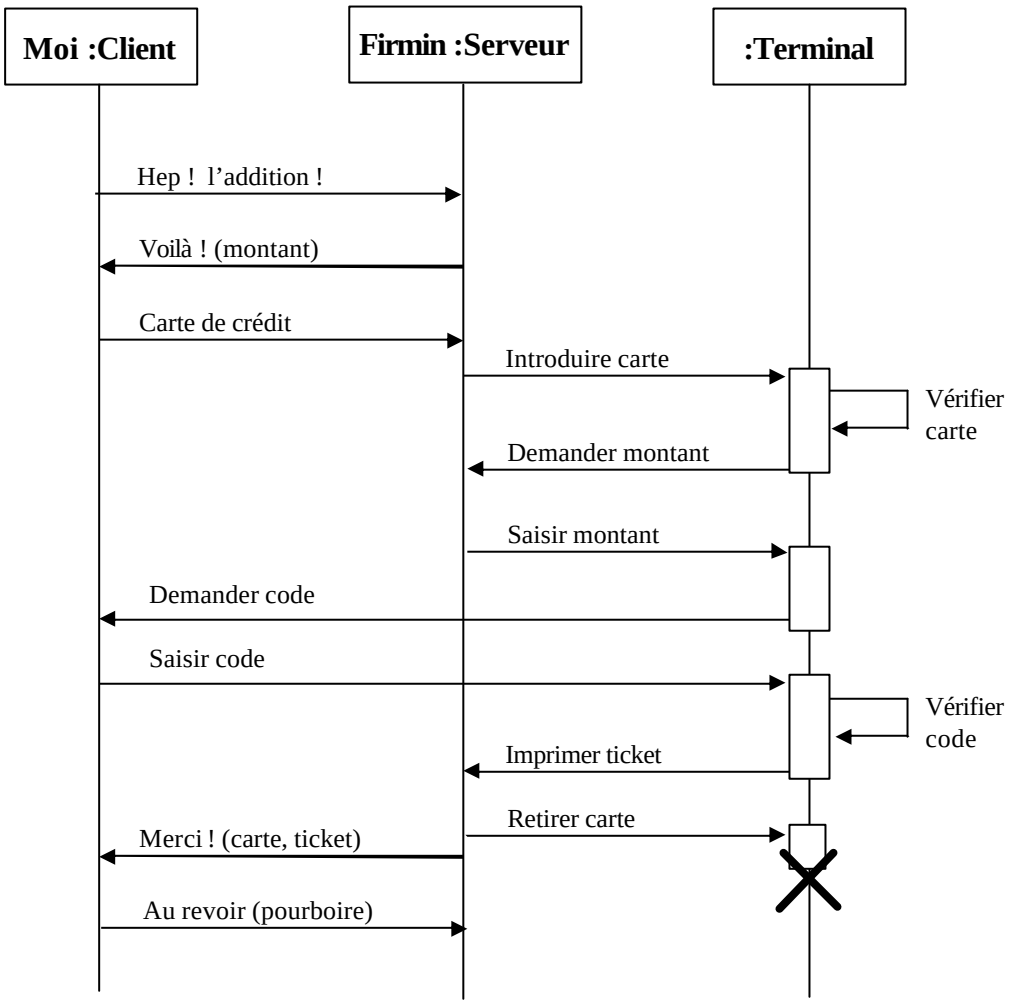


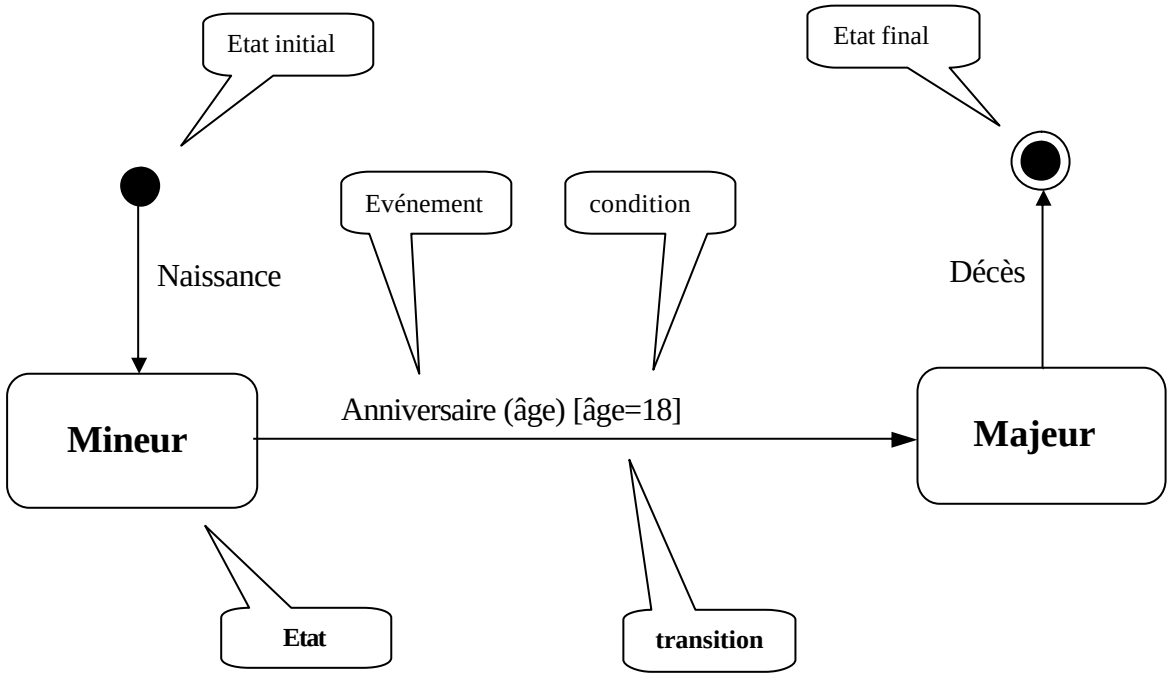
Diagramme d'Etat transition (ou STD: State Transition Diagram)
Source : les *Statecharts* de David HAREL.

Diagramme d'état:
un diagramme d'état décrit l'évolution au cours du temps d'un objet (instance d'une classe) en réponse aux interactions avec d'autres objets ou acteurs.

Il décrit le cycle de vie des objets d'une seule classe .
On établit un diagramme d'état pour chaque classe qui a un comportement dynamique significatif, ce qui signifie que toutes les classes n'en ont pas besoin.

Un diagramme d'état est un graphe dont les nœuds sont des **états** et les arcs orientés des **transitions** portant des paramètres et des noms d'événements.

Notation de base :

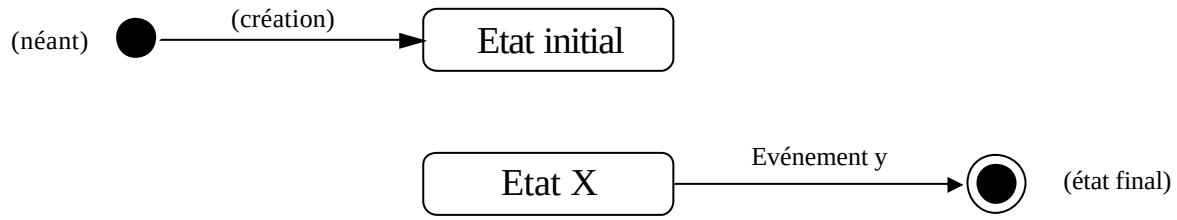


Etat:
un état spécifie la réponse de l'objet aux événements d'entrée. Il est stable.
Il peut être caractérisé par un ensemble de valeurs d'attributs et de liens d'un objet.
Il correspond à l'intervalle de temps entre deux événements reçu par l'objet.

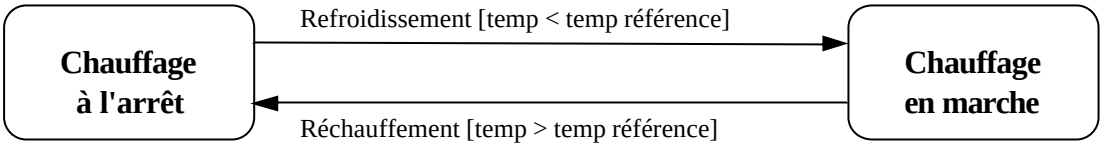
Transition :
Relation entre deux états indiquant qu'un objet dans le premier état va passer dans le deuxième quand un événement particulier apparaîtra, et que certaines conditions seront satisfaites.

Evénement:
un événement est un stimulus pouvant transporter des informations (sous forme de paramètres). Il est considéré comme n'ayant pas de durée.
Un événement peut être émis par un objet du système ou par un objet externe au système. Il peut également provenir de l'interface du système, par exemple un clic de souris. Quelle que soit son origine, un événement peut être soit dirigé vers un ou plusieurs objets, soit émis sans destination précise dans le système, certains objets du système pouvant alors le capter.
La réponse d'un objet à un événement dépend de l'état dans lequel il se trouve autant que de l'événement.
Remarque : la plupart des ouvrages et des outils de modélisation objet emploie les termes *événement* et *message* indifféremment. En UML, un message est un événement particulier, issu de l'interaction entre deux objets (un objet appelle une méthode d'un autre objet). Tout événement n'est donc pas forcément un message.

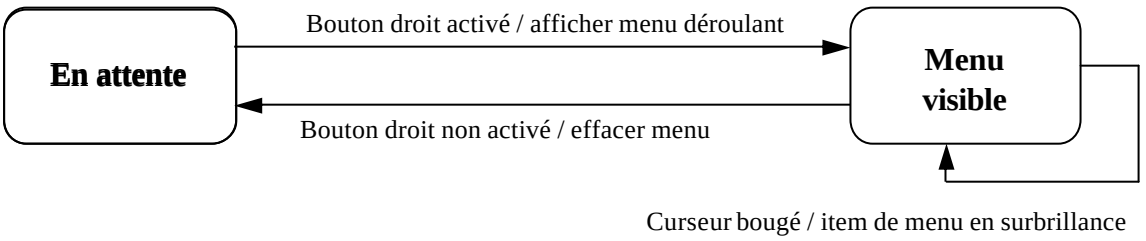
Etats initial et final :
Chaque automate de classe doit préciser un état initial (état d'une instance juste après sa création).
On peut également préciser les condition de destruction en ajoutant un état final.



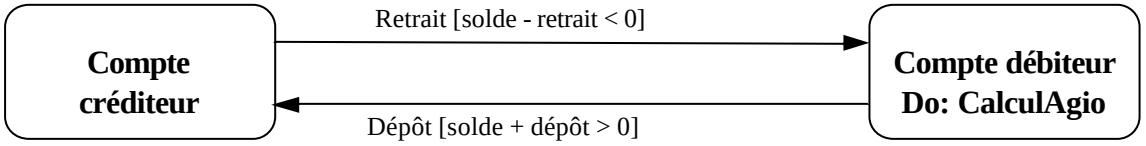
Condition :
une condition est une expression booléenne, attachée à un événement, qui doit être vraie pour le déclenchement de la transition.
Elle est indiquée entre crochets à la suite du nom de l'événement.



Action :
une action est une opération instantanée. Elle est associée à un événement.
Elle représente une opération dont la durée est insignifiante comparée à la résolution (niveau de définition) du diagramme d'état.
La notation d'une action sur une transition est précédée d'une barre oblique ("/").



Activité :
une activité est une opération qui dure un certain temps.
Elle est associée à un état. Elle peut être continue ou finie (se termine d'elle même).
Notation : "Do : activité".



Les diagrammes d'états peuvent s'exécuter en une seule fois ou en boucle continue.

Les diagrammes **"une seule fois"** (one-shot diagrams) représentent les objets qui ont une vie finie.

Un diagramme "une seule fois" possède un état initial et un état final..

L'état initial est entré à la création de l'objet; l'entrée dans l'état final (qui peut éventuellement prendre plusieurs conditions) implique, elle, la destruction de l'objet.

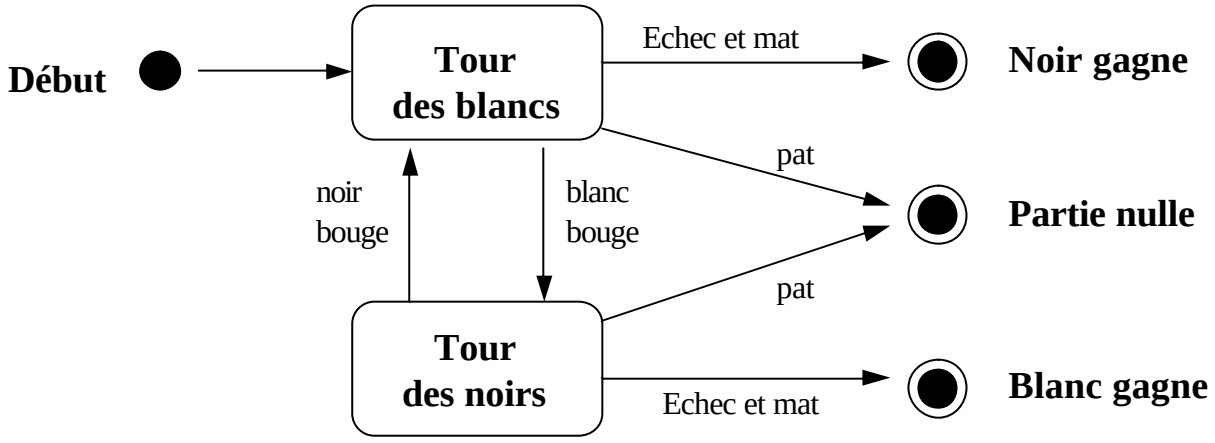
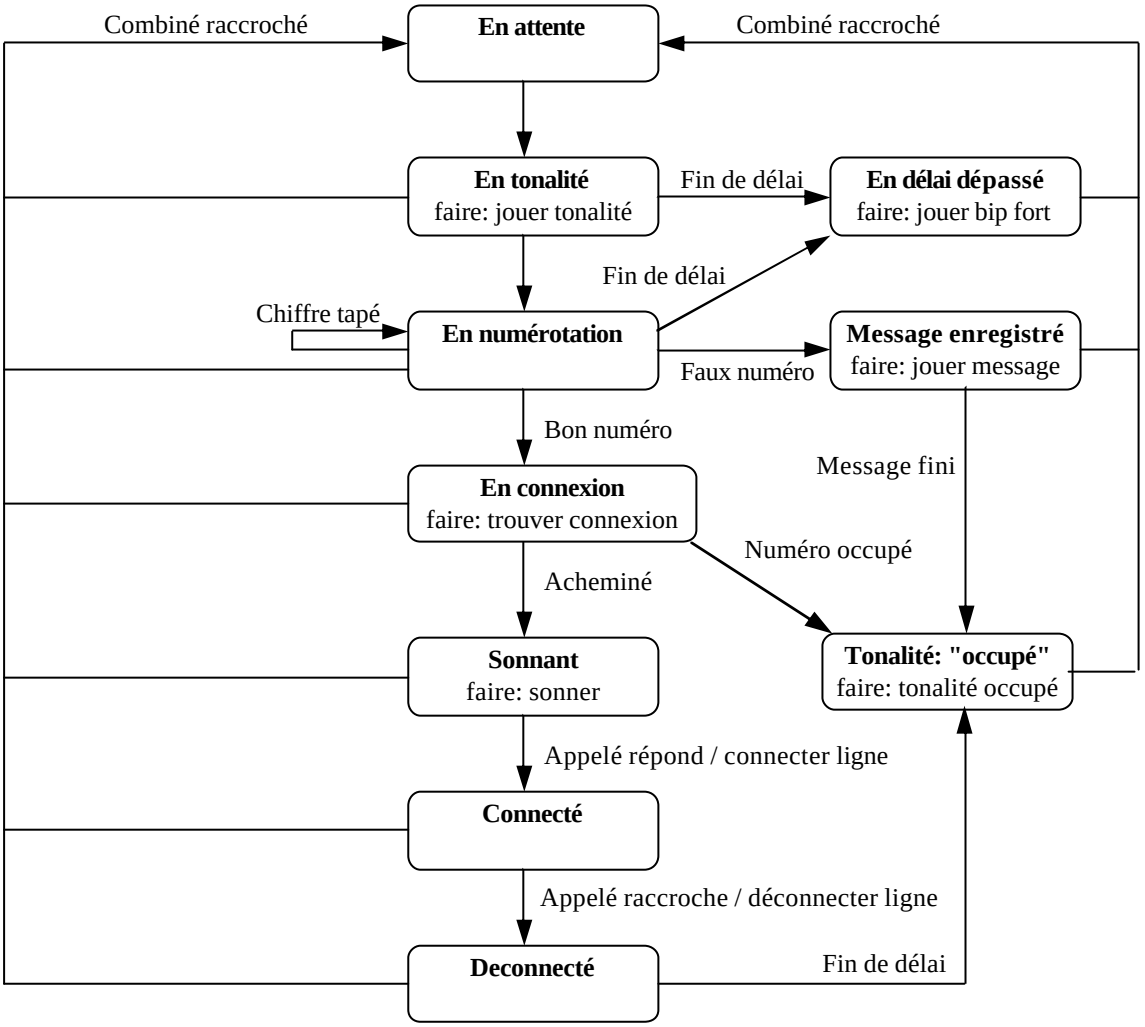


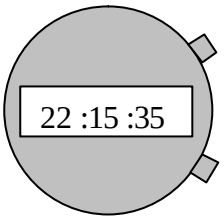
Diagramme en **boucle continue :**
Ce diagramme qui décrit le comportement d'une ligne téléphonique, est une *boucle continue.* On ne se soucie pas de savoir comment la boucle a démarré.



L'événement "Combiné raccroché" provoque une transition de n'importe quel état vers l'état "En attente".
Noter que les états ne définissent pas toutes les valeurs d'un objet; par exemple, l'état "En numérotation" inclut toutes les séquences de numéros incomplets.
Ce n'est pas la peine de les différencier puisqu'ils ont tous le même comportement.

Exercice

La montre



Bouton mode

Bouton avance

En fonctionnement normal, la montre affiche les heures, les minutes et les secondes qui défilent.
Une pression sur le bouton mode sélectionne le chiffre des heures qui se met à clignoter. A ce moment une pression sur le bouton avance incrémente le compteur d'une heure.
Une nouvelle pression sur le bouton mode sélectionne le chiffre des minutes qui se met à clignoter. A ce moment une pression sur le bouton avance incrémente le compteur d'une minute.
Une nouvelle pression sur le bouton mode revient à l'affichage normal.

1) Dessiner le diagramme état-transition de cette montre.

2) On veut modifier le comportement de la montre pour que si on maintient la pression sur le bouton avance au moins deux secondes, l'avancement des heures ou des minutes se fasse en continu.

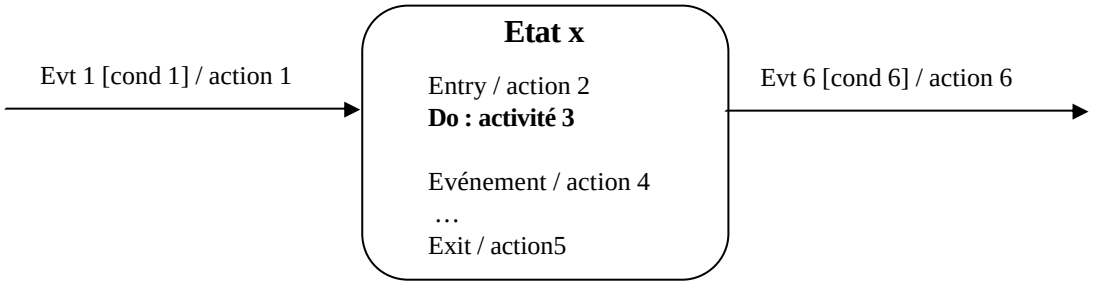
Complément sur les actions

En entrée :
L'action est exécutée chaque fois que l'on entre dans l'état.
Notation : **entry** / action

En sortie :
L'action est exécutée chaque fois que l'on quitte l'état.
Notation : **exit** / action

Action interne :
L'action est exécutée suite à un événement sans qu'il y ait de changement d'état.
Notation : événement / action

Notation complète et ordonnancement :



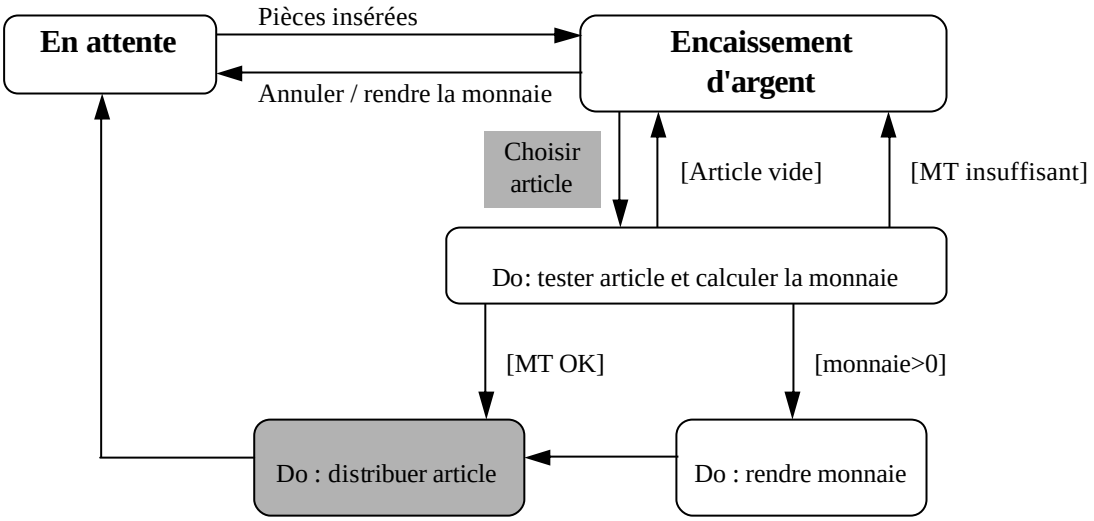
Ordre temporel d'exécution :
En entrée :
? Action sur la transition d'entrée (action 1)
? Action d'entrée (action 2)
? Activité (activité 3)

Dans l'état :
? Interruption de l'activité (activité 3)
? Exécution de l'action interne (action 4)
? Reprise de l'activité (activité 3)

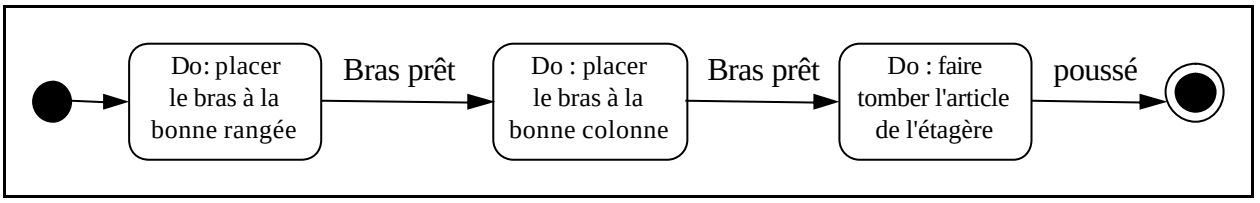
En sortie :
? Arrêt de l'activité (activité 3)
? Action de sortie (action 5)
? Action sur la transition de sortie (action 6).

Diagrammes d'états hiérarchiques ou imbriqués

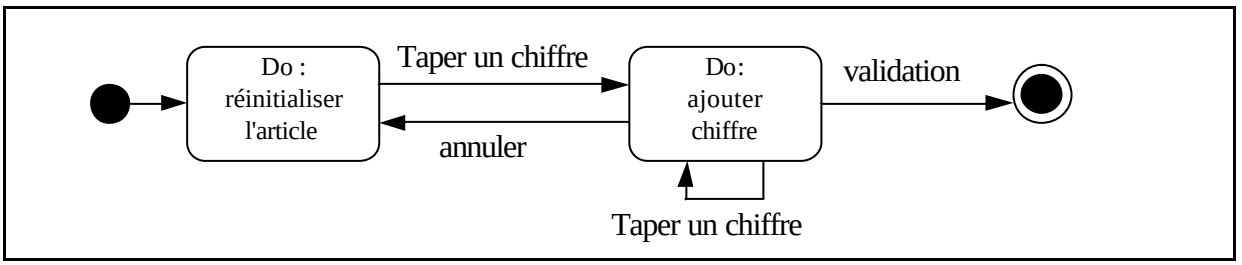
Dans le cas d'un comportement dynamique complexe, les diagrammes d'états sur un niveau deviennent rapidement illisibles.
Pour éviter ce problème, il est nécessaire de structurer les diagrammes d'états.
Cela permet à une activité liée à un état, d'être décomposée hiérarchiquement en sous-activités.



L'activité "distribuer article" est décrite dans ce sous-diagramme :

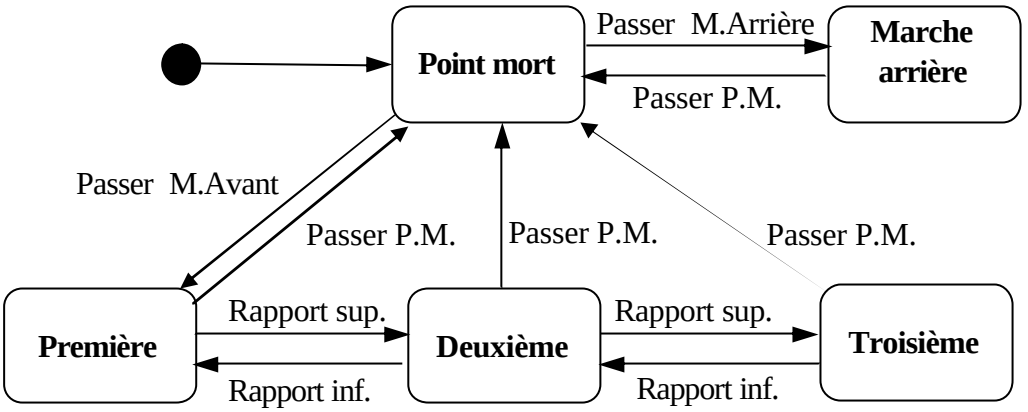


Les événements peuvent aussi être étendus dans des diagrammes d'états subordonnés. L'événement "choisir article" implique en fait plusieurs événements de plus bas niveau :

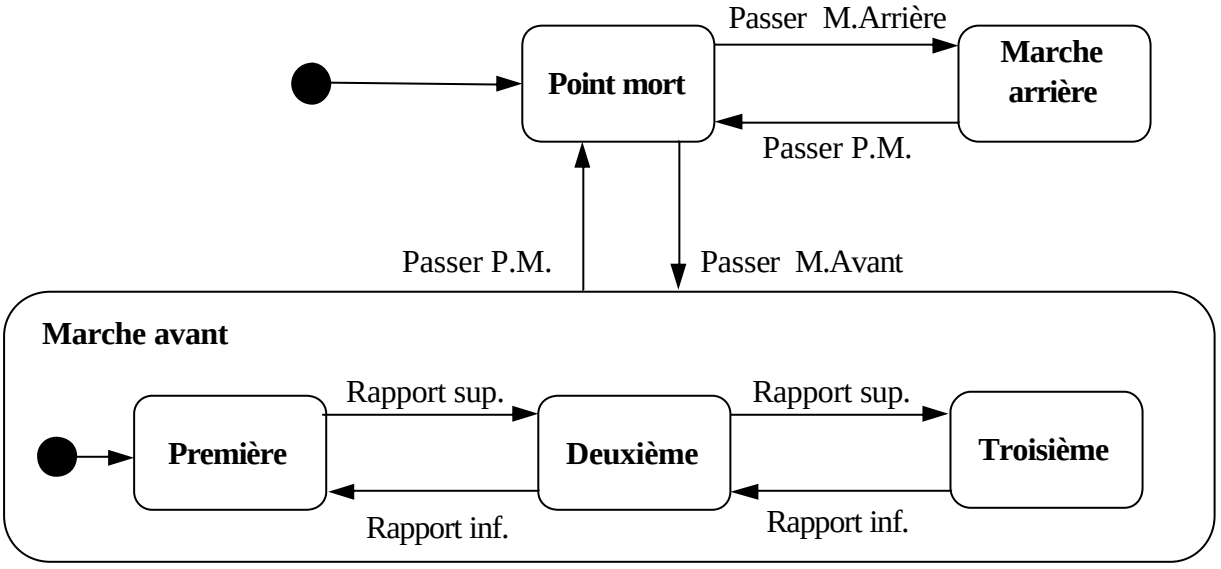


Généralisation d'états

On l'utilise généralement pour réduire la complexité de la représentation :



… en construisant un diagramme d'états imbriqué :

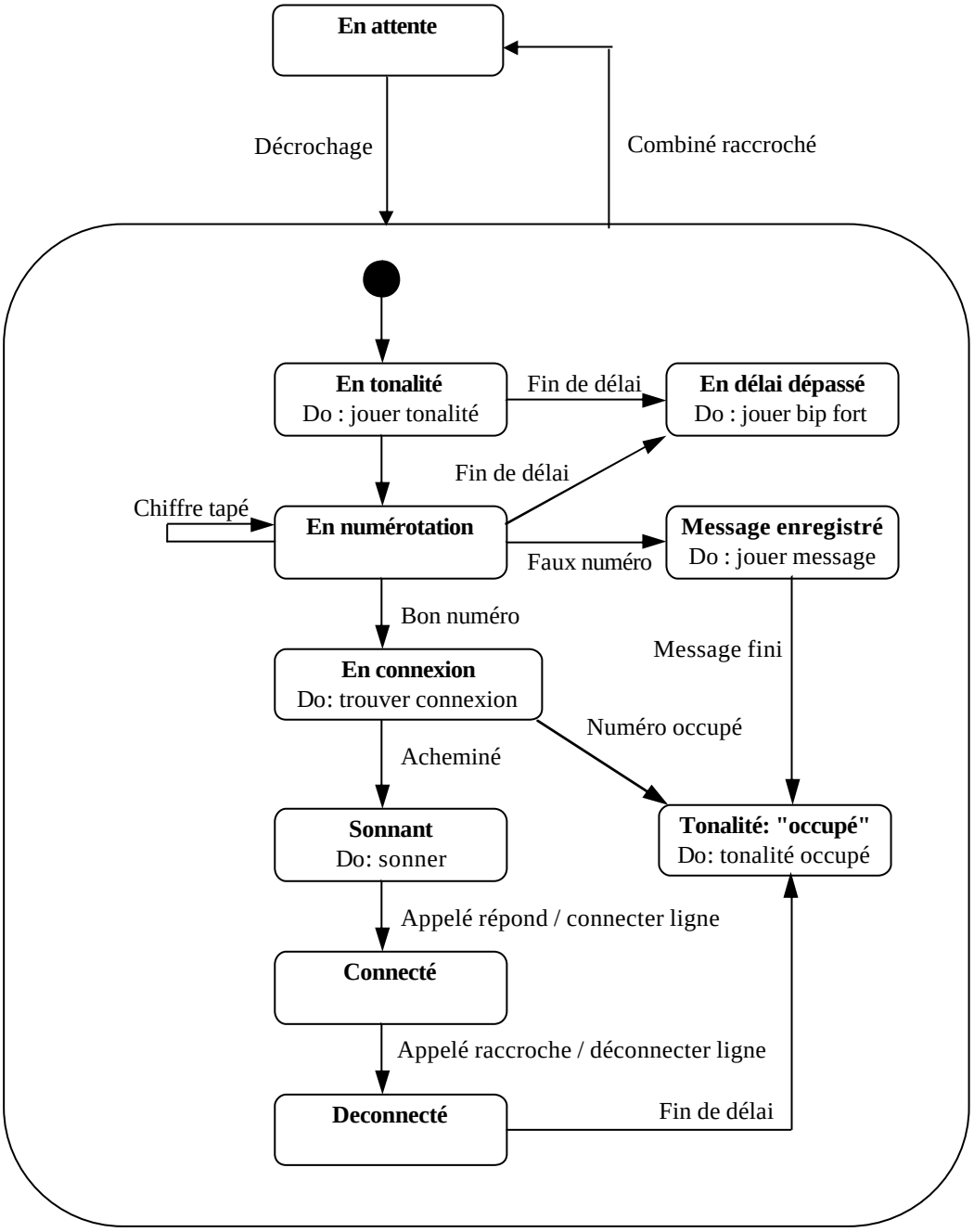


La généralisation exprime la "relation-ou". Ici, dans l'état *MarcheAvant*, la transmission se trouve dans le sous-état *Première* **ou** *Deuxième* **ou** *Troisième.*

Remarque : on peut repasser au point mort depuis n'importe quel sous-état, mais on entre toujours dans l'état *MarcheAvant* en première.

Exemple de la ligne téléphonique

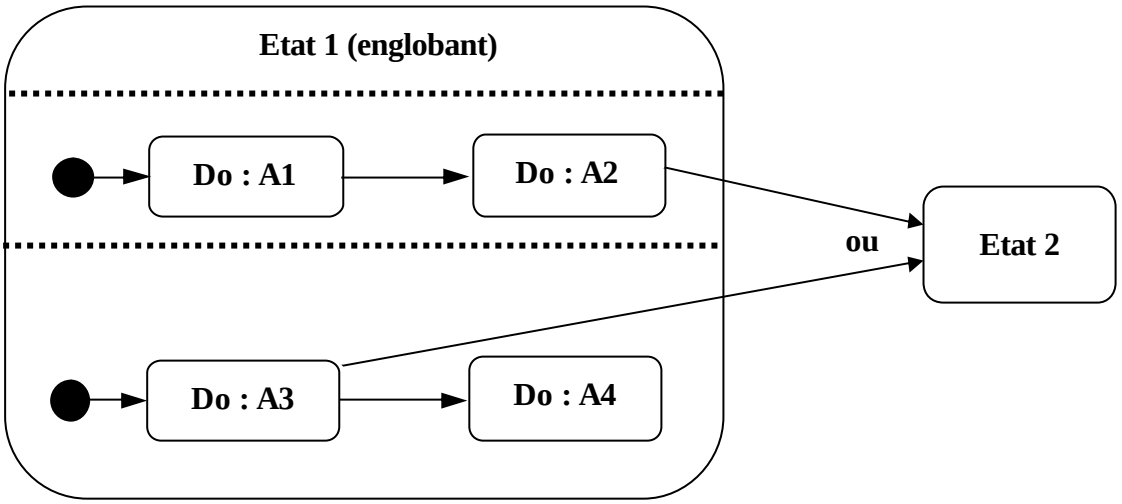
La généralisation permet de représenter plus simplement les différents états de l'objet étudié, sans surcharger le diagramme avec tous les retours vers un état final depuis n'importe quel état de l'objet.
Cette situation est très fréquente.



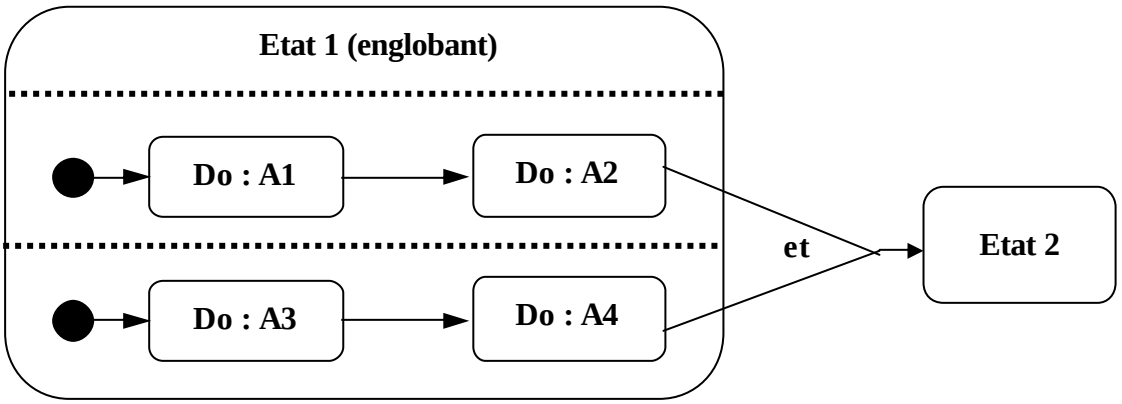
Sous-états parallèles

Plusieurs sous-diagrammes d'états peuvent intervenir en parallèle dans un même état. Ils sont alors :

Soit en concurrence : le premier sous-diagramme permettant de faire la transition de l'état englobant vers un autre état, interrompt les autres sous-diagrammes et fait quitter l'état englobant.

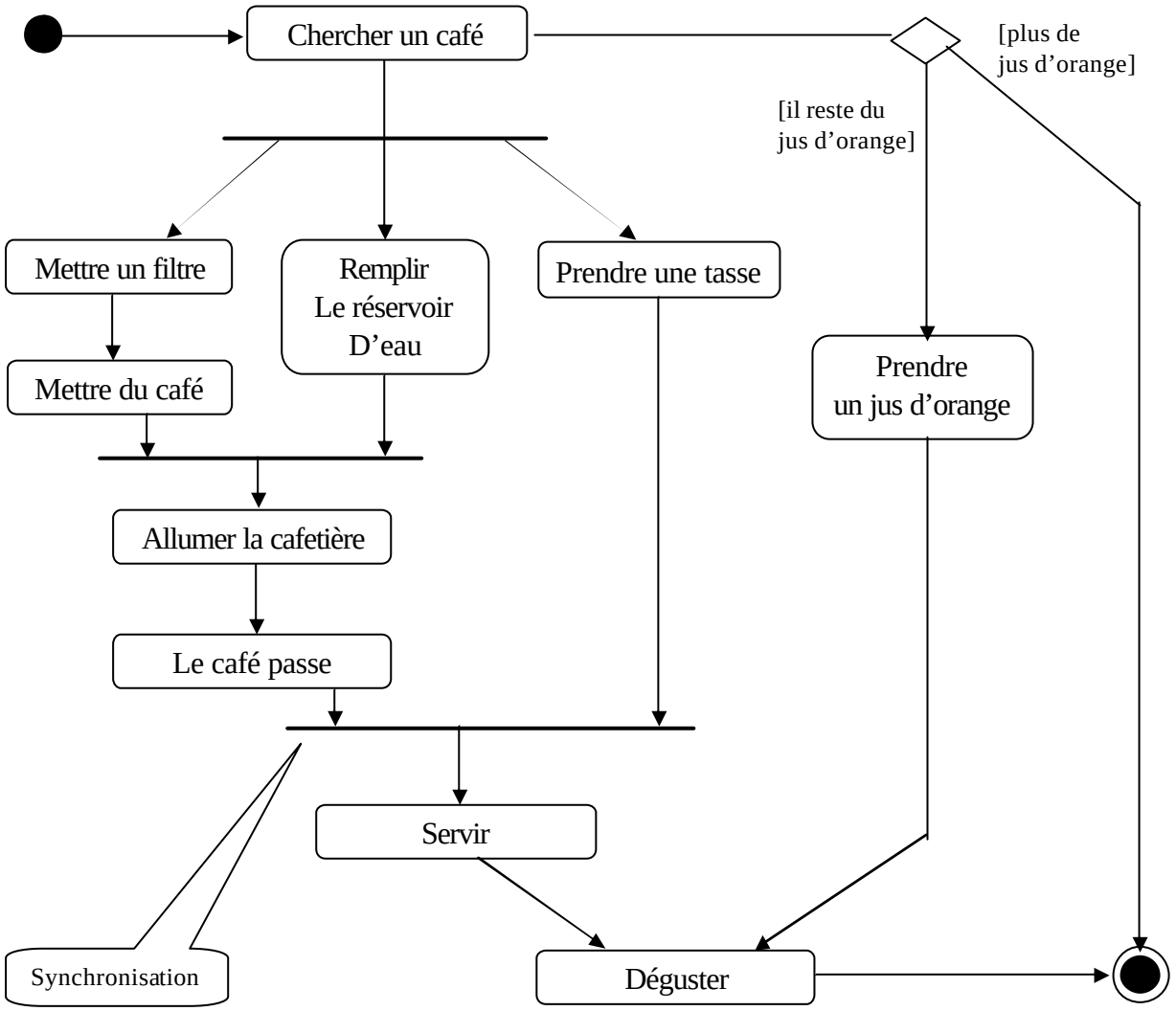


Soit en synchronisation : la transition de l'état englobant vers un autre état n'est effectuée que lorsque tous les sous-diagrammes le permettent, aucun sous-diagramme ne pouvant être interrompu.



Diagrammes d'activité
Ce sont des cas particuliers de diagramme d'état dans lesquels les états représentent des **activités**, et les transitions des **transitions automatiques**.

Il sont plutôt rattachés à une opération ou un Use Cases qu'à une classe.
Utilisés pour décrire l'algorithmique d'une opération : séquences d'étapes, avec représentation de décisions et de la synchronisation des flots de contrôles.
Focalisés sur les traitements internes et non pas sur la réception d'événements externes.



Responsabilités

Un diagramme d'activité peut être divisé en couloirs verticaux (« swinlanes ») assignant la responsabilité de certaines activités à des objets particuliers.

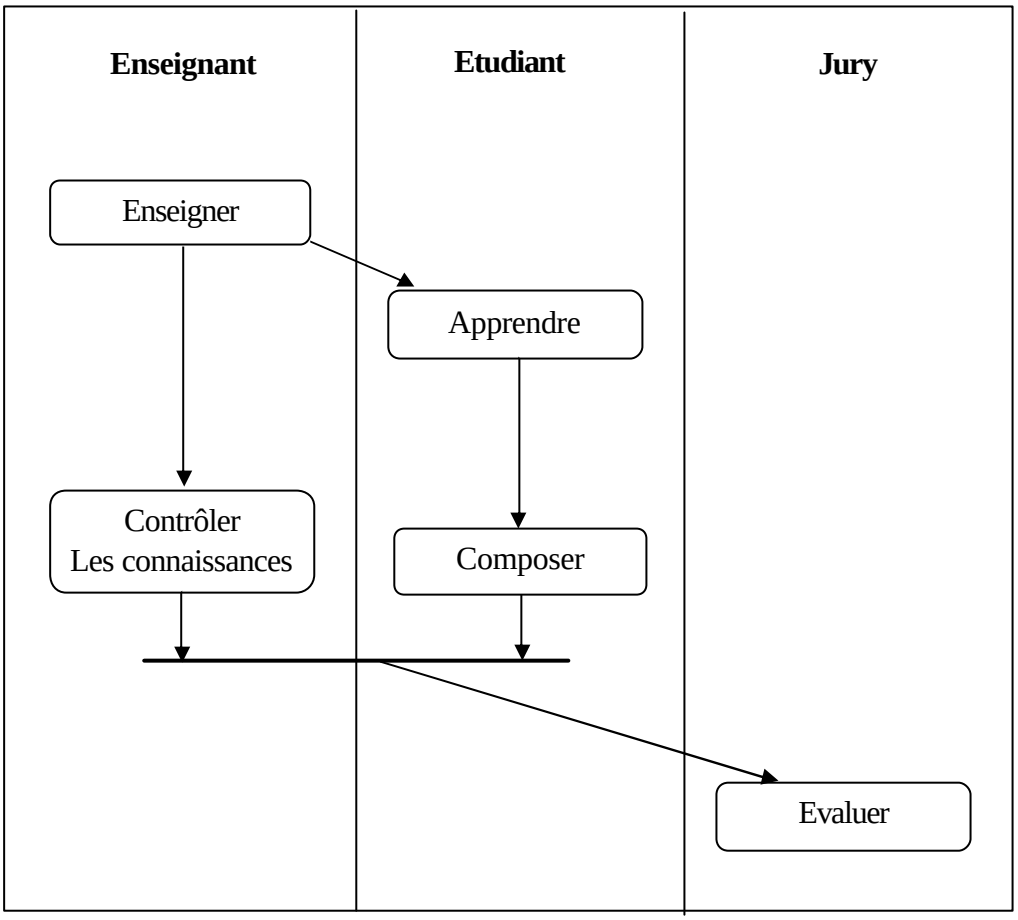
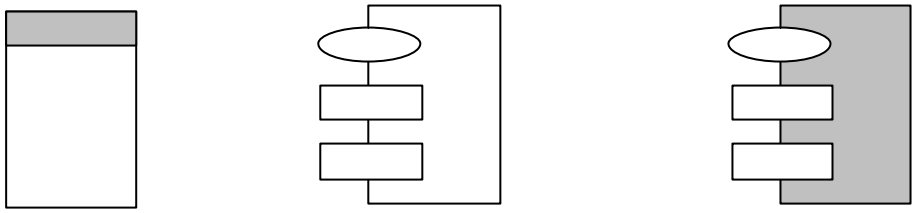


Diagramme de composants

Un diagramme de composants décrit des modules et leurs relations dans l'environnement de réalisation.
Les modules peuvent être de simples fichiers, des paquetages du langage Ada, ou encore des bibliothèques chargées dynamiquement.
La notation utilise 3 icônes pour désigner respectivement :

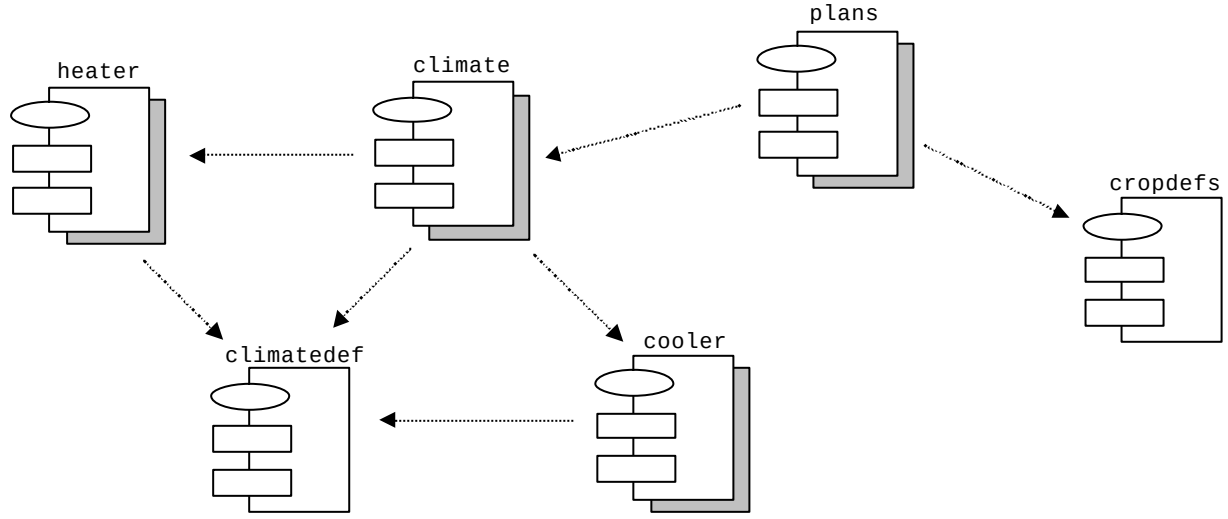
- ? Le programme principal qui correspond à la fonction *main* du C++
- ? La spécification qui contient les déclarations ; fichier `.h` du C++
- ? Le corps (body) qui correspond au fichier `.cpp` du C++.



Main *spécification* *body*

Les relations, représentée par une flèche pointillée qui pointe de l'utilisateur vers le fournisseur, montrent les dépendances de compilation ou les dépendances d'exécution entre les différents modules.

Exemple tiré du livre de Grady Booch



Deux modules, `climatedefs` et `cropdefs` ne sont que des spécifications et exportent des types et des constantes. Les quatre autres sont représentés avec leurs spécifications et leurs corps groupés car ils sont très liés.
Les dépendances suggèrent un ordre partiel de compilation. Le corps de `climate` dépend de la spécification de `heater`, laquelle dépend de celle de `climatedefs`.

Diagramme de déploiement (1)

Un diagramme de déploiement montre la disposition physique des différents matériels (les nœuds) qui entre dans la composition d'un système et la répartition des programmes exécutables sur ces matériels.

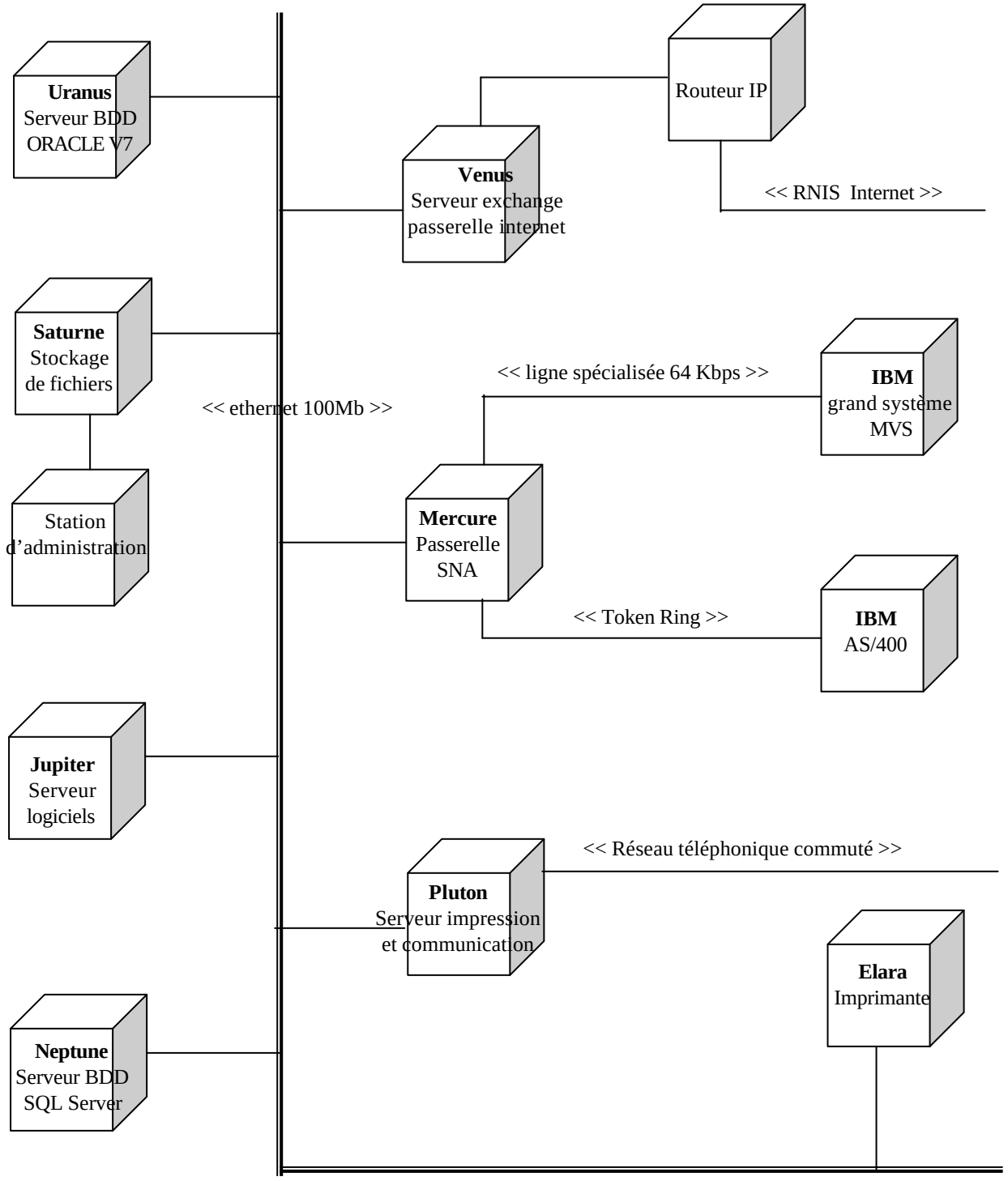
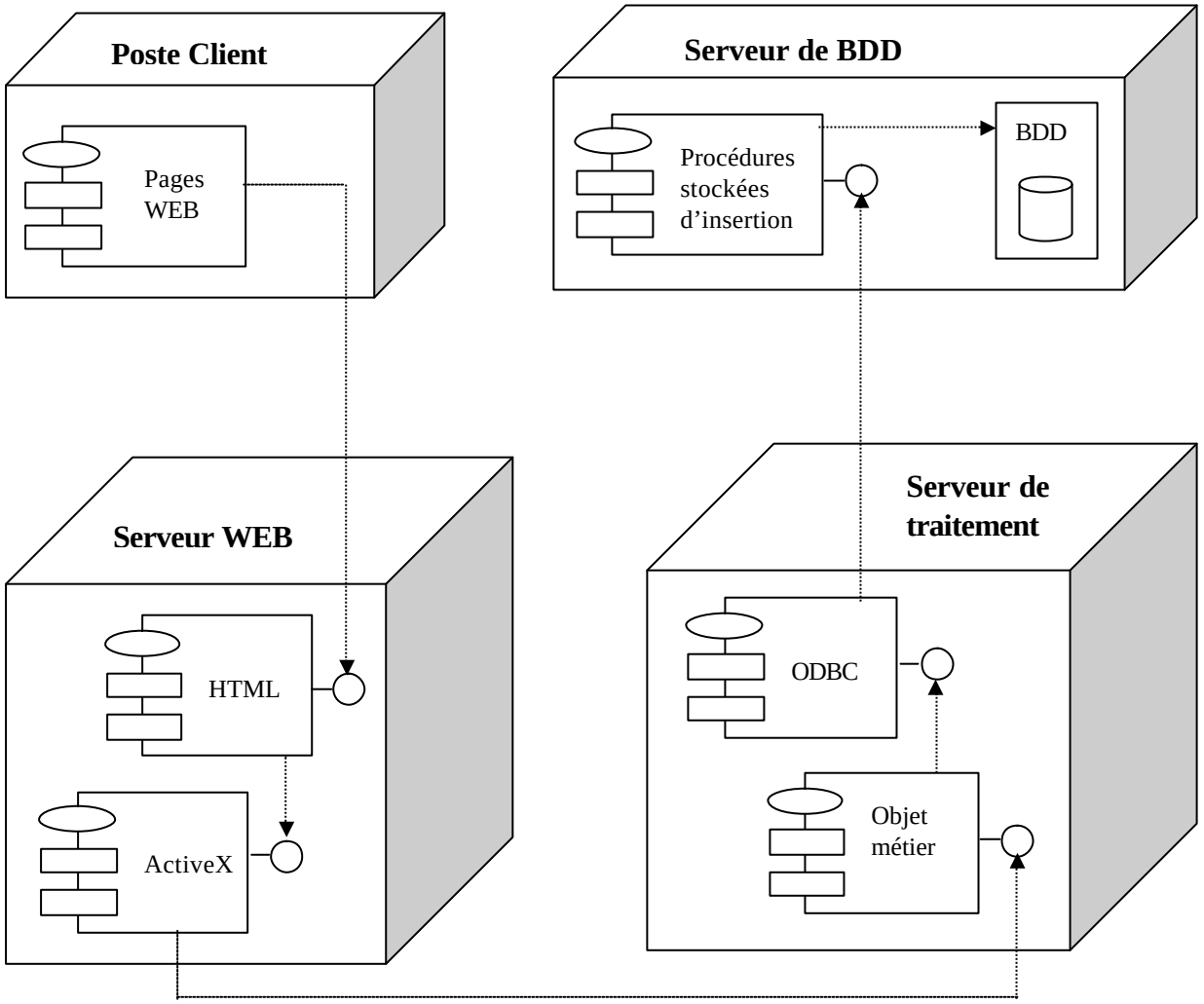


Diagramme de déploiement (2)

Un diagramme de déploiement peut spécifier la répartition des traitements dans une architecture client-serveur.



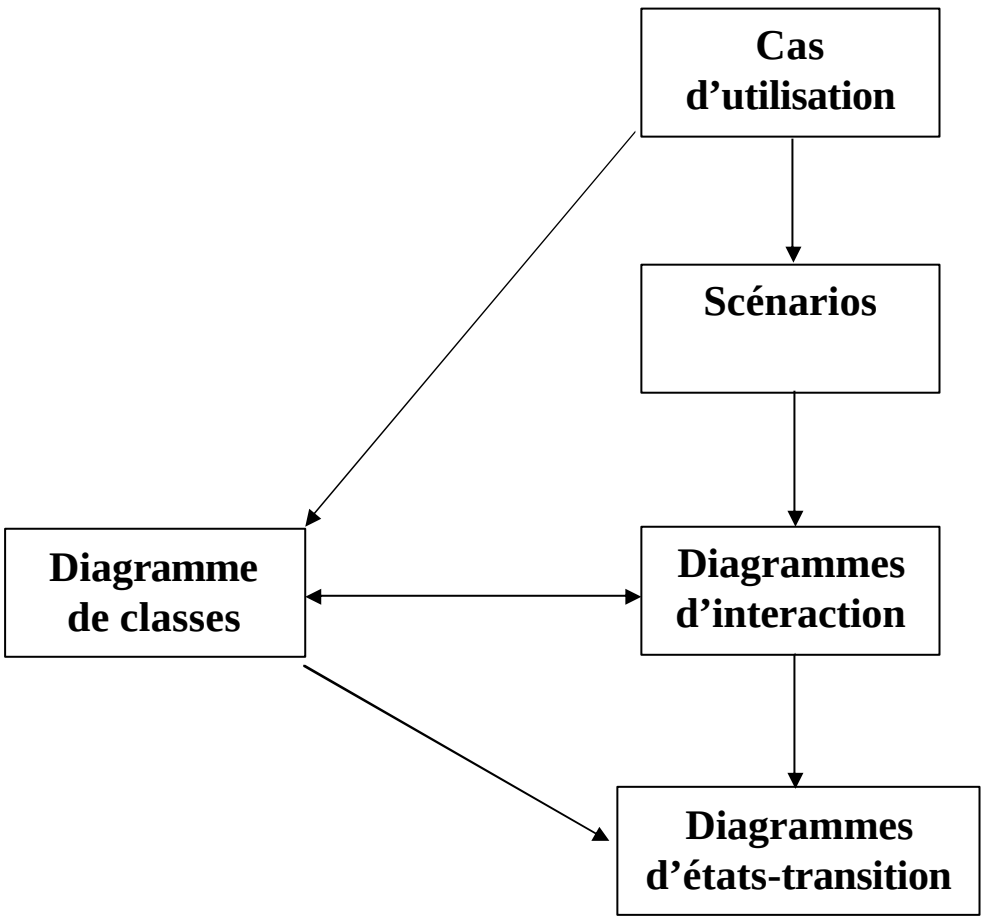
Synthèse : les 9 diagrammes UML

La démarche UML

UML propose un standard de notation (autour des 9 diagrammes) pour représenter l'analyse et la conception orientée objet. Ce n'est pas une notation fermée : elle est générique, extensible et configurable par l'utilisateur. Une grande liberté est donnée aux outils pour le filtrage et la visualisation d'information.

Il n'y a pas de démarche standard dans UML. On peut cependant en proposer une :

- définir et décrire les cas d'utilisations
- pour chaque cas d'utilisation, choisir les scénarios
- construire les diagrammes d'interaction pour chaque scénarios
- élaborer en parallèle le diagrammes de classes.



Exercice

Téléphone public

Imaginer un téléphone public très simple, à pièces.
Le prix d'une communication est de 2 francs.
Après l'introduction de la monnaie, l'utilisateur a 1 minute pour numéroter.

La ligne peut être libre ou occupée. L'appelé peut raccrocher le premier.

Le téléphone consomme de l'argent dès que l'appelé décroche et à chaque unité de temps. On peut rajouter des pièces n'importe quand.