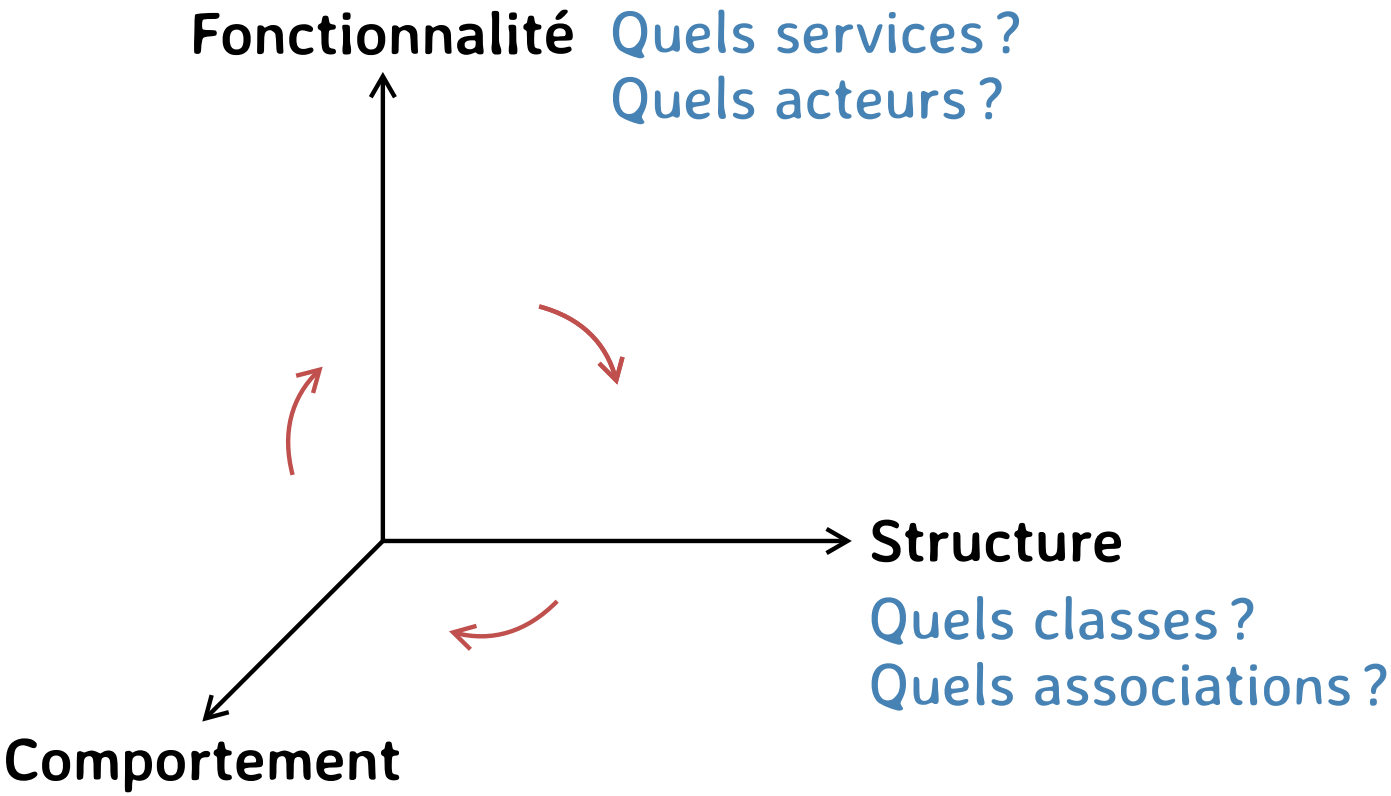
UML - La vue dynamique

Ingéniérie des systèmes d'information

Paul Brunet

L'approche UML



Les diagrammes d'interaction permettent de faire la liaison entre les vues fonctionnelle et statique : ils montrent comment les objets du système interagissent pour réaliser une exigence fonctionnelle.

Le point de vue dynamique

- Modéliser la **vue comportementale** d'un système :
  - Comment les objets de l'application interagissent-ils ?
- Il s'agit de représenter un système pendant son exécution.
  - On raisonne sur les **objets**.
  - Les objets qui composent une application **pendant son exécution** et leurs **échanges de messages** permettent à l'application de réaliser les fonctionnalités (i.e. cas d'utilisation) pour lesquelles elle est développée.

Diagrammes dynamiques

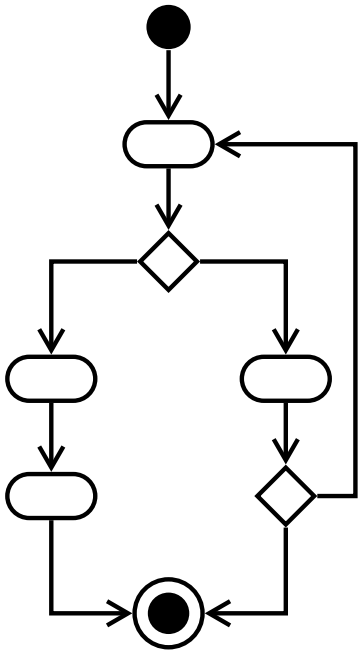


Diagramme d'états/transitions

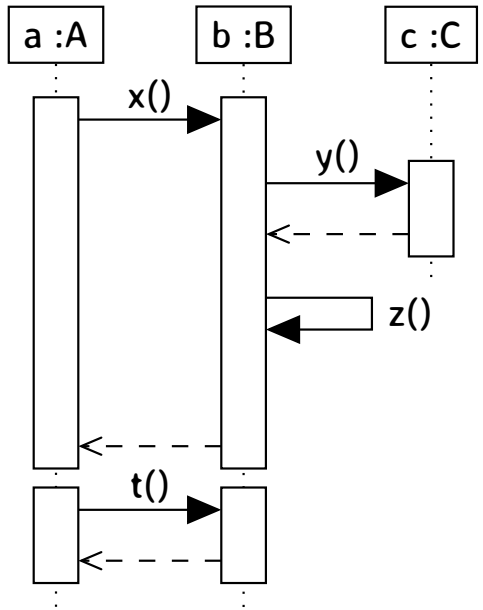


Diagramme de séquence

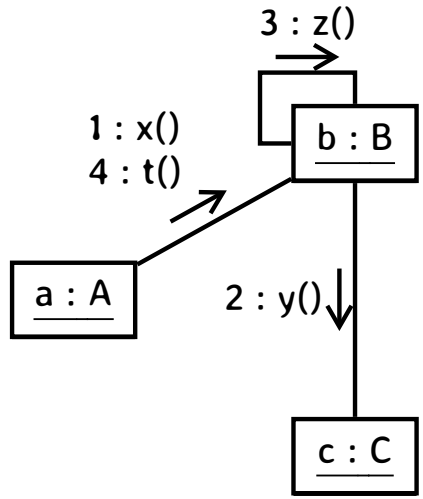


Diagramme de collaboration

UML - La vue dynamique
Plan

Diagramme d'états-transitions
Principes

- Formalisme des automates à état finis « Statecharts ». (David Harel 1987)
- Un automate est un comportement qui spécifie la séquence des états d'un objet durant son cycle de vie, en réponse à des événements ainsi que les réactions à ces événements
- Principaux concepts : États et transitions
- Différents niveaux d'abstraction

UML - La vue dynamique
Plan

Diagramme d'états-transitions
Définition

définition

Un diagramme d'états-transitions décrit tous les comportements possibles des objets d'une classe. Il est associé à une classe.

exemple : **Une lampe avec 2 boutons On et Off**

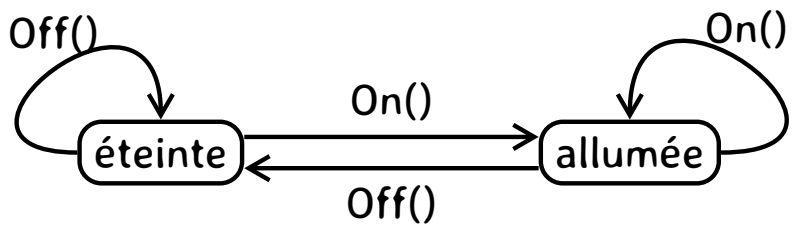


Diagramme d'états-transitions
Définition

définition

Un diagramme d'états-transitions décrit tous les comportements possibles des objets d'une classe. Il est associé à une classe.

exemple : Une lampe avec 2 boutons On et Off

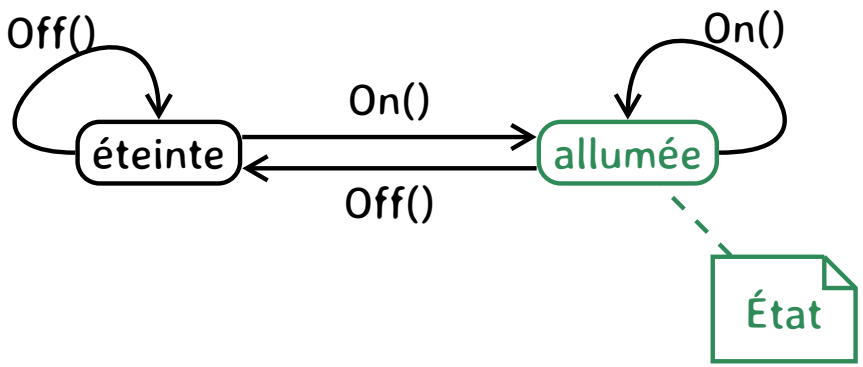
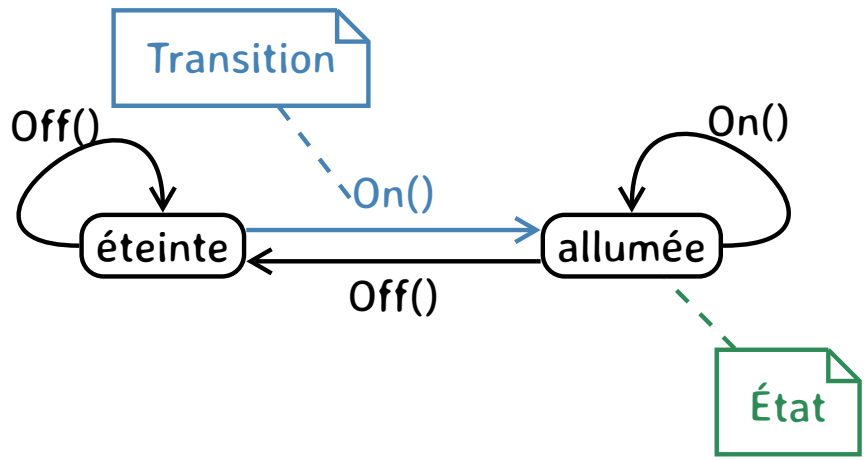


Diagramme d'états-transitions
Définition

définition

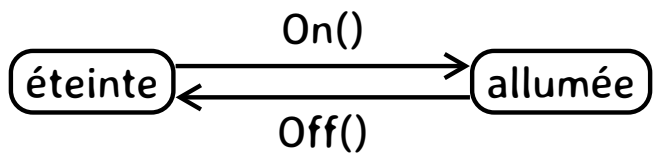
Un diagramme d'états-transitions décrit tous les comportements possibles des objets d'une classe. Il est associé à une classe.

exemple : Une lampe avec 2 boutons On et Off



États, transitions et évènements
Concept : état

- Un **état** est une situation dans la vie d'un objet durant laquelle il vérifie une certaine condition, effectue une activité ou attend un événement.
- Chaque état possède un nom qui l'identifie.
- Chaque objet est à un moment donné dans un seul état.
- Un objet passe par une succession d'états durant son existence
- L'état d'un objet a une **durée finie**, il est valide dans un espace de temps (généralement séparé par deux événements).

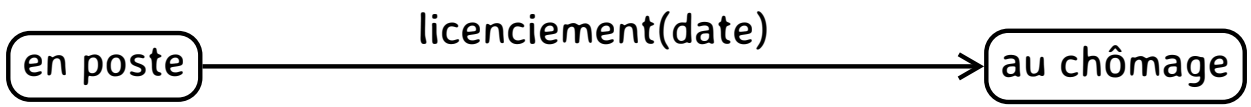


États, transitions et évènements
Concept : transition

- Une transition est définie par un état source, un événement déclencheur, et un état cible.
- De manière optionnelle, on peut y ajouter une condition de garde et/ou une action.

syntaxe

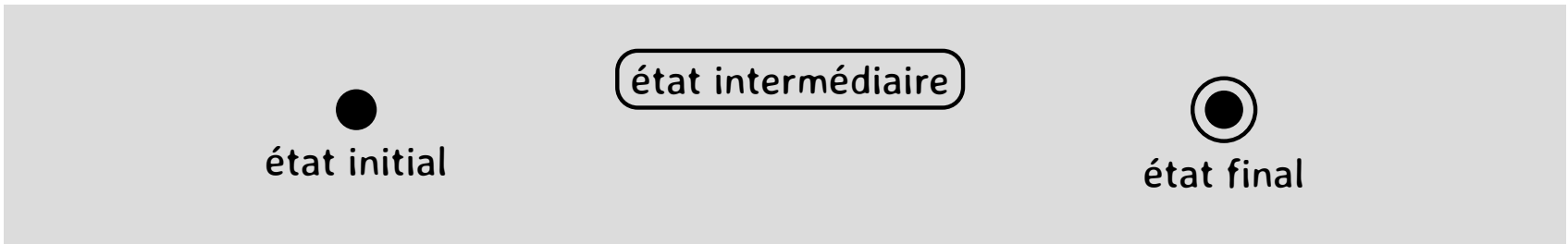
- Quand un événement survient, une transition peut être déclenchée ; elle fait passer l'objet dans un nouvel état.
- La liste des attributs correspond à des informations ou à des paramètres portés par les événements.



États, transitions et évènements
Concepts : pseudo-états initiaux et finaux

- Les automates retenus par UML sont déterministes. Donc :
  - Il y a toujours un et un seul état initial.
  - En revanche, il est possible d'avoir plusieurs états finaux.
  - Ils correspondent chacun à une condition de fin différente.
- Il est également possible de n'avoir aucun état final, dans le cas par exemple d'un système qui « ne s'arrête jamais » (système dit **réactif**).

syntaxe



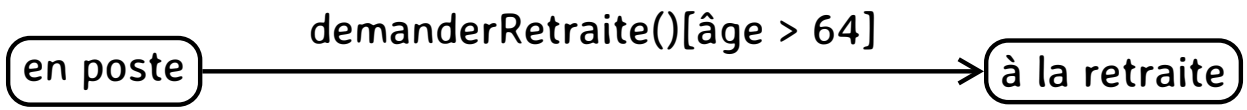
États, transitions et évènements
Concept : évènement

- Un événement est un fait qui survient à un moment donné
- Un événement n'a pas de durée contrairement à un état
- Les types d'événements :

Signal	Description	Évènement
Signal	Réception d'un signal	collision, frappe sur un clavier
Call	Appel d'une méthode	valider(dateDépartPrévue)
After	Évènement temporel	after(2 secondes)
Change	Évènement de changement : se produit quand une condition devient vraie	when(isEmpty(todoList))

États, transitions et évènements
Concept : condition de garde

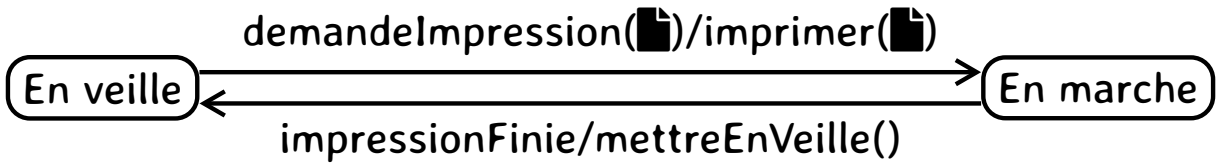
- Une condition de garde est une condition booléenne qui valide ou non le déclenchement d'une transition lors de l'occurrence d'un événement.
- La condition porte sur les attributs de l'objet ainsi que les paramètres de l'événement déclencheur.
- La condition de garde est évaluée uniquement **après** l'occurrence de l'événement.



États, transitions et évènements
Concept : action

- Une action est un effet déclenché suite à un événement.
- L'action est considérée comme un traitement exécutable **atomique**, c'est à dire **instantané** et **ininterruptible**.
- L'action correspond à un appel d'opération (sur un autre objet ou sur lui-même).

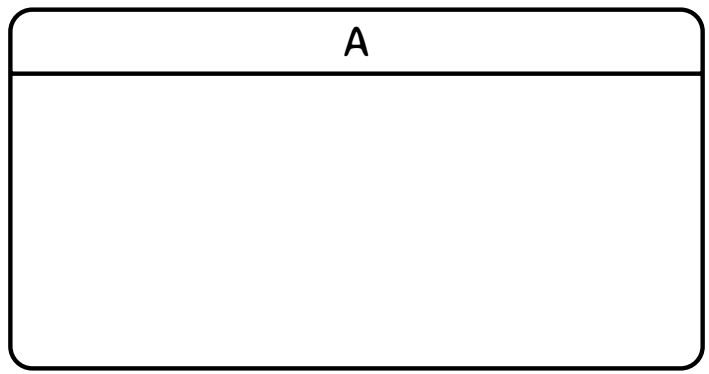
exemple : **Une imprimante**



UML - La vue dynamique
Plan

Actions internes
Principes

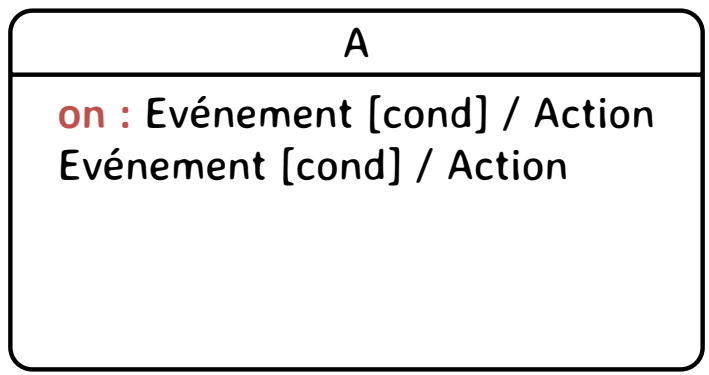
☞ Les états peuvent également contenir des actions ; elles sont exécutées à l'entrée ou à la sortie de l'état ou lors de l'occurrence d'un événement pendant que l'objet est dans l'état.



1. Le mot-clé « on : » peut être omis.

Actions internes
Principes

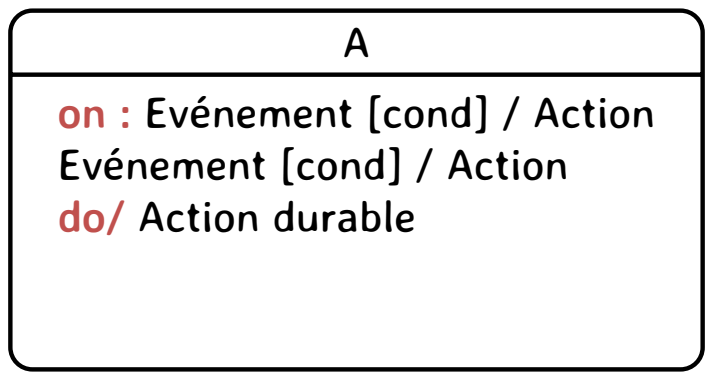
- Les états peuvent également contenir des actions ; elles sont exécutées à l'entrée ou à la sortie de l'état ou lors de l'occurrence d'un événement pendant que l'objet est dans l'état.
- L'action sur événement interne (on :) est exécutée lors de l'occurrence d'un événement qui ne modifie pas l'état. [1]



1. Le mot-clé « on : » peut être omis.

Actions internes
Principes

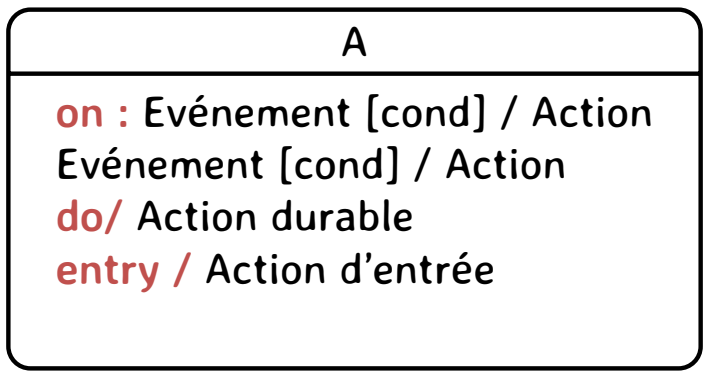
- Les états peuvent également contenir des actions ; elles sont exécutées à l'entrée ou à la sortie de l'état ou lors de l'occurrence d'un événement pendant que l'objet est dans l'état.
- L'action sur événement interne (on :) est exécutée lors de l'occurrence d'un événement qui ne modifie pas l'état. [1]
- L'action durable (do/) est exécutée en continu, tant qu'on reste dans le même état.



1. Le mot-clé « on : » peut être omis.

Actions internes
Principes

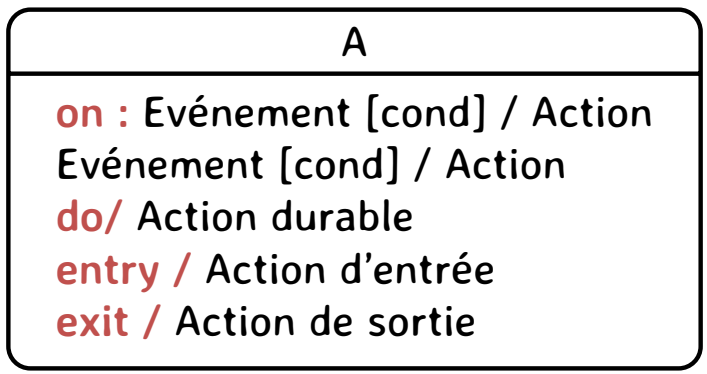
- Les états peuvent également contenir des actions ; elles sont exécutées à l'entrée ou à la sortie de l'état ou lors de l'occurrence d'un événement pendant que l'objet est dans l'état.
- L'action sur événement interne (on :) est exécutée lors de l'occurrence d'un événement qui ne modifie pas l'état. [1]
- L'action durable (do/) est exécutée en continu, tant qu'on reste dans le même état.
- L'action d'entrée (entry/) est exécutée de manière instantanée et atomique dès l'entrée dans l'état.



1. Le mot-clé « on : » peut être omis.

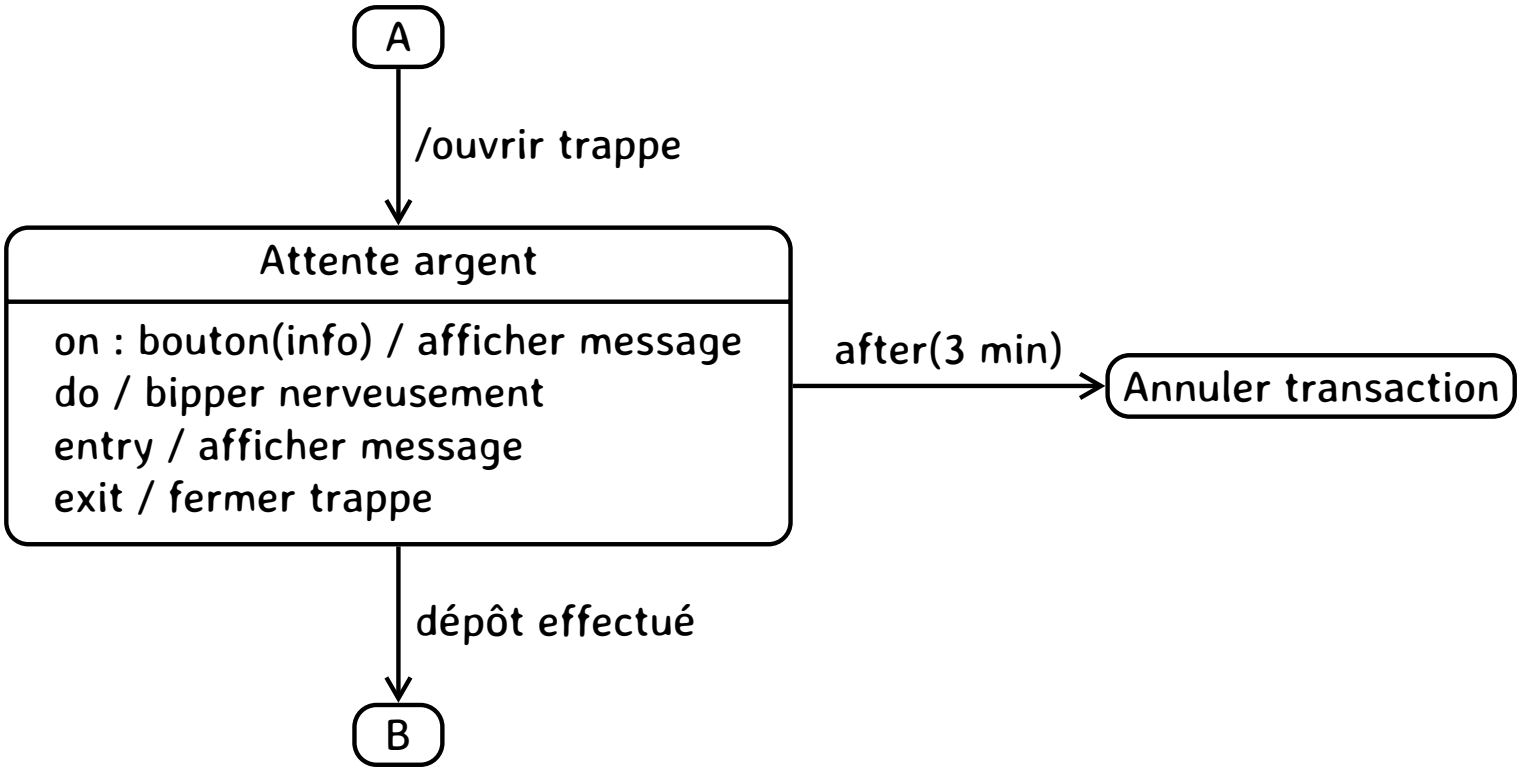
Actions internes
Principes

- Les états peuvent également contenir des actions ; elles sont exécutées à l'entrée ou à la sortie de l'état ou lors de l'occurrence d'un événement pendant que l'objet est dans l'état.
- L'action sur événement interne (on :) est exécutée lors de l'occurrence d'un événement qui ne modifie pas l'état. [1]
- L'action durable (do/) est exécutée en continu, tant qu'on reste dans le même état.
- L'action d'entrée (entry/) est exécutée de manière instantanée et atomique dès l'entrée dans l'état.
- L'action de sortie (exit/) est exécutée à la sortie de l'état.

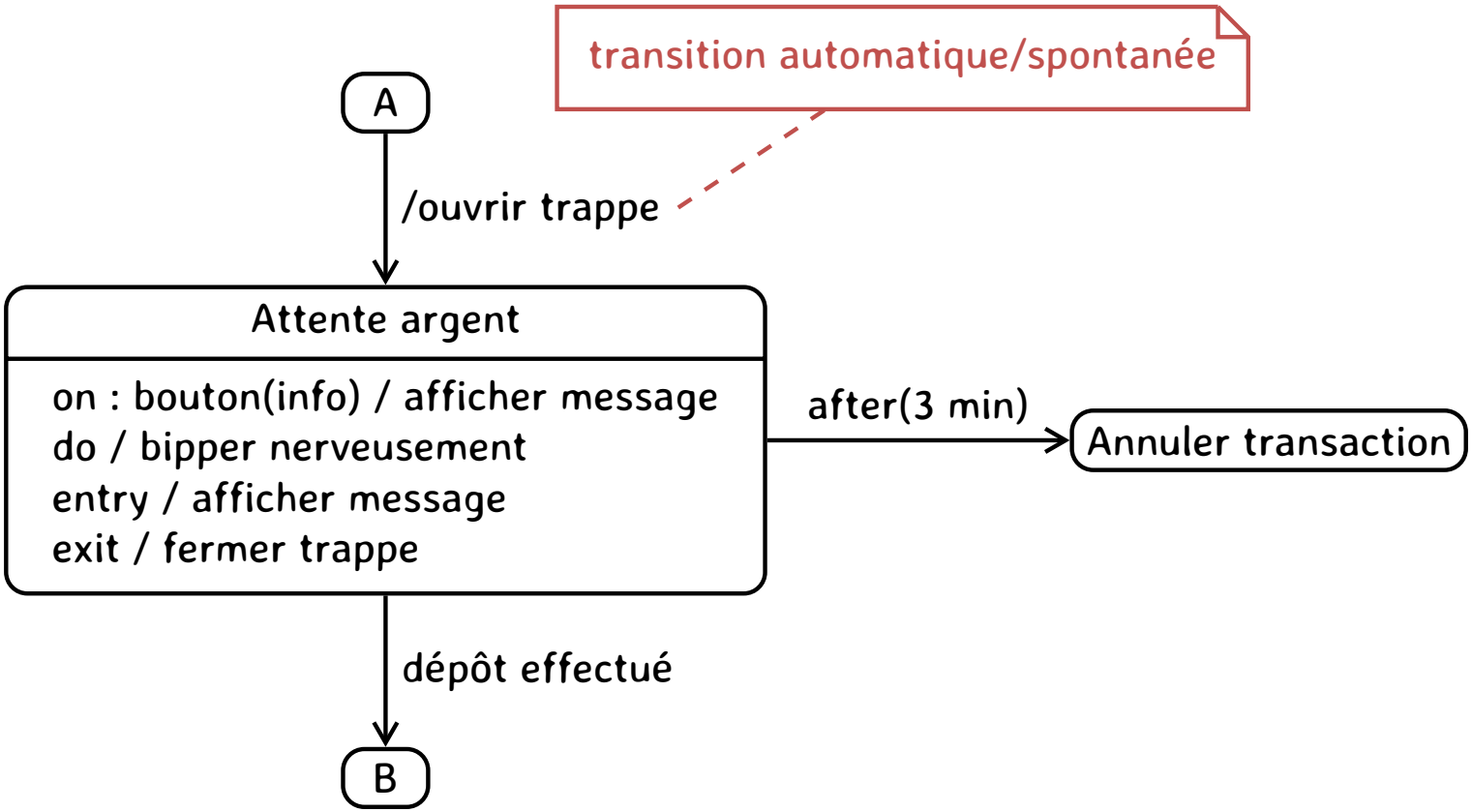


1. Le mot-clé « on : » peut être omis.

Actions internes
Exemple

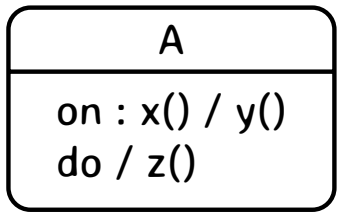


Actions internes
Exemple



Actions internes
Élimination

☞ Les actions sur évènement interne et les actions durables peuvent être remplacées par des boucles [2].



2. Sauf pour les états composites, voir p. 25.

Actions internes
Élimination

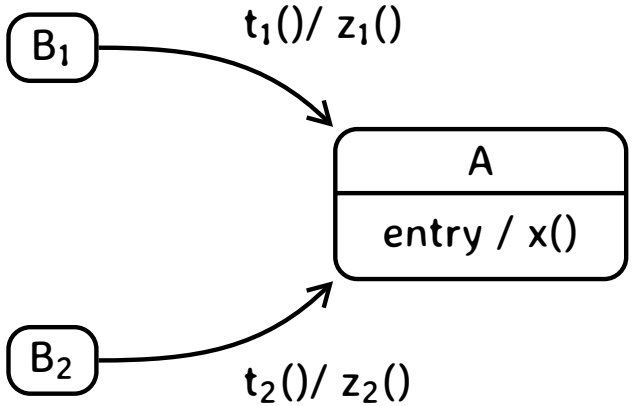
☞ Les actions sur évènement interne et les actions durables peuvent être remplacées par des boucles [2].

2. Sauf pour les états composites, voir p. 25.

Actions internes
Élimination

- Les actions sur évènement interne et les actions durables peuvent être remplacées par des boucles [2].

- Les actions d'entrée peuvent être remplacées en ajoutant un état d'entrée :

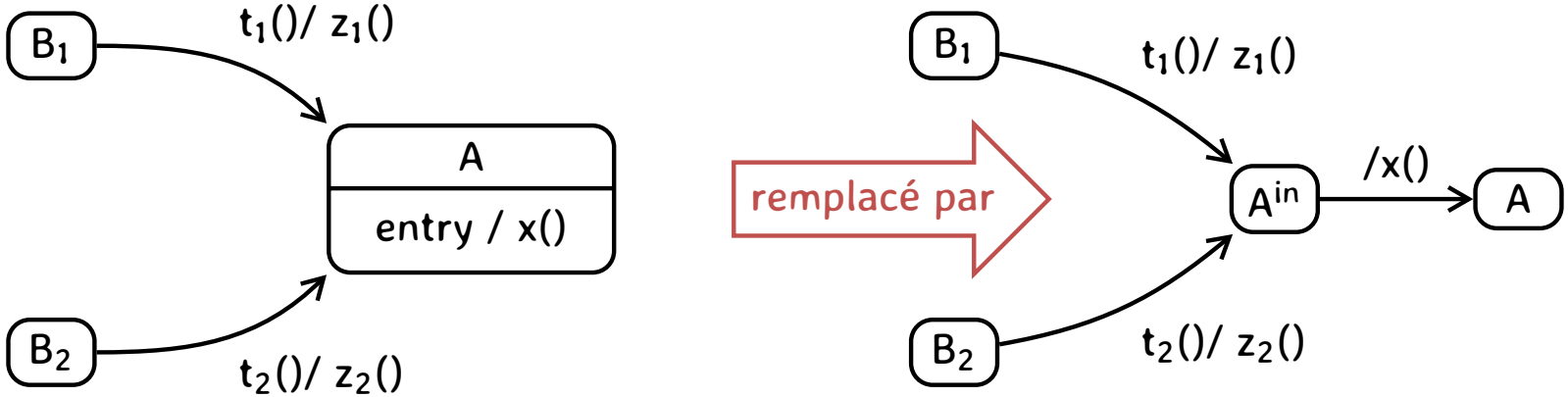


2. Sauf pour les états composites, voir p. 25.

Actions internes
Élimination

- Les actions sur évènement interne et les actions durables peuvent être remplacées par des boucles [2].

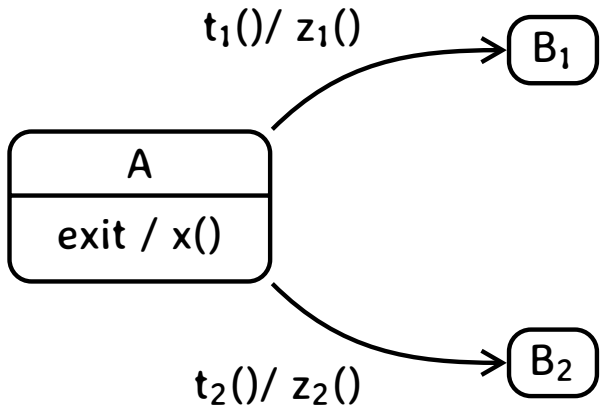
- Les actions d'entrée peuvent être remplacées en ajoutant un état d'entrée :



2. Sauf pour les états composites, voir p. 25.

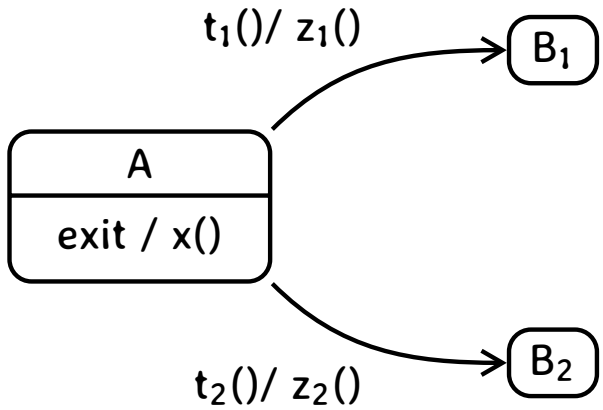
Actions internes
Élimination

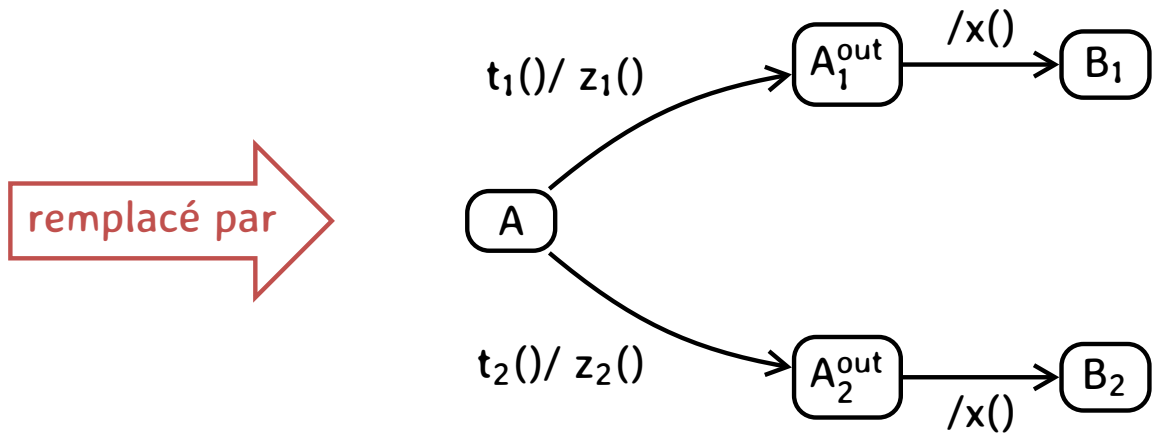
☞ Les actions de sortie peuvent être remplacées en ajoutant des états de sortie :



Actions internes
Élimination

☞ Les actions de sortie peuvent être remplacées en ajoutant des états de sortie :



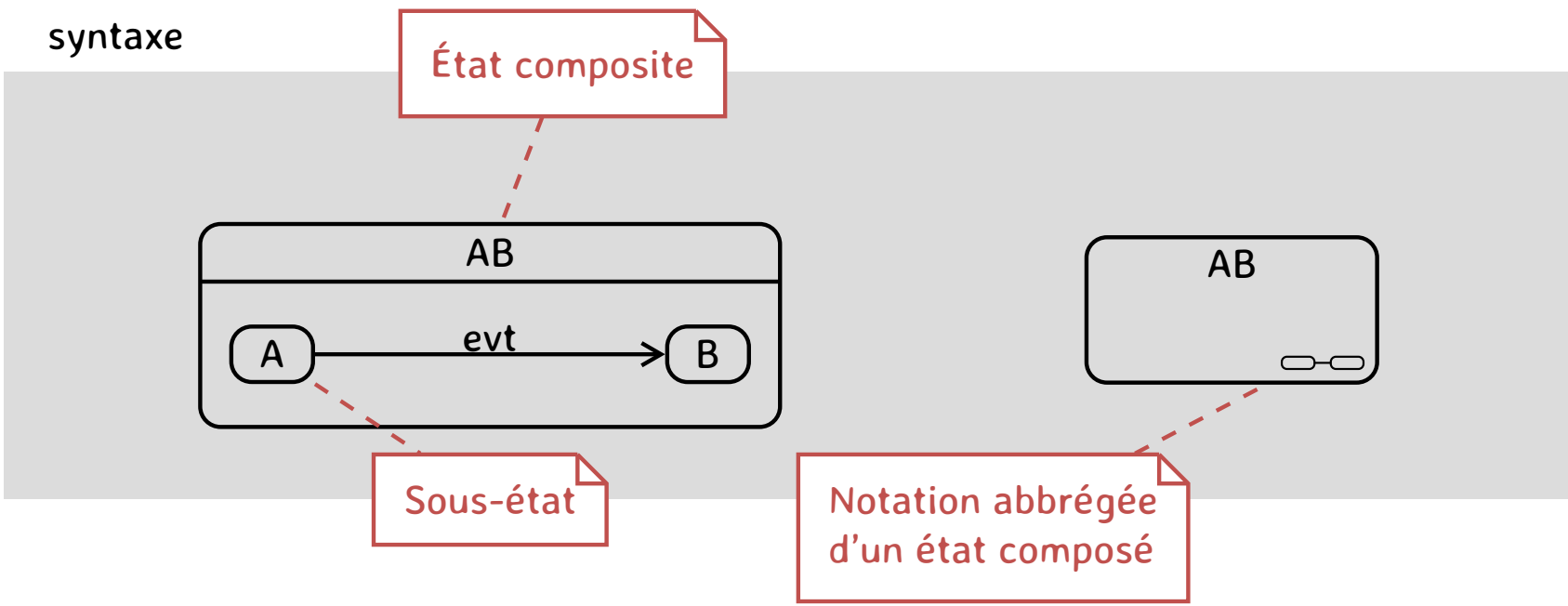


UML - La vue dynamique
Plan

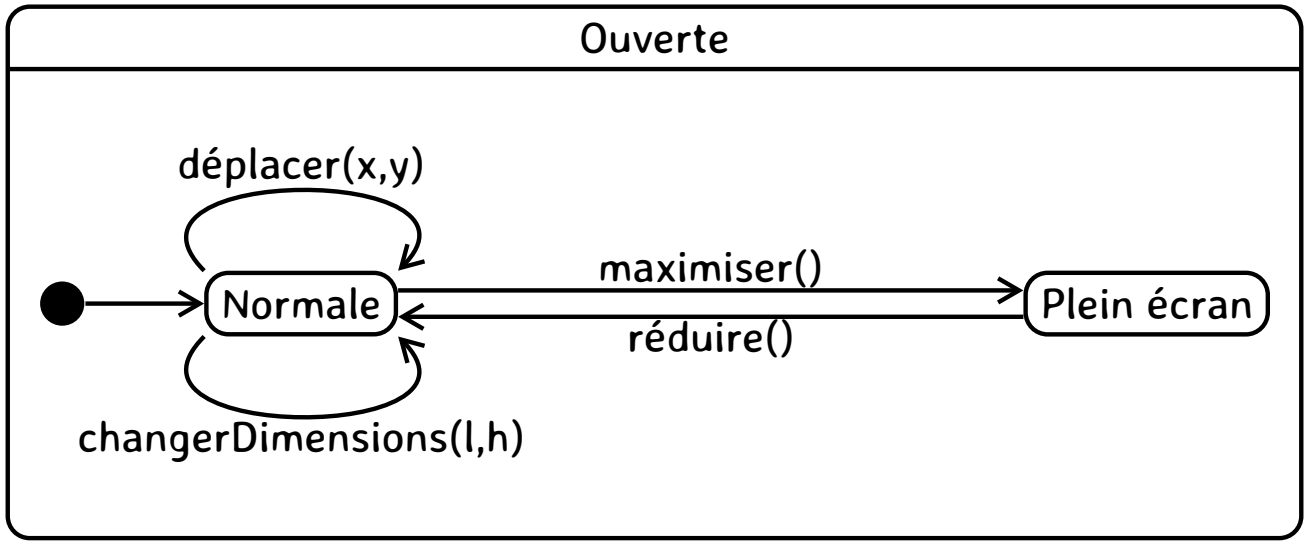
États composites
Principes

- Possibilité de représenter des états complexes
- Décomposition disjonctive : l'objet doit être dans un et un seul sous-état à la fois.
- Les états composites peuvent être utilisés quand certains états partagent des actions internes et des transitions communes.

syntaxe

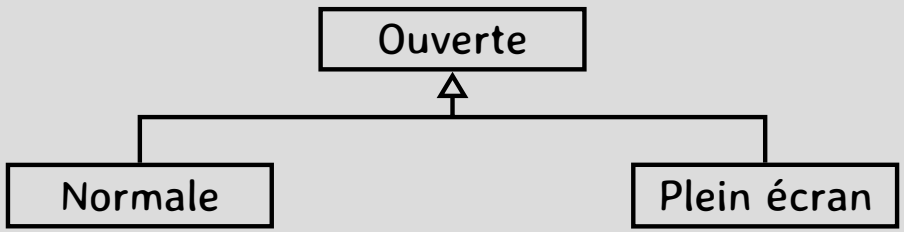


États composites
Exemple : une fenêtre graphique

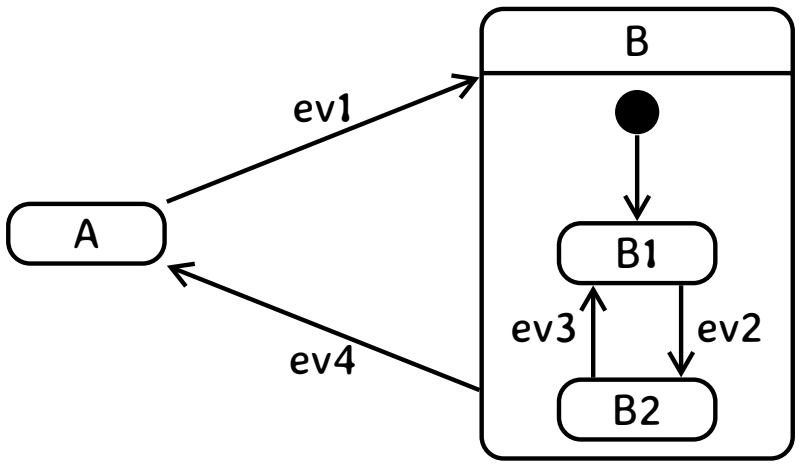


intuition

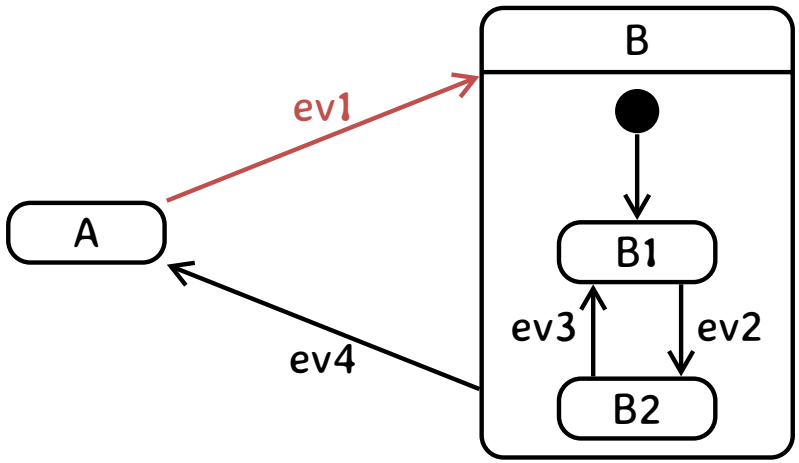
L'état composite est une généralisation de ses sous-états.



États composites
Transitions

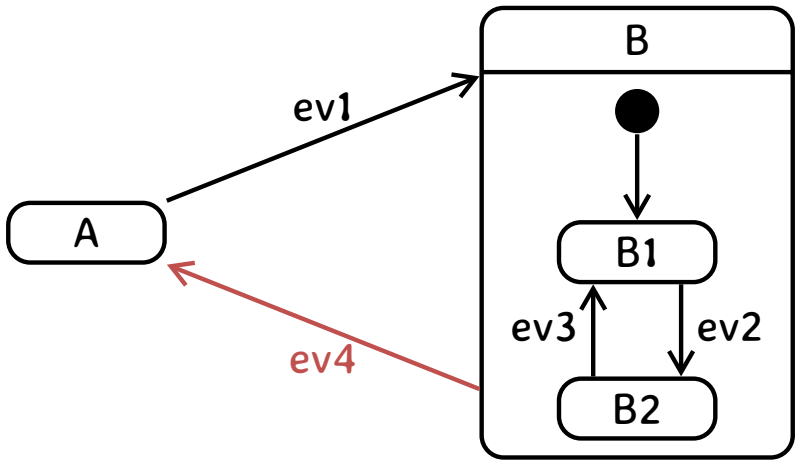


États composites
Transitions



☞ ev1 a pour cible l'état initial de l'état composite B.

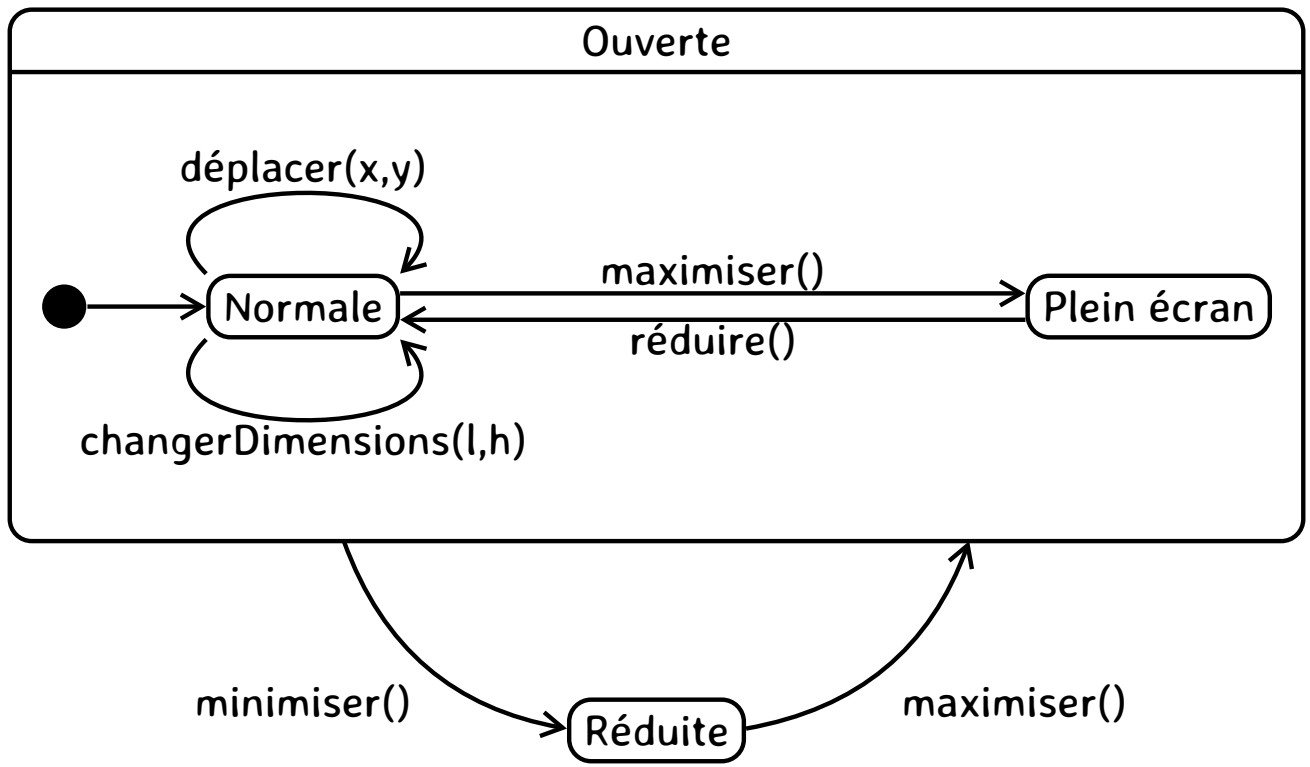
États composites
Transitions



- ev1 a pour cible l'état initial de l'état composite B.
- ev4 s'applique à (peut être activée depuis) tous les sous-états de l'état composite B.

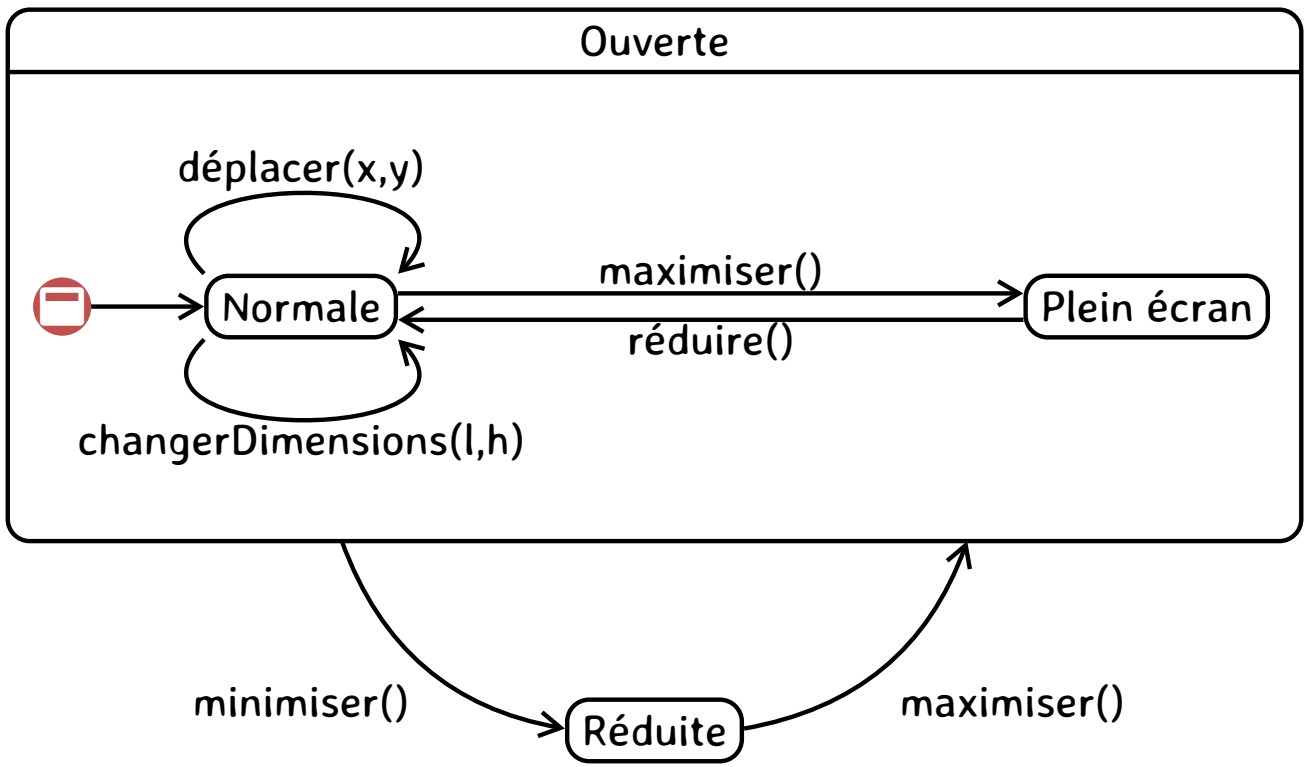
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsqu'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



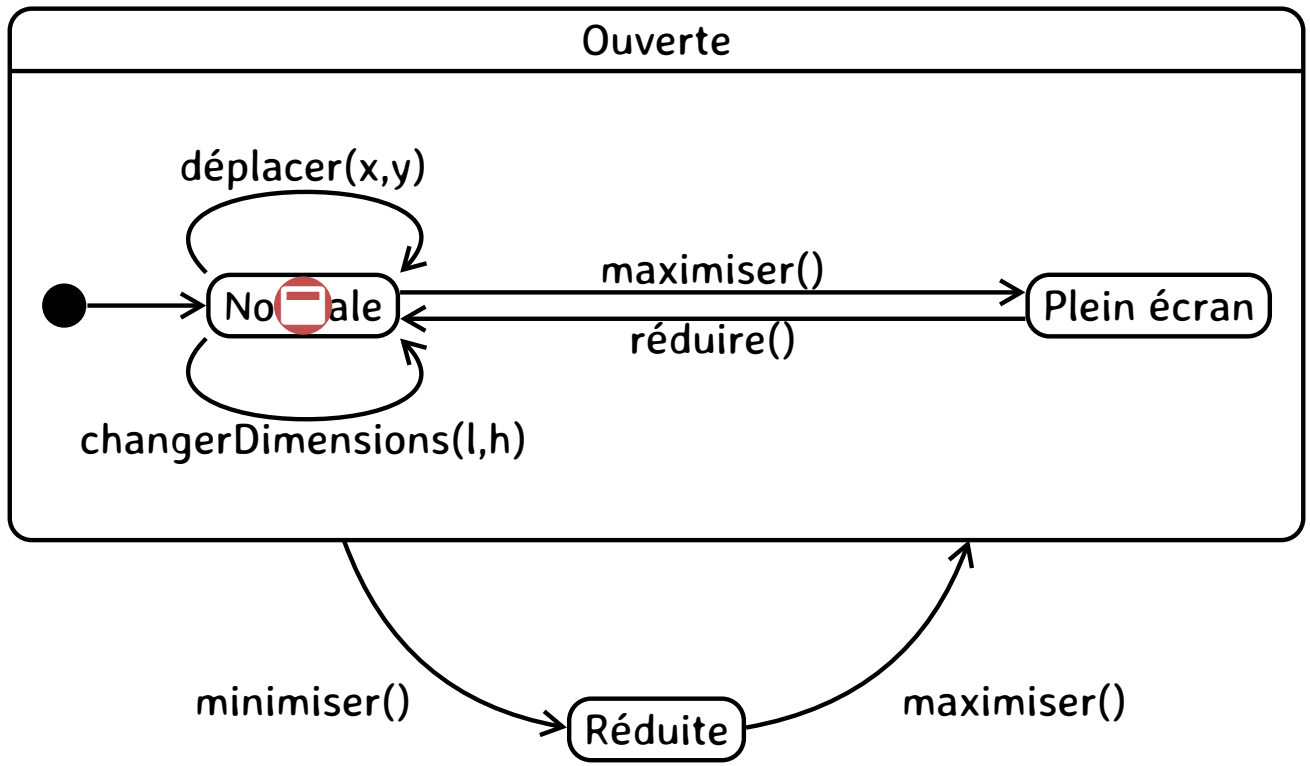
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



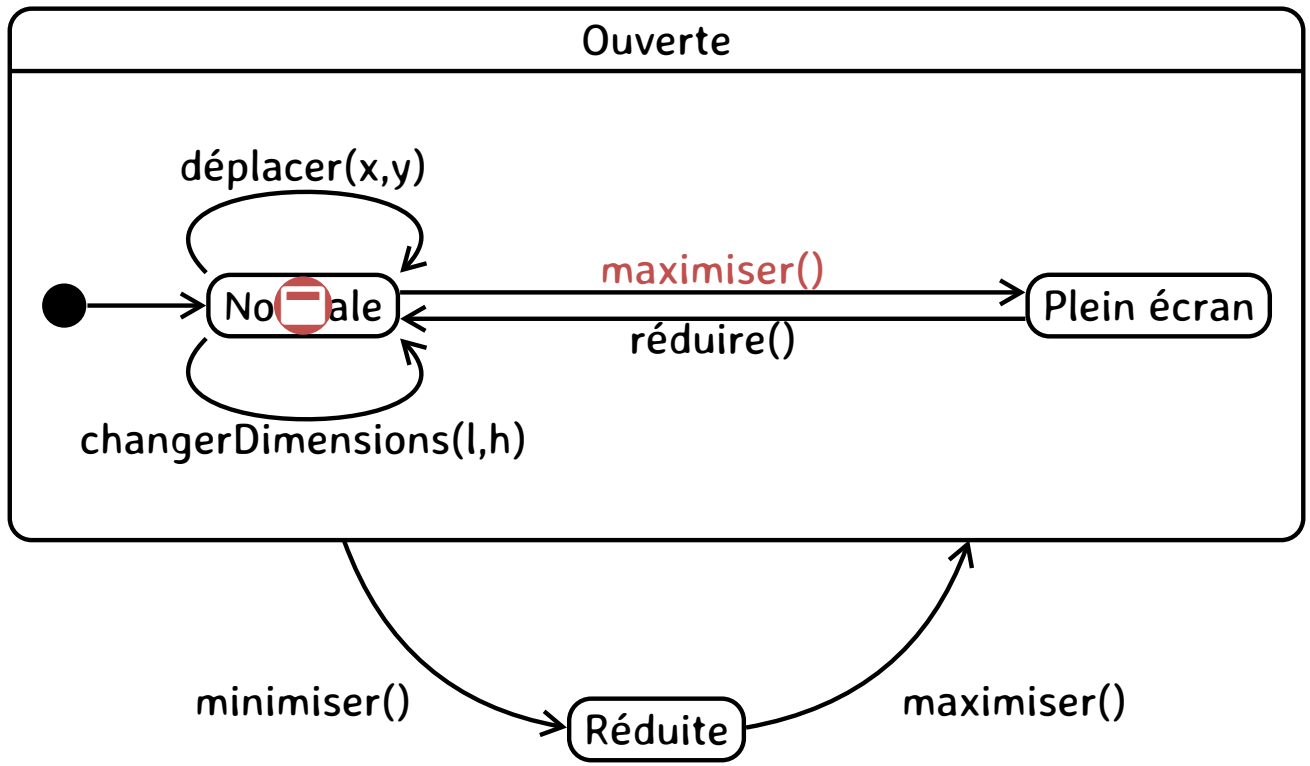
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



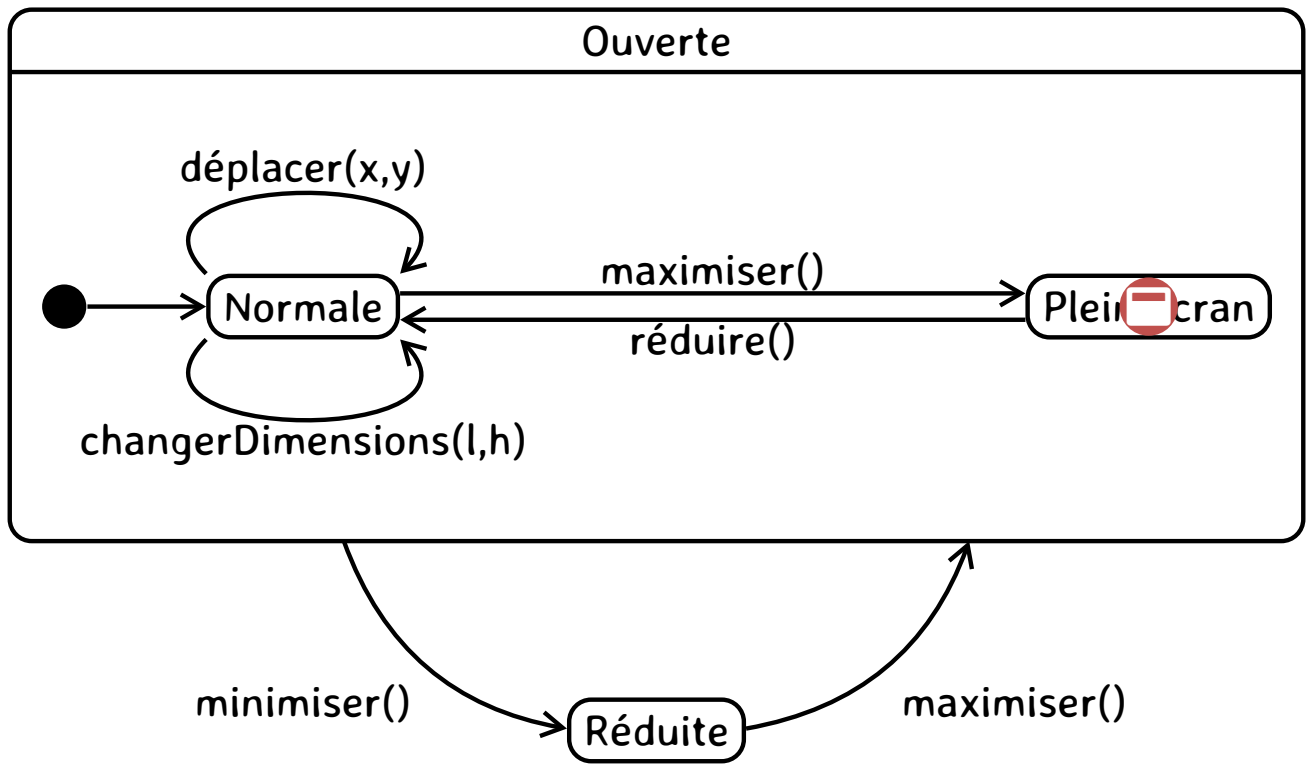
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



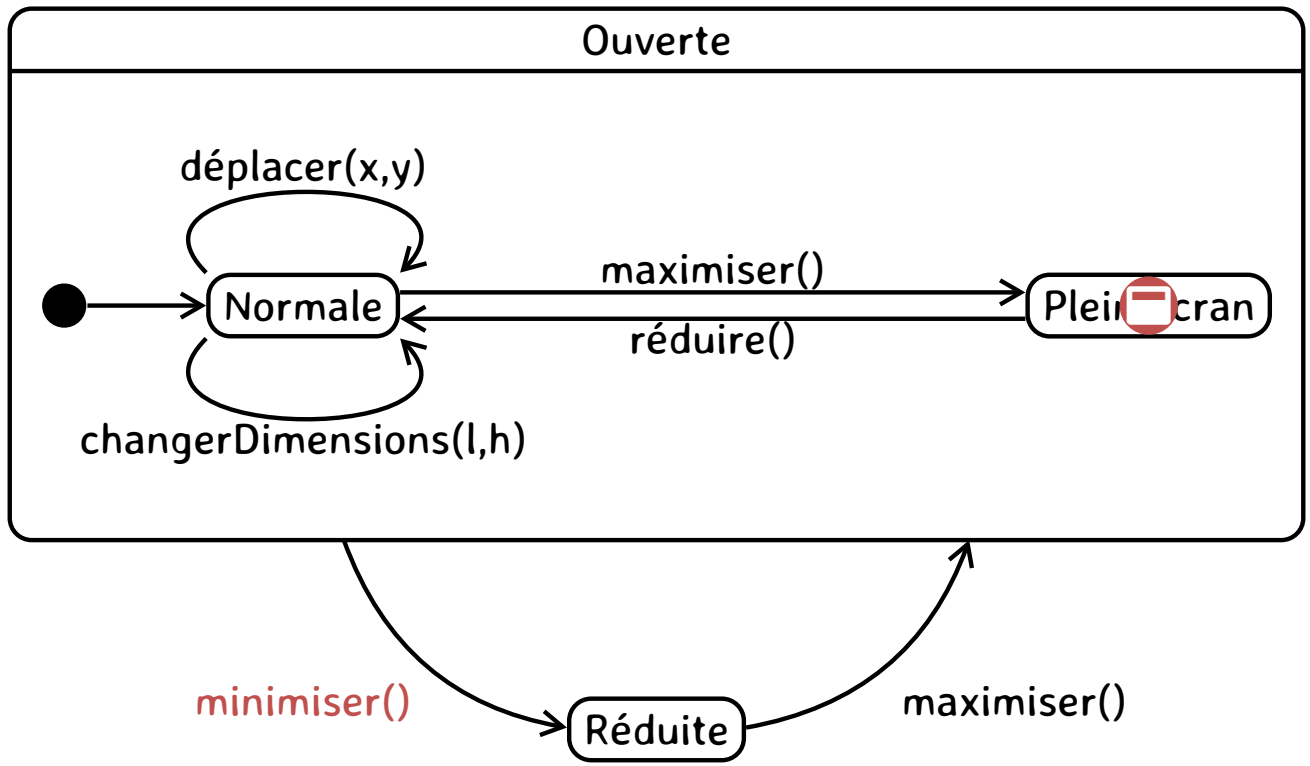
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



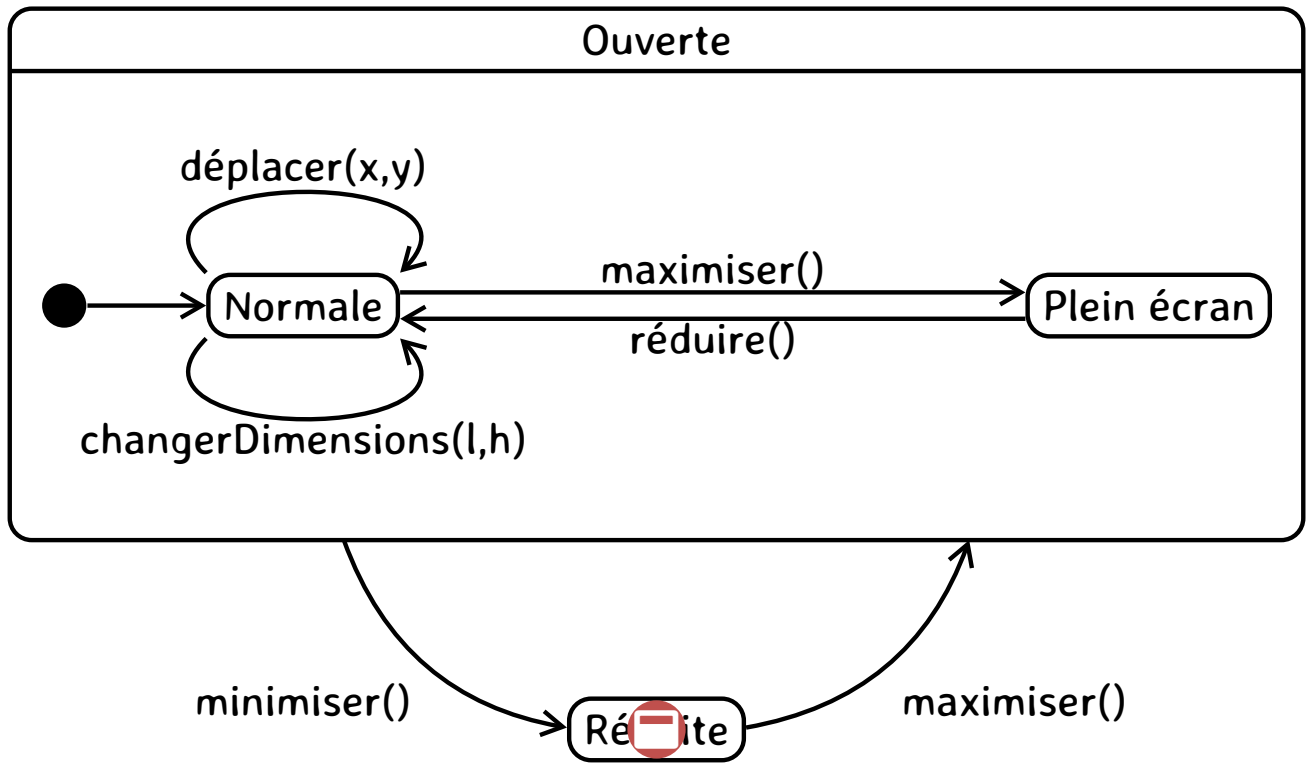
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



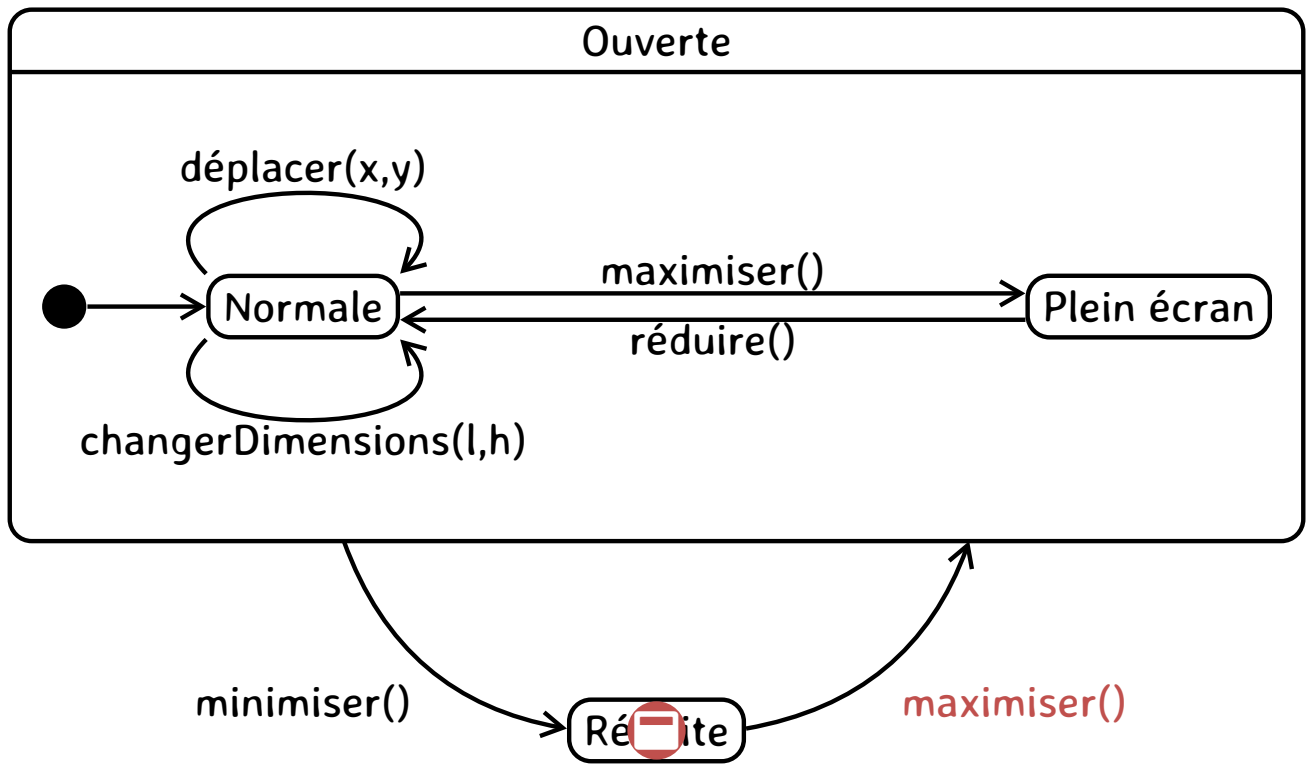
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



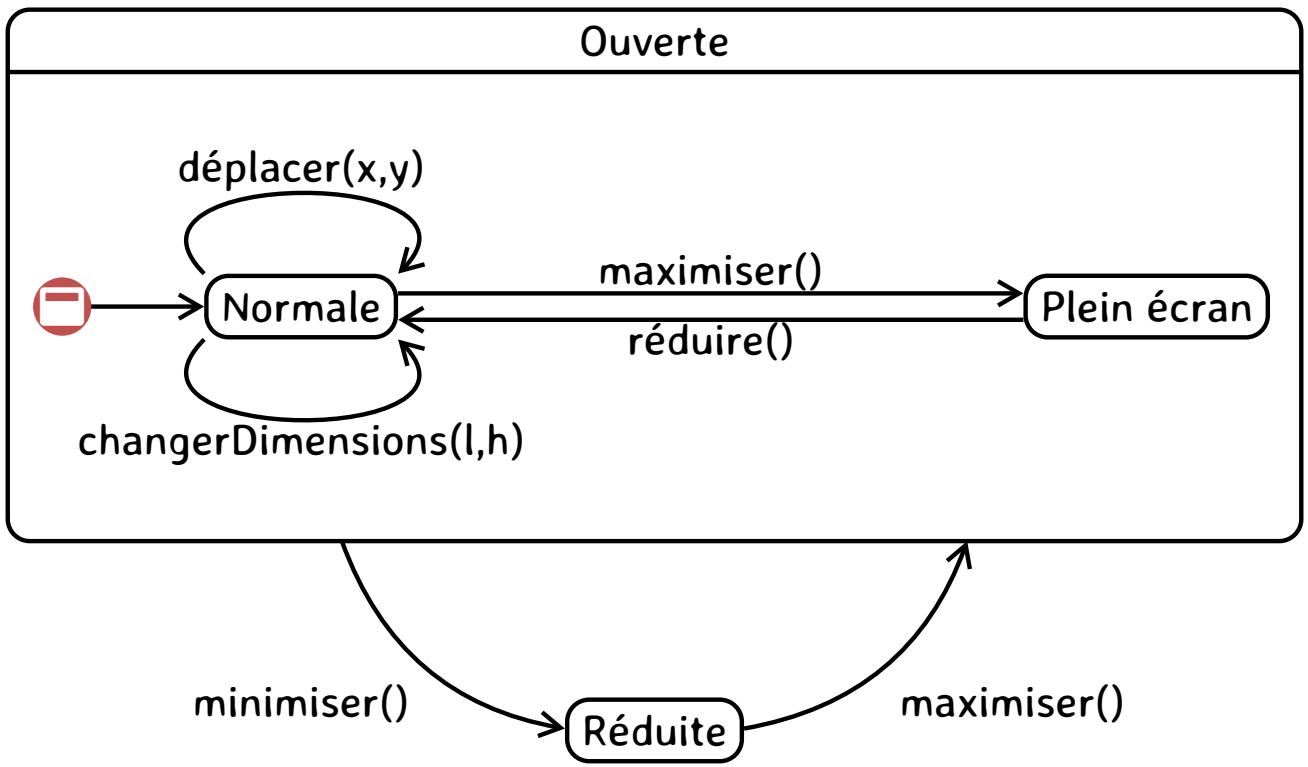
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



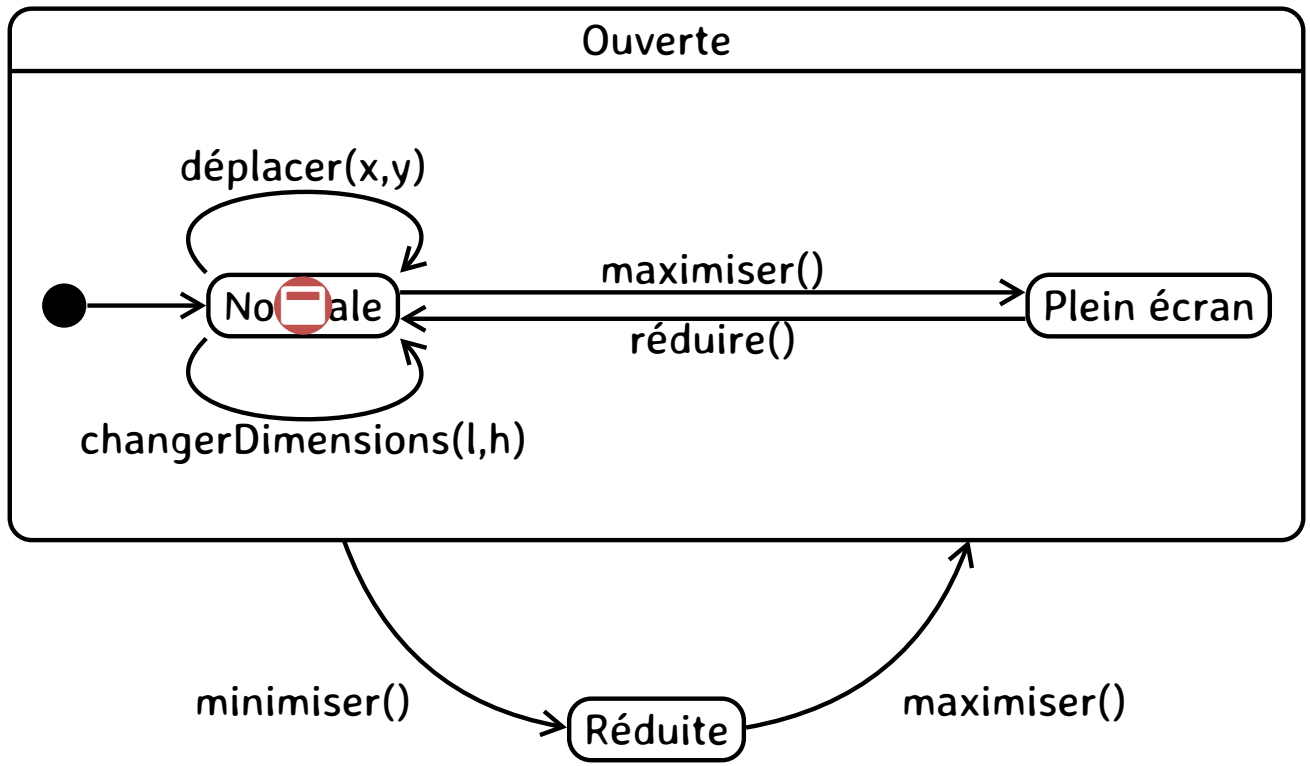
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



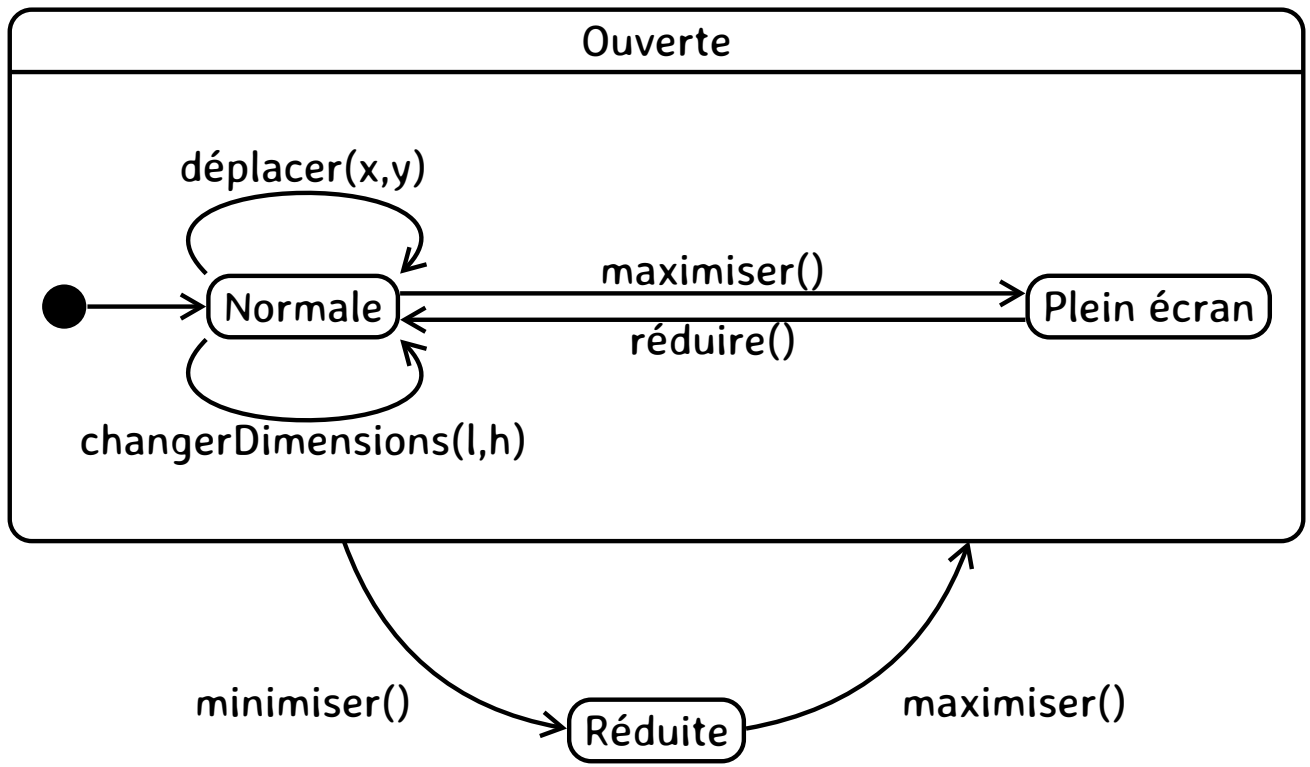
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.



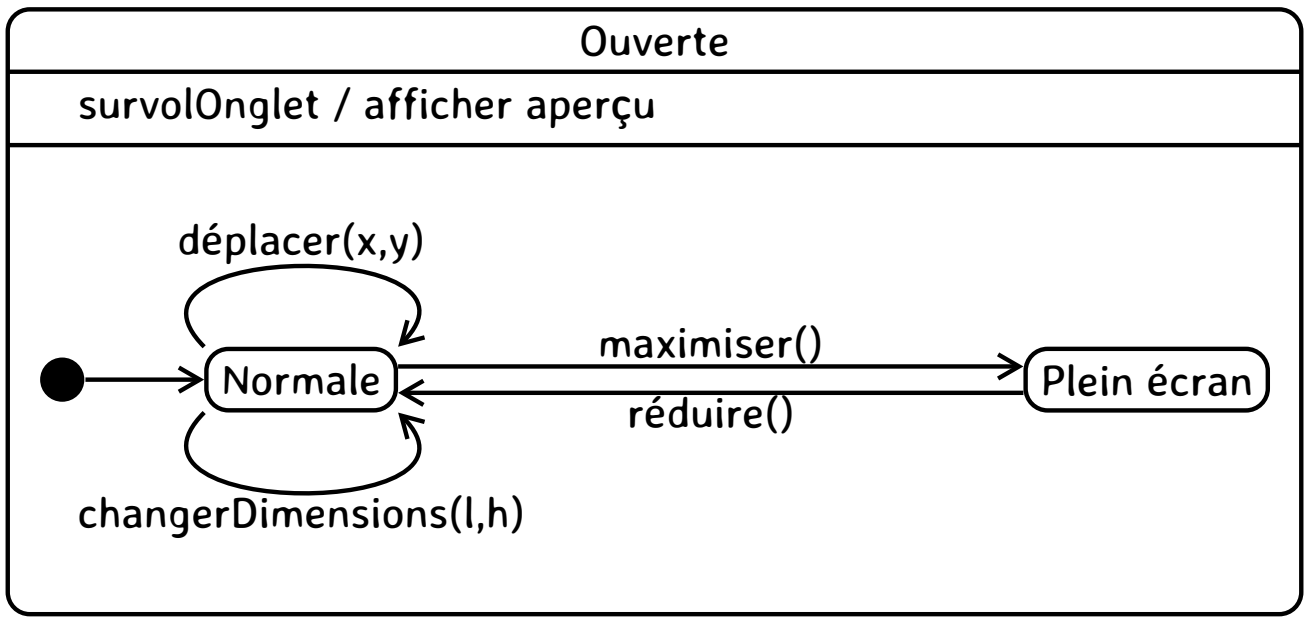
États composites
Transitions

- Une transition arrivant dans un état composite cible implicitement son état initial.
- En particulier, lorsequ'une transition est effectuée d'un état composite vers lui-même, celui-ci revient implicitement à son état initial.
- Si ce comportement est indésirable, deux solutions s'offrent à nous :
  1) Les transitions internes
  2) Le pseudo-état historique



États composites
Actions internes

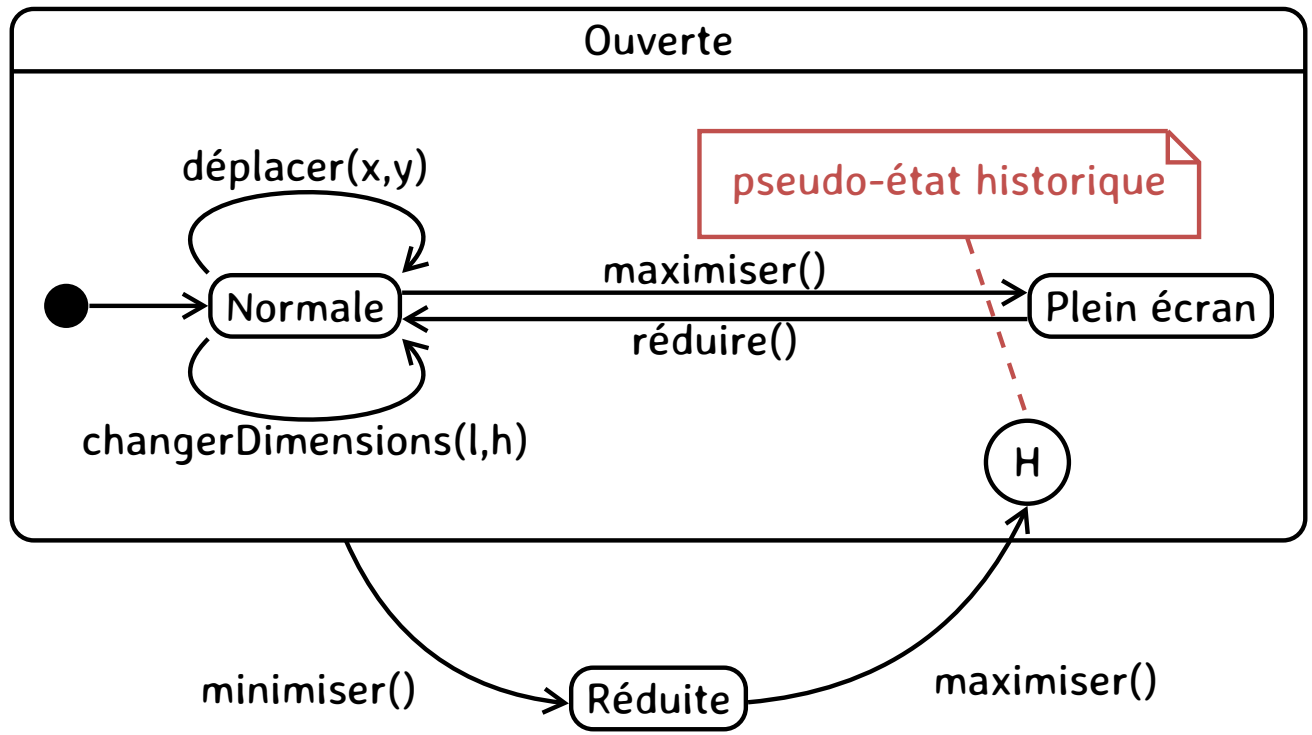
Dans un état composite, tous les sous-états héritent des actions internes du super état.



États composites
Concept : pseudo-état historique

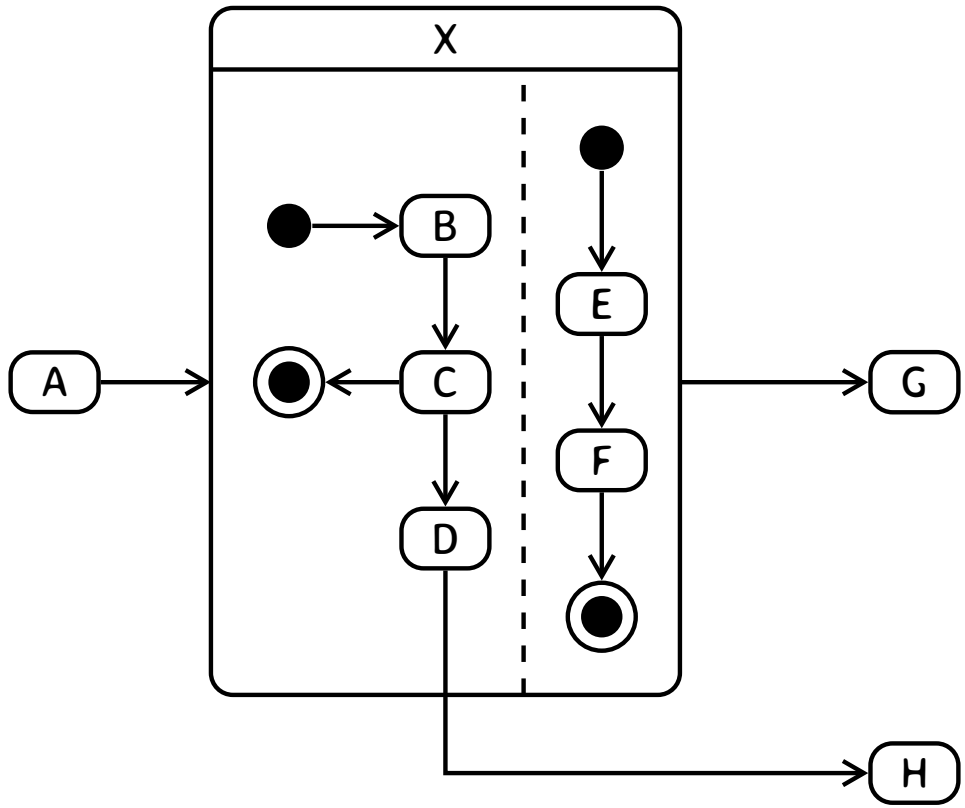
définition

Il est possible, lorsqu'un objet quitte un état composé, de mémoriser le sous-état actif pour y revenir. On utilise pour cela le pseudo-état historique qui représente dans l'état composé le dernier sous-état actif mémorisé.



Régions concurrentes

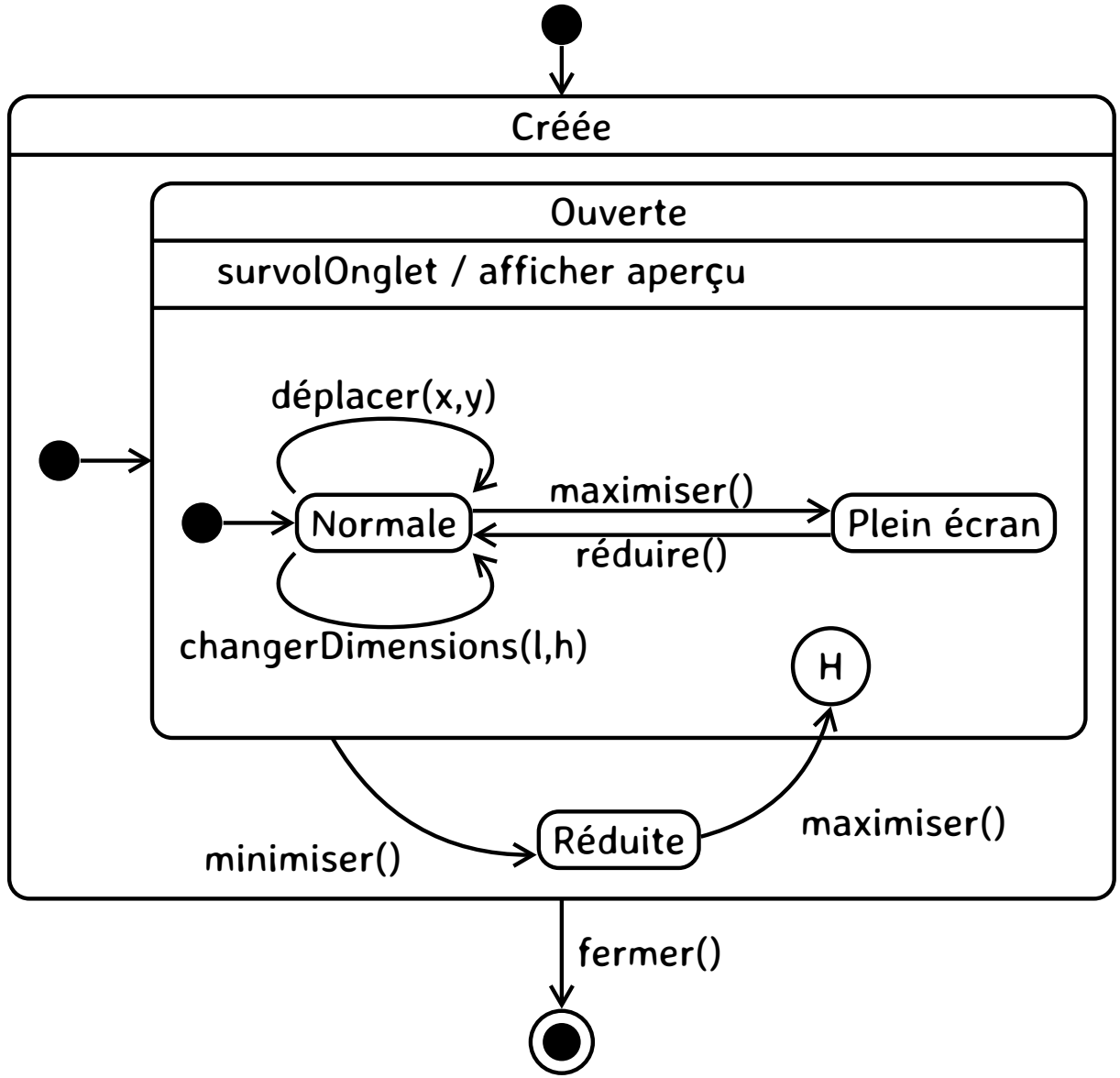
Composition (simultanée) de plusieurs sous-états appelés régions.



UML - La vue dynamique
Plan

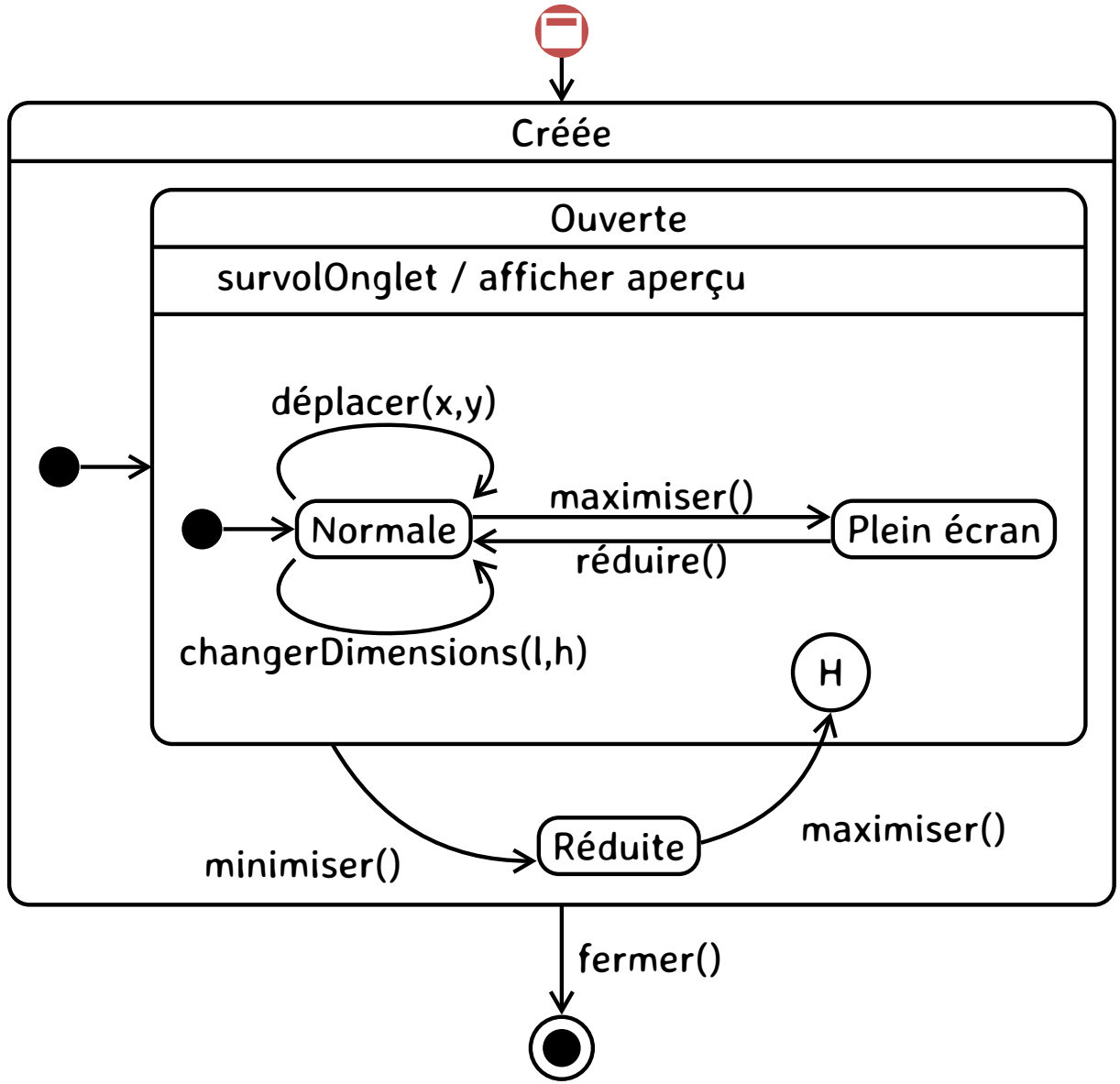
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



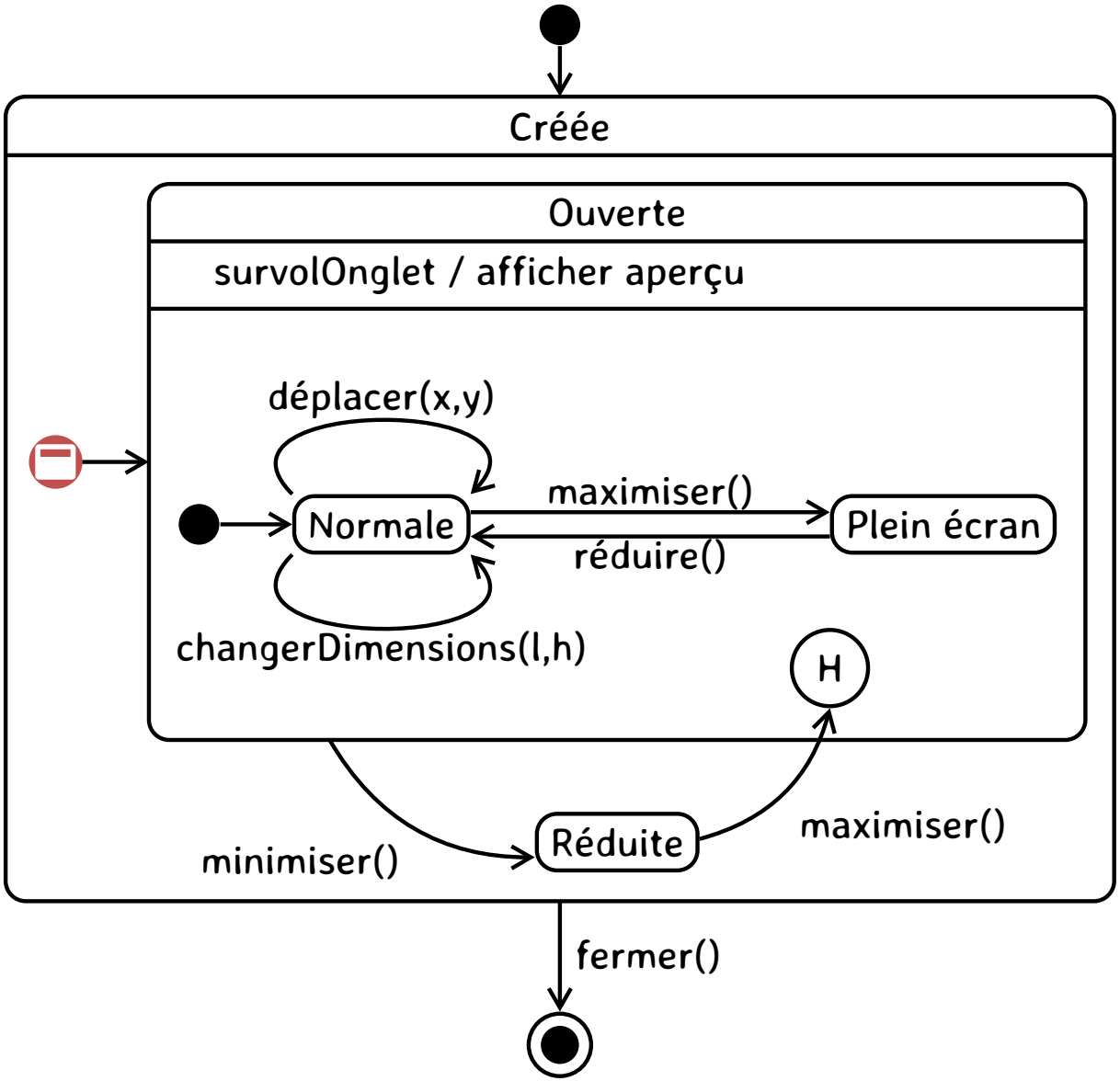
La classe fenêtre complète
Exemple

→ maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



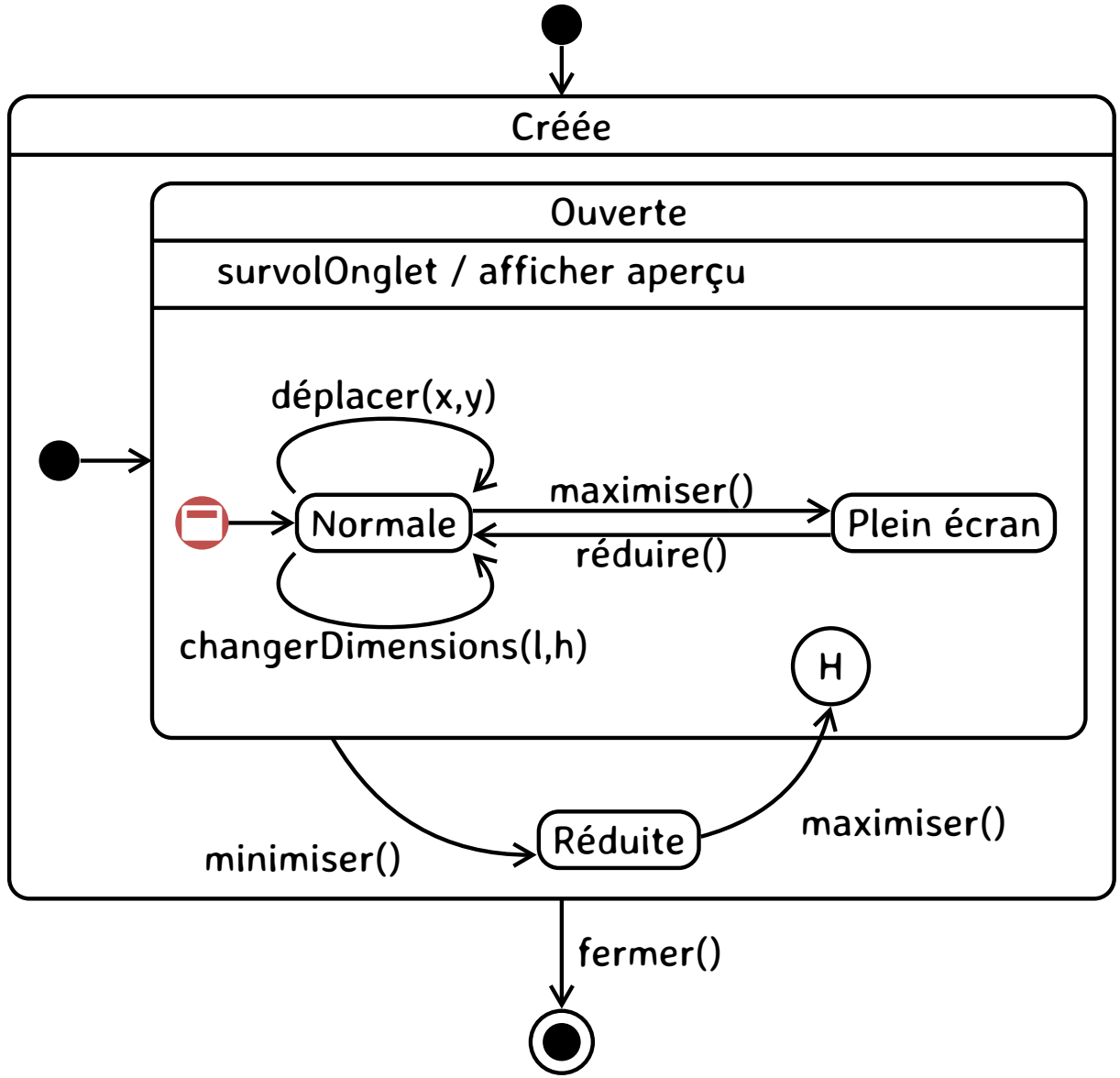
La classe fenêtre complète
Exemple

→ maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



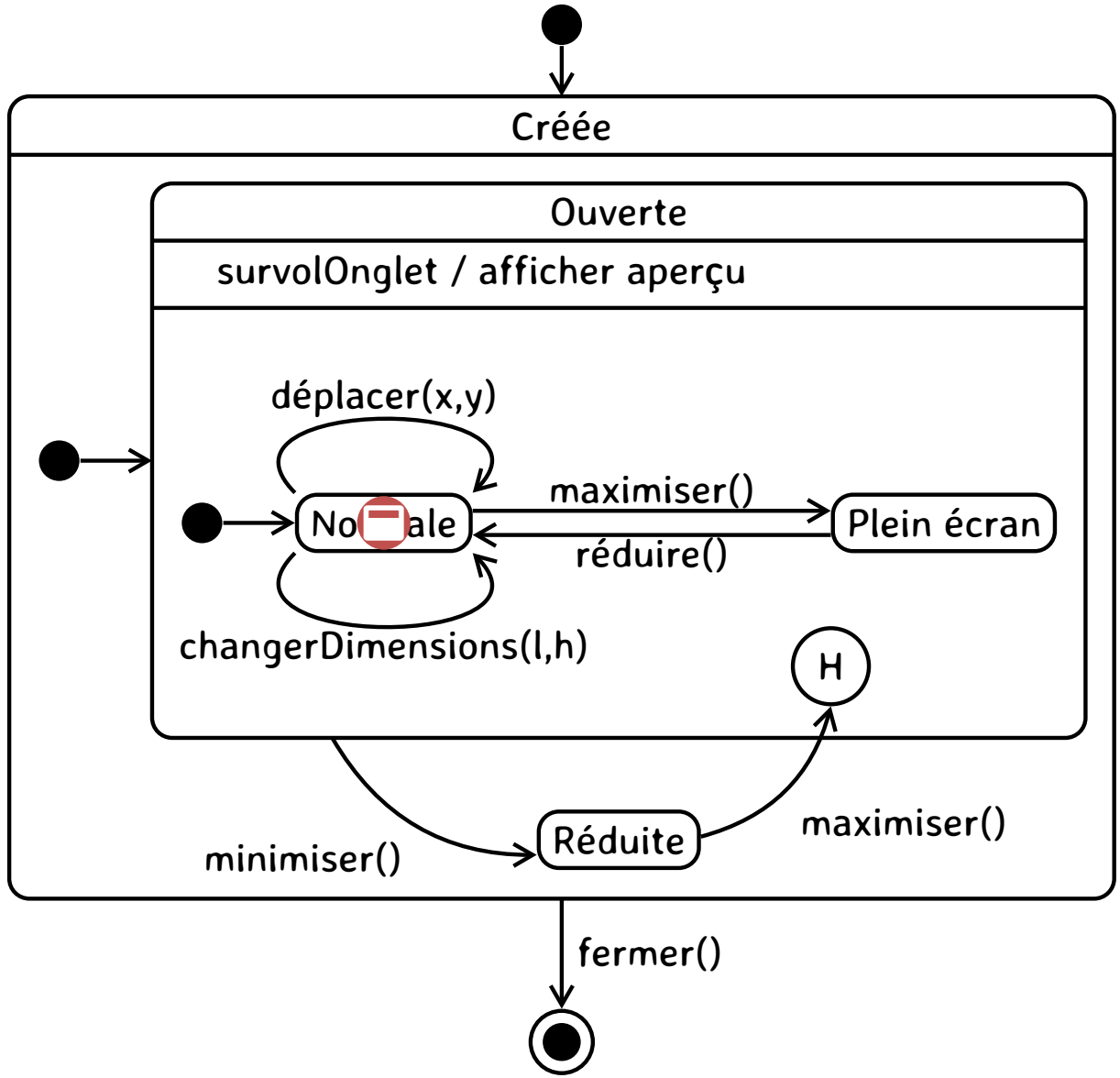
La classe fenêtre complète
Exemple

→ maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



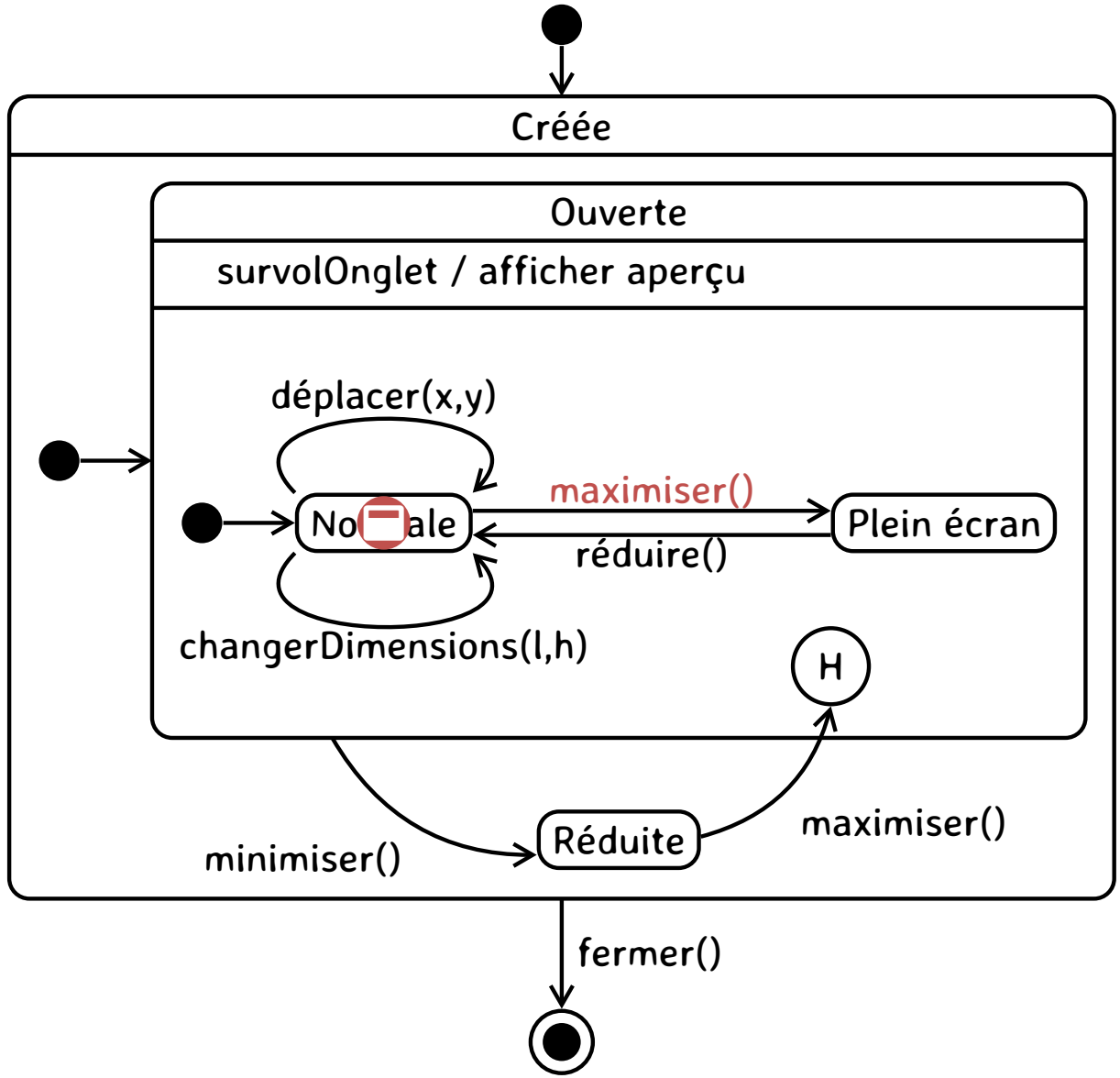
La classe fenêtre complète
Exemple

→ maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



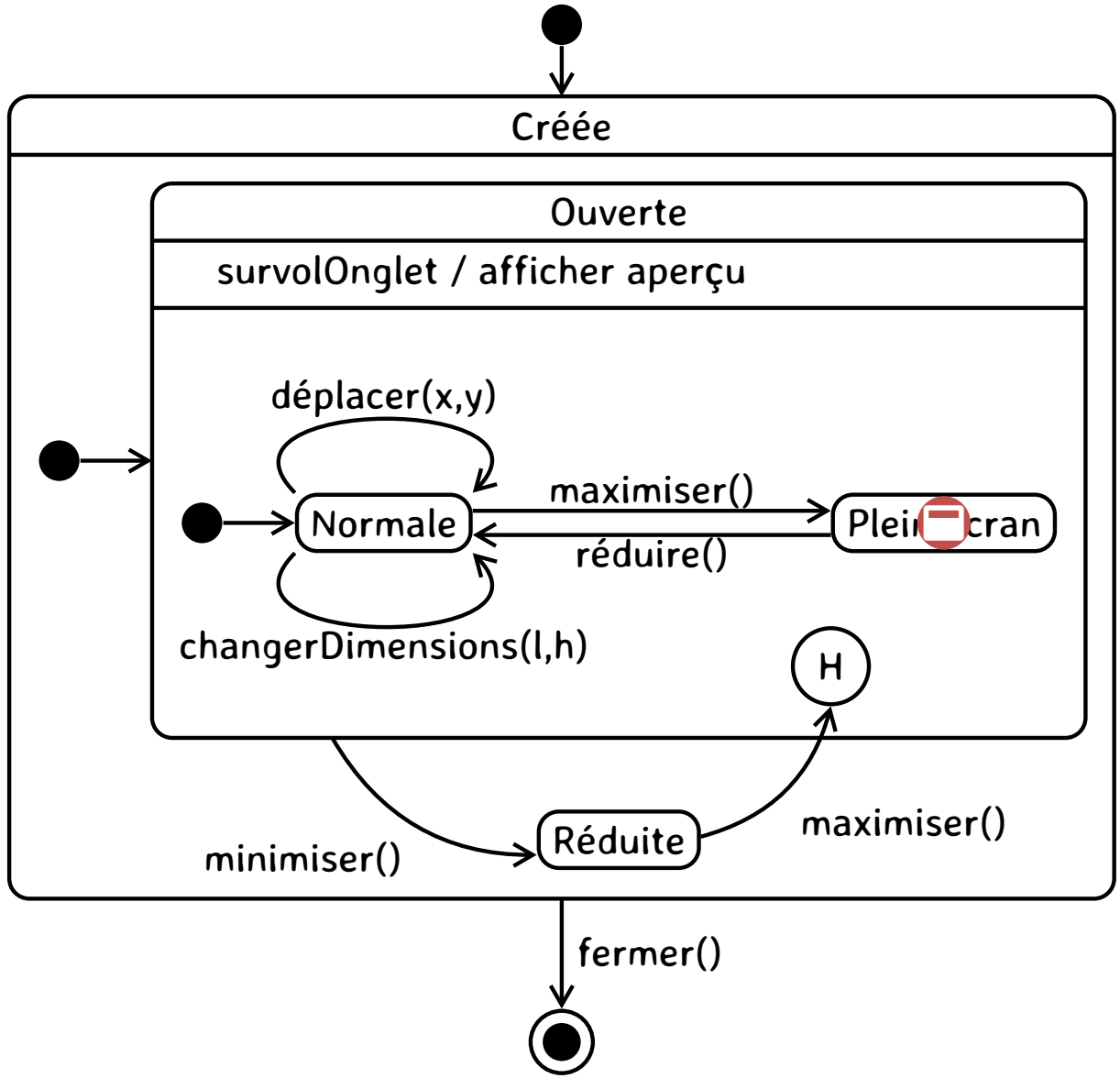
La classe fenêtre complète
Exemple

→ maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



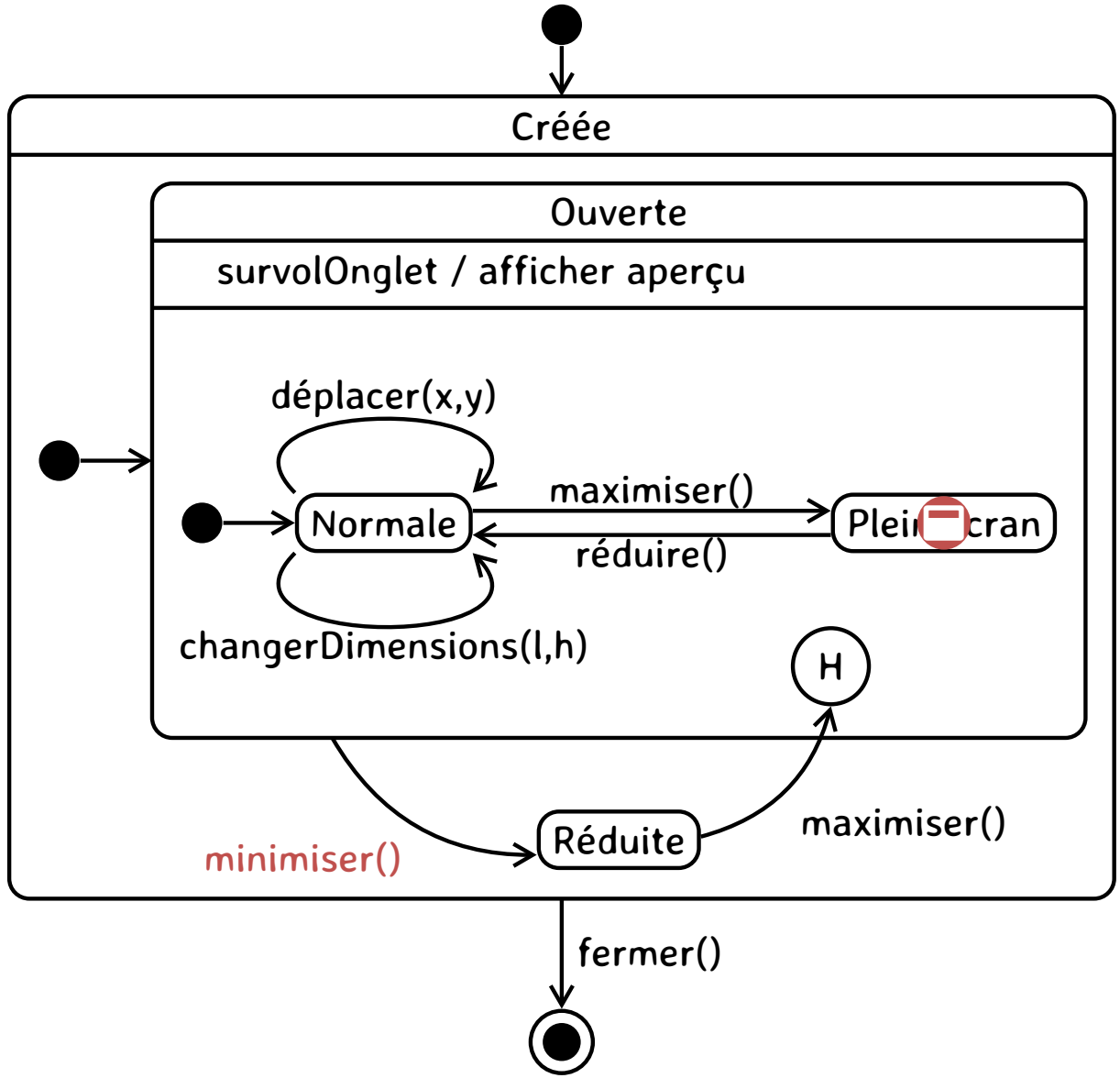
La classe fenêtre complète
Exemple

maximiser()
→ minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



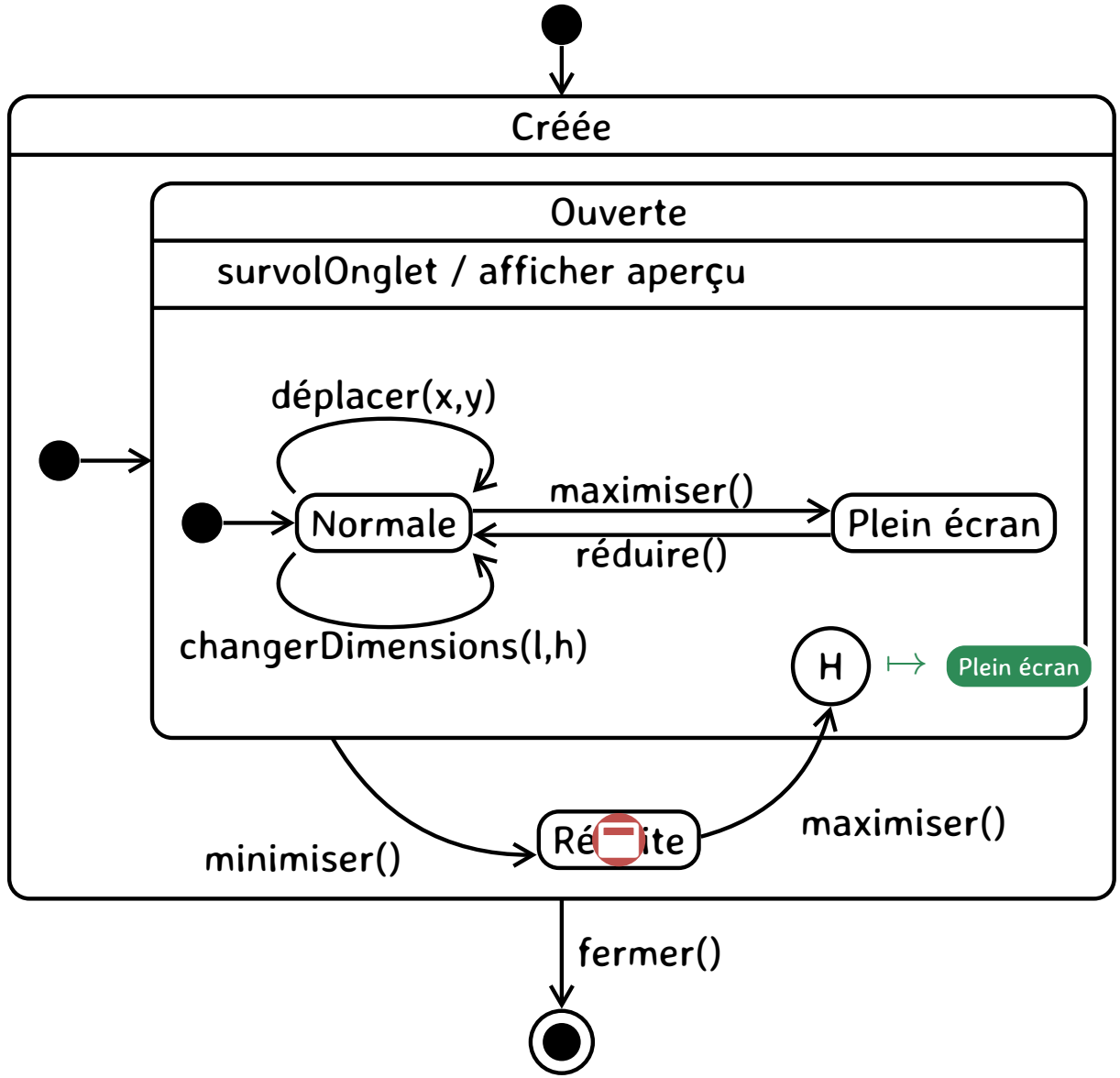
La classe fenêtre complète
Exemple

maximiser()
→ minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



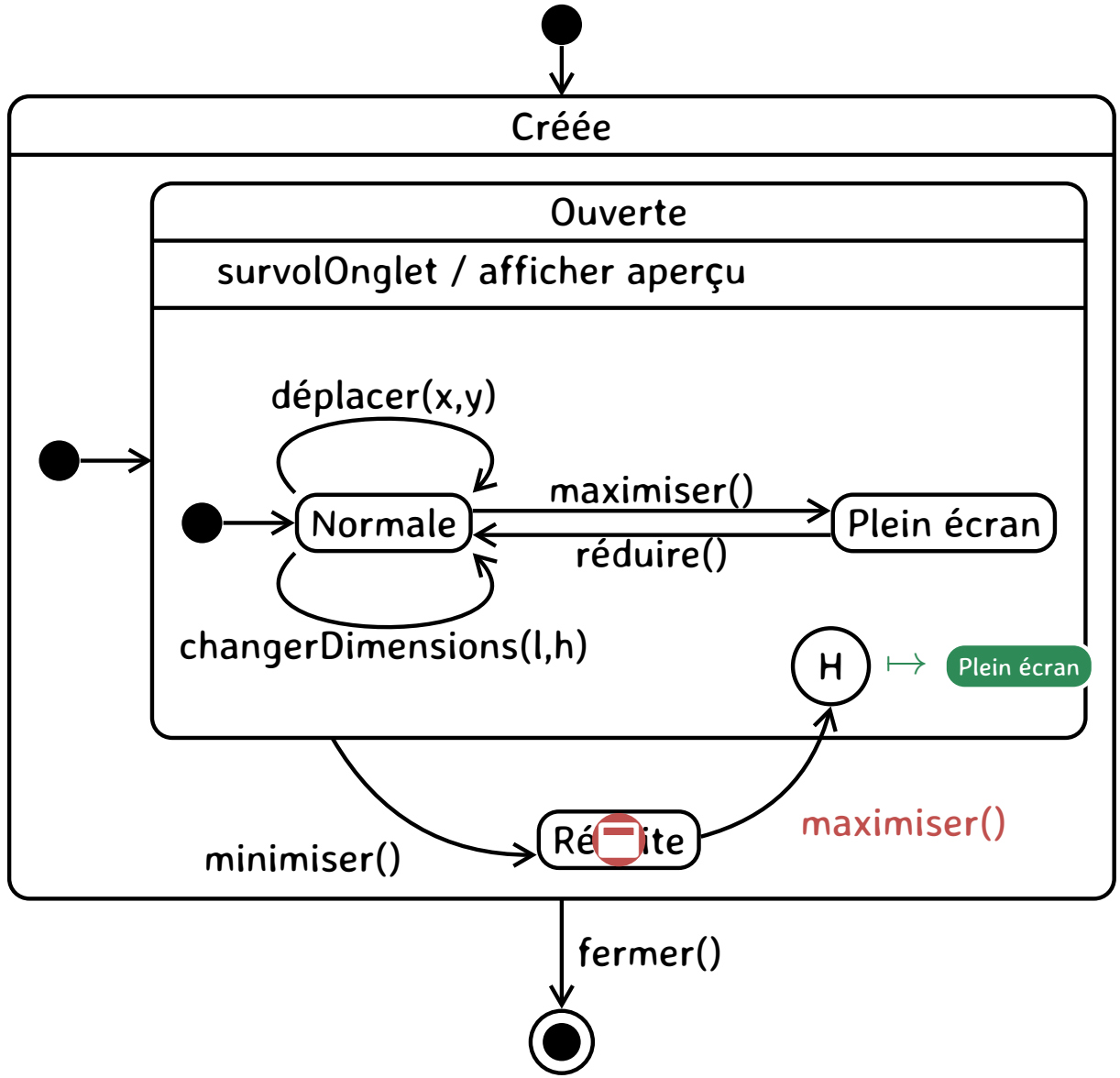
La classe fenêtre complète
Exemple

maximiser()
minimiser()
→ maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



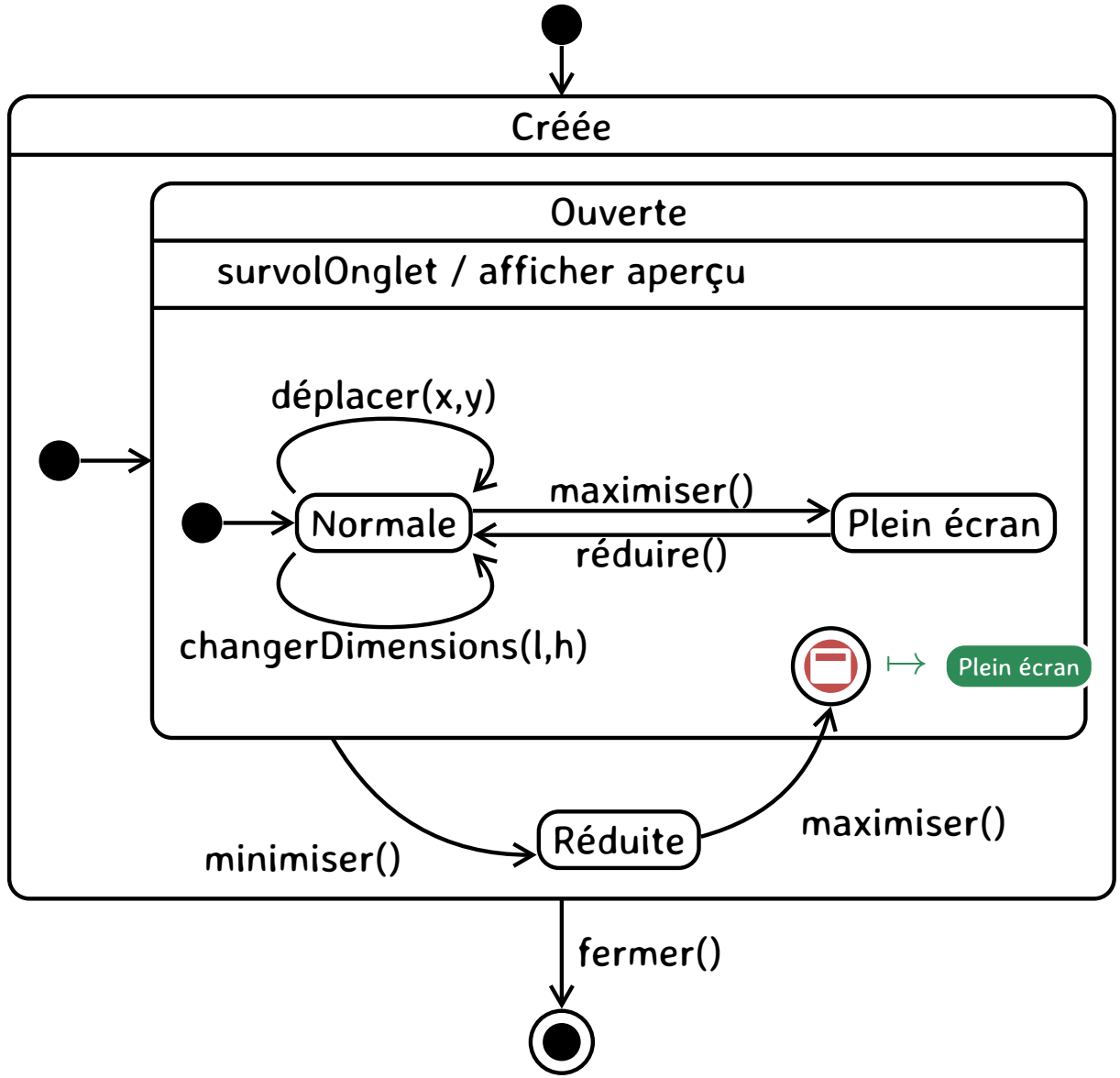
La classe fenêtre complète
Exemple

maximiser()
minimiser()
→ maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



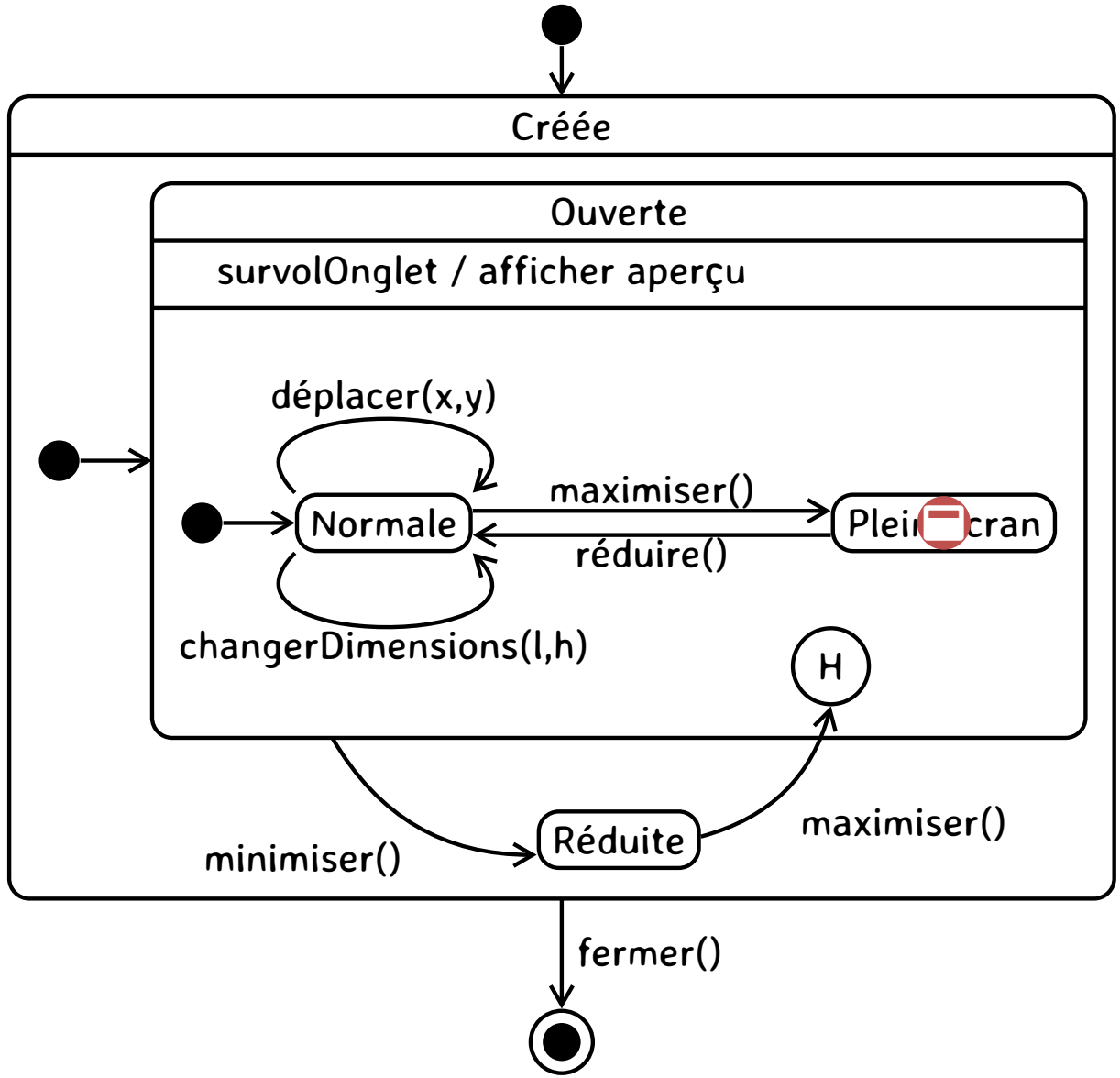
La classe fenêtre complète
Exemple

maximiser()
minimiser()
→ maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



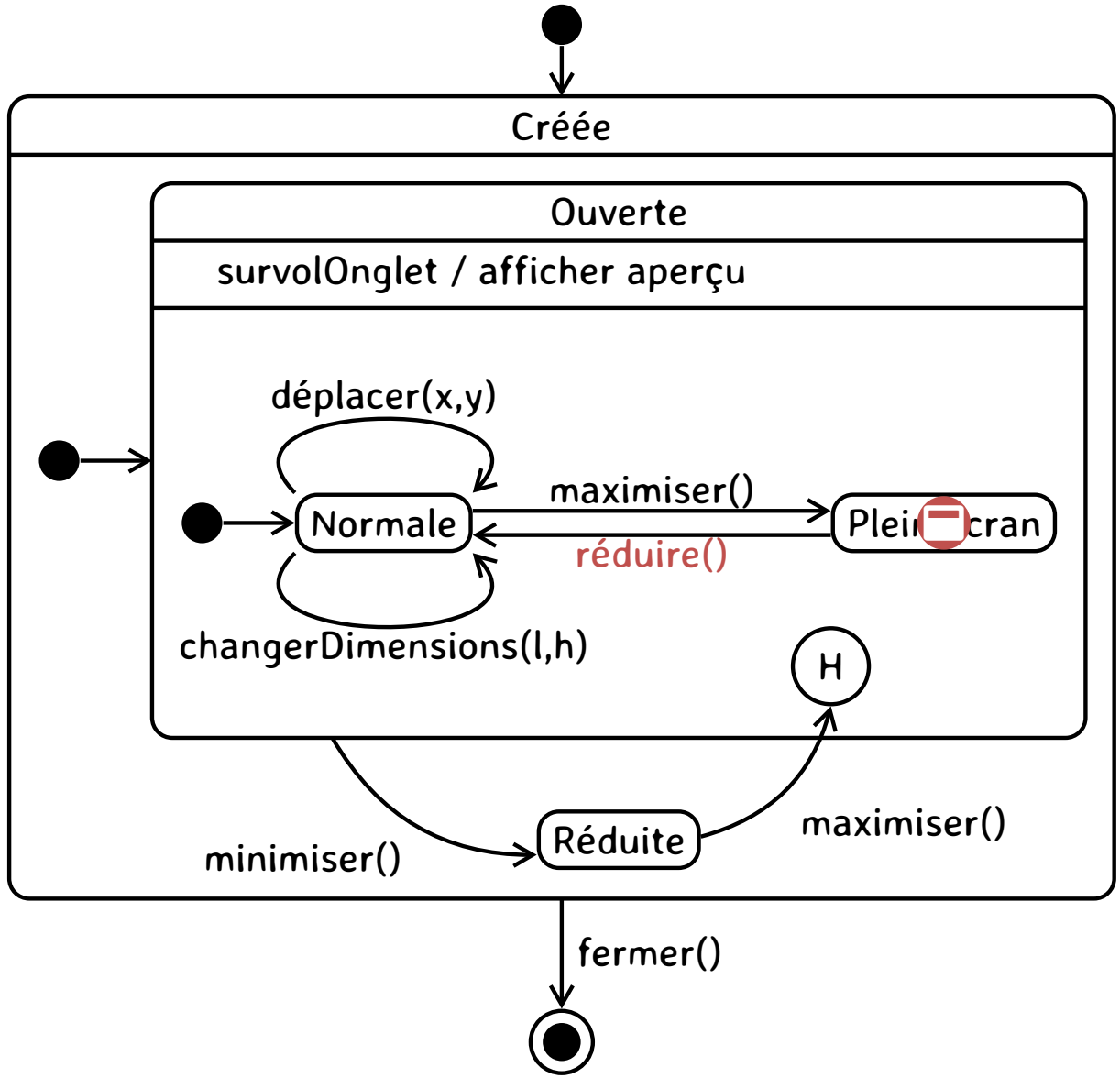
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
→ réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



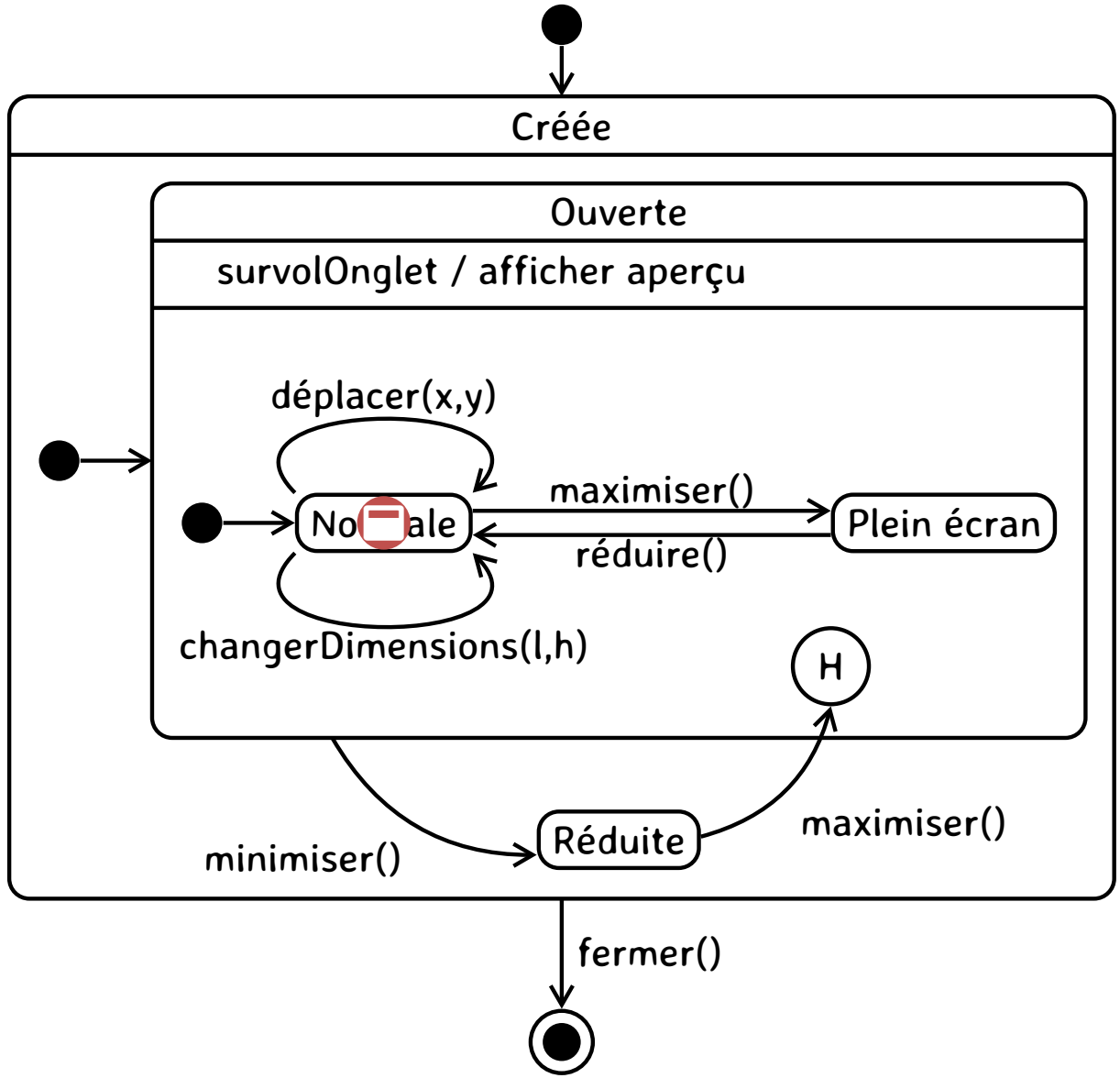
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
→ réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



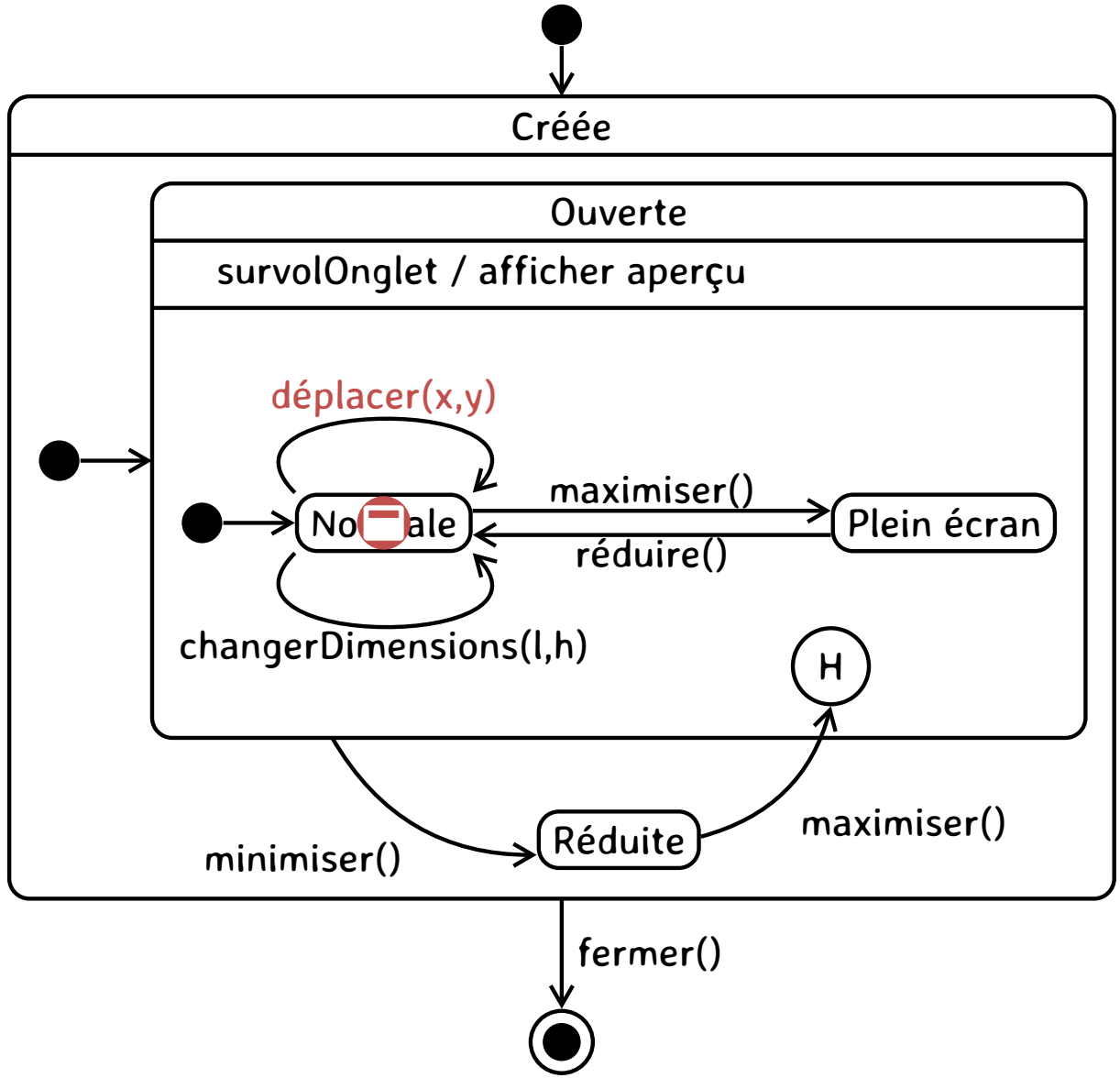
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
→ déplacer(x,y)
minimiser()
maximiser()
fermer()



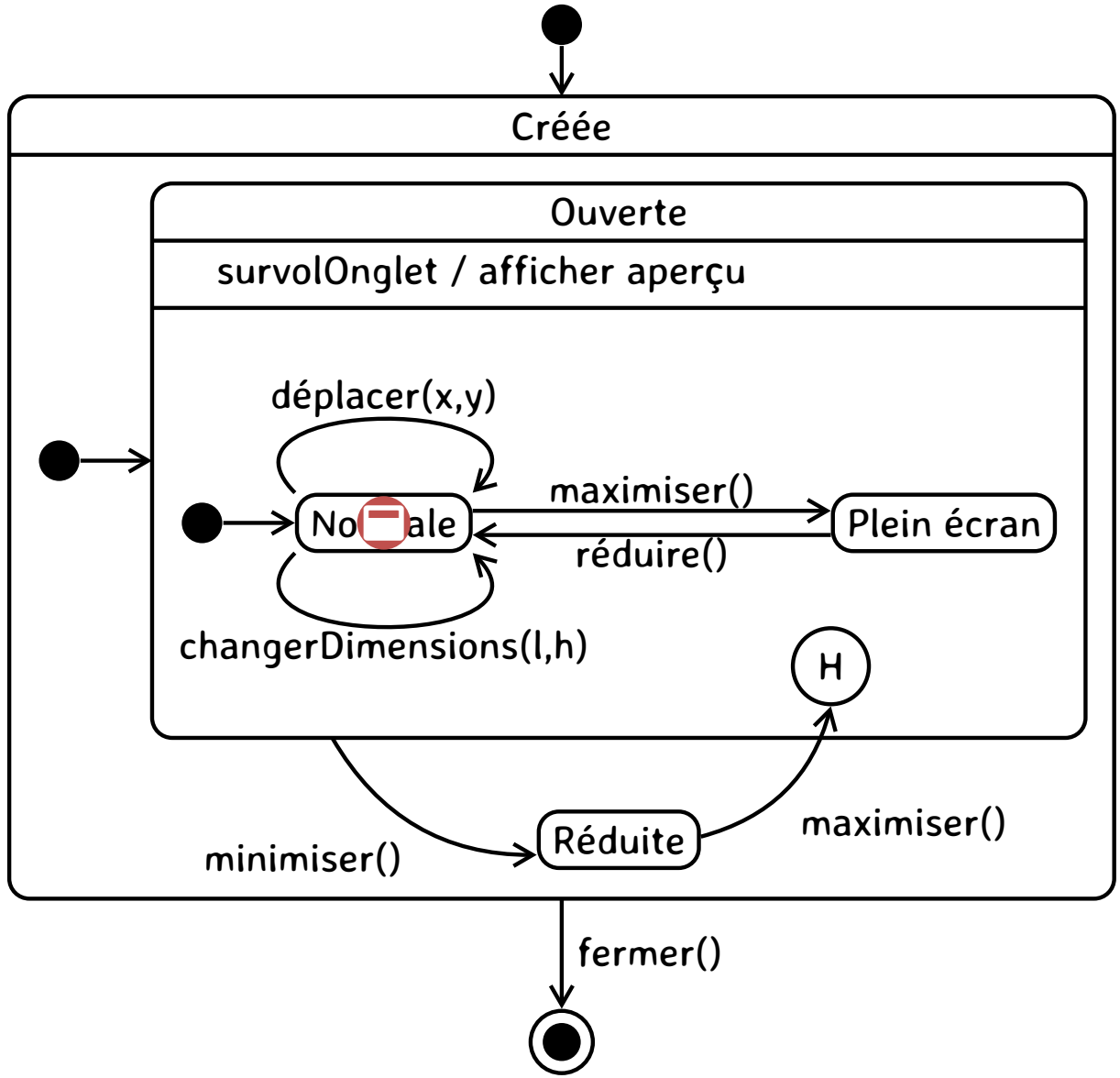
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
→ déplacer(x,y)
minimiser()
maximiser()
fermer()



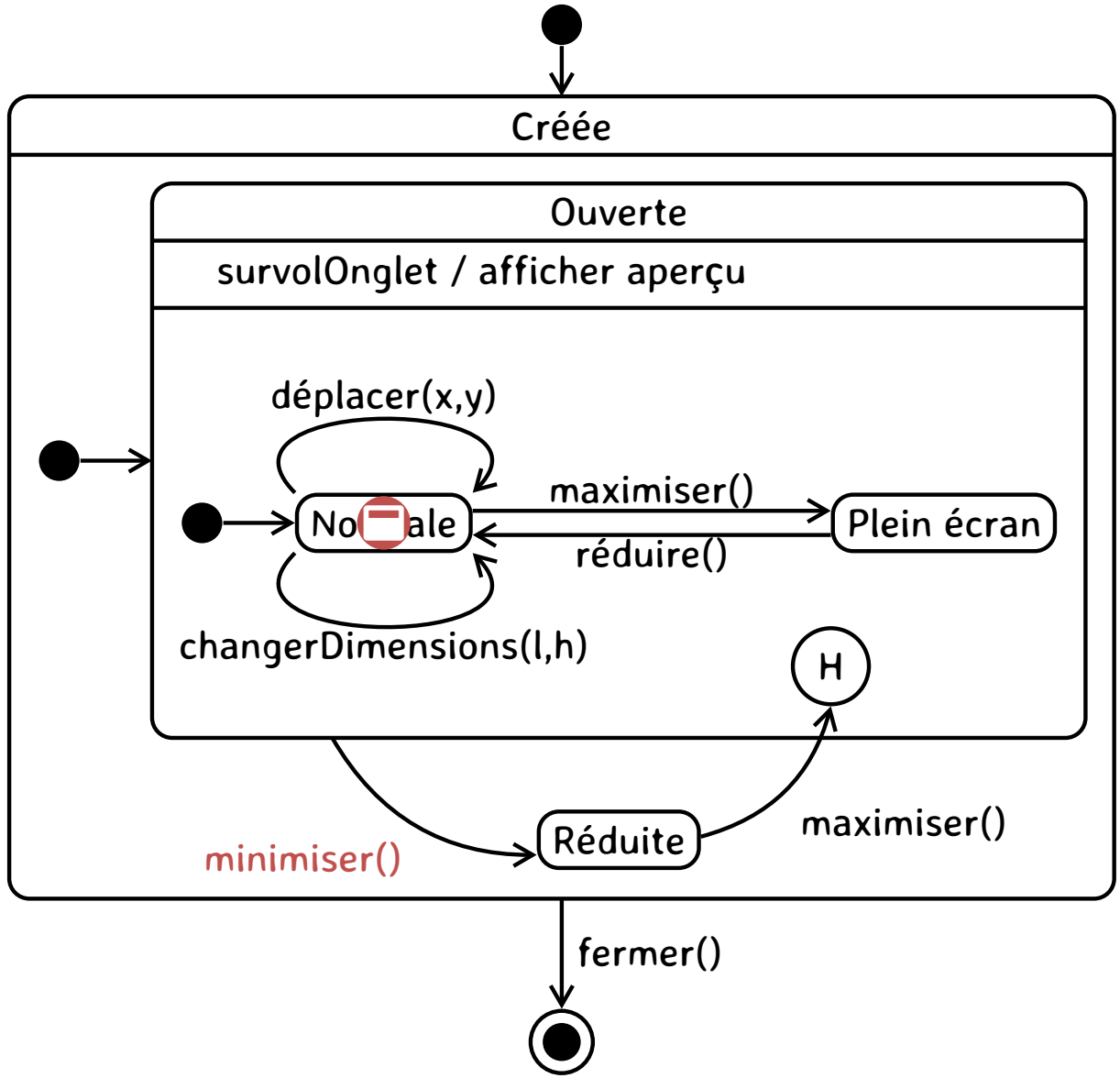
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
→ minimiser()
maximiser()
fermer()



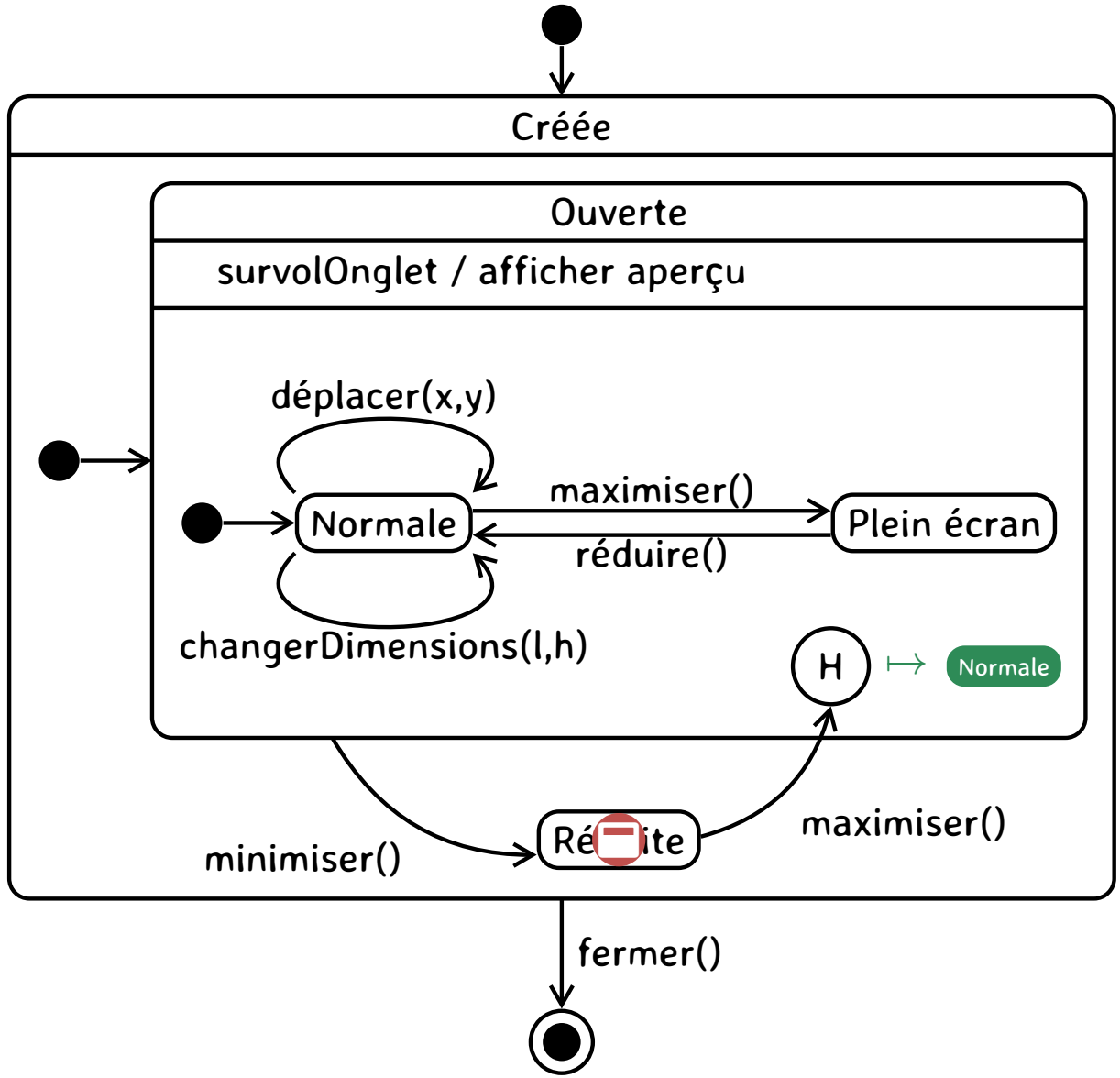
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
→ minimiser()
maximiser()
fermer()



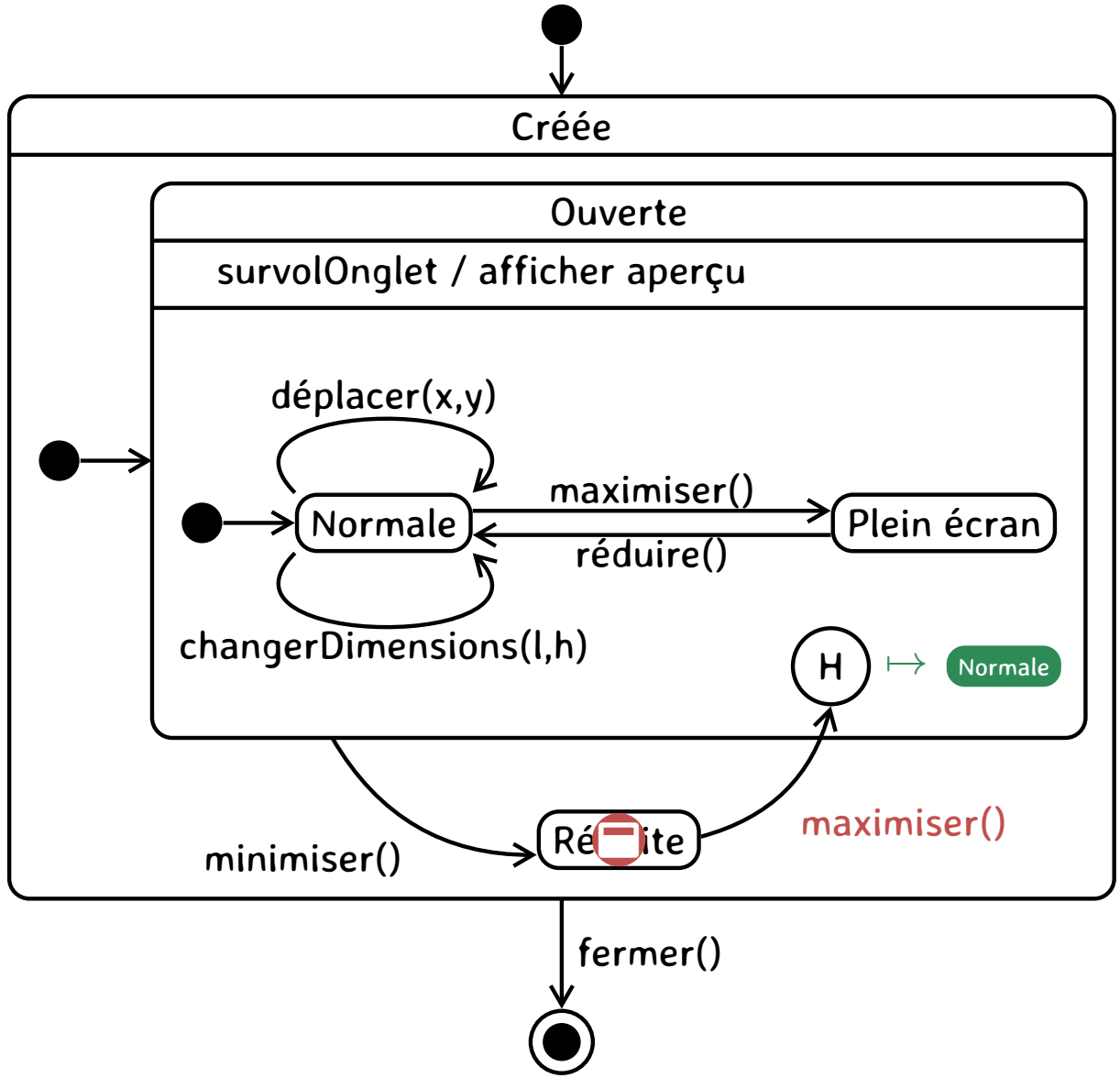
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
→ maximiser()
fermer()



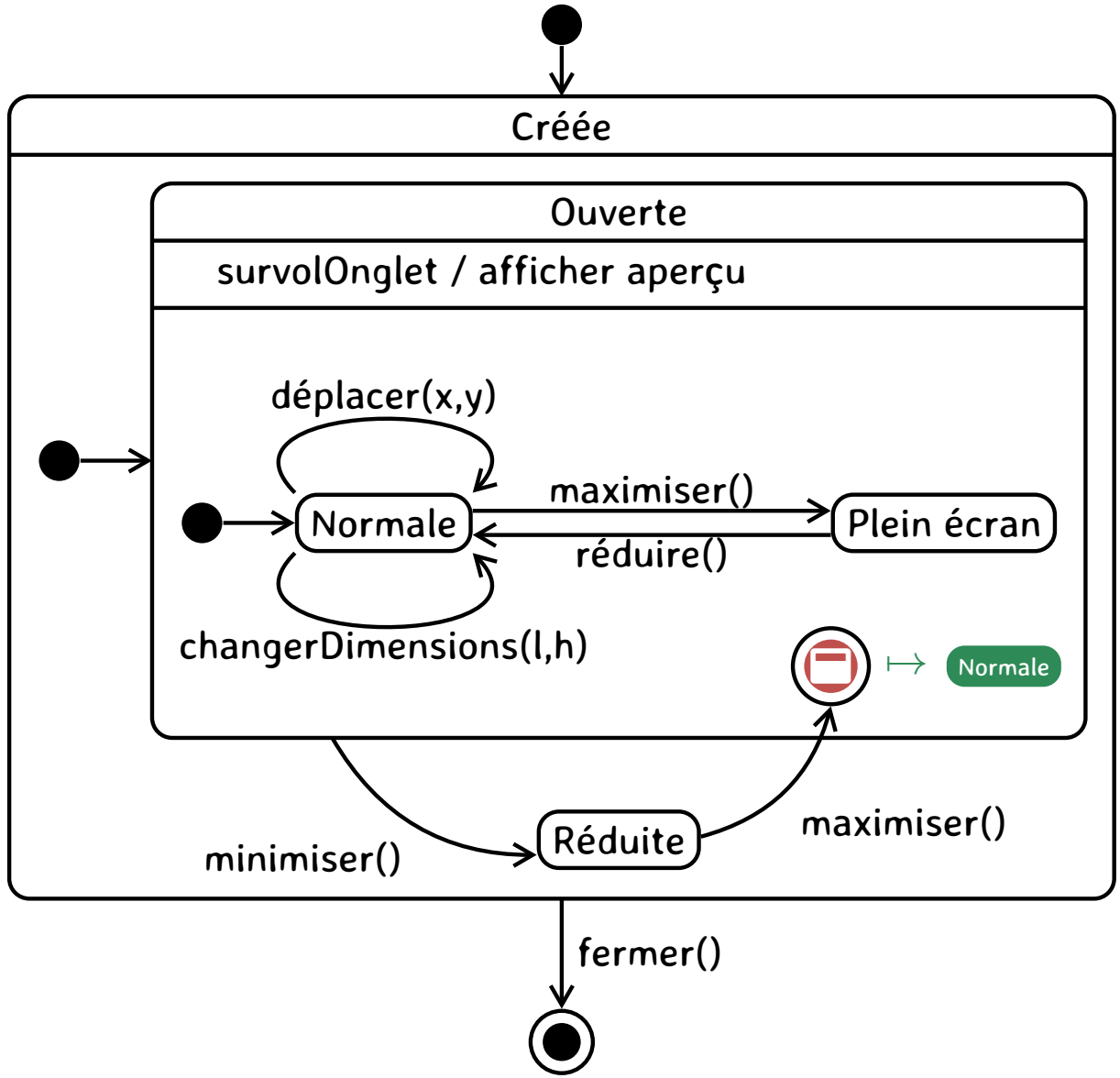
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
→ maximiser()
fermer()



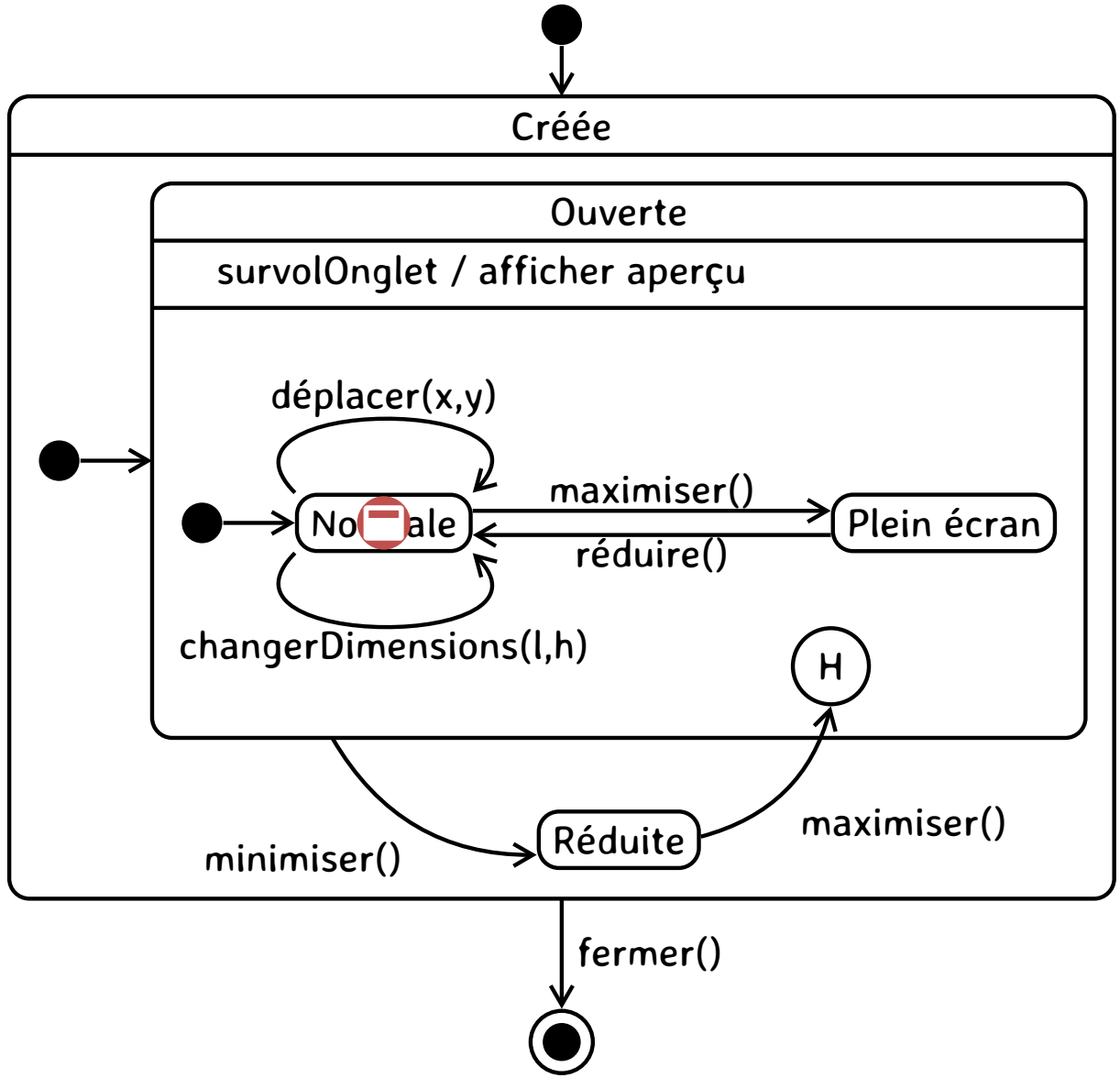
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
→ maximiser()
fermer()



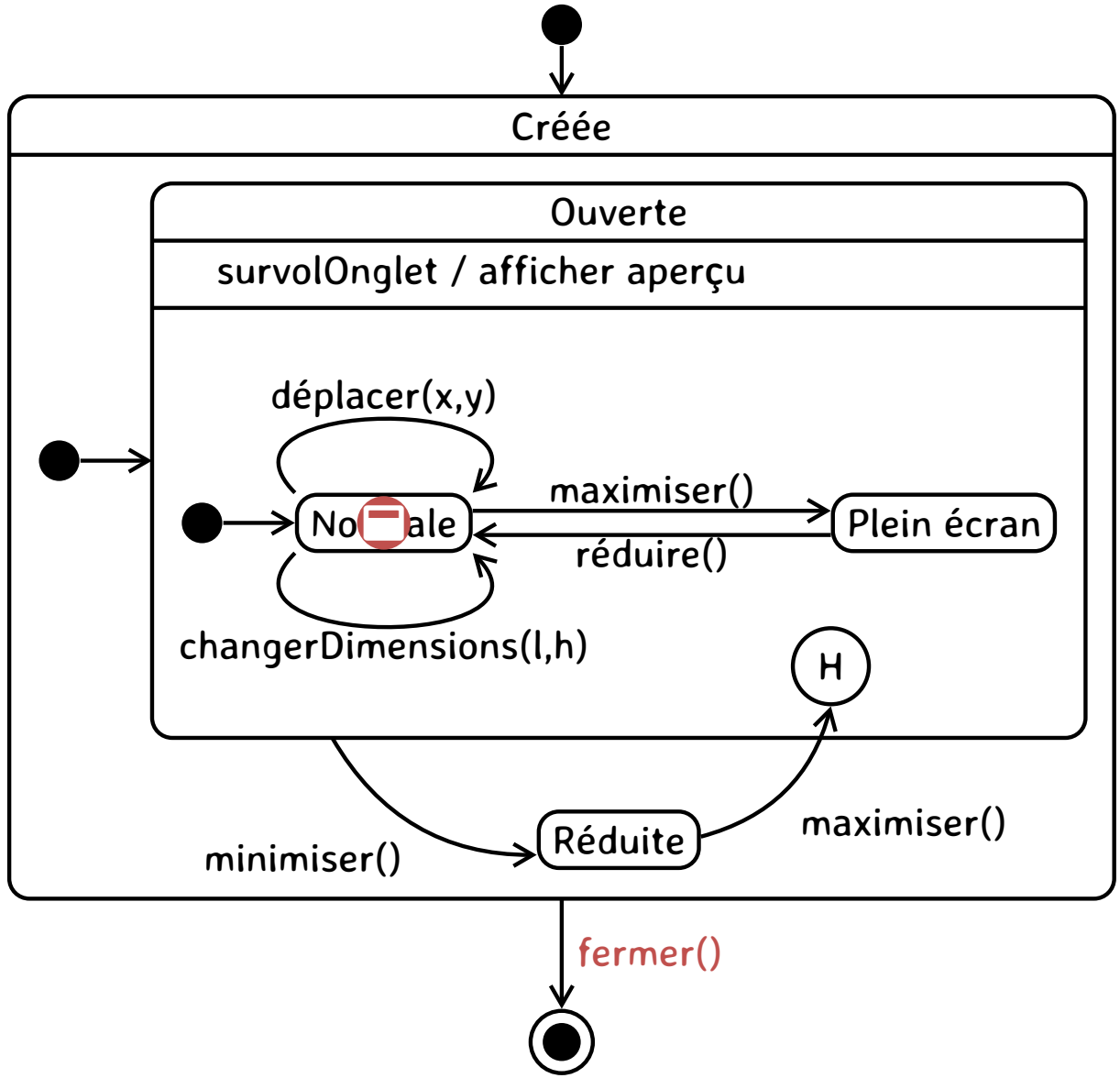
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
→ fermer()



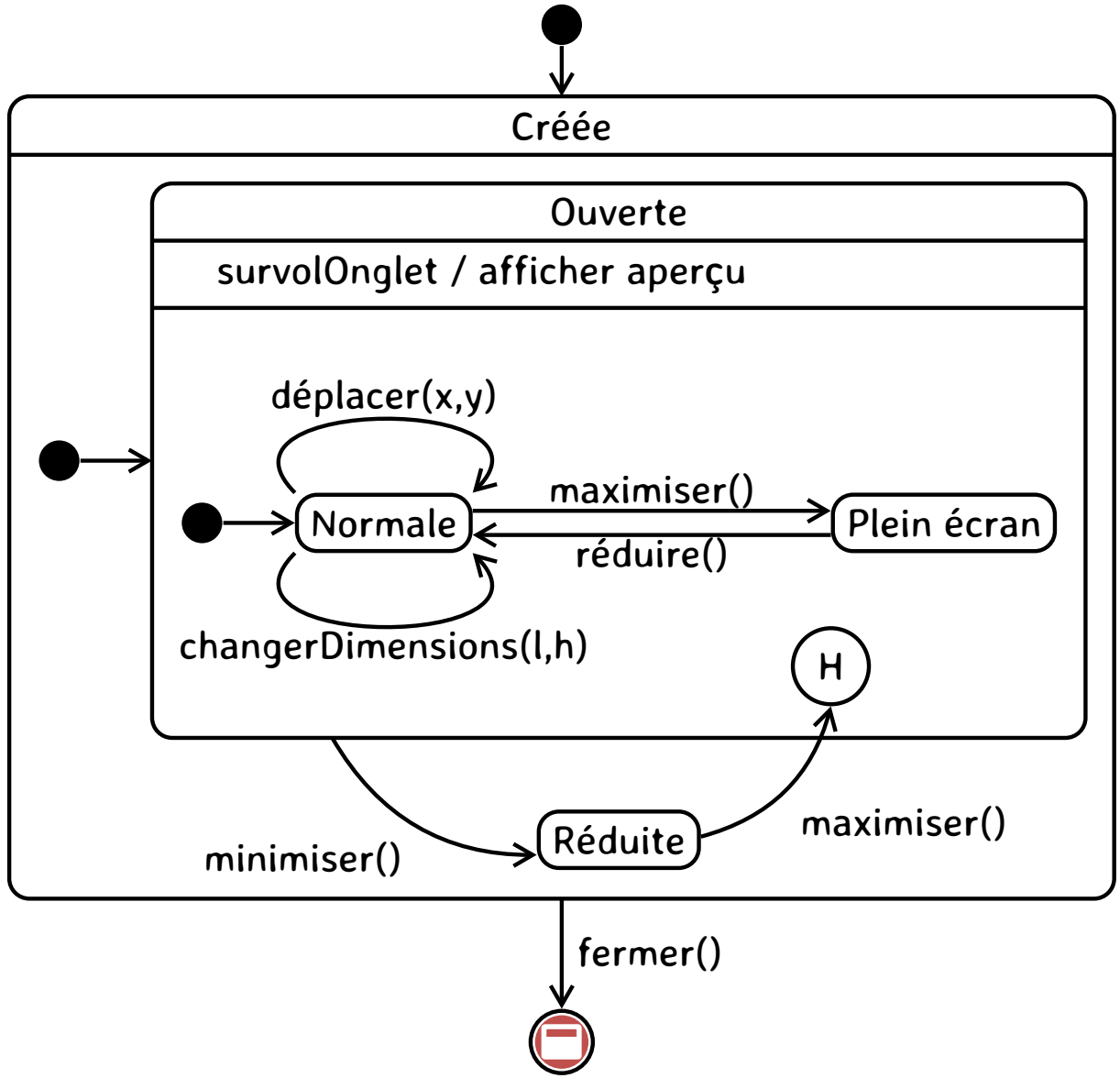
La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
→ fermer()



La classe fenêtre complète
Exemple

maximiser()
minimiser()
maximiser()
réduire()
déplacer(x,y)
minimiser()
maximiser()
fermer()



Quand utiliser les diagrammes d'état-transitions

- Outil méthodologique pour étudier :
  - le comportement des objets d'une classe conceptuelle ou technique ;
  - la succession des événements externes relatif à un cas d'utilisation.
- Outil de validation des autres diagrammes :

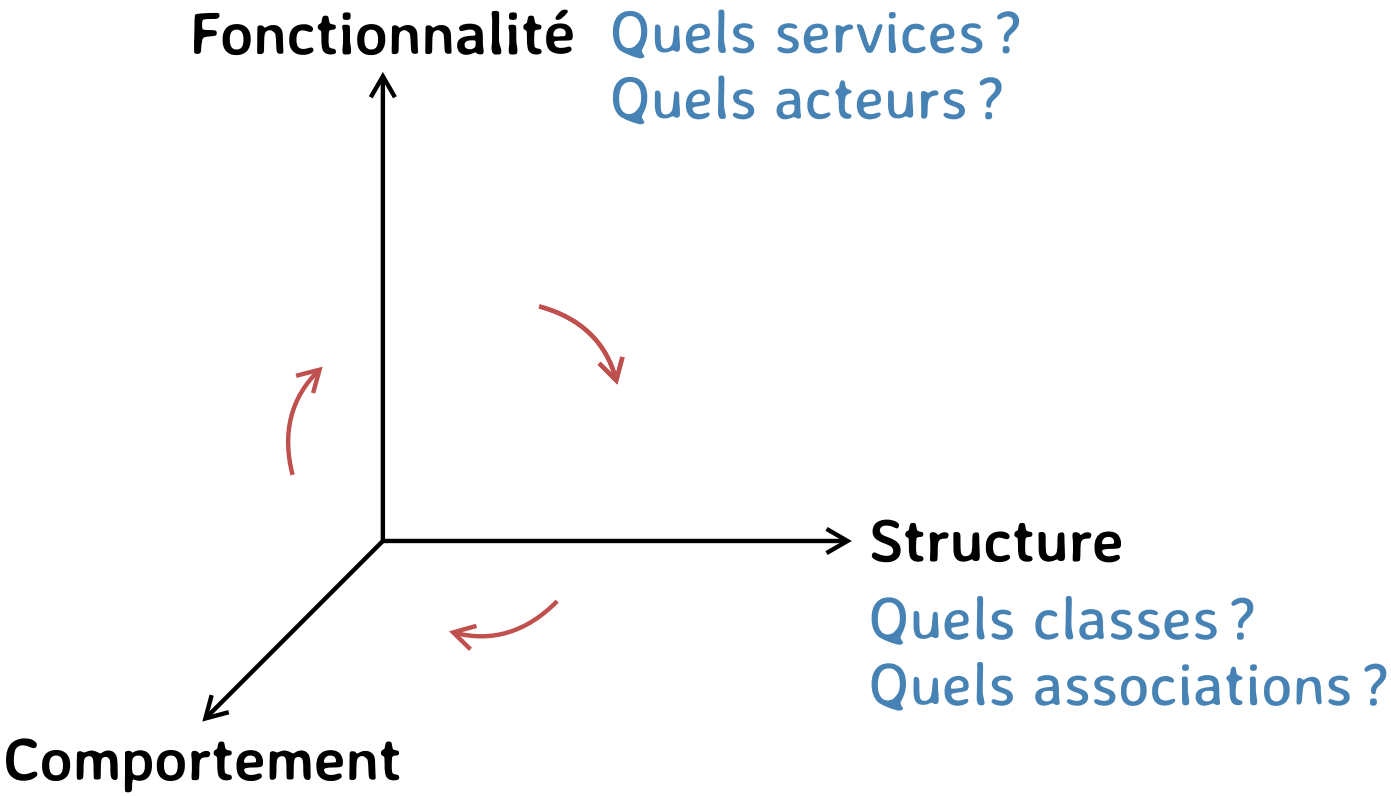
  Il est important de « croiser » les diagrammes états transitions, les diagrammes de séquence et les diagrammes de classes pour détecter des incohérences.
- Ne construire un diagramme d'états transitions que pour des classes dont le comportement est dépendant des états.

conseil

Il est préférable de limiter les liens entre niveaux hiérarchiques d'un automate en définissant systématiquement un état initial pour chaque niveau.

UML - La vue dynamique
Plan

L'approche UML



Les diagrammes d'interaction permettent de faire la liaison entre les vues fonctionnelle et statique : ils montrent comment les objets du système interagissent pour réaliser une exigence fonctionnelle.

Le point de vue dynamique

- Modéliser la **vue comportementale** d'un système :
  - Comment les objets de l'application interagissent-ils ?
- Il s'agit de représenter un système pendant son exécution.
  - On raisonne sur les **objets**.
  - Les objets qui composent une application **pendant son exécution** et leurs **échanges de messages** permettent à l'application de réaliser les fonctionnalités (i.e. cas d'utilisation) pour lesquelles elle est développée.

Diagrammes dynamiques

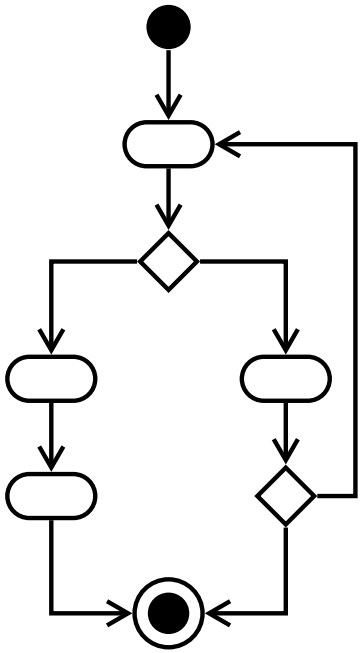


Diagramme d'états/transitions

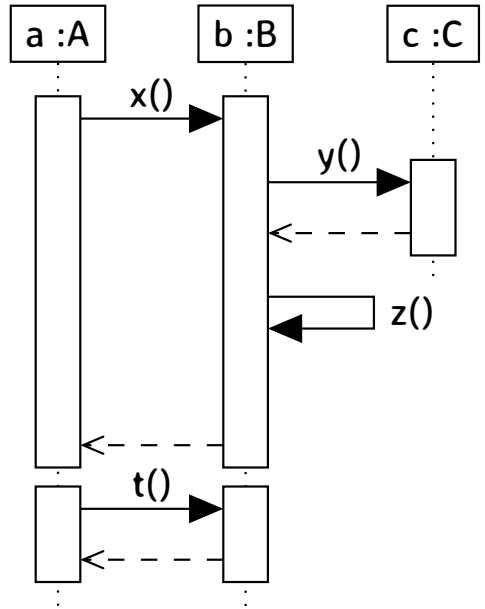


Diagramme de séquence

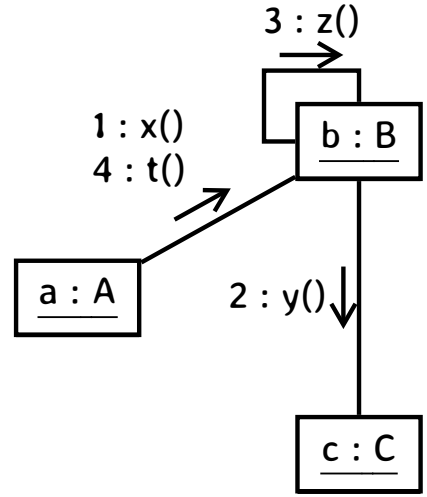


Diagramme de collaboration

Diagrammes d'interaction

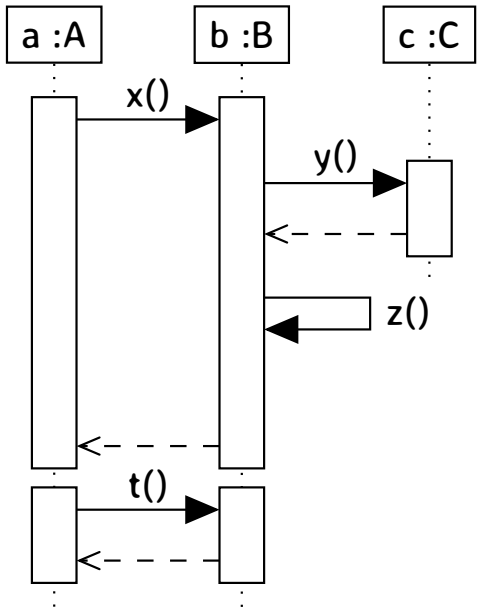


Diagramme de séquence

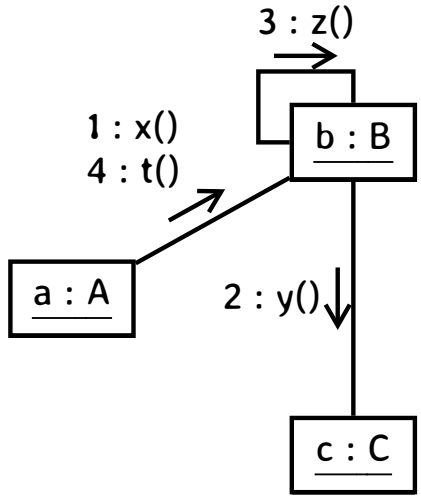


Diagramme de collaboration

☞ Ces deux types de diagramme sont équivalents sémantiquement.

Diagrammes d'interaction

met l'accent sur
la **chronologie**
des messages

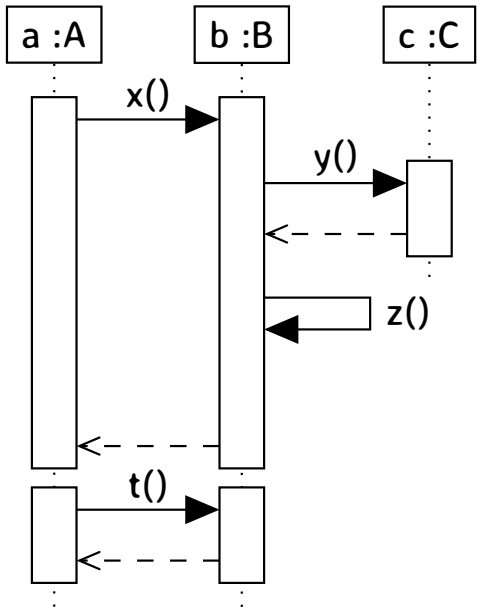


Diagramme de séquence

met l'accent sur
l'**organisation structurelle**
des objets qui émettent et
reçoivent des messages

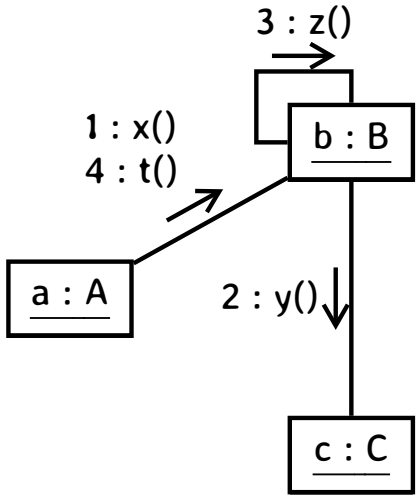


Diagramme de collaboration

☞ Ces deux types de diagramme sont équivalents sémantiquement.

Diagrammes d'interaction

met l'accent sur
la **chronologie**
des messages

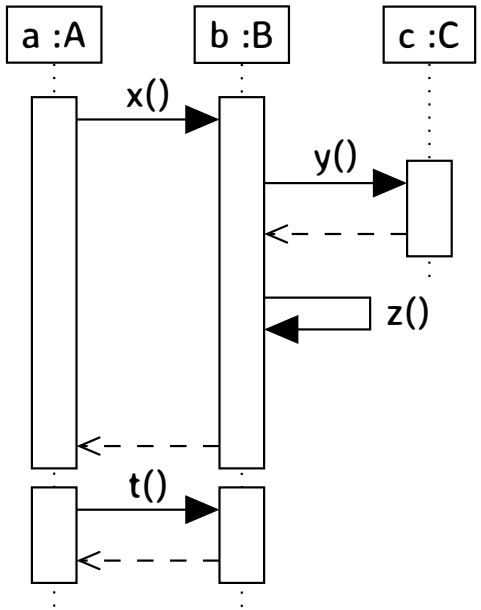


Diagramme de séquence

met l'accent sur
l'**organisation structurelle**
des objets qui émettent et
reçoivent des messages

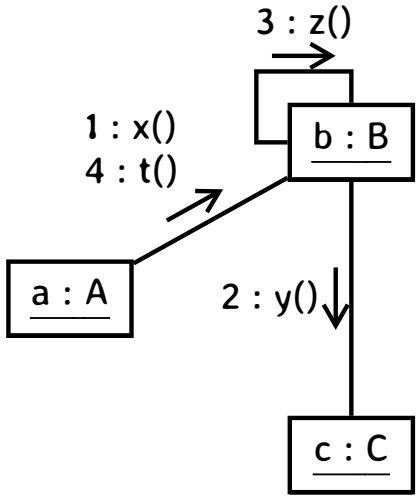


Diagramme de collaboration

☞ Ces deux types de diagramme sont équivalents sémantiquement.

Diagramme de collaboration

- Description des interactions entre les objets composant le système.
- Représentation se concentrant sur les relations d'interaction entre les objets.
- La dimension temporelle est ajoutée grâce à des numéros de séquence.
- Représente un ensemble de rôles joués par les objets dans un contexte particulier, ainsi que les liens entre ces objets.
- Les diagrammes de collaboration sont des **diagrammes d'interaction** comme les diagrammes de séquence.
- Le passage à un diagramme de séquences et inversement est simple.

Diagramme de collaboration
Utilisation

☞ **Documentation des cas d'utilisation :**

- description permettant de **réaliser** les cas d'utilisation.
- décrit le comportement du système pour chacun des scénarios accompagnant les cas d'utilisation.
- Facilite la rédaction des diagrammes des classes, des diagrammes état-transition, ...

☞ **Documentation conceptuelle :**

- description du comportement de classes et d'opérations.
- **Remarque :** si les opérations ont une structure algorithmique, on préférera les décrire avec des diagrammes d'activités. Les diagrammes de collaboration sont plus adaptés quand une opération fait intéragir de nombreux objets.

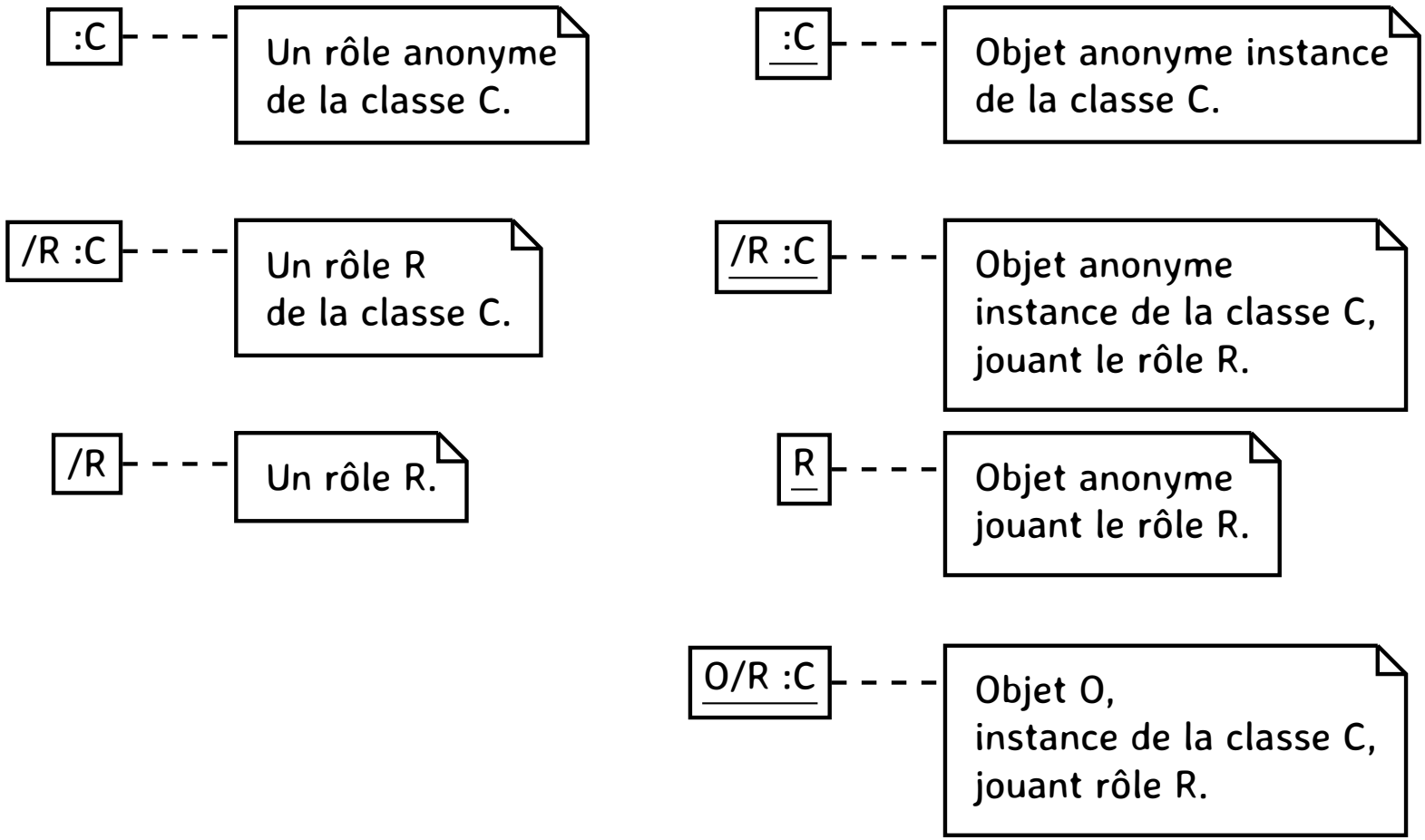
UML - La vue dynamique
Plan

Collaboration

- Définition des éléments utiles pour obtenir un résultat en spécifiant leurs **rôles** dans le contexte de la collaboration.
- Est composée de deux description :
  - description générale au niveau **spécification** qui représente :
    - les **rôles** des cas d'utilisations, des classes, des méthodes et des associations ;
    - une **interaction** : une séquence de messages partiellement ordonnés échangés entre les rôles.
  - description spécifique au niveau **instance** qui représente :
    - une instance particulière d'une interaction composés d'objets et de liens respectant les rôles, et de **stimulus** (instances de messages) échangés entre ces objets.

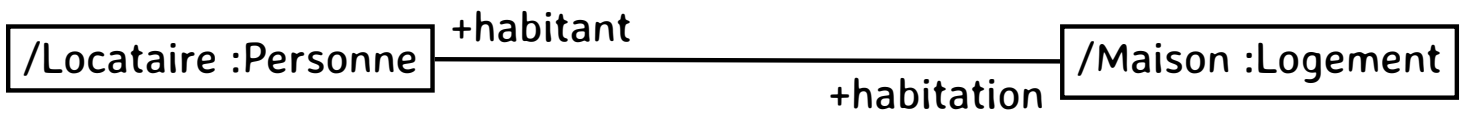
Collaboration
Rôles

- Chaque élément d'une collaboration joue un **rôle**.
- Les rôles des classificateurs (classes, cas d'utilisation, ...) sont représentés par des symboles de classe :



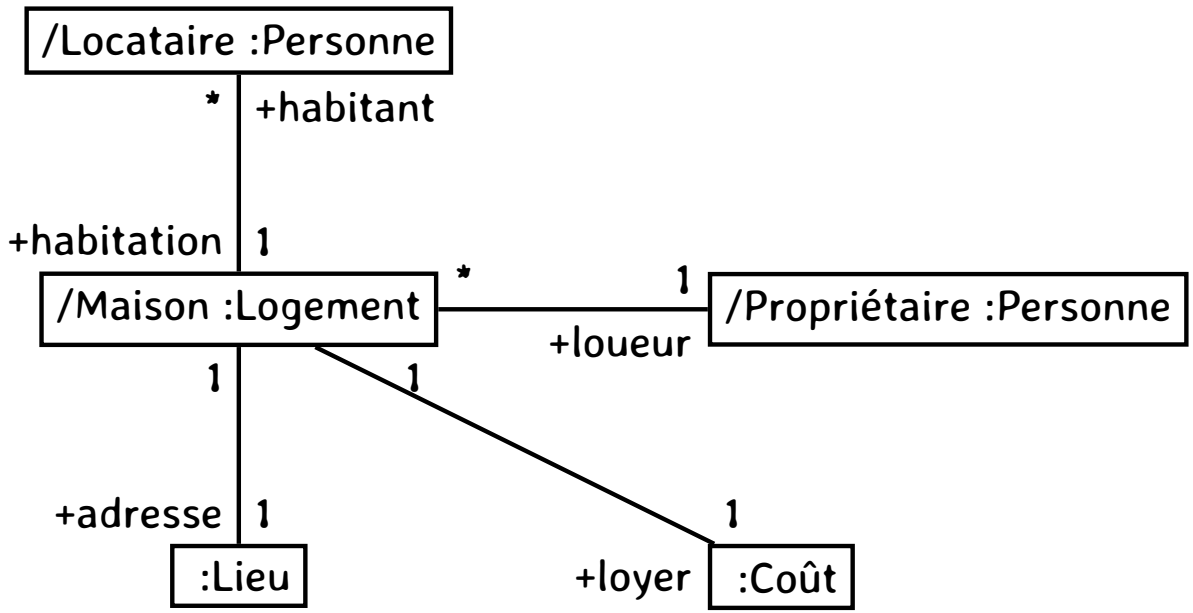
Collaboration
Rôles

Les rôles des associations sont des textes respectant la syntaxe des étiquettes d'associations (diagrammes de classes).



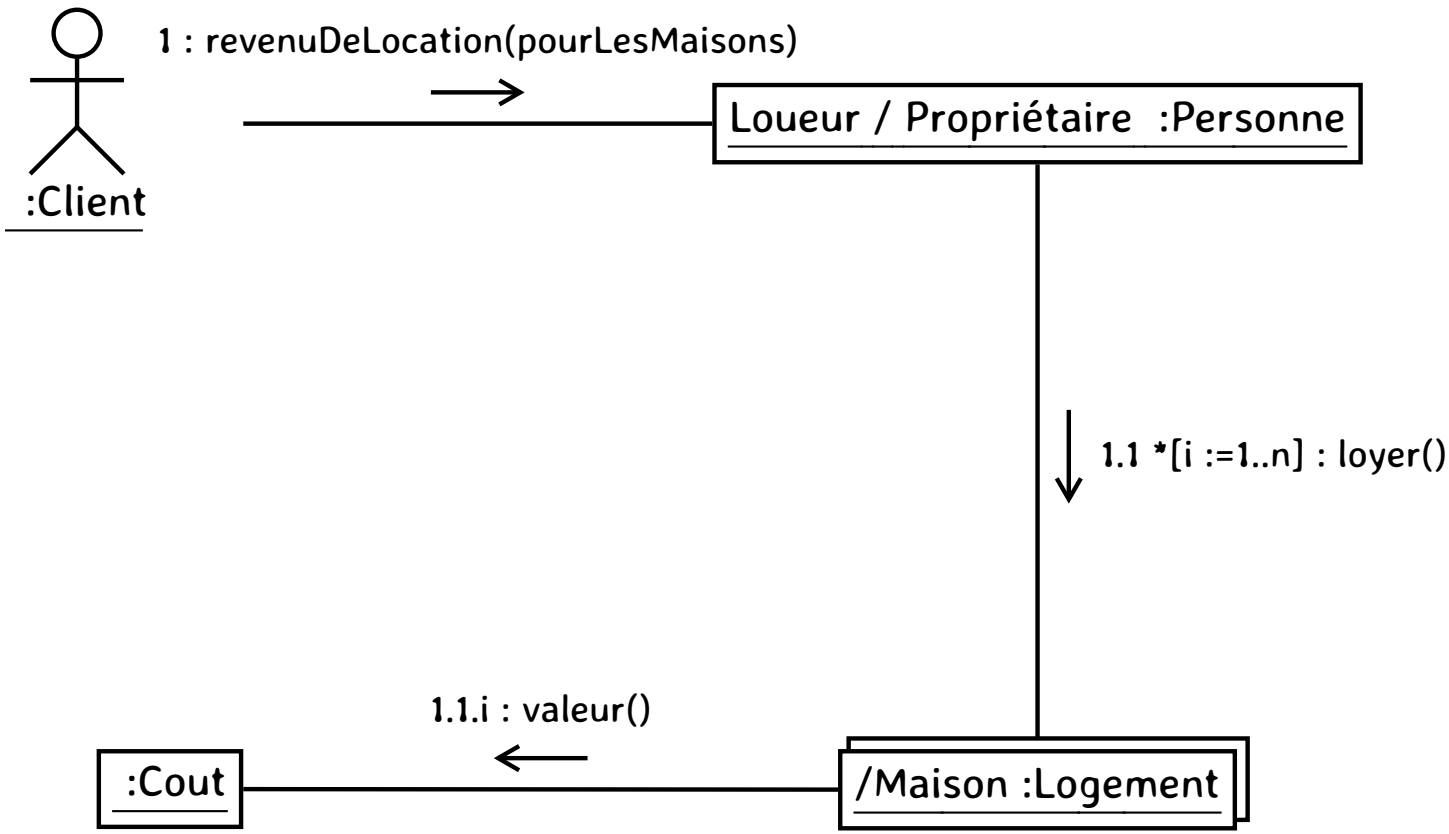
Collaboration
Au niveau spécification

- La collaboration forme un graphe de rôles liés par des rôles d'associations.
- En général, une collaboration au niveau spécification représente un **contexte**.



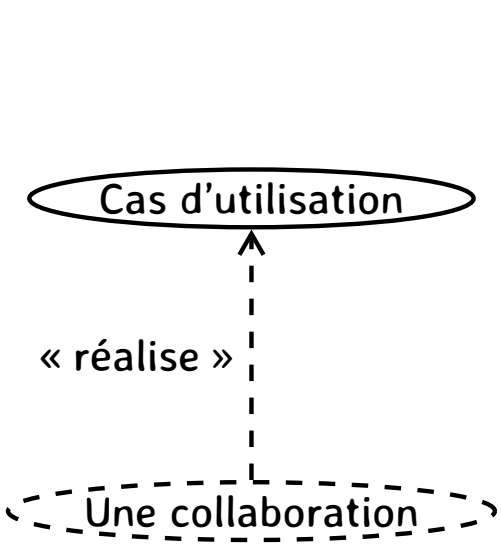
Collaboration
Au niveau instance

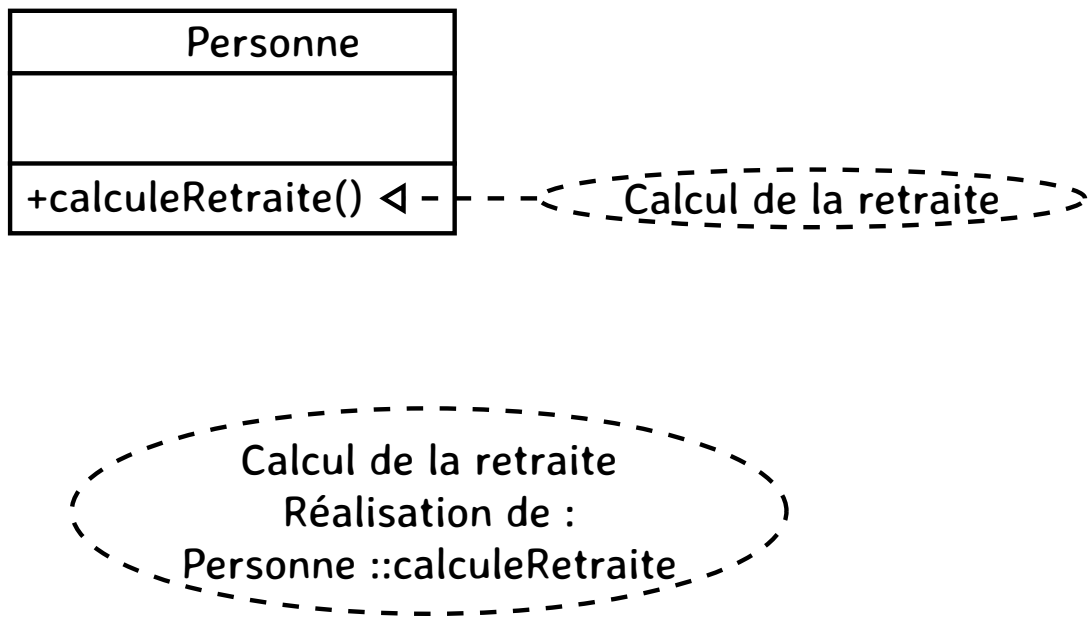
- Diagramme représentant une instance du diagramme au niveau spécification avec des **stimulus**.
- Stimulus : instance d'un message envoyé d'un objet vers un autre.



Collaboration
Représentation condensée

Dans UML, une collaboration est représentée comme suit :





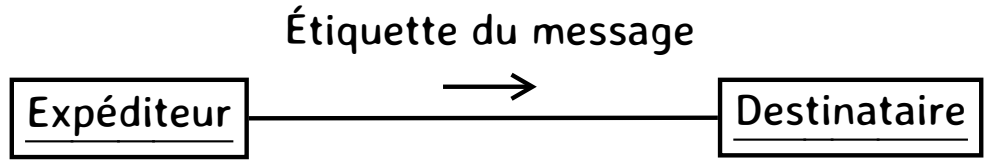
Document de cas d'utilisation

Document de classes/méthodes

UML - La vue dynamique
Plan

Messages

- Les objets communiquent en échangeant des messages représentés sous forme de flèches.
- Les messages sont étiquettés par le nom de l'opération ou du signal invoqué.
- L'envoi d'un message nécessite que le récepteur puisse réaliser l'opération.



Messages
Étiquettes

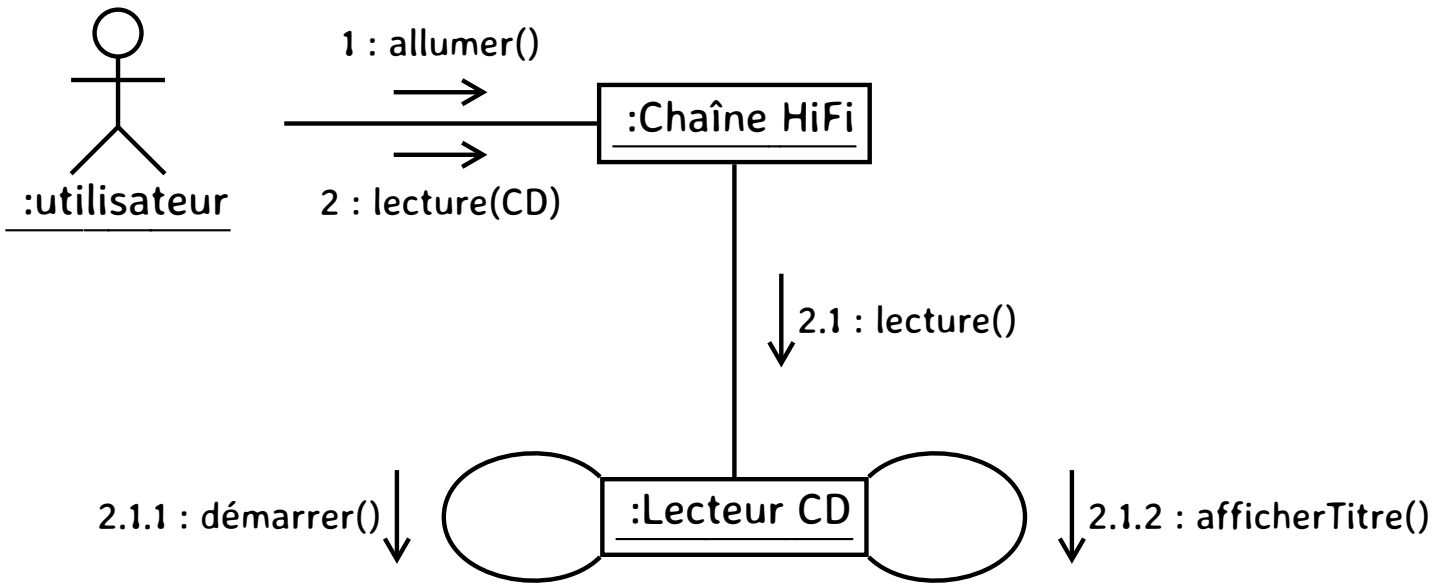
- Les étiquettes décrivent les messages auxquels elles sont attachées.
- Syntaxe générale :

```
['['garde']'] [séquence] [itération] [résultat:=] nomMessage ['('arguments')']
```

- nomMessage : nom de l'opération ou du signal invoqué par l'intermédiaire de ce signal
- garde : condition booléenne et optionnelle (représentée entre crochets) autorisant ou non l'envoi d'un message.

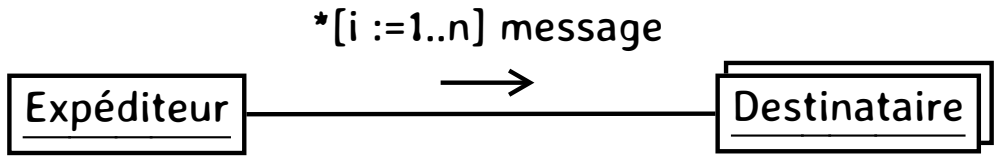
Messages
Séquence

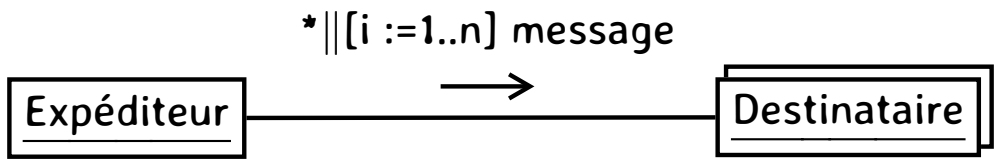
- Ensemble de numéros ordonnant l'envoi des messages (1 puis 2 puis 3 ...)
- Numérotation englobante (cas d'appels de procédure) : 2 (appel initial) puis 2.1 (premier appel imbriqué) puis 2.2 (second sous-appel) puis 3 (appel du mˆeme niveau que le numéro 2).



Messages
Itération

- Itération séquentielle : envoi **séquentiel** de n instances du même message.
  Syntaxe : `*[ clause d'itération ]`
- Itération parallèle : envoi **parallèle** de n instances du même message.
  Syntaxe : `*||[ clause d'itération ]`



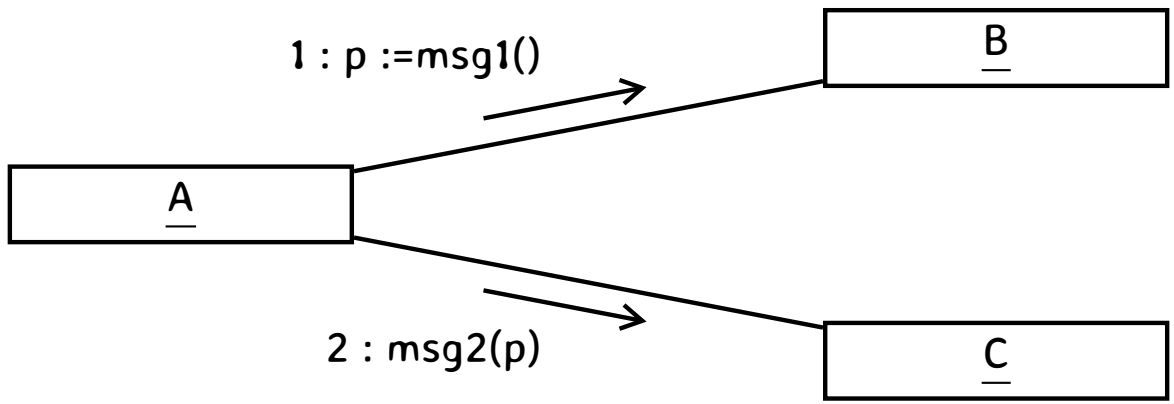


Messages
Arguments

- Liste des paramètres du message séparés par des virgules.
- Les arguments et le nom de l'action déterminent sans ambiguïté l'action à réaliser.
- Les arguments peuvent contenir des valeurs retournées par des messages envoyés précédemment.
- Exemples :
  - `Afficher(x, y)` : affiche les valeurs x et y.
  - `Soustraire(today, DoB)` : calculer le nombre de jours entre deux dates.

Messages
Résultat

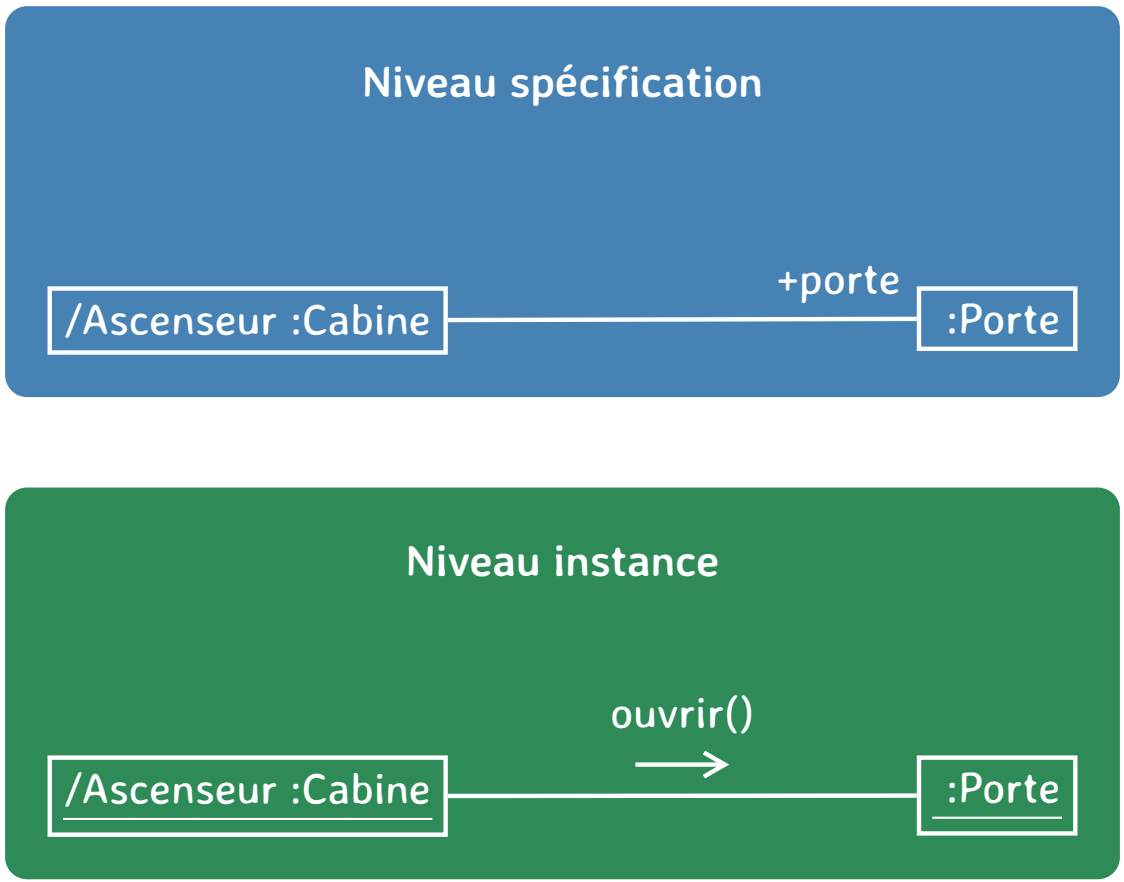
- Le résultat est constitué d'une liste de valeurs retournées par le message.
- Ces valeurs peuvent être utilisées comme paramètres des autres messages.



UML - La vue dynamique
Plan

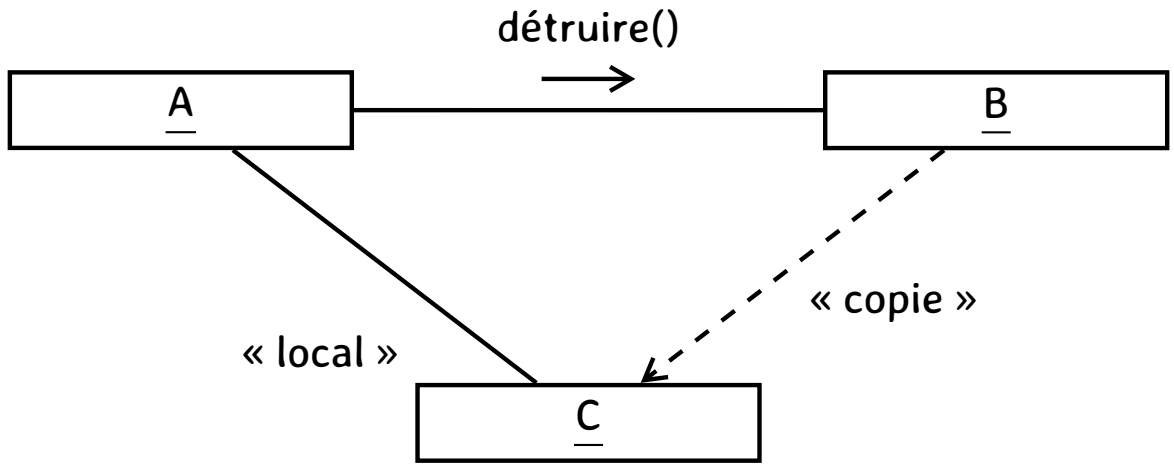
Interactions

- Définit la communication entre les instances des éléments d'une collaboration.
- Plusieurs interactions peuvent s'appliquer à la même collaboration pour exprimer divers comportements.
- Le contexte d'une interaction comprend les arguments, les variables locales, l'état des objets ainsi que les liens entre les objets qui participent à la collaboration.



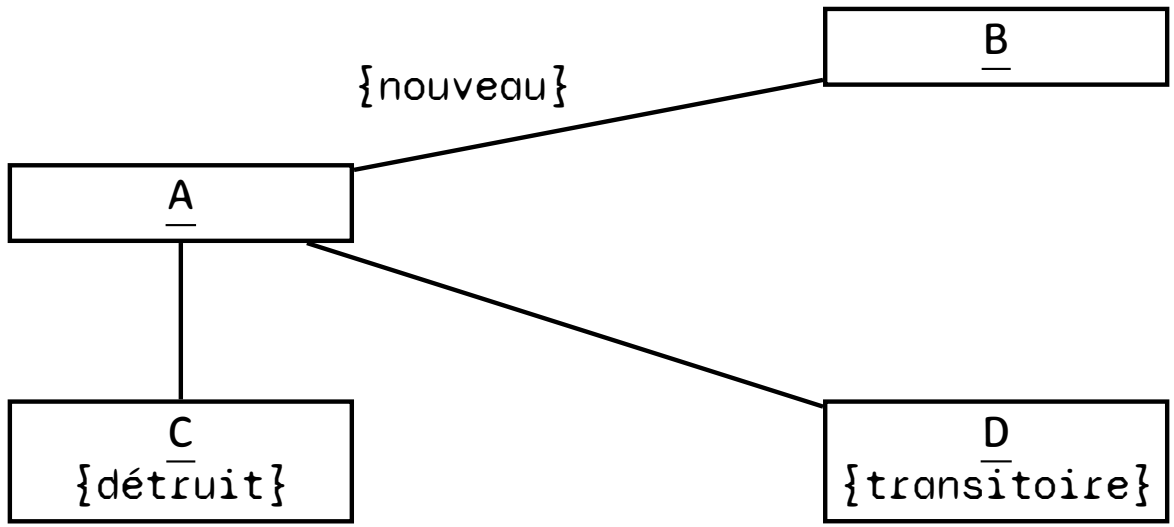
Interactions

Les diagrammes d'interactions montrent les interactions entre les objets et les relations structurelles permettant ces interactions.



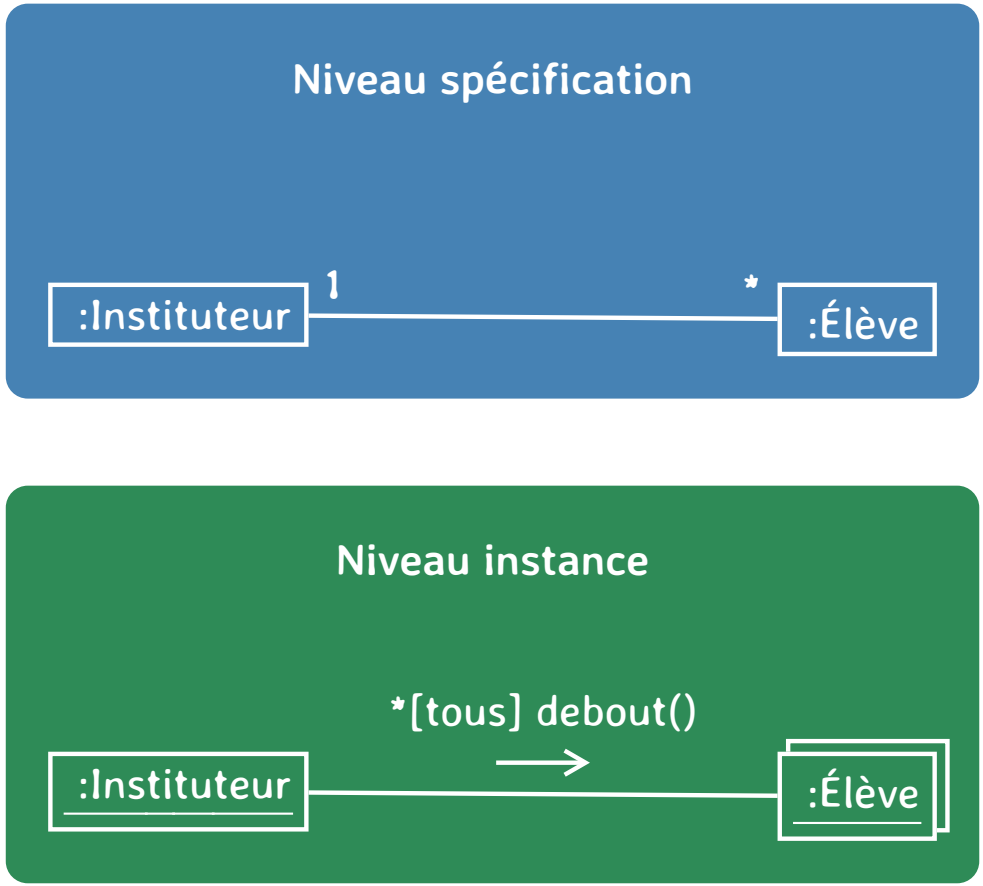
Interactions

- Les objets et les liens créés ou détruits au cours d'une interaction peuvent respectivement porter les contraintes {nouveau} ou {détruit}.
- Les objets créés, puis détruits au sein de la même interaction, portent la contrainte {transitoire}.



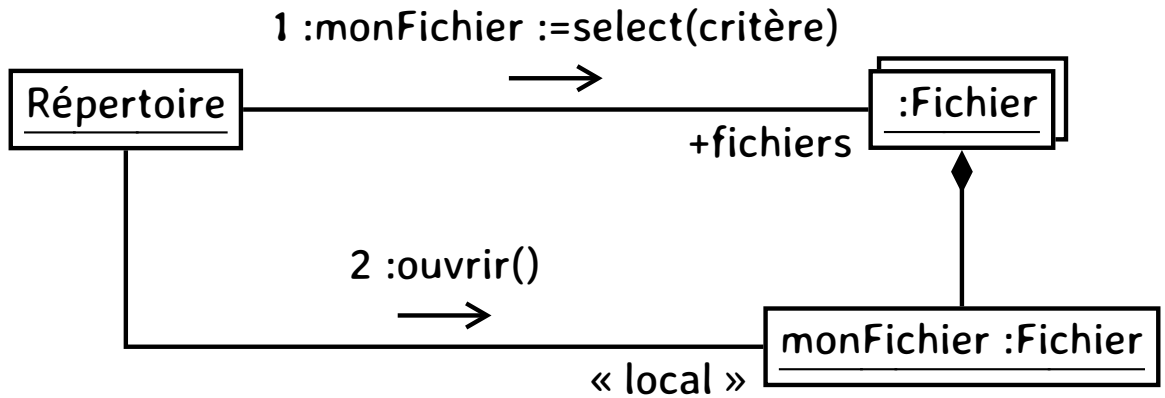
Interactions

- UML permet une représentation condensée d'un ensemble d'objets.
- Utile lorsque tous les objets de l'ensemble doivent être traités de manière uniforme.



Interactions

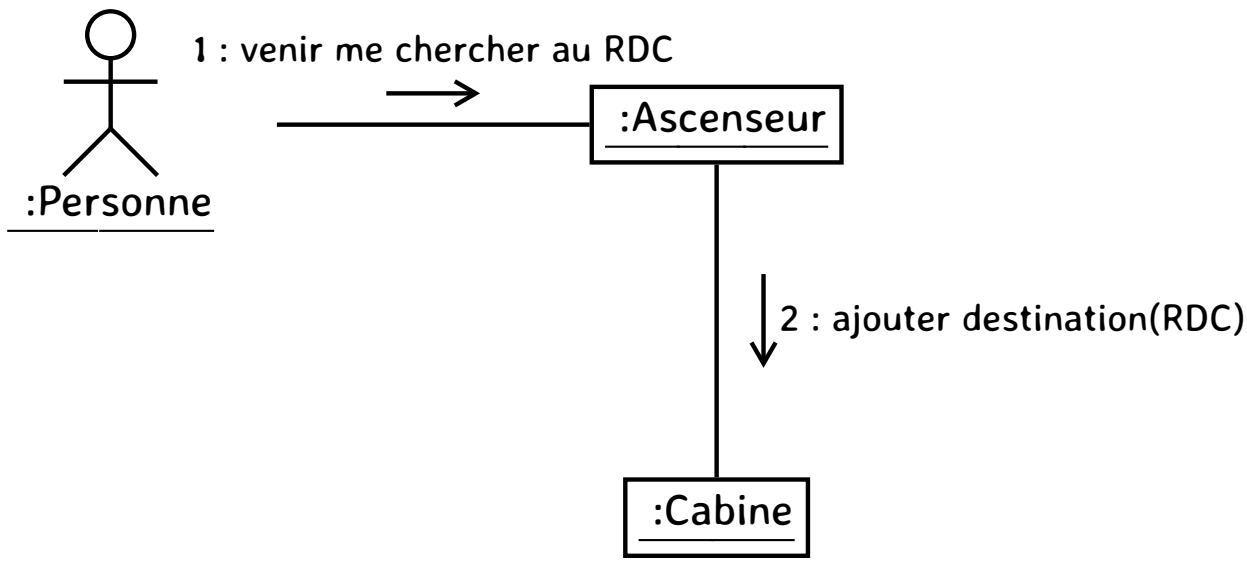
- Possibilité de représenter un objet particulier appartenant à un groupe afin de lui appliquer un message particulier
- Représentation à l'aide d'une composition indiquant que l'objet fait parti de l'ensemble d'objets.



UML - La vue dynamique
Plan

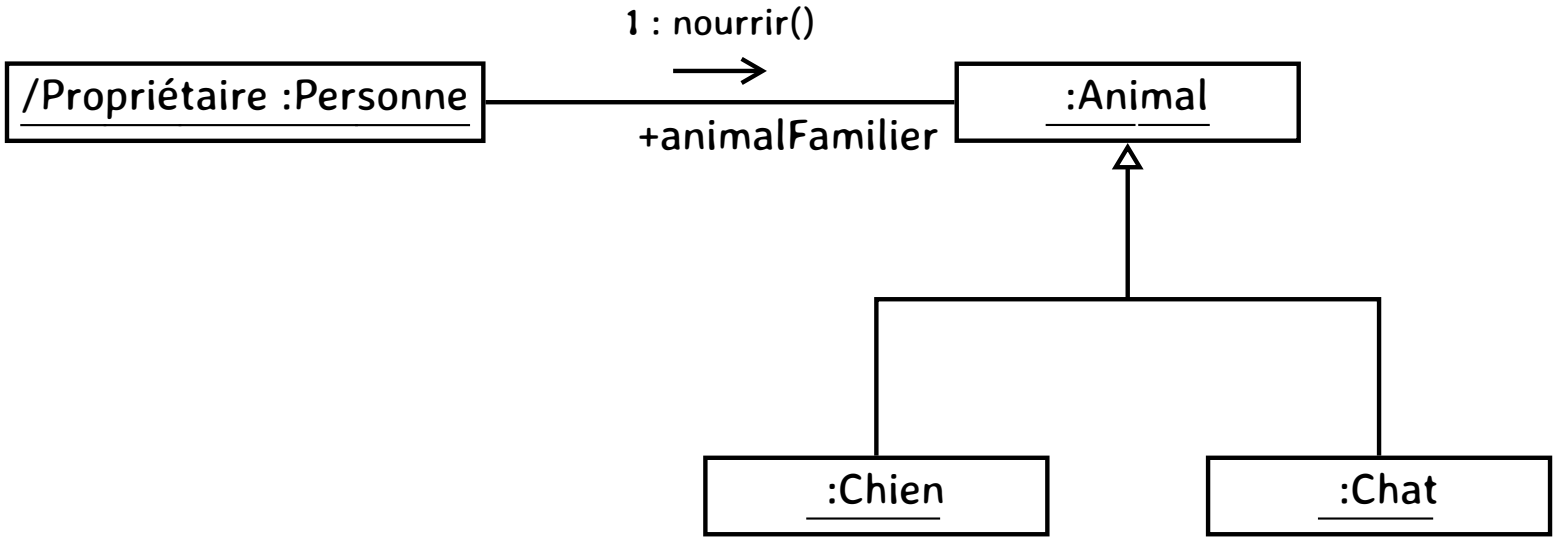
Représentation des acteurs

- La notation UML permet de faire figurer un acteur dans les diagrammes de collaboration.
- Ils permettent de représenter les intéractions déclenchées par un élément extérieur au système.
- Le **premier message** est envoyé par l'acteur.



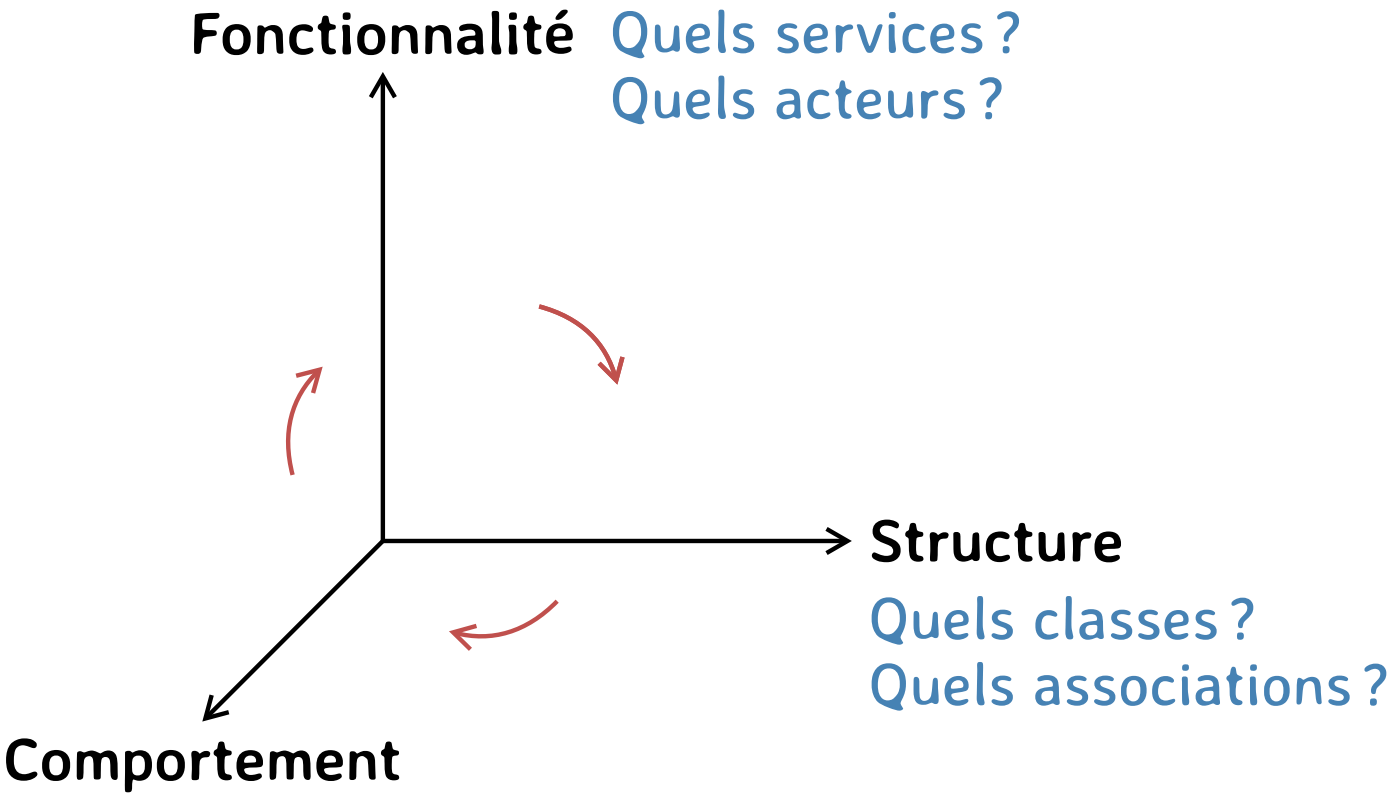
Classes abstraites et interface

- Les classes abstraites et les interfaces peuvent figurer dans les diagrammes de collaboration.
- Elles représentent des informations complémentaires : liens polymorphes, réalisation d'interfaces, ...



UML - La vue dynamique
Plan

L'approche UML



Les diagrammes d'interaction permettent de faire la liaison entre les vues fonctionnelle et statique : ils montrent comment les objets du système interagissent pour réaliser une exigence fonctionnelle.

Le point de vue dynamique

- Modéliser la **vue comportementale** d'un système :
  - Comment les objets de l'application interagissent-ils ?
- Il s'agit de représenter un système pendant son exécution.
  - On raisonne sur les **objets**.
  - Les objets qui composent une application **pendant son exécution** et leurs **échanges de messages** permettent à l'application de réaliser les fonctionnalités (i.e. cas d'utilisation) pour lesquelles elle est développée.

Diagrammes d'interaction

met l'accent sur
la **chronologie**
des messages

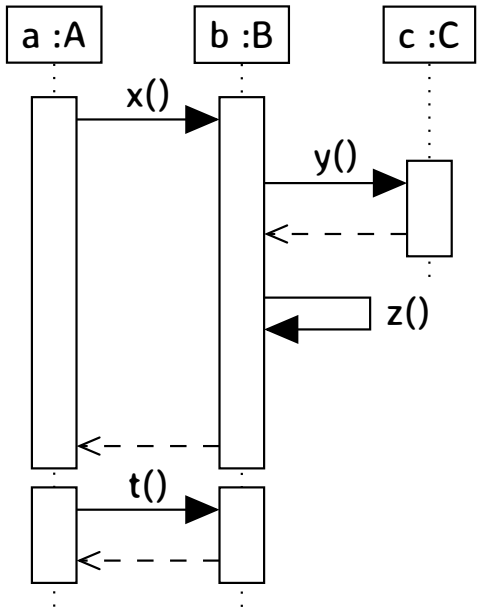


Diagramme de séquence

met l'accent sur
l'**organisation structurelle**
des objets qui émettent et
reçoivent des messages

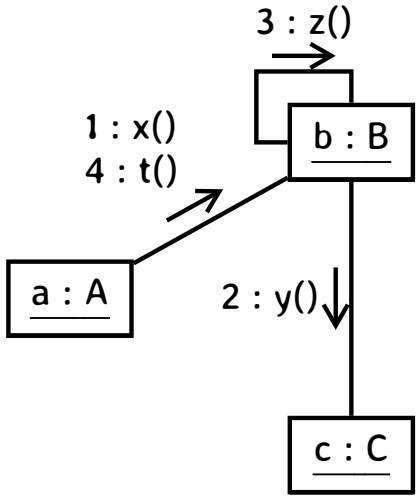


Diagramme de collaboration

☞ Ces deux types de diagramme sont équivalents sémantiquement.

Diagrammes d'interaction

met l'accent sur
la **chronologie**
des messages

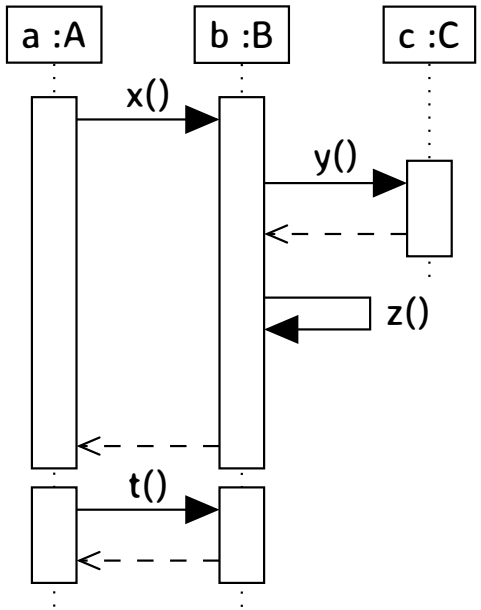


Diagramme de séquence

met l'accent sur
l'**organisation structurelle**
des objets qui émettent et
reçoivent des messages

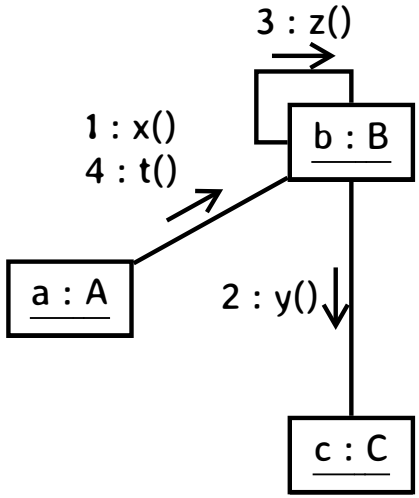


Diagramme de collaboration

☞ Ces deux types de diagramme sont équivalents sémantiquement.

Le diagramme de séquence

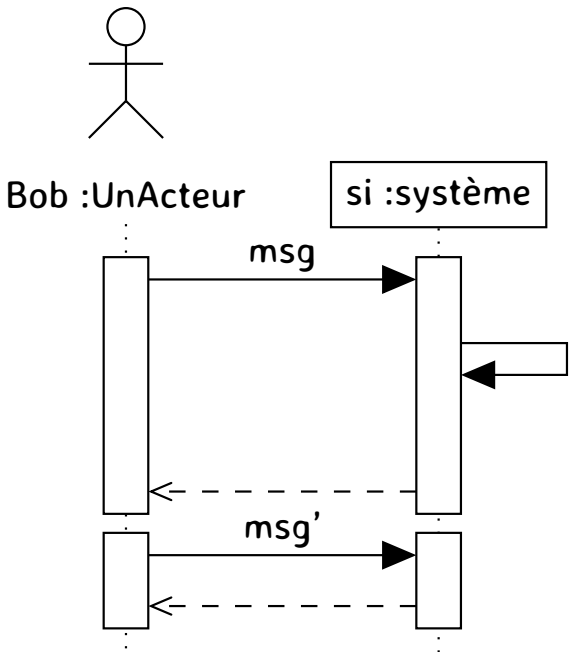
- Utilisés à 2 niveaux :

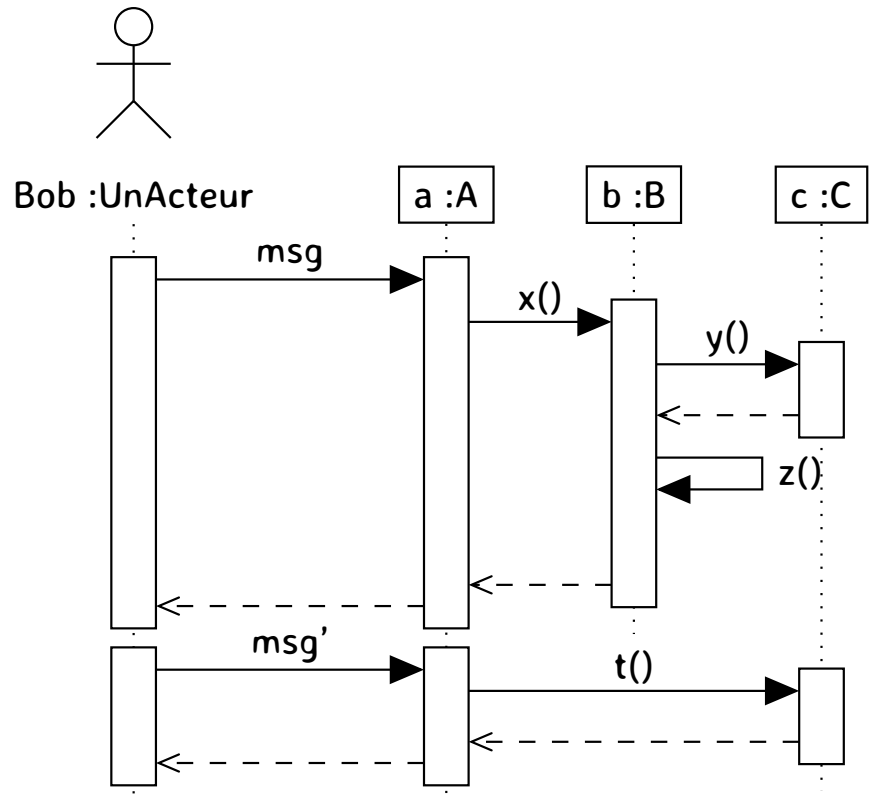
  Analyse et conception

  - **Analyse :** Entre un acteur et le système (Diagramme séquence système DSS)
  - **Conception :** Entre des objets « internes » du système
- Règles de cohérence avec le diagramme de classes
- 3 concepts essentiels : l'**interaction**, l'**objet** et le **message**.

Le diagramme de séquence

La construction d'un diagramme de séquence revient à remplacer dans le DSS le système (la boîte noire) par une collaboration d'objets.

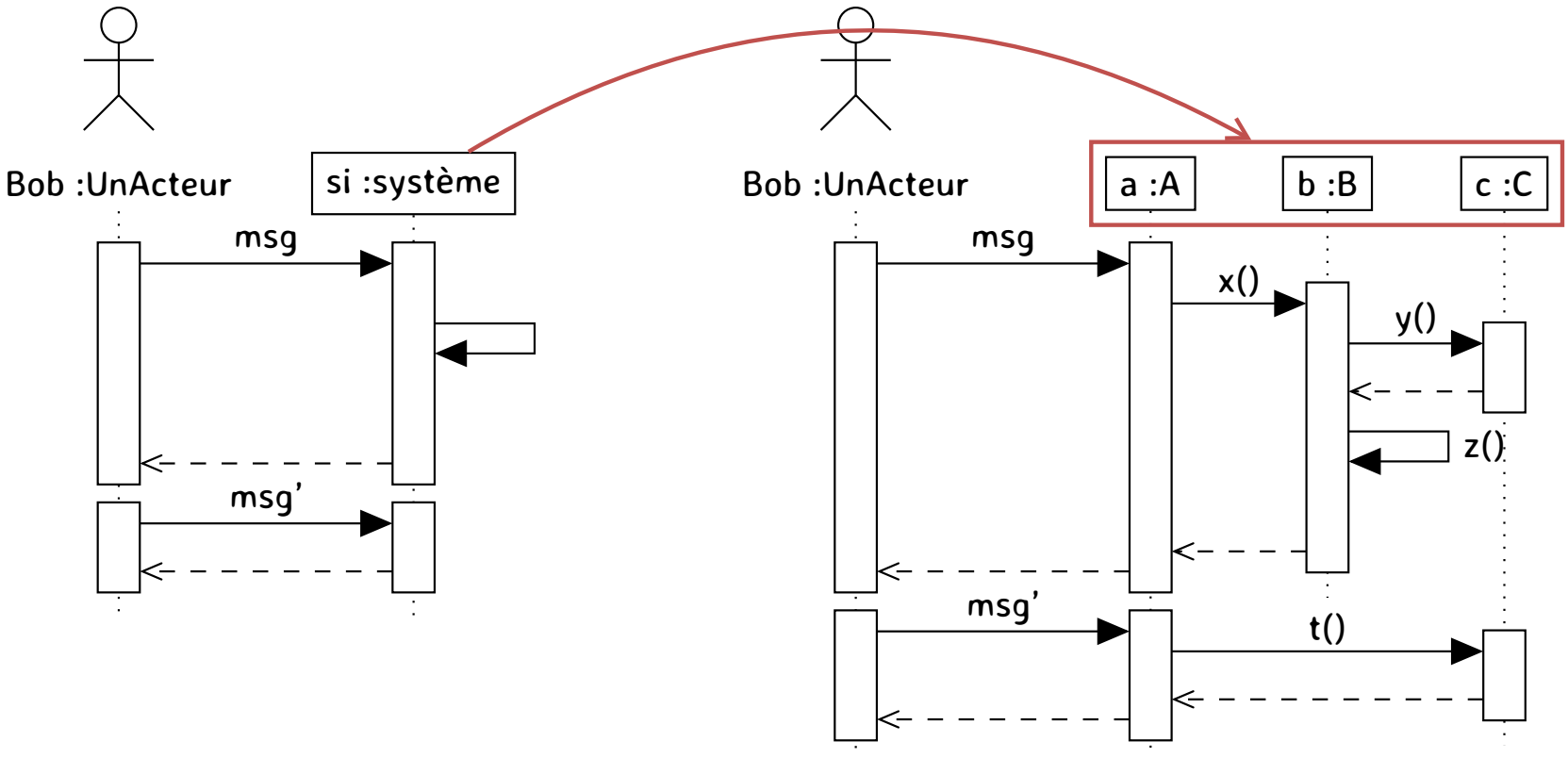




- Quels objets ?
- Comment doivent-ils interagir pour réaliser le cas d'utilisation ?

Le diagramme de séquence

La construction d'un diagramme de séquence revient à remplacer dans le DSS le système (la boîte noire) par une collaboration d'objets.



- Quels objets ?
- Comment doivent-ils interagir pour réaliser le cas d'utilisation ?

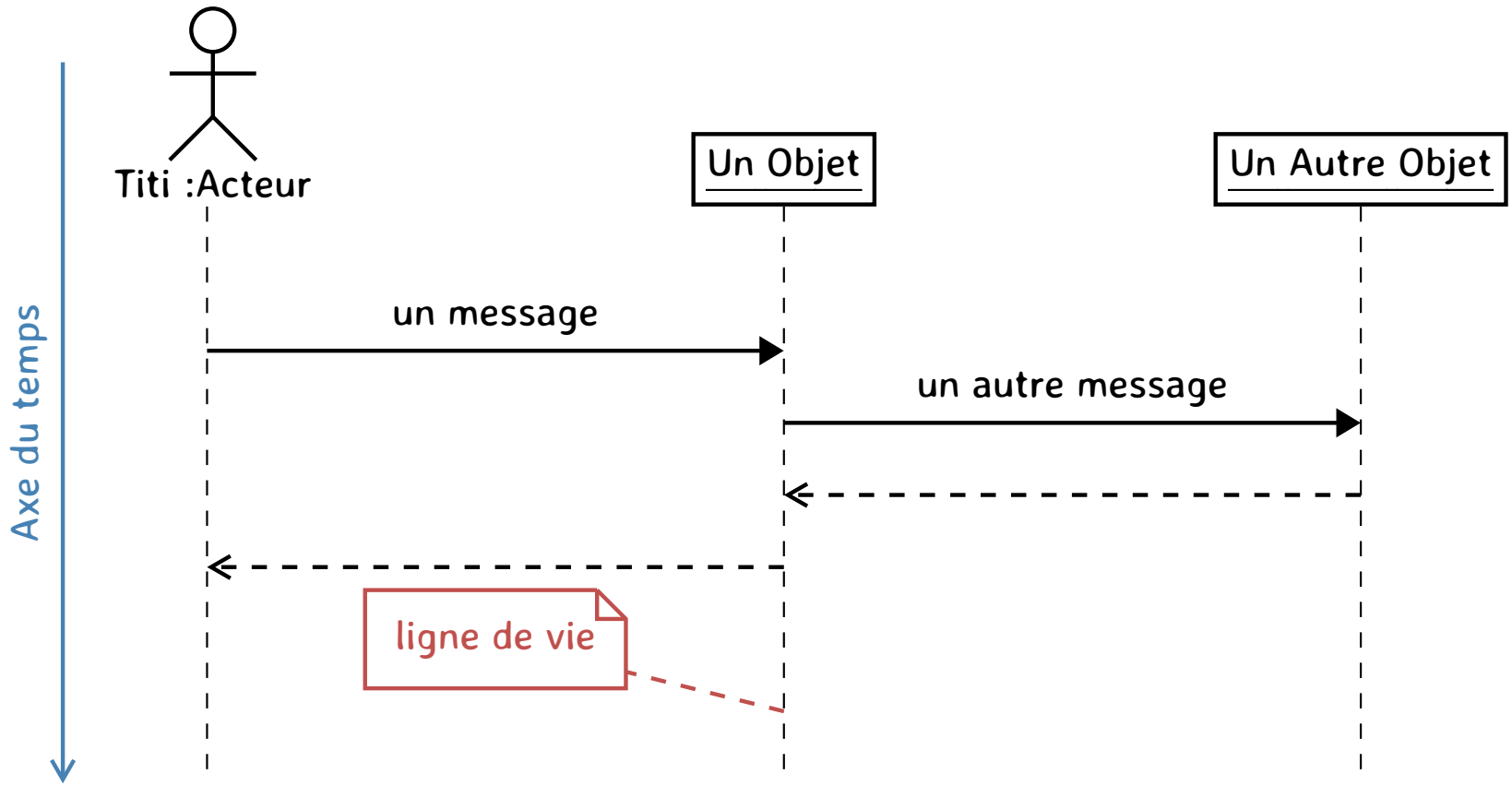
UML - La vue dynamique
Plan

Le diagramme de séquence
Interaction

- Une **interaction** décrit les **messages** échangés par un ensemble d'**objets** dans un certain contexte pour accomplir une certaine tâche.
  - Le contexte est en général celui d'un cas d'utilisation.
- Les **objets** considérés peuvent être des acteurs, des objets du système ou le système lui-même.
- Les **messages** représentent une communication unidirectionnelle entre objets qui transporte de l'information avec l'intention de déclencher une réaction chez le récepteur.

Le diagramme de séquence

Dans une application O.O. les objets communiquent par échanges de messages.



Le diagramme de séquence
Message

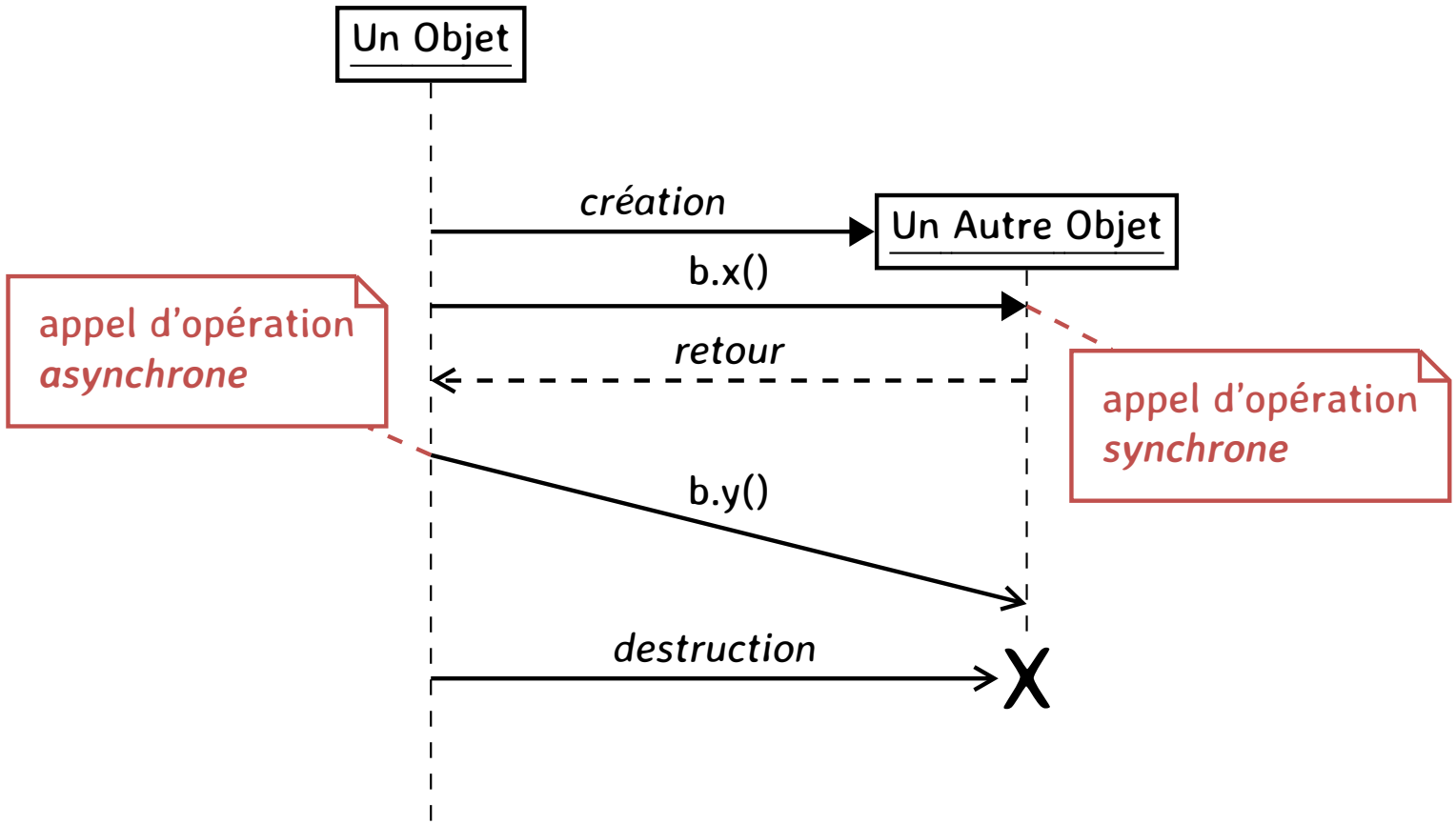
☞ Plusieurs types de message :

- Appel d'opération : demande à un objet ou à lui-même d'exécuter une opération
- Retour : retourne une valeur à l 'appelant
- Création : crée un objet
- Destruction : détruit un objet

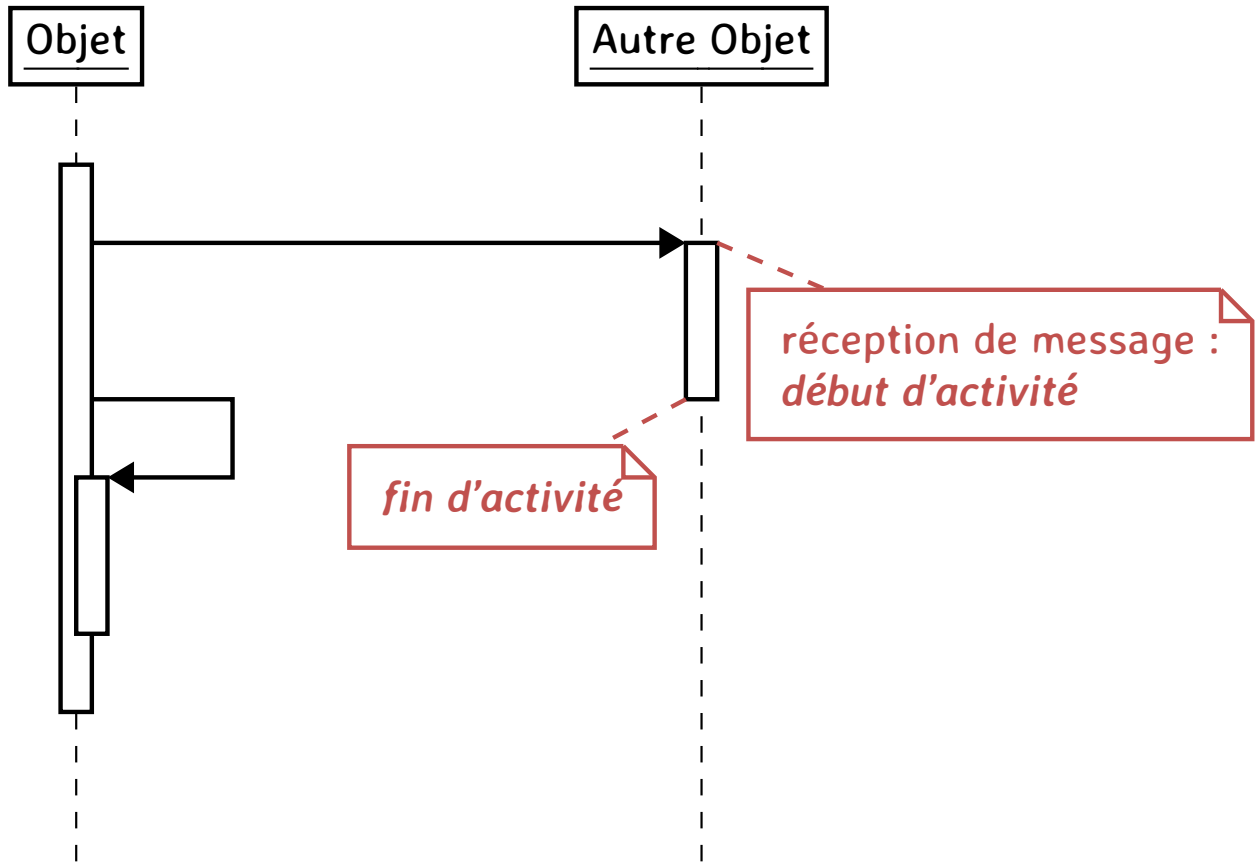
Le diagramme de séquence
Message

- Plusieurs types de message :
  - Appel d'opération : demande à un objet ou à lui-même d'exécuter une opération
  - Retour : retourne une valeur à l 'appelant
  - Création : crée un objet
  - Destruction : détruit un objet
- Message synchrone et message asynchrone
  - Les envois synchrones pour lesquels l'émetteur est bloqué et attend que l'appelé ait fini de traiter le message
  - Les envois asynchrones pour lesquels l'émetteur n'est pas bloqué et peut continuer son exécution

Le diagramme de séquence
Messages



Le diagramme de séquence
Périodes d'activité



Le diagramme de séquence
Description détaillée des messages

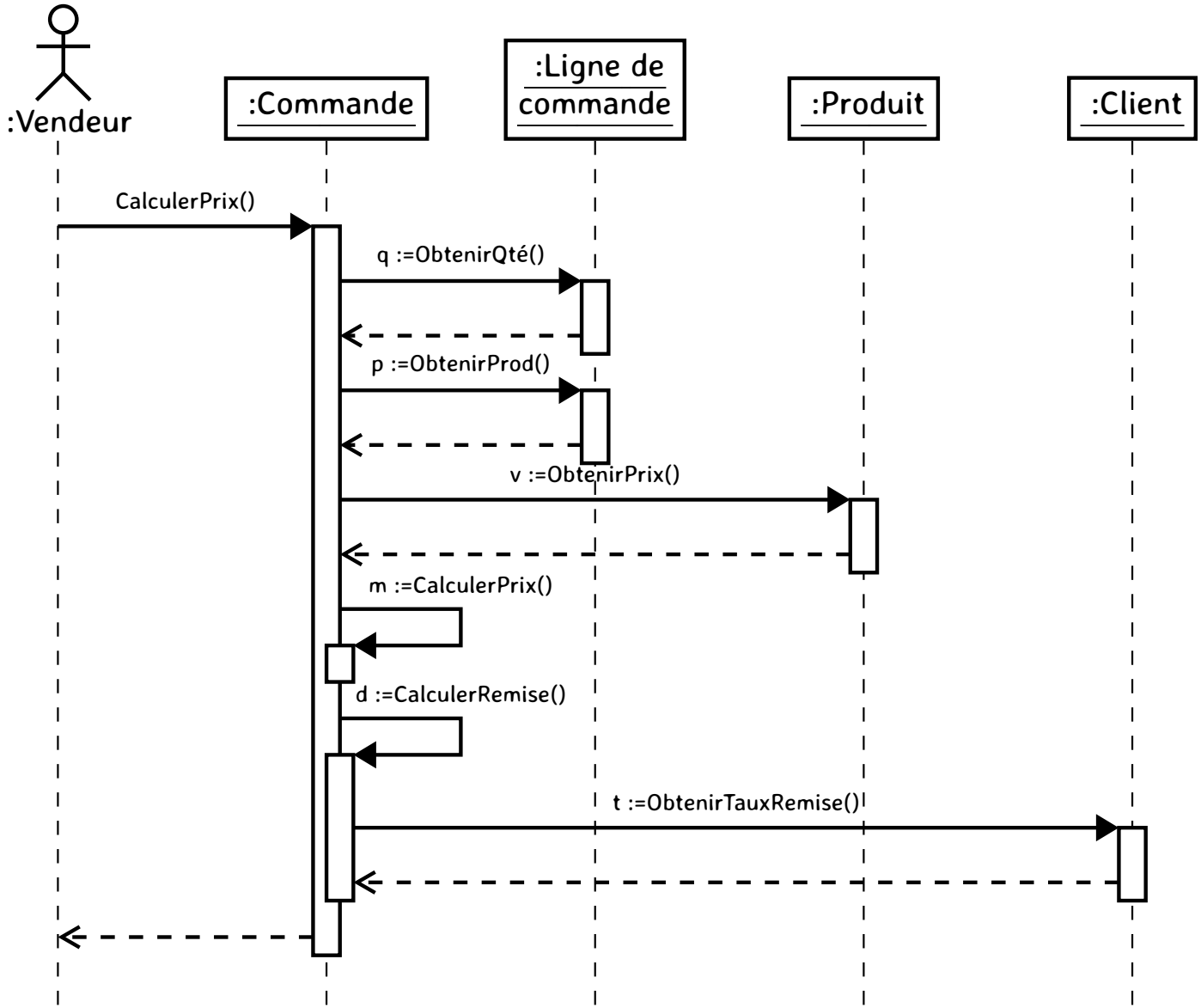
Il s'agit de la liste des paramètres du message. Les arguments et le nom du message identifient de manière unique l'action qui doit être déclenchée dans l'objet destinataire.

```
 <message> ::= [retour:= ] message [(<argument>,…)] [: typeRetour]
<argument> ::= <nom-paramètre> = [<valeur de l'argument>][:type]
```

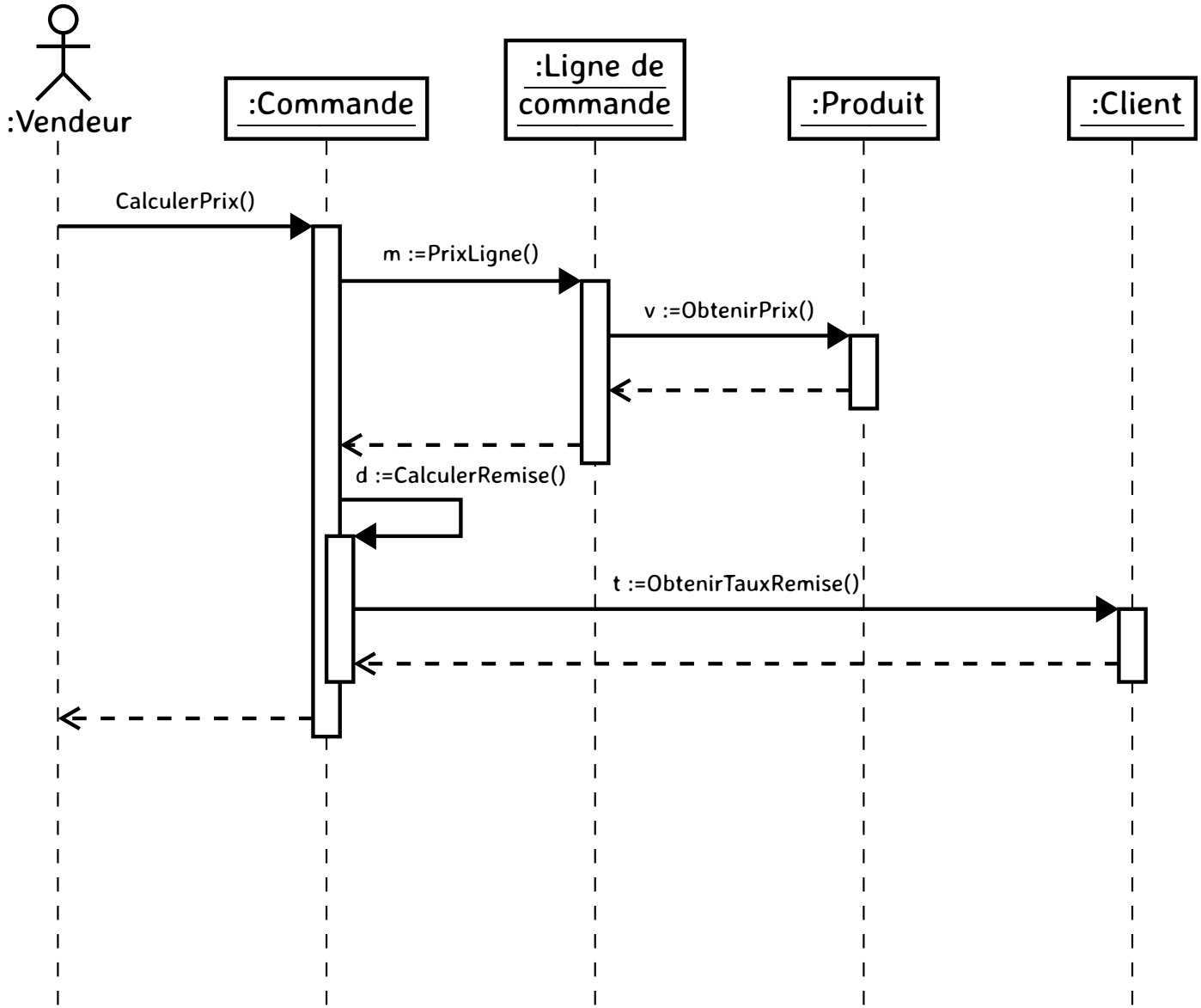
Exemples :

- `Afficher(X, Y)`
- `Initialiser(x=100)`
- `SouhaiterAnniversaire(roiDeLaFête=Paul:Personne,âge=35:Entier):Carte`
- `Quantité := CombienTEnVeux() : Nombre`

Le diagramme de séquence
Exemple

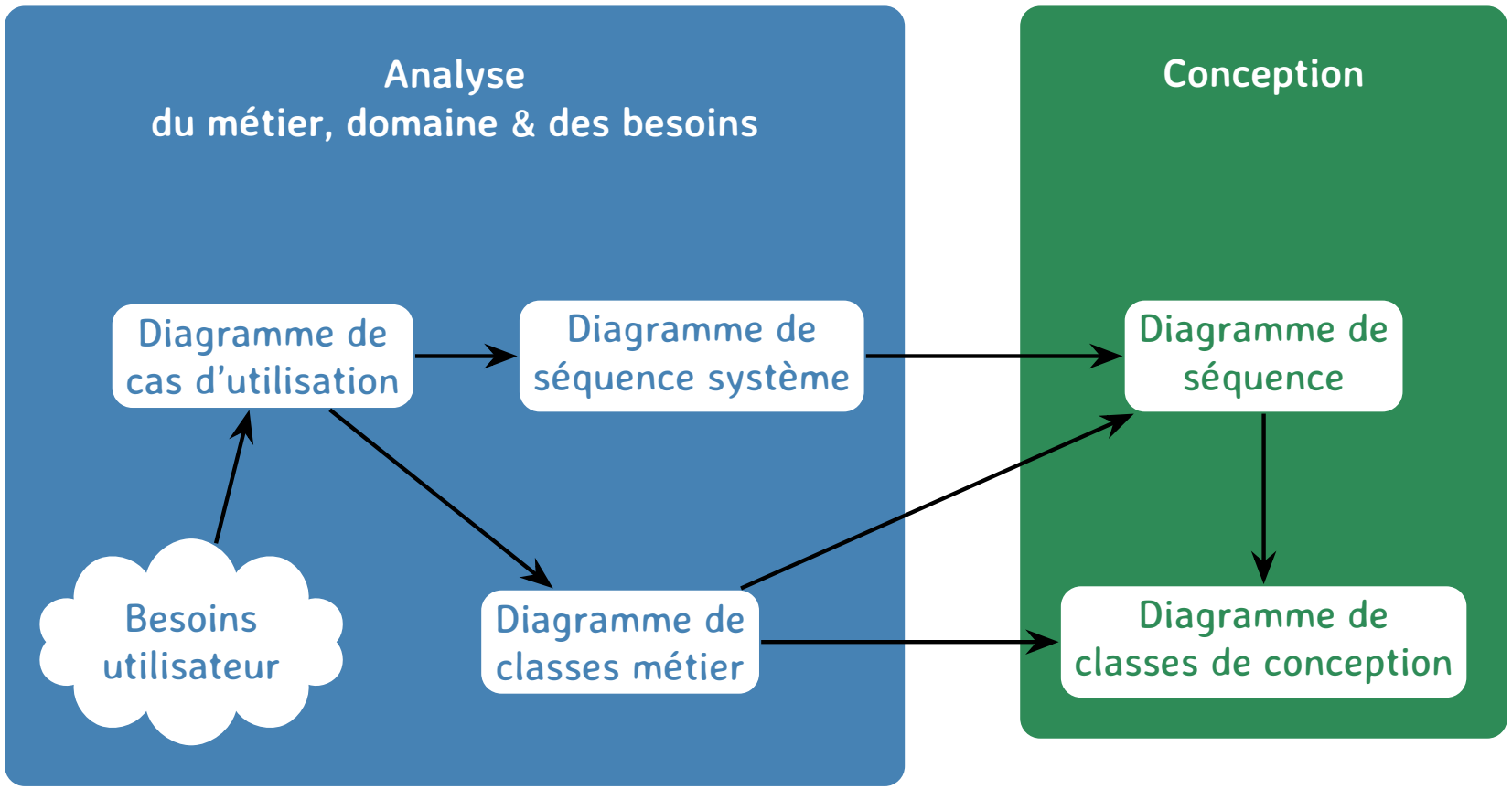


Le diagramme de séquence
Exemple



UML - La vue dynamique
Plan

DS : analyse et conception
Processus de développement



DS : analyse et conception

☞ Le DS au niveau **analyse** : **Diagramme de séquence système (DSS)**.

- Il s'agit d'un diagramme de séquence particulier :
  - acteur principal, acteurs secondaires et système,
  - messages entre acteurs et système.
- On construit un (ou plusieurs) DSS par cas d'utilisation.

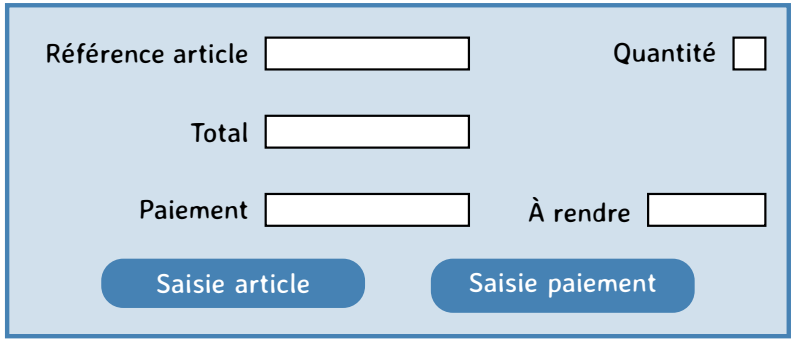
☞ Le D.S. au niveau **conception** : quels sont les **objets** du système ?

Basé sur une typologie des classes d'analyse

- **Classes « IHM » ou « présentation »**
  permettent les interactions entre le système et les utilisateurs du système : les formulaires de saisie, les résultats de recherche
- **Classes « contrôleur »**
  font la transition entre les classes IHM et les classes métier, ce sont des objets qui permettent de manipuler des informations détenues par plusieurs objets métier.
- **Classes « entité »**
  englobent les objets métier provenant du domaine, ce sont les entités qui doivent persister

Le D.S. au niveau conception
Cas d'une caisse enregistreuse

Présentation IHM

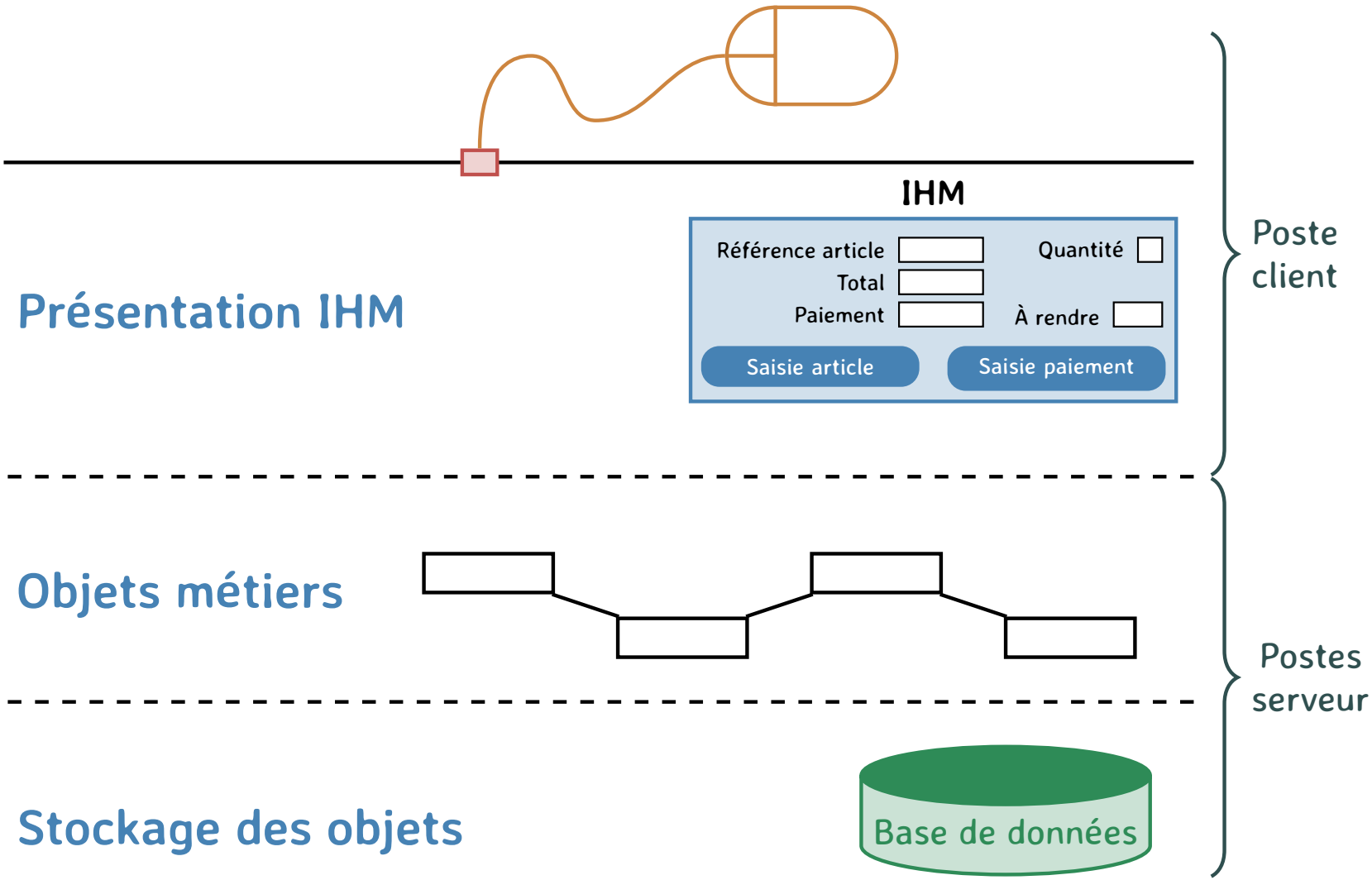


Objets métiers

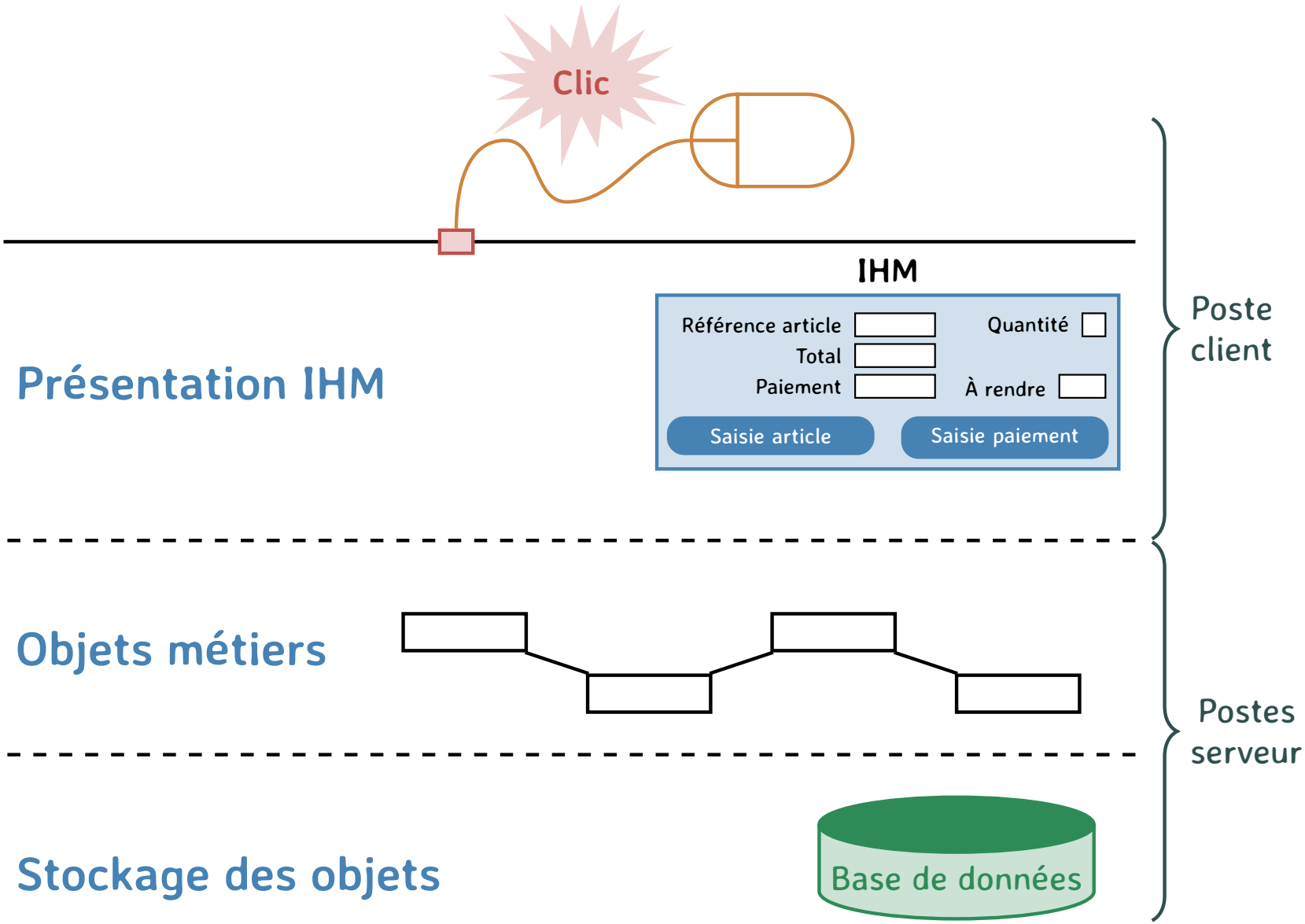
Traiter le passage en caisse

Stockage des objets

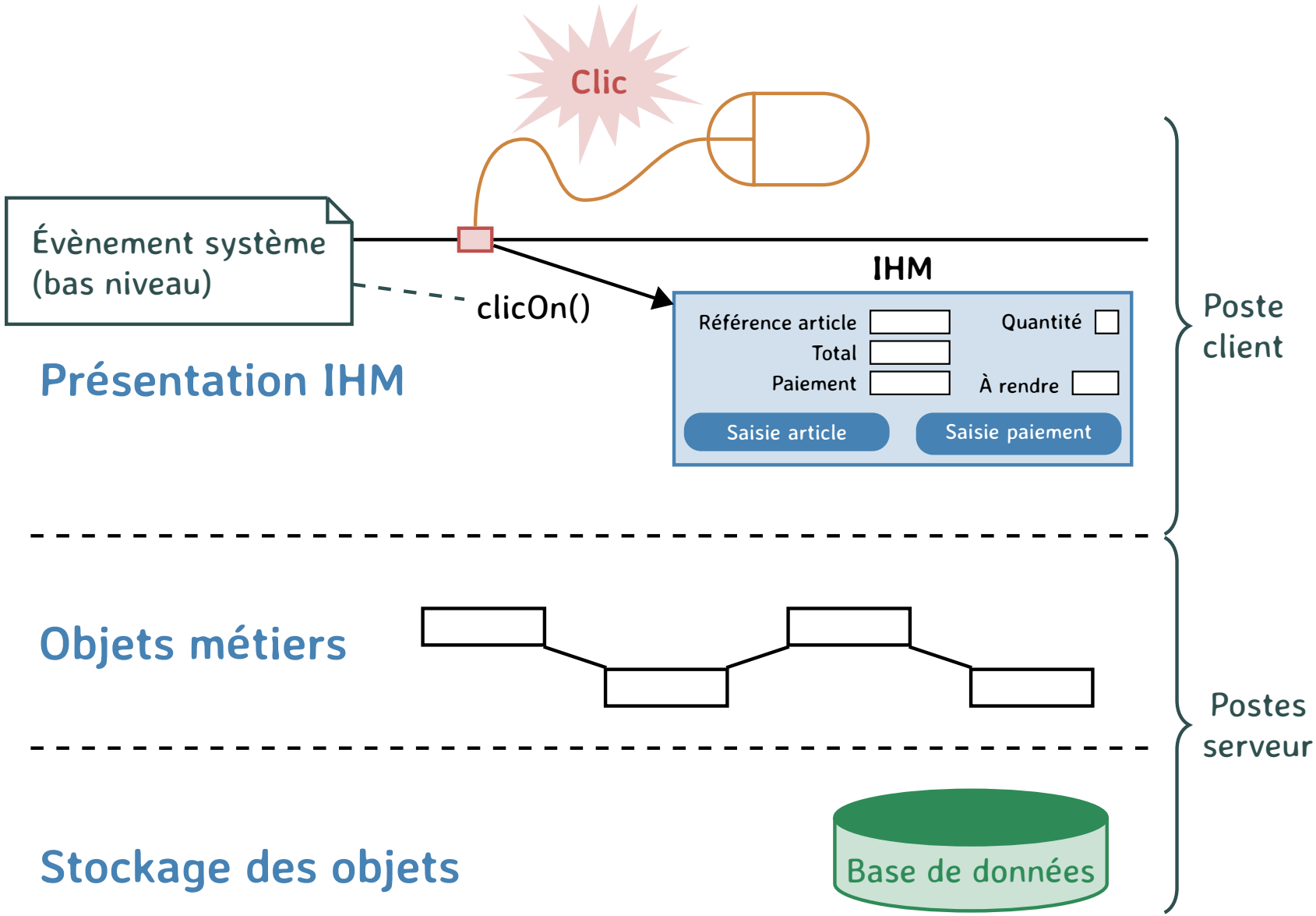
Le D.S. au niveau conception
Cas d'une caisse enregistreuse



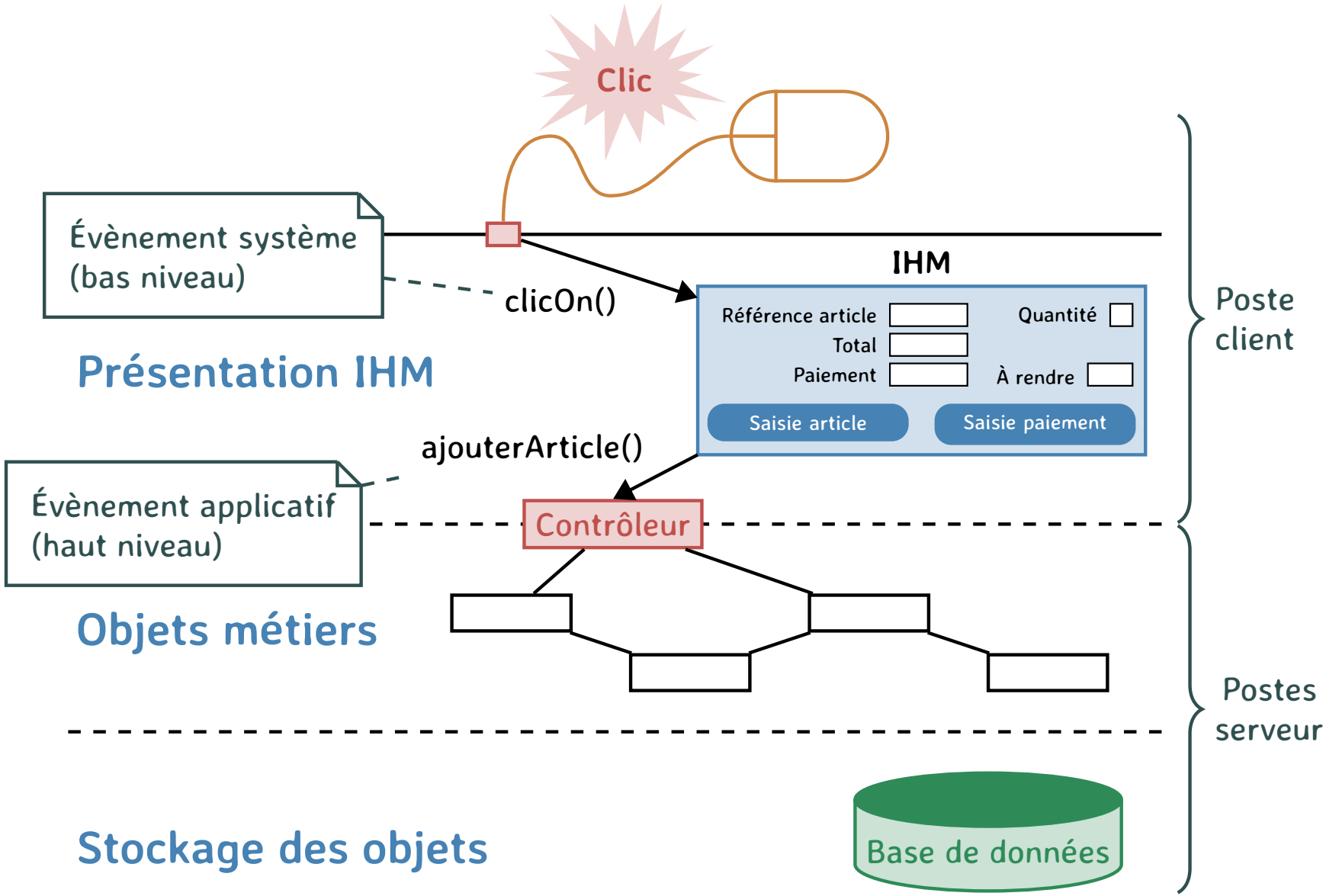
Le D.S. au niveau conception
Cas d'une caisse enregistreuse



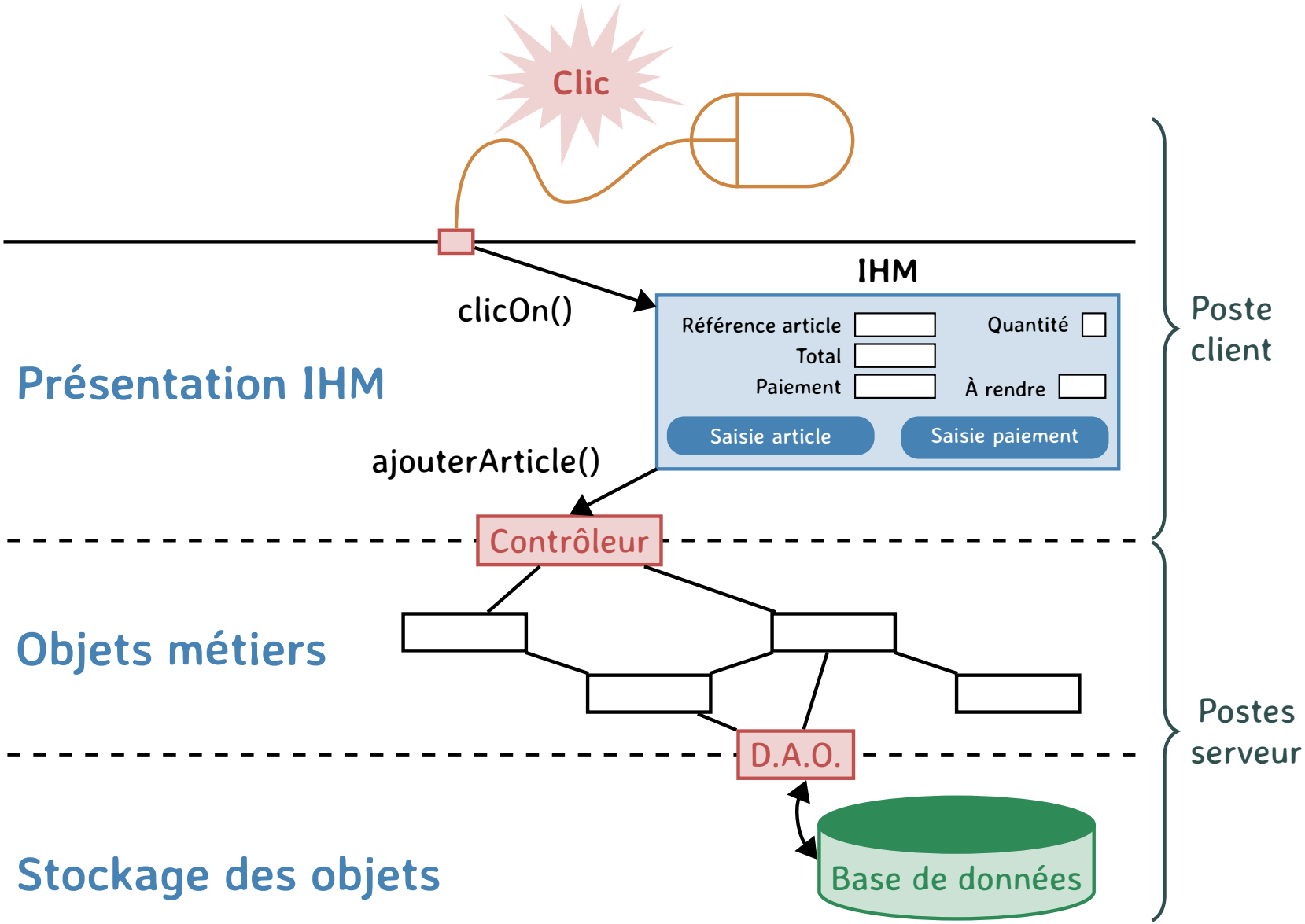
Le D.S. au niveau conception
Cas d'une caisse enregistreuse



Le D.S. au niveau conception
Cas d'une caisse enregistreuse



Le D.S. au niveau conception
Cas d'une caisse enregistreuse



Du D.S.S. (analyse) au D.S. (conception)

Diagramme de séquence système

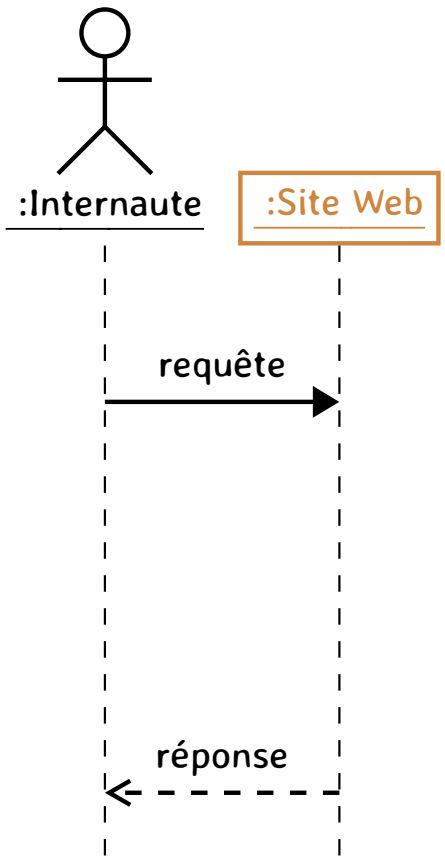
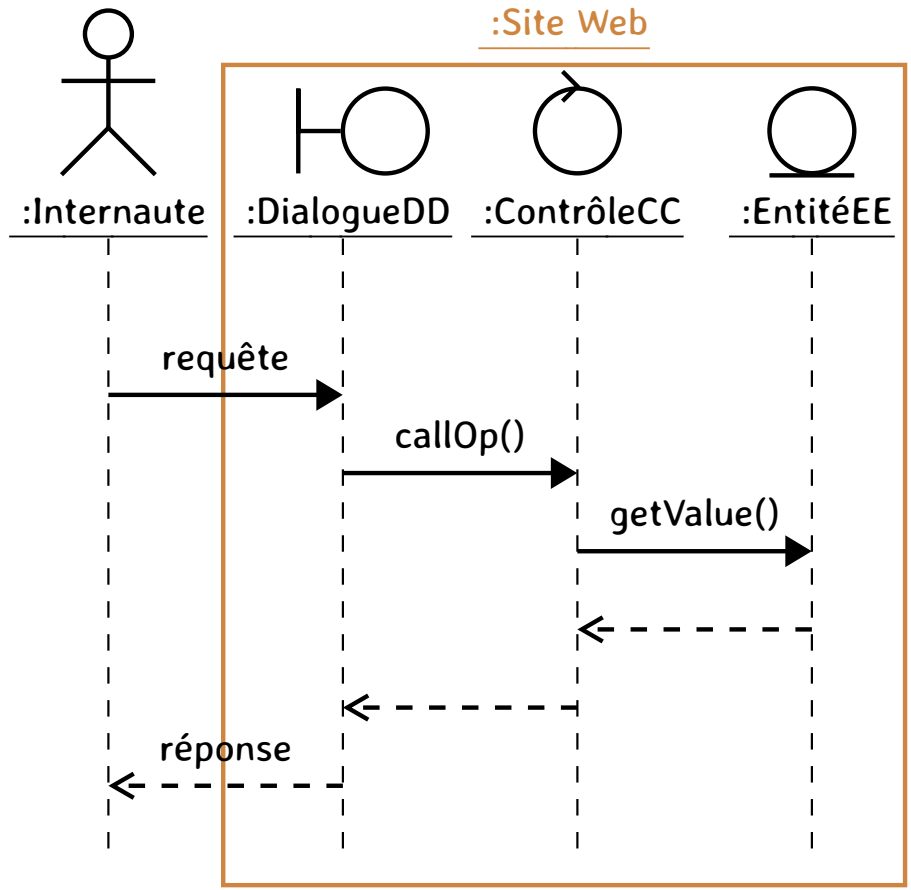


Diagramme de séquence



Le diagramme de séquence

Pourquoi ?

- ☞ Choix sur le partage et l'affectation des responsabilités entre classes
- ☞ Quelles opérations ? Pour quelles classes ?

Comment ?

- ☞ La représentation se concentre sur l'**expression des interactions**.
  - Les objets qui participent à l'interaction
  - Les messages entre les objets
  - La ligne de vie de chaque objet
  - Les périodes d'activité des objets
- ☞ Ne pas utiliser les diagrammes de séquence :
  - Pour décrire le comportement d'un seul objet
  - Pour décrire des algorithmes
  - Pour décrire le contrôle de l'application

Cadres d'interaction

☞ Introduits dans UML 2.0

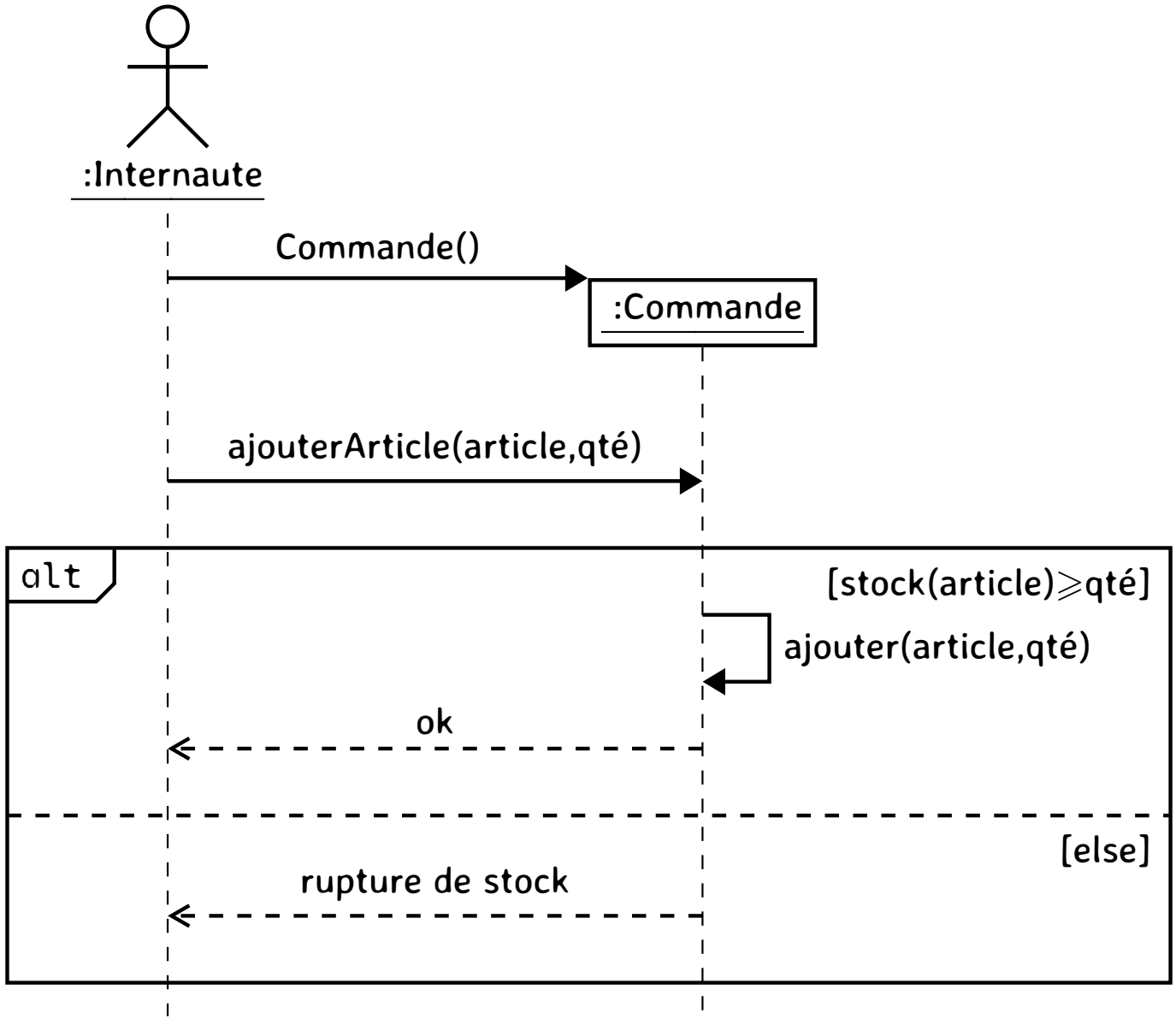
définition (Cadre d'interaction)

Un « fragment » du diagramme d'interaction pouvant être référencé et réutilisé.

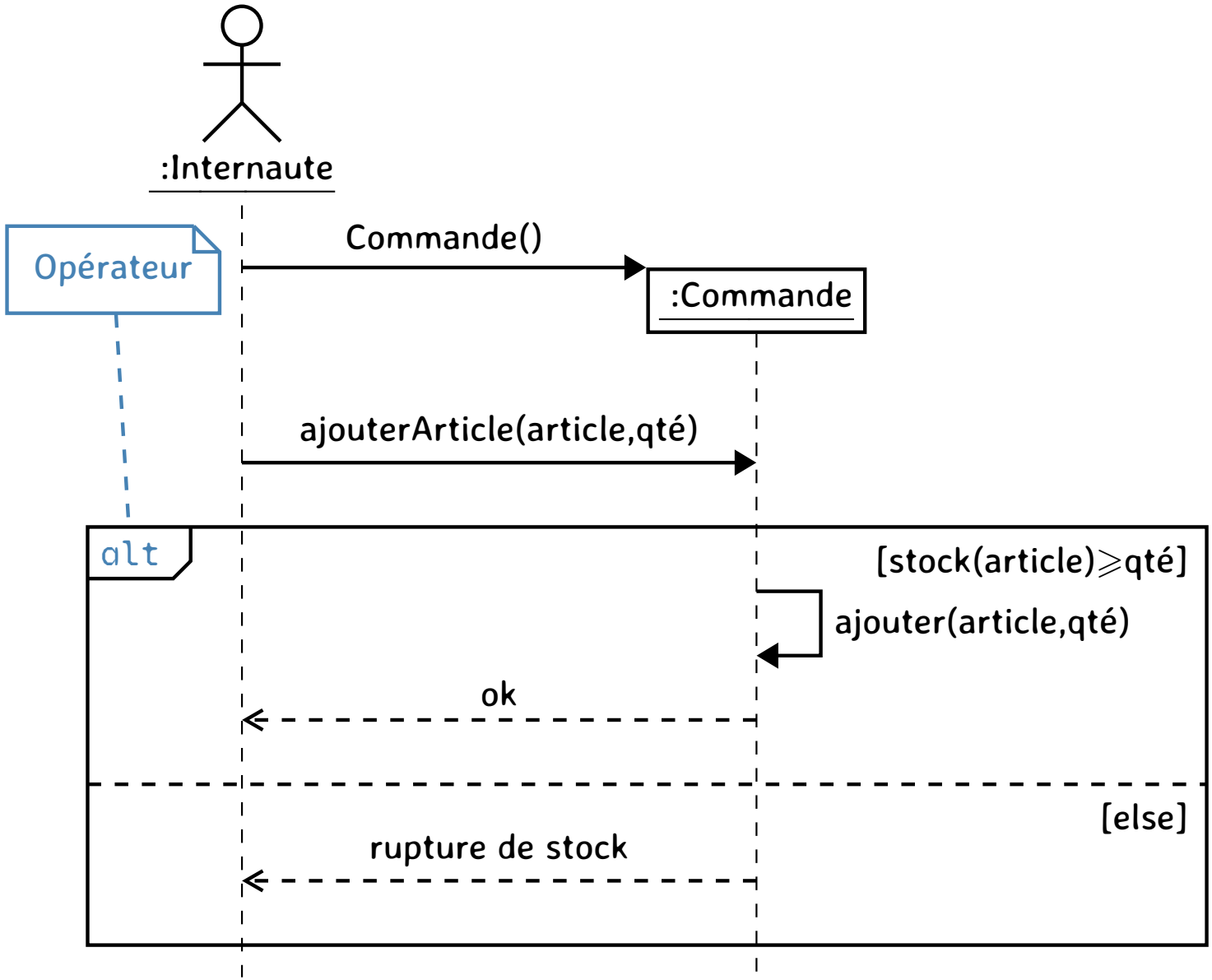
Opérateurs associés à un cadre d'interaction :

```
alt , opt, par, loop, ref, sd, break
```

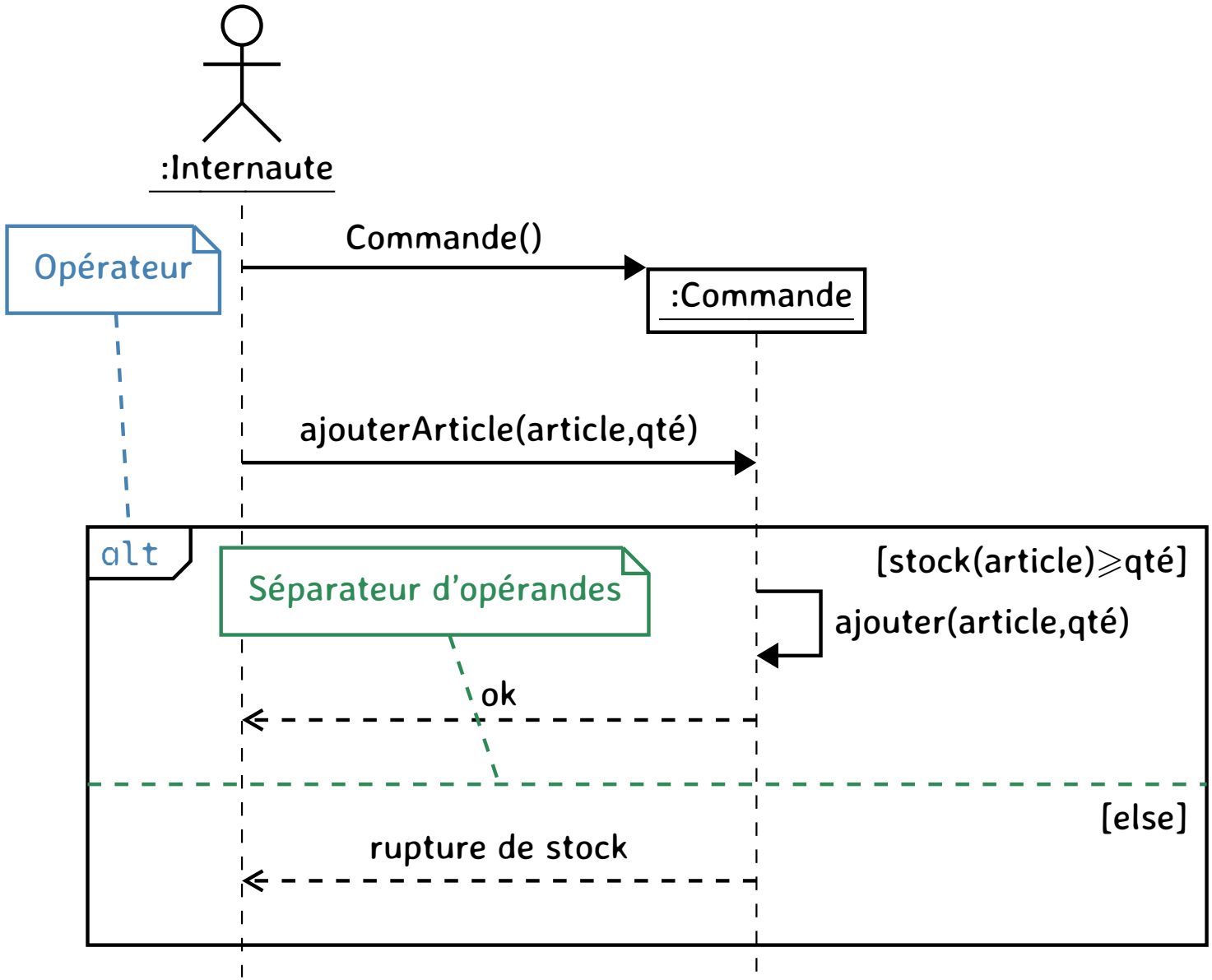
Cadres d'interaction
alt : branchement conditionnel



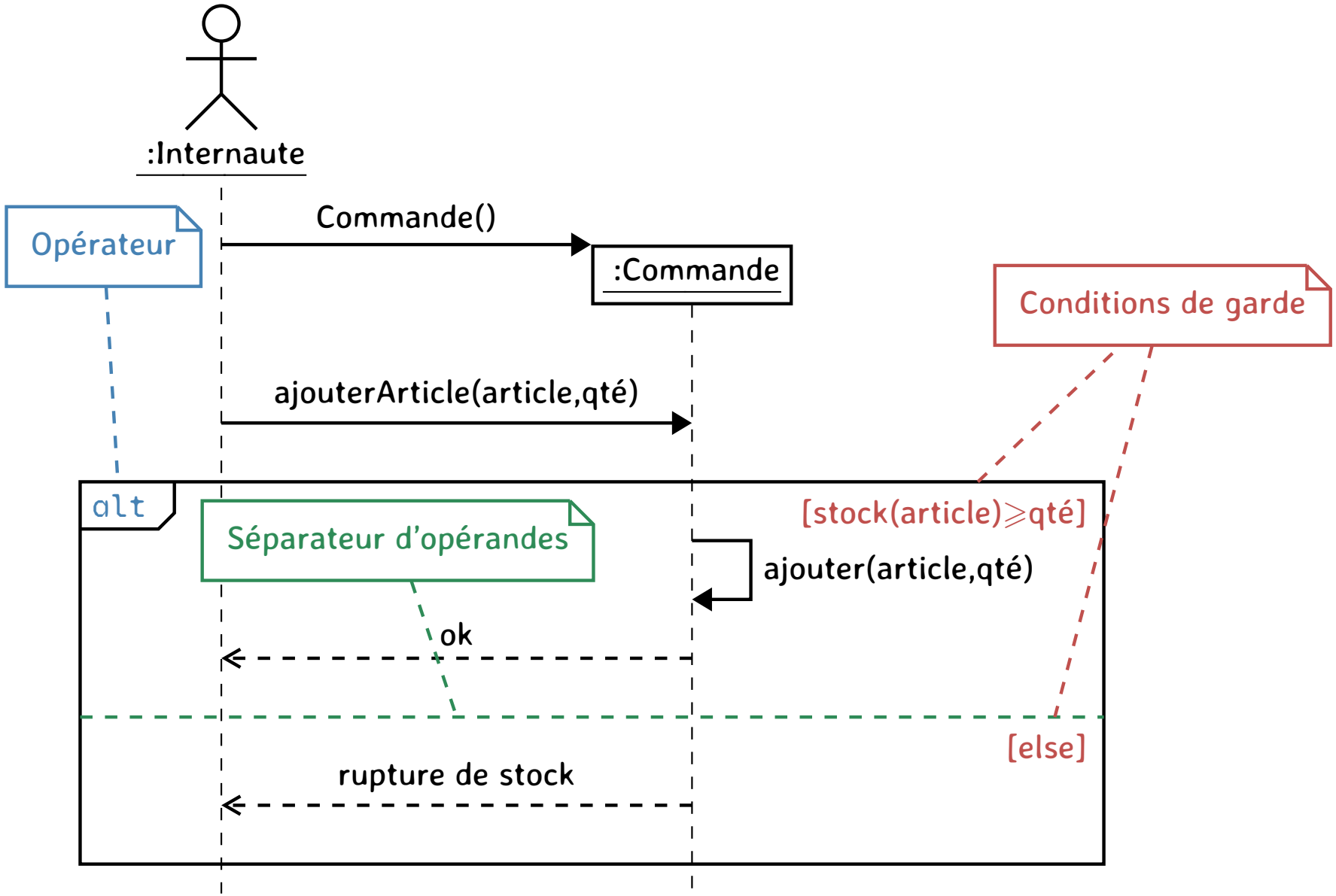
Cadres d'interaction
alt : branchement conditionnel



Cadres d'interaction
alt : branchement conditionnel

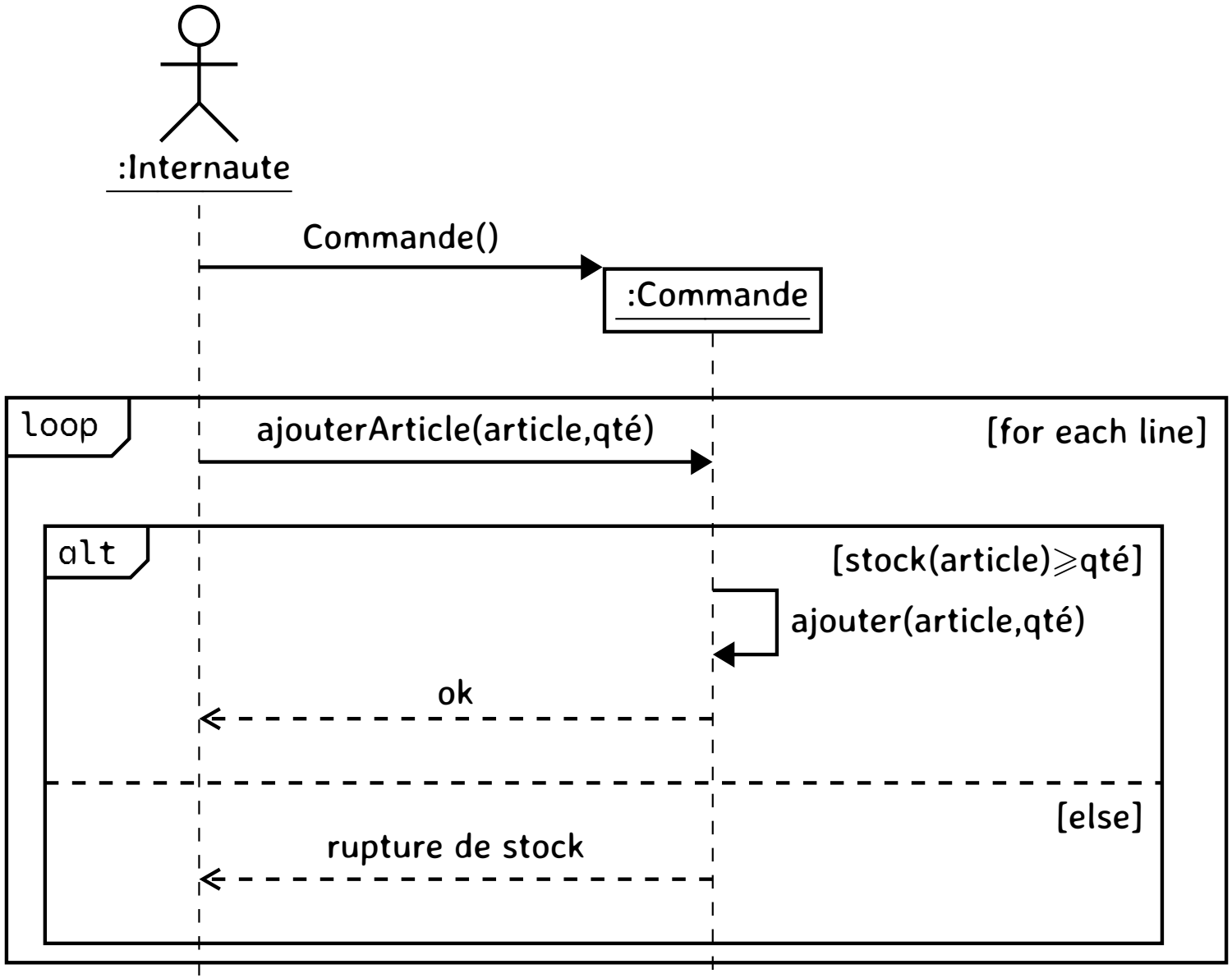


Cadres d'interaction
alt : branchement conditionnel



Cadres d'interaction

loop : boucles



Cadres d'interaction
Opérateurs importants

- `alt` (Alternative)
  plusieurs fragments alternatifs, seul celui dont la condition est vraie s'exécutera.
- `opt` (Optionnel)
  le fragment s'exécute si la condition est vraie.
- `loop` (Boucle)
  le fragment peut s'exécuter plusieurs fois et la garde indique le nombre d'itérations (boucle `for`) ou la condition d'arrêt (boucle `while`).
- `sd` (Sequence Diagram) utilisé pour entourer et nommer un fragment de diagramme ou un diagramme entier.
- `ref` (Référence)
  se réfère à une interaction définie sur un autre diagramme dans un cadre `sd`.

UML - La vue dynamique
Plan

UML - La vue dynamique
Plan