Modélisation objet et diagrammes UML dynamique

Aurélien Tabard
Département Informatique
Université Claude Bernard Lyon 1

Basé sur les cours de Yannick Prié, Julien Mille et Laurent Audibert

Plan général

1. Les cas d'utilisations
2. Introduction au langage de modélisation UML
3. Modélisation objet et diagrammes UML statiques
4. Modélisation UML dynamique

Plan général

1. Les cas d'utilisations
2. Introduction au langage de modélisation UML
3. Modélisation objet et diagrammes UML statiques
4. Modélisation UML dynamique

Objectifs de ce cours

- Apprendre la syntaxe et la sémantique des diagrammes dynamiques et d'interaction les plus importants
- Améliorer au passage la compréhension de différents principes objets

Plan du cours

- **Diagramme de séquence :** représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou les acteurs
- **Diagramme d'états−transitions :** description, sous forme de machine à états finis, du comportement d'un système ou de l'un de ses composants (ex: une classe)

Diagramme de séquence

Principal diagramme d'interaction

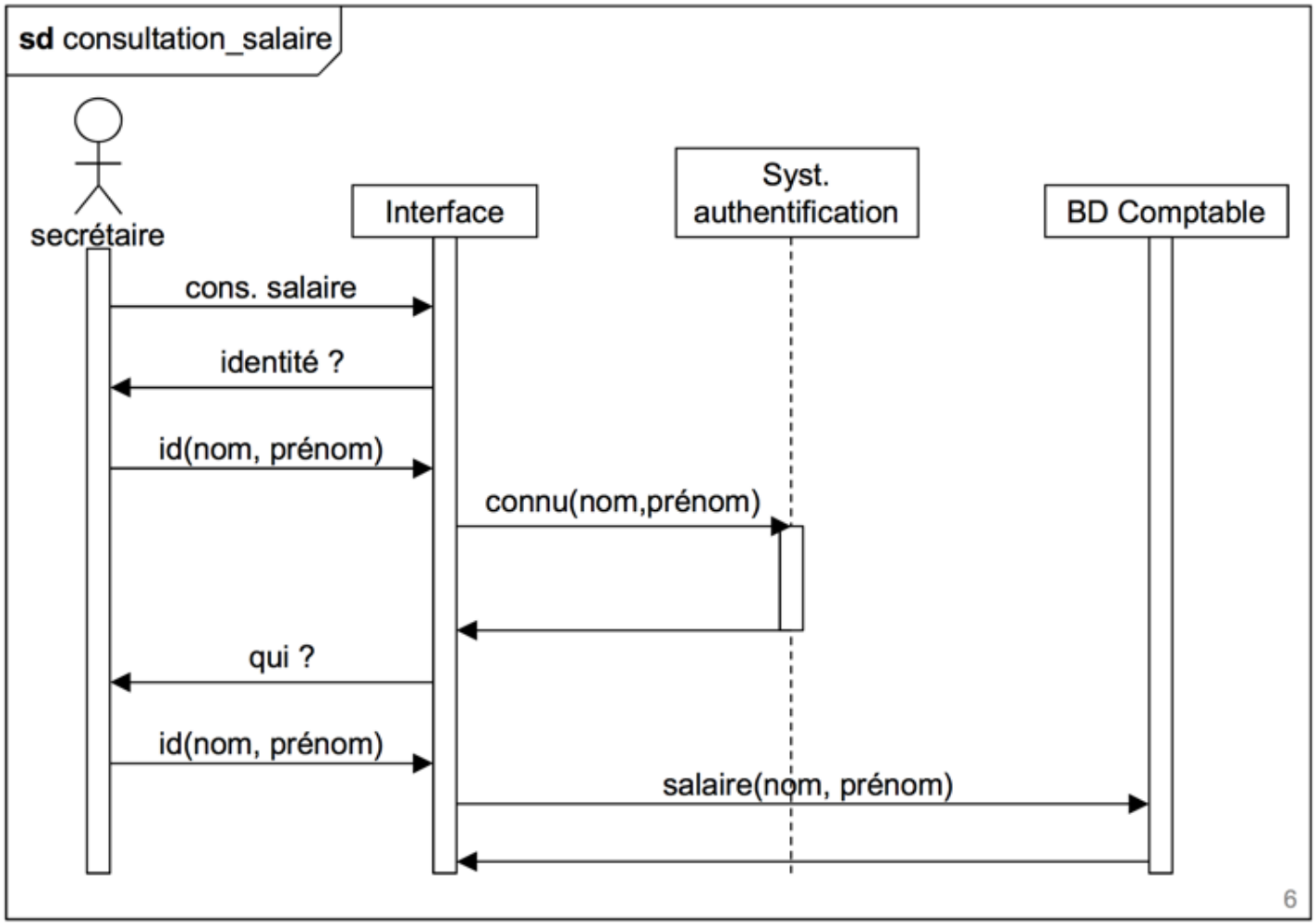
Objectifs :

- Représentation du déroulement des traitements et des interactions entre les éléments du système et/ou les utilisateurs
- Centré sur l'expression des interactions et l'échange de messages

Portée :

- un diagramme de séquence se rapporte par exemple à un ou plusieurs scénario(s) identifié(s) dans les cas d'utilisation
- toujours nommer un diagramme de séquence !
- ce n'est pas une description exhaustive du comportement du système

Exemple


sd consultation_salaire
secrétaire
Interface
Syst.
authentification
BD Comptable
cons. salaire
identité ?
id(nom, prénom)
connu(nom,prénom)
qui ?
id(nom, prénom)
salaire(nom, prénom)
6

Composants des diagrammes de séquence

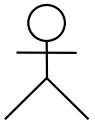
Boîte

- Coin supérieur gauche : étiquette avec sd (sequence diagram) suivi du nom (= du scénario)

Axe vertical

- Représentation implicite du temps
  - le temps augmente lorsqu'on se déplace vers le bas
- Non gradué (la durée des traitements n'est pas prise en compte, c'est l'enchainement qui est important)

Acteurs :

- Utilisateur : 
  Objet : 

Messages : flèches entre acteurs

Acteurs et axes verticaux

L’Axe vertical est la "ligne de vie" de l'acteur dans le scénario

Emplacement de l'acteur (utilisateur ou objet)

- Si l'acteur existe dès le début du scénario, en haut
- Sinon, à l'extrémité du message donnant lieu à sa création

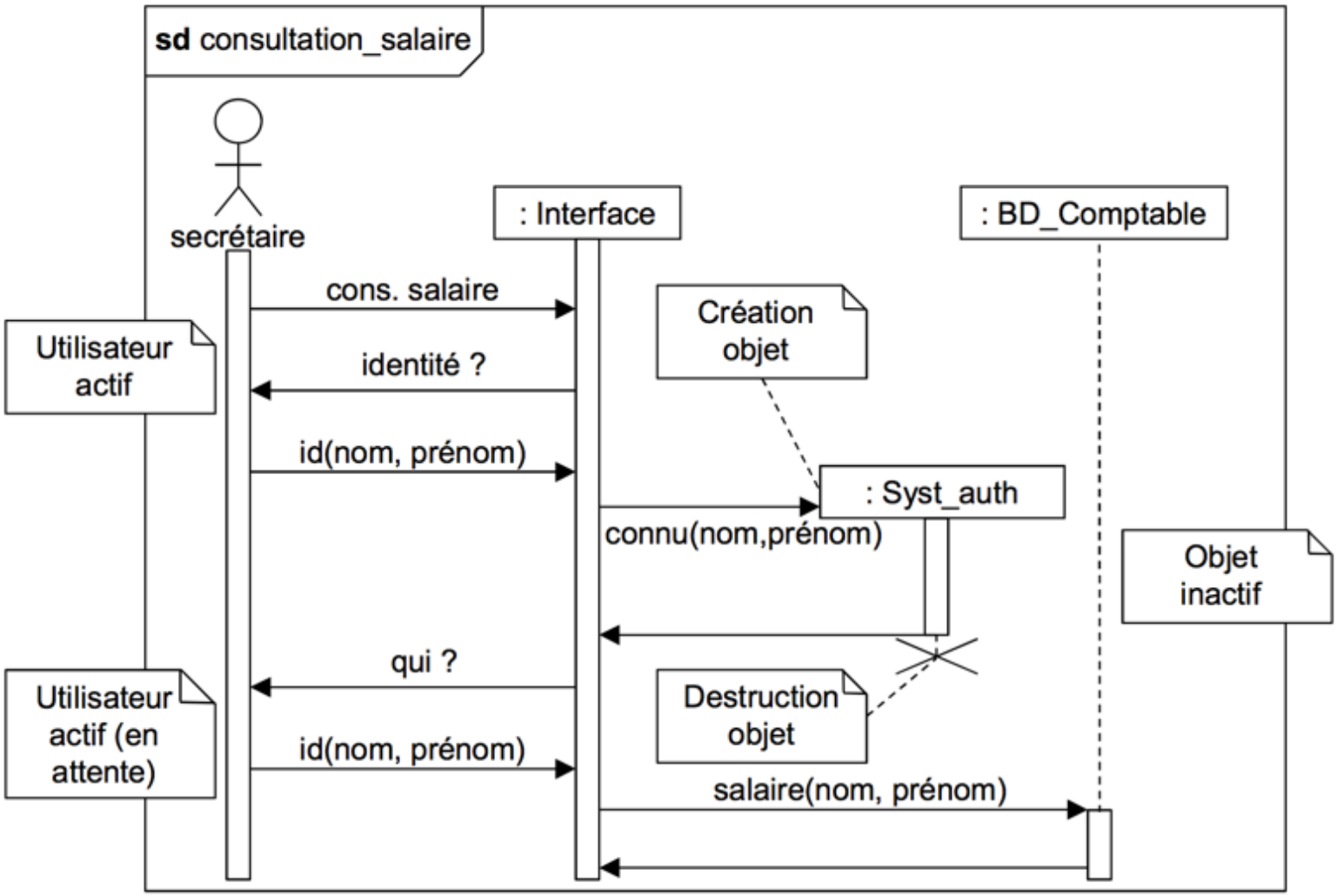
Fin de l'acteur

- Si l'acteur n'est pas détruit : aucune
- Sinon, une croix là ou l'objet est détruit

Activité

- Un acteur peut être actif ou inactif
- Un acteur actif effectue une opération ou attend le retour d'un envoi de message en mode synchrone (ex: appel de fonction)
- Trait épais creux sur les périodes pendant lesquelles l'objet est actif
- Trait en pointillé sur les périodes pendant lesquelles l'objet est inactif

Exemple

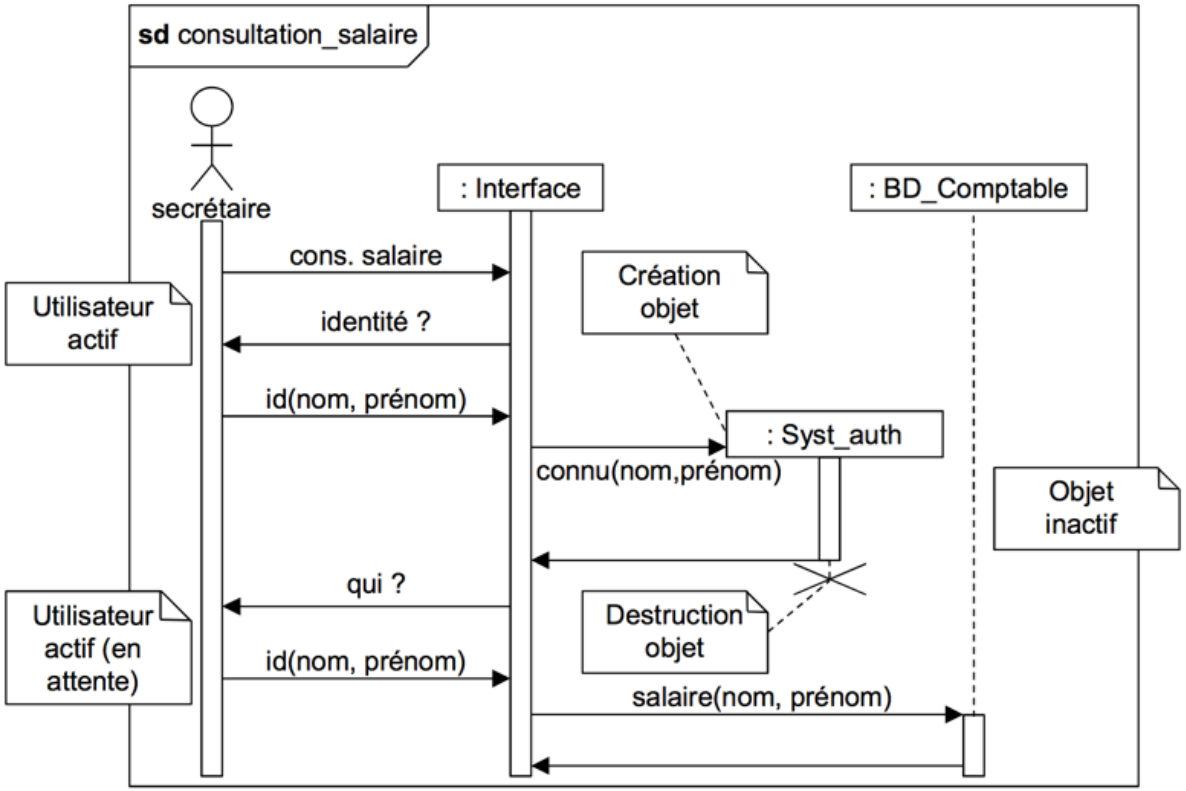

sd consultation_salaire
secrétaire
: Interface
: BD_Comptable
cons. salaire
Création objet
Utilisateur actif
identité ?
id(nom, prénom)
: Syst_auth
connu(nom,prénom)
Objet inactif
qui ?
Utilisateur actif (en attente)
Destruction objet
id(nom, prénom)
salaire(nom, prénom)

Exercice

Achat d’un ticket sur une borne TCL

Exercice : Achat d'un ticket sur une borne TCL

Exemple :



Message

Matérialisation d'une communication, avec transmission d'information entre :

- un émetteur (source)
- un récepteur (destination)

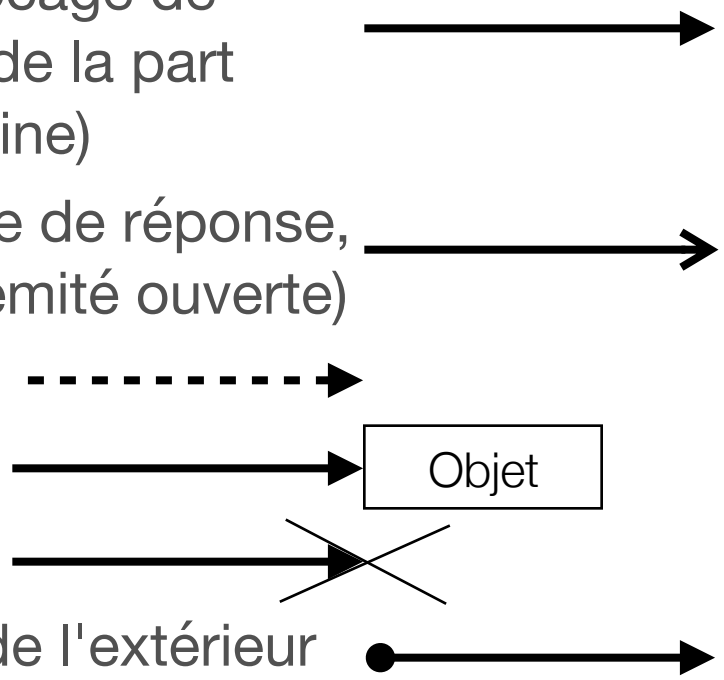
Un message peut :

- déclencher une opération
  - rôle le plus utilisé en programmation objet. Le déclenchement de l'opération peut être synchrone (cas le plus fréquent : l'émetteur reste bloqué le temps que dure l'invocation de l'opération) ou asynchrone (création d'un thread)
- représenter l'émission d'un signal
  - l'envoi d'un signal déclenche une réaction chez le récepteur, de façon asynchrone et sans réponse : l'émetteur du signal ne reste pas bloqué le temps que le signal parvienne au récepteur. Il ne sait pas quand, ni même si le message sera traité par le destinataire
- entrainer la création/destruction d'un objet

Messages : représentations

Flèche = envoi de message

- Message standard (synchrone) : blocage de l'émetteur en attendant la réponse de la part du récepteur (flèche à extrémité pleine)
- Message asynchrone : pas d'attente de réponse, poursuite de la tâche (flèche à extrémité ouverte)
- Réponse
- Création d'un objet
- Destruction d'un objet
- Lancement de l'interaction venant de l'extérieur



Message : Syntaxe

([ ] = facultatif)

NomSignalOuOpération ['(' argument1, [argument2, ...] ')']

Syntaxe d'un argument

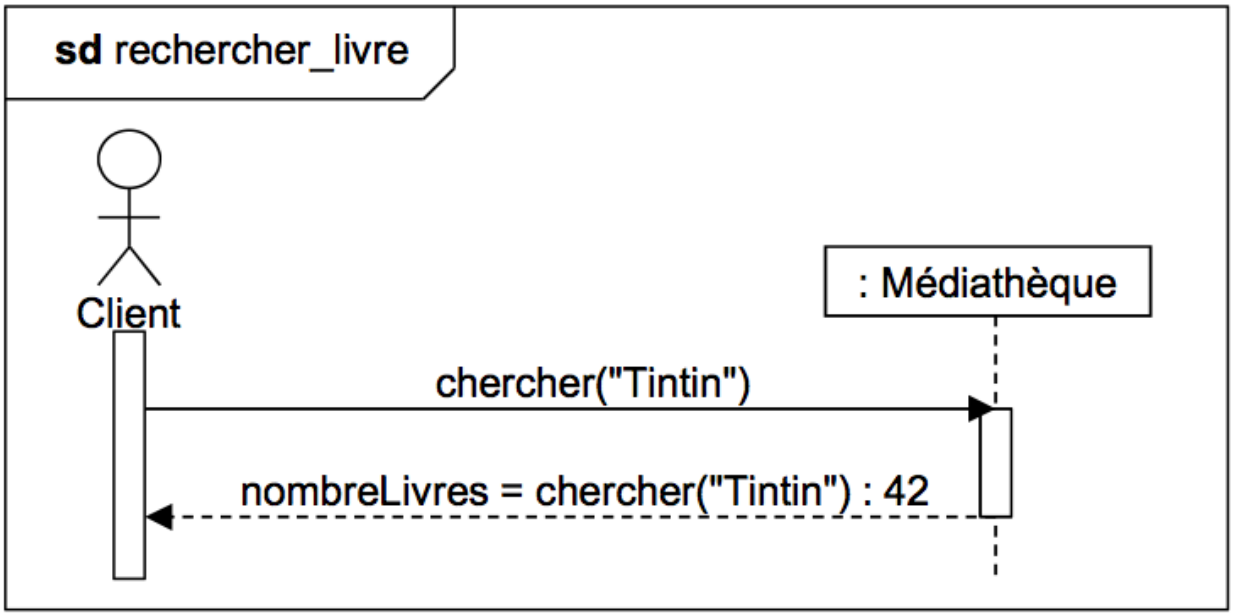
- Si entrée : [NomParamètre '='] ValeurArgument
- Si sortie ou entrée-sortie : NomParametre [':' ValeurArgument]
- Exemples :
  - `initialiser(x=100)` : l'argument en entrée reçoit la valeur 100
  - `f(x:12)` : l'argument en entrée-sortie est x, avec pour valeur initiale 12

Syntaxe de réponse à un message

- [NomAttribut '='] Message [':' ValeurRetour]
  où Message représente le message envoyé

Syntaxe des messages, exemple

le message de réponse signale que 42 occurrences de la référence "Tintin" figurent dans une médiathèque



Messages, remarques

- Flèches horizontales = l'envoi d'un message est considéré comme instantané (le temps de transfert n'est pas pris en compte)
- Les messages asynchrones peuvent être reçus dans un ordre différent de l'ordre d'envoi

Fragments d’interaction combinés

- Permettent de décomposer une interaction en fragments simples
- Représentation graphique des tests et des boucles des langages de programmation
- Représentation identique à celle d'une interaction
  - **Boîte**, englobant TOUTES les lignes de vies des acteurs concernés
  - **Etiquette** dans le coin supérieur gauche, contenant le type de combinaison (appelé "opérateur d'interaction")
  - Eventuellement, plusieurs opérandes séparés par une ligne pointillée (exemple : dans le cas d'un fragment de type alt (alternative), les opérandes sont les différents choix)

Fragments d’interaction combinés

Regroupement des opérateurs d'interaction par fonctions :

- choix et boucle :
  - **alt, option, break, loop**
- contrôle de l'envoi de messages en parallèle :
  - **parallel**, critical region
- contrôle de l'envoi de messages :
  - ignore, consider, **assert**, negative
- ordre d'envoi des messages :
  - weak sequencing, strict sequencing
- référencement
  - ref

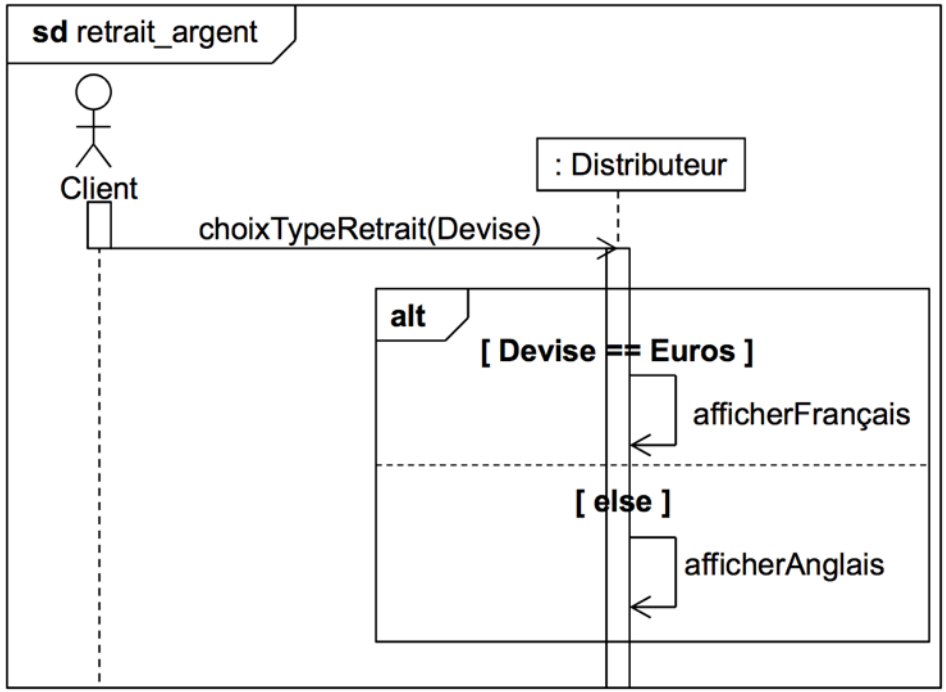
Fragments d’interaction combinés

Regroupement des opérateurs d'interaction par fonctions :

- **choix et boucle :**
  - **alt, option, break, loop**
- contrôle de l'envoi de messages en parallèle :
  - parallel, critical region
- contrôle de l'envoi de messages :
  - ignore, consider, assert, negative
- ordre d'envoi des messages :
  - weak sequencing, strict sequencing
- référencement
  - ref

Fragments, alternative simple

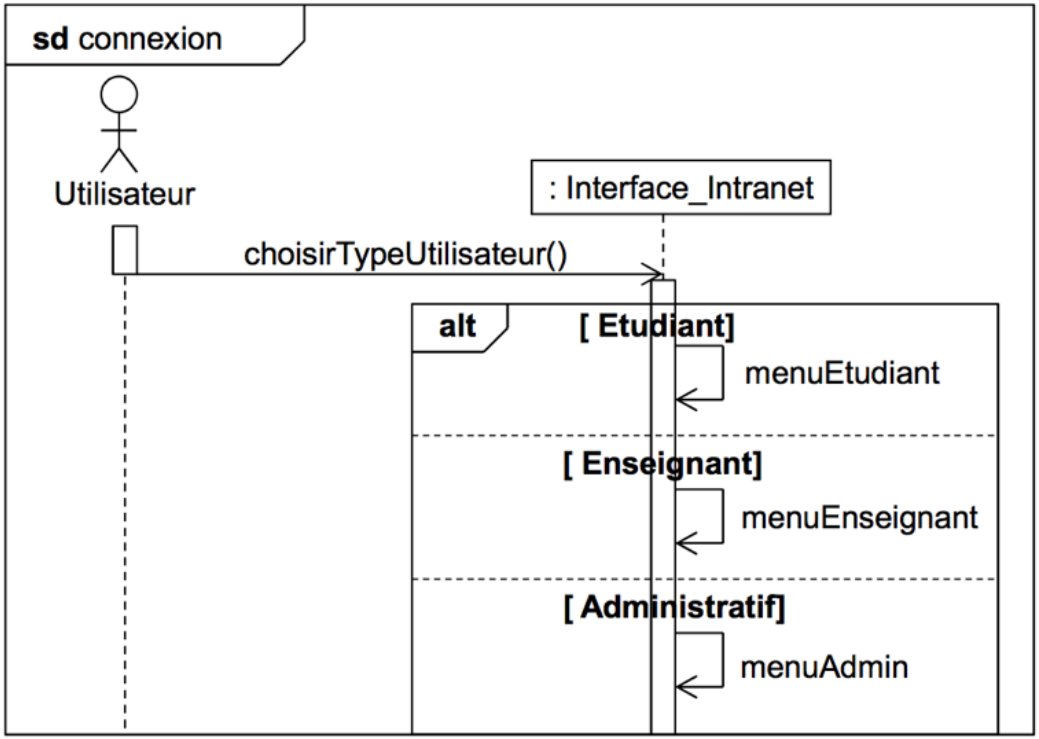
- Equivalent à un bloc `if (...) {bloc1} else {bloc2}`
- Condition booléenne entre crochets [ ]
- Opérandes : bloc1, bloc2



Fragments, alternative complexe

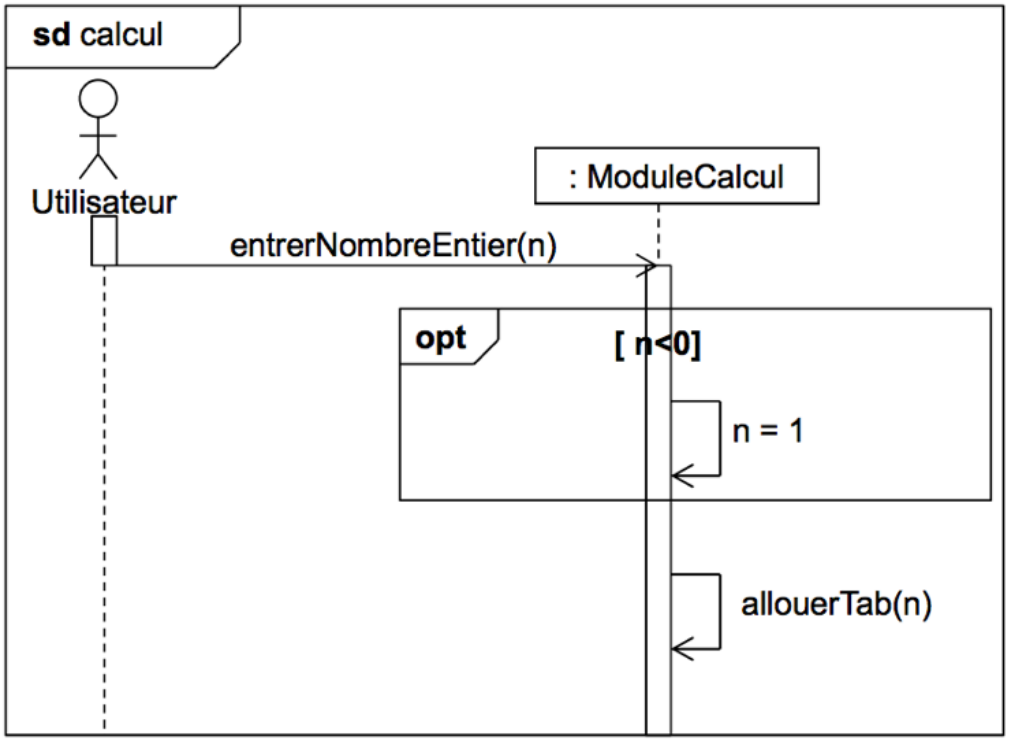
Plus de deux opérandes,

- signification équivalente à `switch (...) { case: ...}`
- Valeur entre crochets dans chaque opérande



Fragments, option

- Equivalent à un bloc `if (...) {bloc1}`
- Condition booléenne entre crochets [ ]
- Un seul opérande



Fragments, boucles

Syntaxe : loop ['(' MinInt [',' MaxInt] ')']

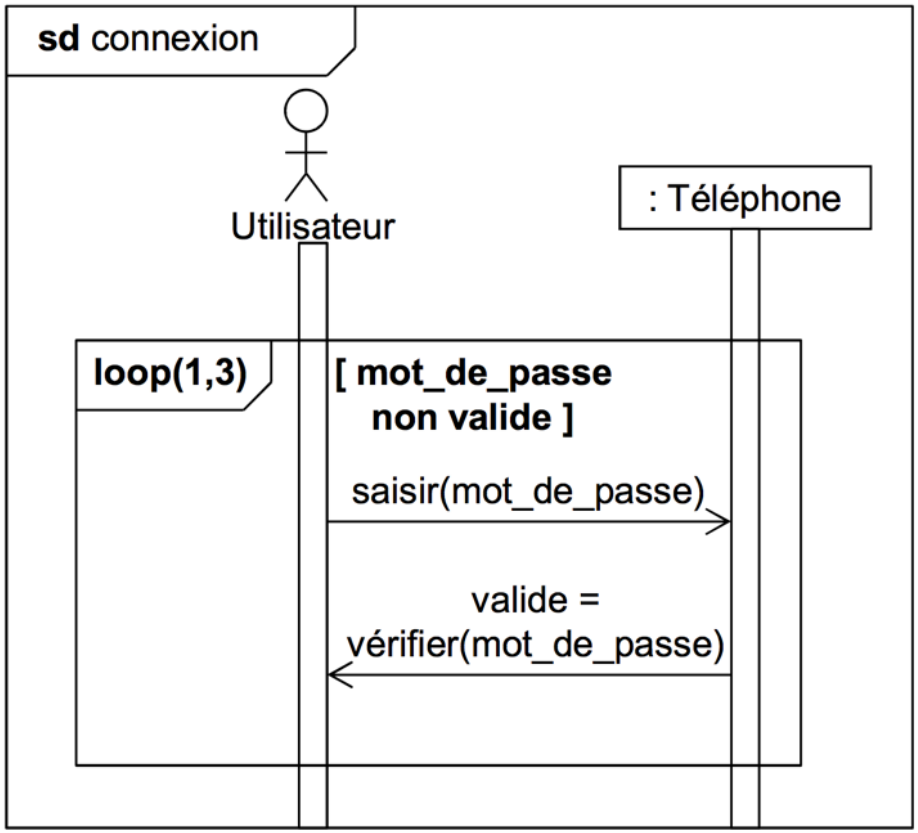
- La boucle est répétée au moins MinInt fois avant qu'une éventuelle condition booléenne ne soit testée (si elle est présente, la condition est placée entre crochets en haut du fragment). Tant que la condition est vraie, la boucle continue, au plus MaxInt fois

Remarques

- MinInt ≥ 0, MaxInt ≥ MinInt
- loop(valeur) est équivalent à loop(valeur, valeur)
- loop est équivalent à loop(0,*) (* = "illimité")
- **MinInt et MaxInt ne sont pas les bornes inférieures et supérieures d'un indice** ! (≠ `for (i=MinInt; i<MaxInt; i++)`)

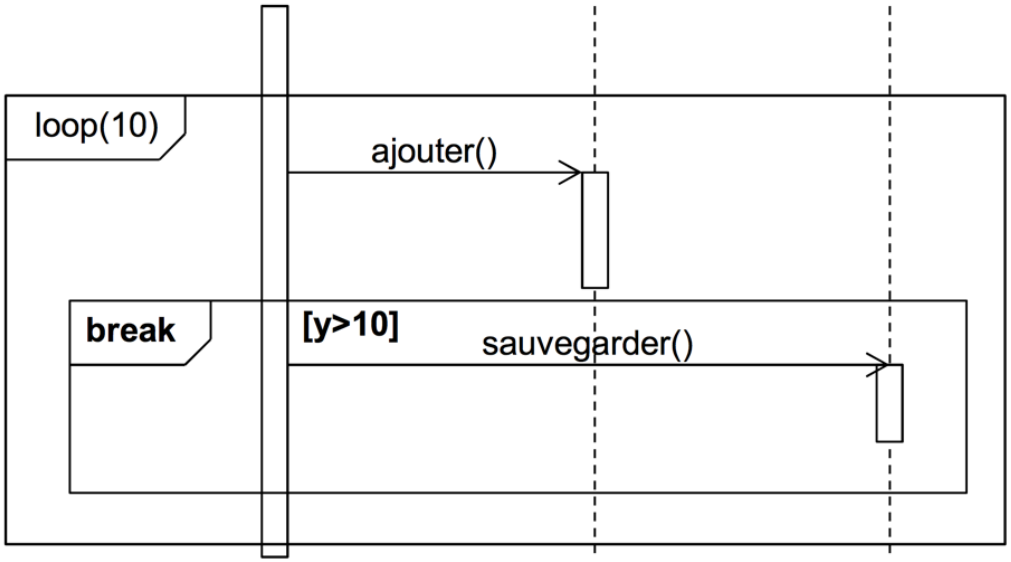
Fragments, boucle exemple

boucle avec condition » 3 essais maximum



Fragments, Interruption

Avec condition : exécuté lorsque la condition est VRAIE.
Le reste du fragment d'interaction contenant (ex: loop) est ignoré



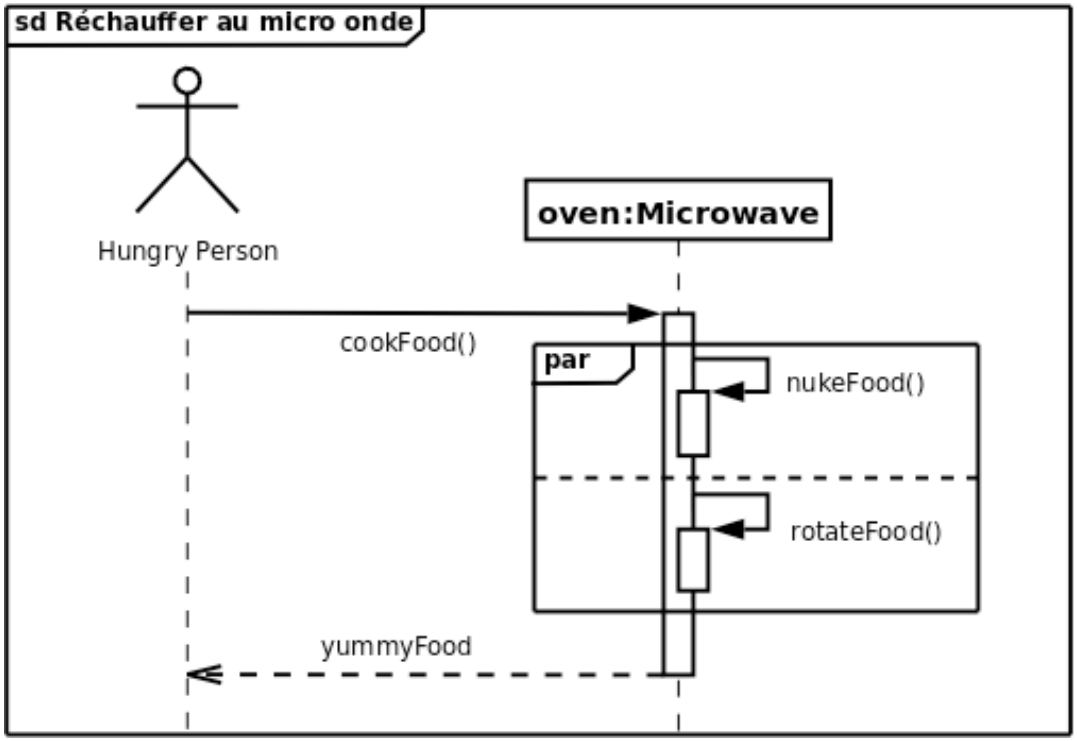
Fragments d’interaction combinés

Regroupement des opérateurs d'interaction par fonctions :

- choix et boucle :
  - alt, option, break, loop
- **contrôle de l'envoi de messages en parallèle** :
  - **parallel**, critical region
- contrôle de l'envoi de messages :
  - ignore, consider, assert, negative
- ordre d'envoi des messages :
  - weak sequencing, strict sequencing
- référencement
  - ref

Fragments, parallélisation

Envoi de messages en parallèle. Les opérandes du fragment se déroulent en parallèle



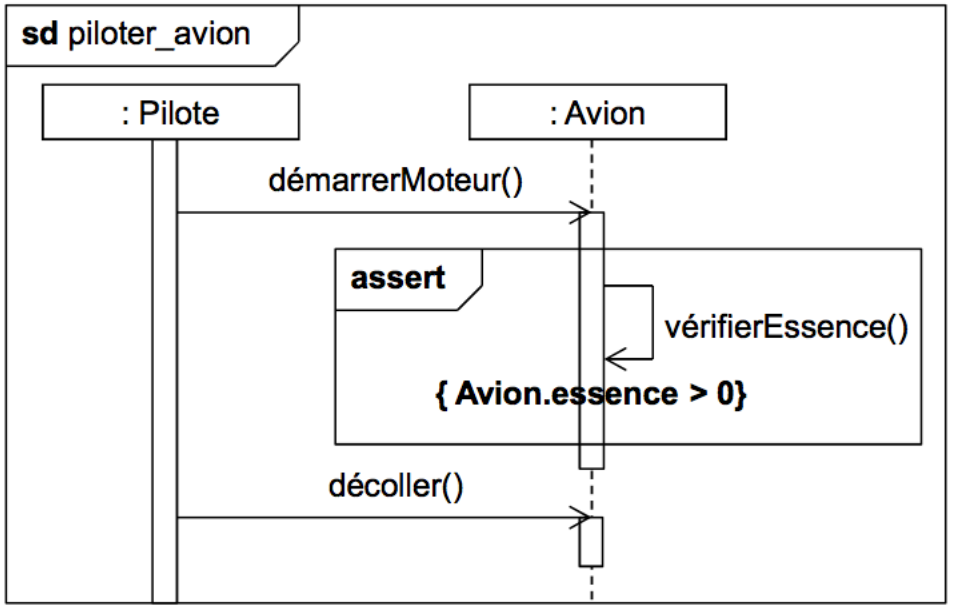
Fragments d’interaction combinés

Regroupement des opérateurs d'interaction par fonctions :

- choix et boucle :
  - alt, option, break, loop
- contrôle de l'envoi de messages en parallèle :
  - parallel, critical region
- **contrôle de l'envoi de messages :**
  - ignore, consider, **assert**, negative
- ordre d'envoi des messages :
  - weak sequencing, strict sequencing
- référencement
  - ref

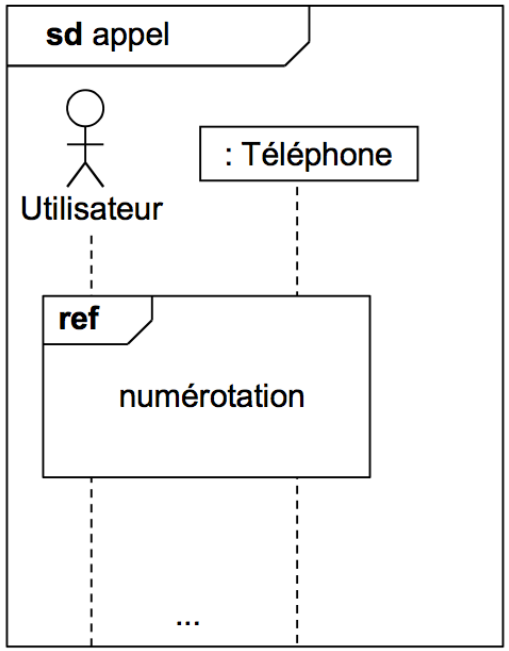
Contraintes sur les lignes de vie, validité des messages

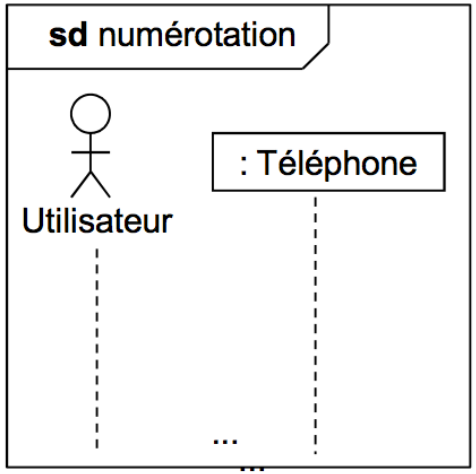
- Une contrainte est indiquée sur une ligne de vie par **{un texte entre accolades}**
- L'opérateur **d'assertion** rend indispensable l'envoi d'un message. Si la condition de la contrainte n'est pas vérifiée, les occurrences des évènements qui suivent sont considérés comme invalides. `
- Exemple :
  si la quantité d'essence est nulle, décoller() devient un message invalide



Fragments, Référencement / Réutilisation

Appel à une interaction décrite dans un autre diagramme de séquence existant



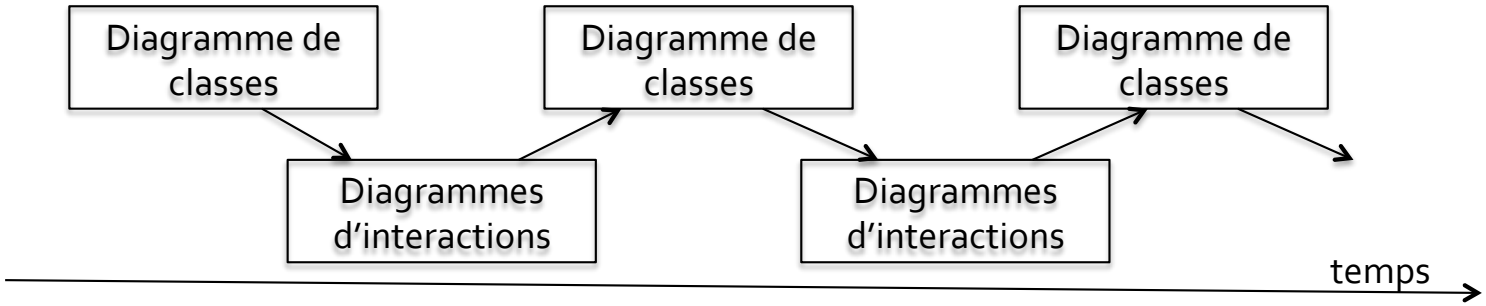


Exercice

Reprendre l’achat d’un ticket sur une borne TCL proprement

Co-conception des classes et des interactions

- Les objets utilisés dans les interactions pour réaliser les scénarios proviennent
  - des classes déjà décrites dans le diagramme de classes
  - des besoins nouveaux en objets pour l'interaction spécifique
- A partir des diagrammes d'interaction, on complète le diagramme de classes
  - précisions (attribut, méthodes)
  - nouvelles classes
  - etc.
- On essaye de réaliser tous les scénarios en convergeant vers un diagramme de classes stables



Plan du cours

- **Diagramme de séquence :** représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou les acteurs
- **Diagramme d'états−transitions :** description, sous forme de machine à états finis, du comportement d'un système ou de l'un de ses composants (ex: une classe)

Diagramme d’état transition / Machines à état

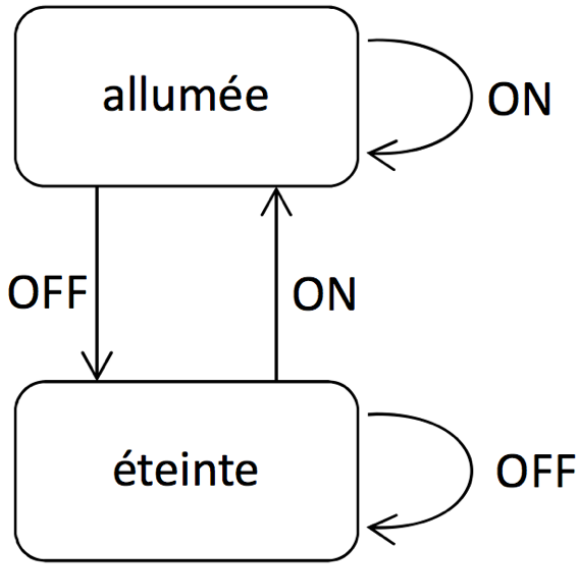
- Description, sous forme d'automates à états finis, du comportement d'un des composants du système (ex: un objet) ou, plus rarement, d'un sous-système
- Automates à états finis = graphes d'états, reliés par des arcs orientés qui décrivent les transitions.
- Représentation des changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs.
- Un état se caractérise par sa durée et sa stabilité, il représente une conjonction instantanée des valeurs des attributs d'un objet.

Diagrammes de machines d’états

- Abstraction des comportements possibles pour une classe
  - automate à états finis décrivant les chemins possibles dans le cycle de vie d’un objet
- Etat d’un objet
  - situation stable d’un objet
  - d'une certaine durée
  - associée à un nom
- Transition entre états
  - réponse de l’objet dans un certain état à l’occurrence d’un événement
  - passage d’un état à un autre sur événement + condition respectée,
  - action à exécuter

Exemple de machine à état

Une lampe avec deux boutons-poussoirs ON et OFF.

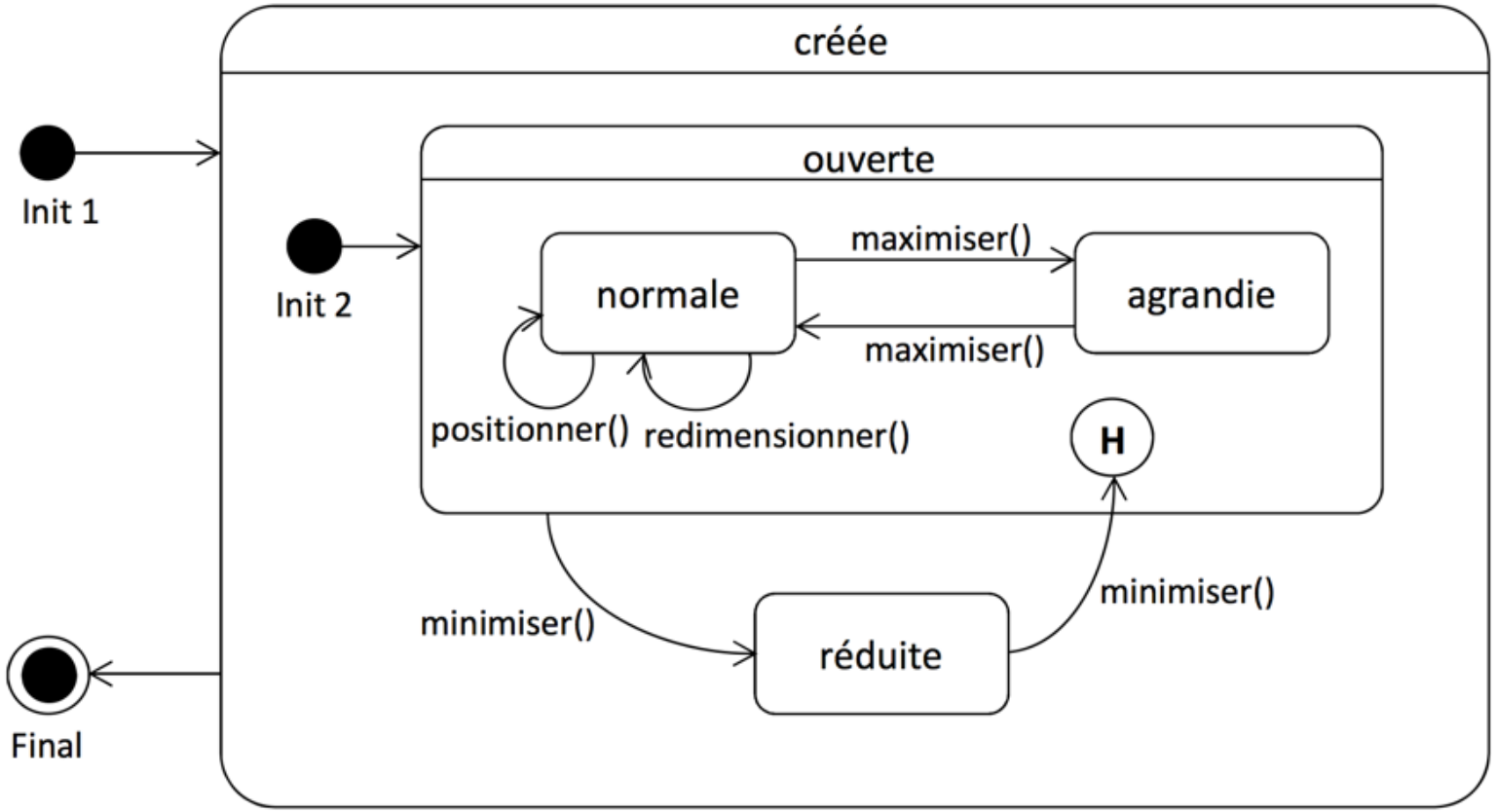


Exemple avancé : fenêtre d'ordinateur (1/2)

Une fenêtre d'application peut être dans trois états :

- Normale, Agrandie ou Réduite
- A l'ouverture, elle est créée dans son état normal.
- Elle peut alors être déplacée et re-dimensionnée.
- Lorsqu'elle est agrandie, elle occupe toute la surface de l'écran.
- Lorsqu'elle est réduite, elle est représentée par une icône dans la barre des tâches.
- A l'état agrandie ou réduite, elle ne peut être ni déplacée ni re-dimensionnée

Exemple avancé : fenêtre d'ordinateur (2/2)



État : définition

- Caractérisé par sa durée et sa stabilité = période dans la vie d'un objet
- Dans un état, un objet accomplit une activité ou attend un évènement
- Représente une conjonction instantanée des valeurs des attributs d'un objet
- Seuls certains états caractéristiques du domaine sont étudiés : soit disjoints, soit imbriqués (pas de recouvrement partiel)
- Exemple : pour un employé d'une entreprise, les états intéressants peuvent être recruté, en activité, en congé, malade

État : représentation

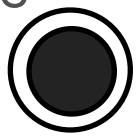
Représentation (état intermédiaire) :

Etat initial

- Obligatoire
- Tout objet est créé dans cet état
- Notation :

Etat final

- Facultatif
- Passage obligé pour tout objet à détruire
- Notation :



Transition

- Représente le passage instantané d'un état (source) vers

- Déclenchée par un événement : *C'est l'arrivée d'un événement qui conditionne la transition*.
- Quand il n'y a pas d'évènements qui la déclenche, la transition est automatique.
- On peut conditionner une transition à l'aide de **gardes** : conditions booléennes entre [ ]
- Syntaxe de déclaration
  `NomEvènement '(' [ListeParamètres] ')' [ '[' garde ']' '/' Action ]`

Évènement

Définition :

- Déclencheur d'une transition

Type d’évènements

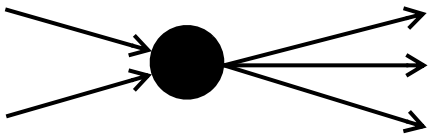
- **Call :** appel de méthode sur l'objet courant (méthode déclarée dans le diagramme de classe)
- **Change :** condition booléenne passe de FAUX à VRAI
- **Signal :** réception d'un signal asynchrone, explicitement émis par un autre objet (classe avec stéréotype <<signal>> déclarée dans le diagramme de classe, sans opération et dont les attributs sont interprétés comme arguments)
- **After :** écoulement d'une durée déterminée après un évènement donné (par défaut, le temps commence à s'écouler dès l'entrée dans l'état courant)
- **Completion event :** fin d'une activité liée à un état, de type do/ (déclenchement d'une transition "automatique", sans évènement déclencheur explicite)

Syntaxe évènements

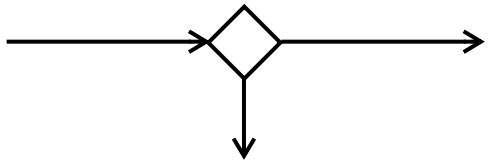
- Déclaration d'un évènement de type call ou signal :
  `NomEvènement '(' [ NomParamètre1, NomParamètre2, ... ] ')'`
  où chaque paramètre est de la forme :
  `NomParamètre ':' TypeParamètre`
- Déclaration d'un évènement de type change
  `when '(' ConditionBooléenne ')'`
- Déclaration d'un évènement de type after
  `after '(' Paramètre ')'`
  où le paramètre s'évalue comme une durée a priori écoulée depuis l'entrée dans l'état courant, ex: after(10 secondes)

Points de décision

- Possibilité de représenter des alternatives pour le franchissement d'une transition
- Utilisation de deux pseudo-états particuliers :
  - point de jonction



  - point de choix

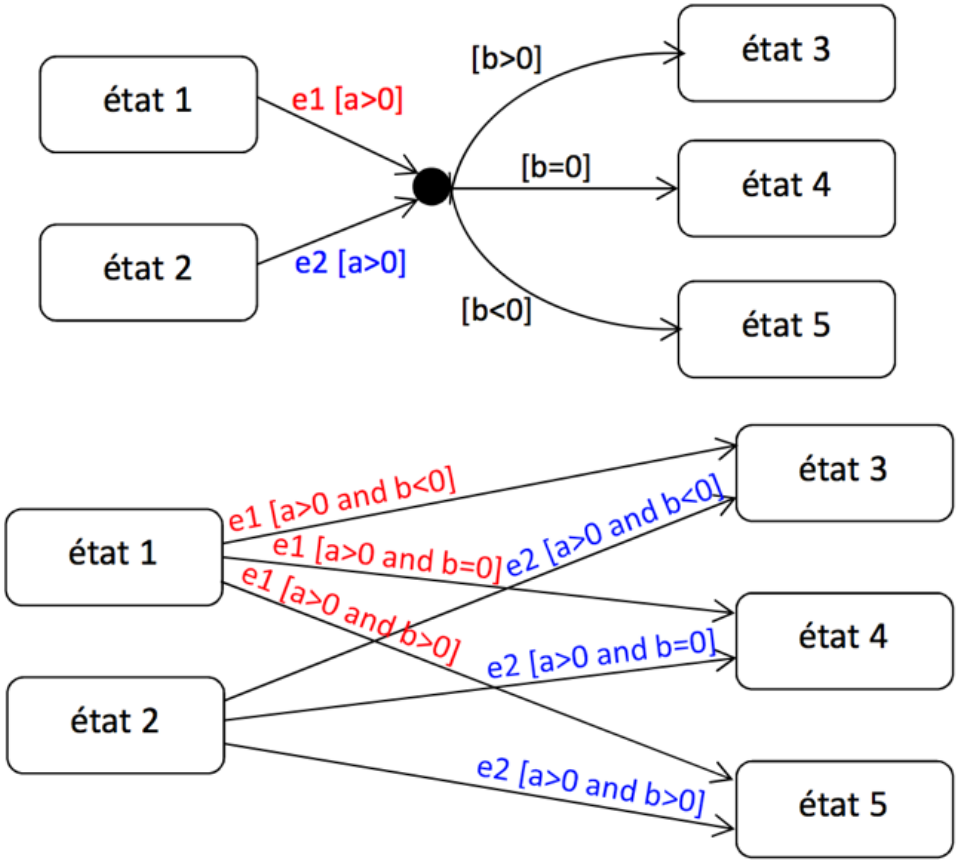


Point de jonction

- Permet de partager des segments de transition (notation plus compacte, amélioration de la visibilité des chemins alternatifs)
- Plusieurs transitions peuvent viser et/ou quitter un point de jonction
- Tous les chemins (suites de segments) à travers le point de jonction sont potentiellement valides (on peut donc représenter un comportement équivalent en créant une transition pour chaque paire de segment avant et après le point de jonction)
- Pour pouvoir emprunter un chemin, toutes les gardes le long de ce chemin doivent s'évaluer à VRAI **dès le franchissement du premier segment**

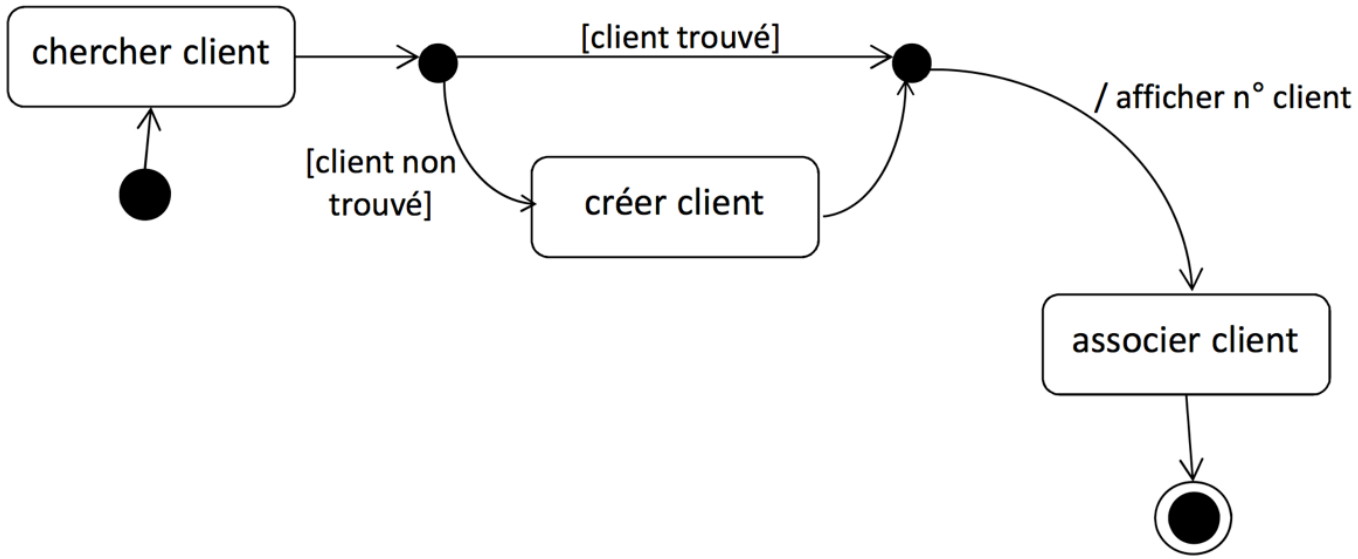
Points de jonction

- Deux représentations équivalentes :



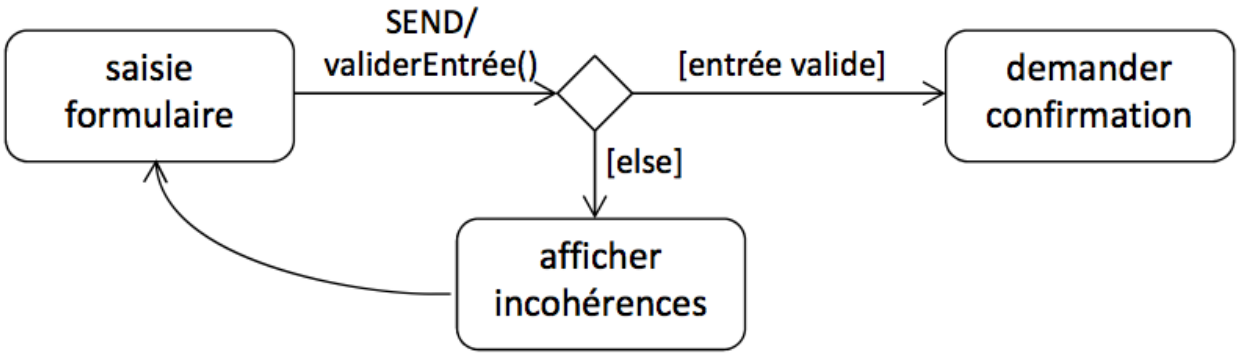
Points de jonction

- Bien adapté pour représenter des chemins optionnels
  `if (...) {...}`
- Utilisation de deux points de jonction :



Point de choix

- Contrairement au point de jonction, le point de choix est "dynamique" : les gardes après le point de choix sont évaluées au moment où il est atteint
- Exemple :
  - *un formulaire est rempli en ligne par un utilisateur. Quand il valide son formulaire en appuyant sur le bouton SEND, une vérification de la cohérence des données fournies est réalisée par validerEntrée(). Si les données sont cohérentes, on lui demande de valider. Sinon, on affiche les incohérences et lui demande de remplir à nouveau le formulaire.*



Activités internes à un état

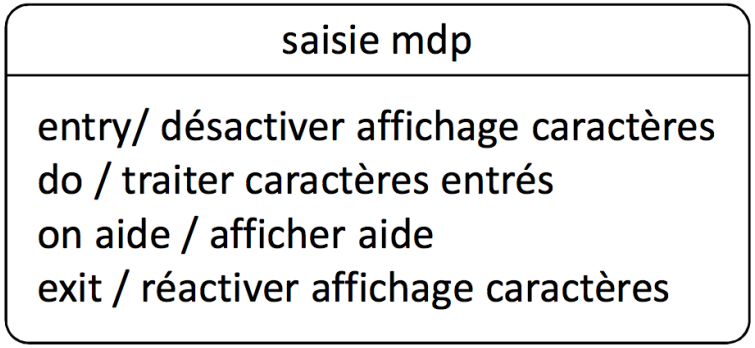
Un état peut être séparé en deux compartiments

- nom de l'état
- activités internes

Une transition interne ne modifie pas l'état courant, mais suit globalement les règles d'une transition simple entre deux états

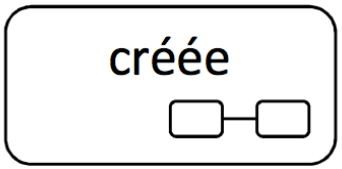
État, déclencheurs prédéfinis

- entry : activité à effectuer à chaque fois que l'objet rentre dans l'état
- do : activité réalisée tant que l'objet est dans l'état courant, pouvant être interrompue
- on evt-ext : activité interne instantanée, l'objet reste dans l'état courant
- exit : activité à effectuer à chaque fois que l'objet quitte l'état



Etat composite

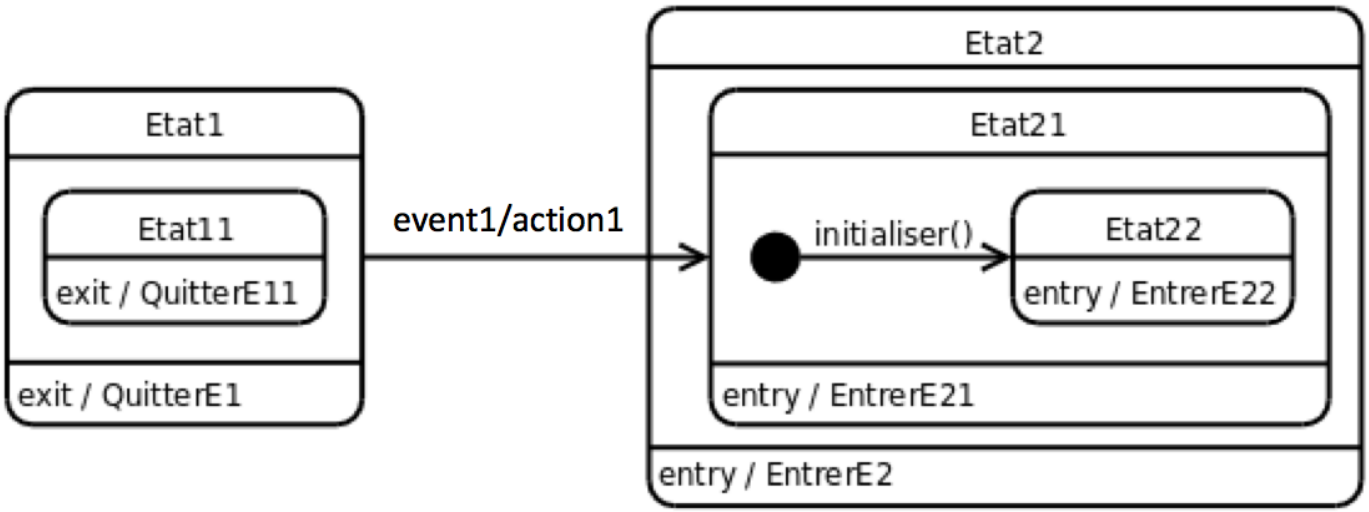
- Tout état peut être décomposé en sous-états enchaînés sans limite a priori de profondeur
- Un état composite (EC) peut posséder trois compartiments : nom de l'état, sous-diagramme et liste des activités internes
- L'utilisation d'états composites permet de développer une spécification par raffinements
- Pour ne pas avoir à représenter les sous-états à chaque utilisation de l'état composite, emploi d'une notation abrégée :



avec l'état "créée" développé dans un autre diagramme

Exemple

Depuis l'état Etat11, la réception de l'évènement event1 provoque la séquence d'activités QuitterE11, QuitterE1, action1, EntreeE2, Entree21, initialiser, EntreeE22 et place le système dans l'état Etat22



Etat composite et concurrence

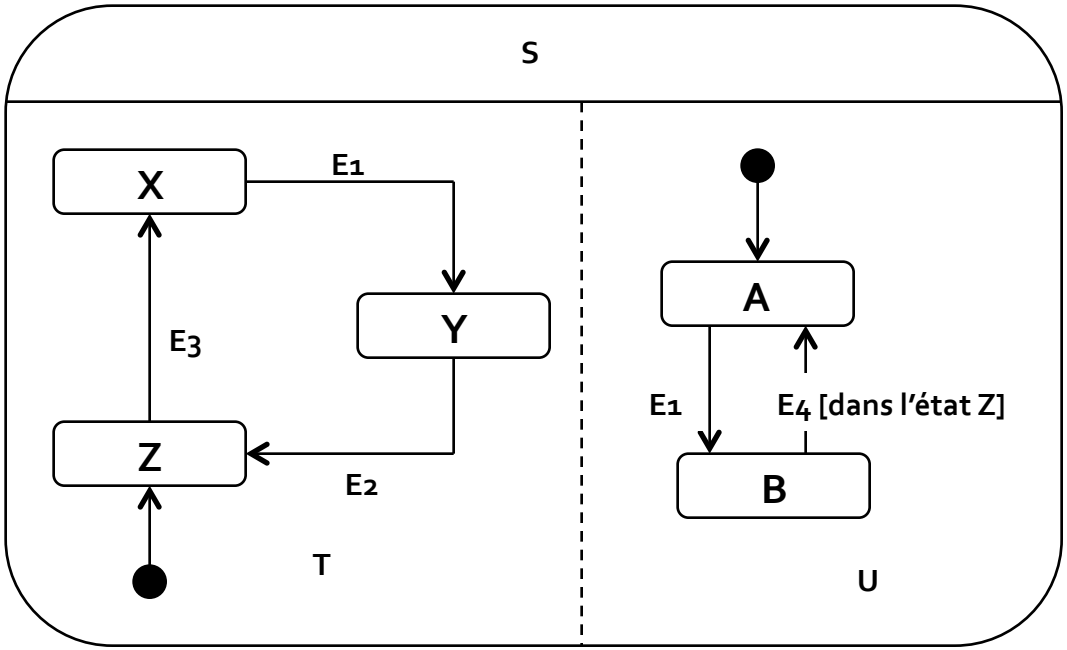
Un état composite peut comporter plus d'une région

- L'état est alors dit "orthogonal"
- Régions séparées par une ligne pointillée
- Chaque région représente un flot d’exécution
- Les régions au sein de l'état orthogonal sont dites "concurrentes", car elles sont exécutées en parallèle
- Le nombre de sous-états peut être différent selon les régions
- Toutes les régions concurrentes d’un état composite orthogonal doivent atteindre leur état final pour que l’état composite soit considéré comme terminé.

États concurrents

Pour décomposer des états complexes

Exercice : trouver le diagramme d'état « à plat » équivalent



Méthodes d’utilisations des machines a état

Méthode itérative et incrémentale (Larman)

- commencer par le scénario nominal
- ajouter les exceptions
- factoriser dès que ça devient trop compliqué.

Utilisation des états

Pour se concentrer sur le fonctionnement d'une classe

- décrire / fixer le comportement concret de la vie d'une objet lié à un ou plusieurs scénarios

Pour les classes complexes

- objets réactifs complexes (objets métier…)
- protocole et séquences légales (sessions…)
- en général pas plus de 10% des classes d'une application
  - plus en télécommunication / moins en informatique de gestion

Larman

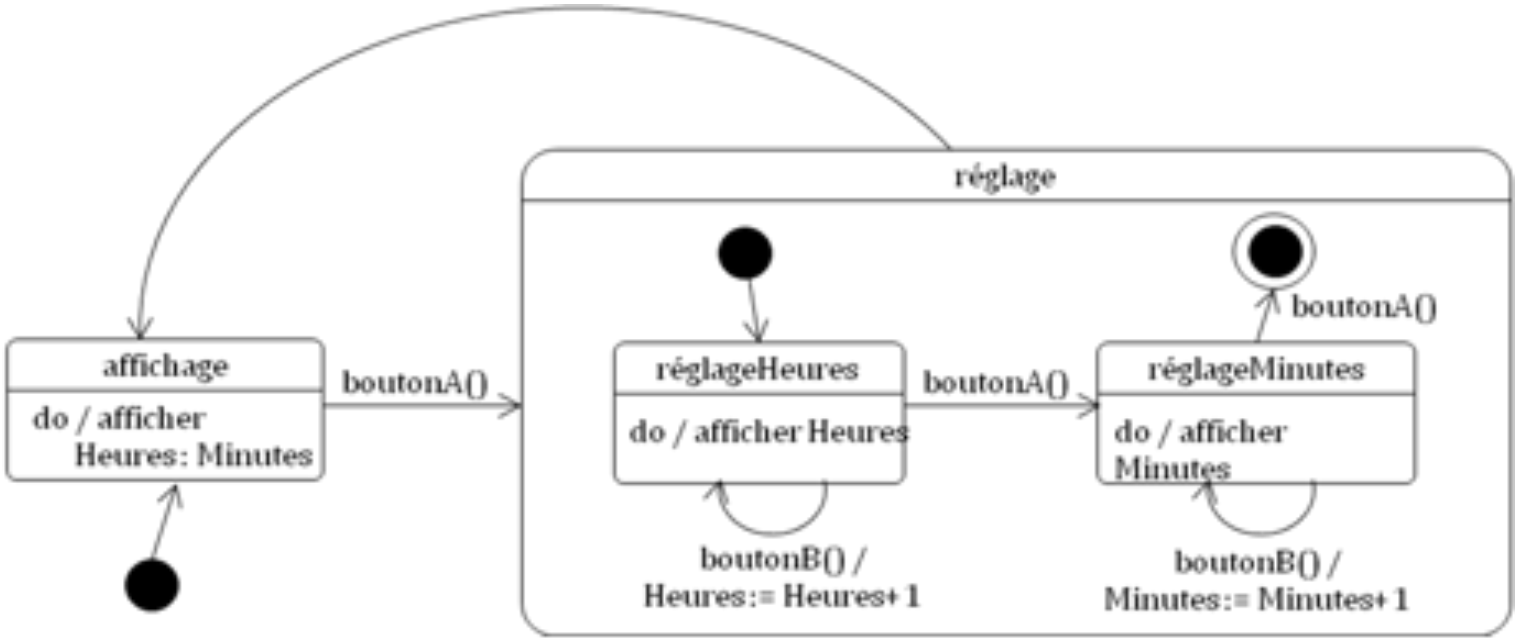
- navigation dans un site web, Interfaces utilisateurs
  - enchaînement de pages/fenêtres

Exercice

Une montre digitale possède un cadran et deux boutons A et B pour la mettre à l'heure.
La montre a deux modes : l'affichage et le réglage de l'heure. En mode affichage, les heures et les minutes sont affichées. Le mode de réglage a deux sous-modes : réglage des heures, et réglage des minutes. Le bouton A s'utilise pour changer de mode. A chaque fois que l'on appuie dessus, le mode change suivant la séquence (affichage, réglage heures, réglage minutes). Dans un sous-mode de réglage, le bouton B s'utilise pour avancer les heures ou les minutes à chaque pression.

On suppose que, dans le diagramme de classe correspondant, la classe Montre a, entre autres, deux attributs Integer `heures` et `minutes`.

Construisez le diagramme d'états-transitions décrivant le fonctionnement de la montre.


réglage
affichage
do / afficher
Heures: Minutes
boutonA()
réglageHeures
do / afficher Heures
boutonA()
réglageMinutes
do / afficher
Minutes
boutonA()
boutonB() /
Heures:= Heures+ 1
boutonB() /
Minutes:= Minutes+ 1

Conclusion sur UML

Propriétés

- Unification de concepts de modélisation
- Formalisme puissant et détaillé
- Standard très répandu

Limites

- UML reste un langage
  - Pas une méthode de modélisation
  - N'indique pas comment construire les modèles
- La totalité du SI n'est pas nécessairement bien représentée par des diagrammes
  - Besoin de conserver des descriptions textuelles