L'outil SysML pour la modélisation des systèmes complexes

Vincent Albert

Université Paul Sabatier

2013-2014

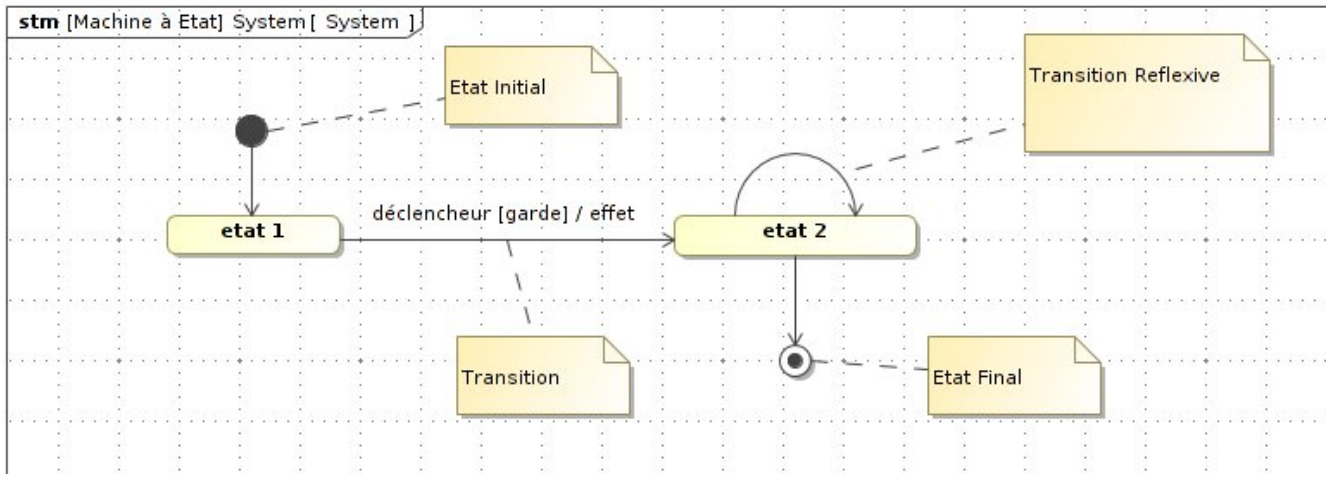
Programme

1. Introduction 2h
2. Le langage SysMl 4h
   - Les diagrammes de comportements
   - Les diagrammes de structure
   - Les diagrammes d'exigences
   - Les relations transversales
3. Étude de cas 2h
4. TP 8h

Programme

Les diagrammes de comportements - Machine à états

Les diagrammes d'états - La transition



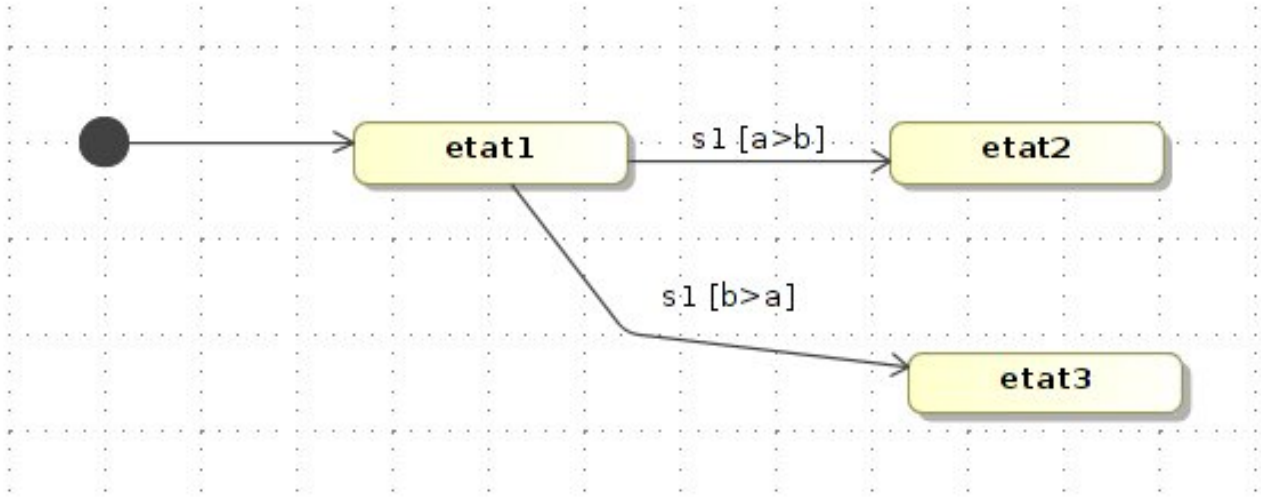
- déclencheur : Trigger[0..*]
- garde : Constraint[0..1]
- effet : Behaviour[0..1]

Les diagrammes d'états - Transitions - Les déclencheurs

Un *Trigger* est associé à un *Event* et il y a 4 types d'*Event* :

- *TimeEvent* : L'événement survient au bout d'un temps donné (mot clé *at*).
- *CallEvent* : L'événement survient lors de la réception d'un message d'appel d'opération.
- *SignalEvent* : L'événement survient lors de la réception d'un signal.
- *ChangeEvent* : L'événement survient lorsque une condition devient vraie (mot clé *when*).

Les diagrammes d'états - Transitions - Les gardes



Les diagrammes d'états - Transitions - Les effets

Il y a au moins 4 types de *Behaviour* :

- *State Machine*.
- *Activity*.
- *Interaction*.
- *Opaque Behaviour*.

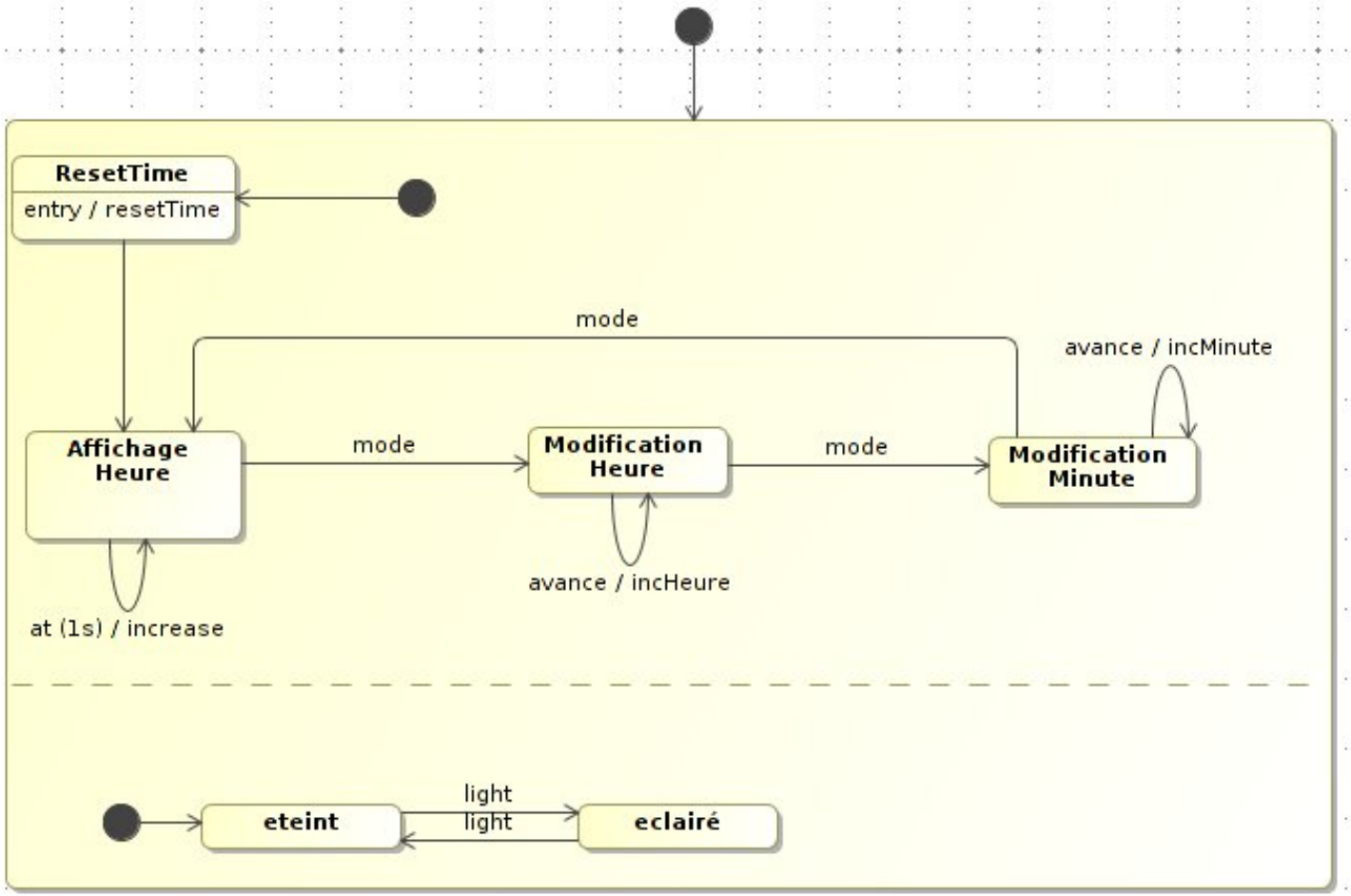
L'effet du franchissement d'une transition peut donc être décrit par un diagramme de machine à état, un diagramme de séquence, un diagramme d'activité ou du code.

Les diagrammes d'états - L'état - Les comportements

- doActivity : Behavior[0..1] : Un comportement qui est exécuté en boucle lorsque l'état est actif (mot clé *do*).
- entry : Behavior[0..1] Un comportement qui est exécuté lorsqu'on entre dans l'état.
- exit : Behavior[0..1] Un comportement qui est exécuté lorsqu'on sort de l'état.

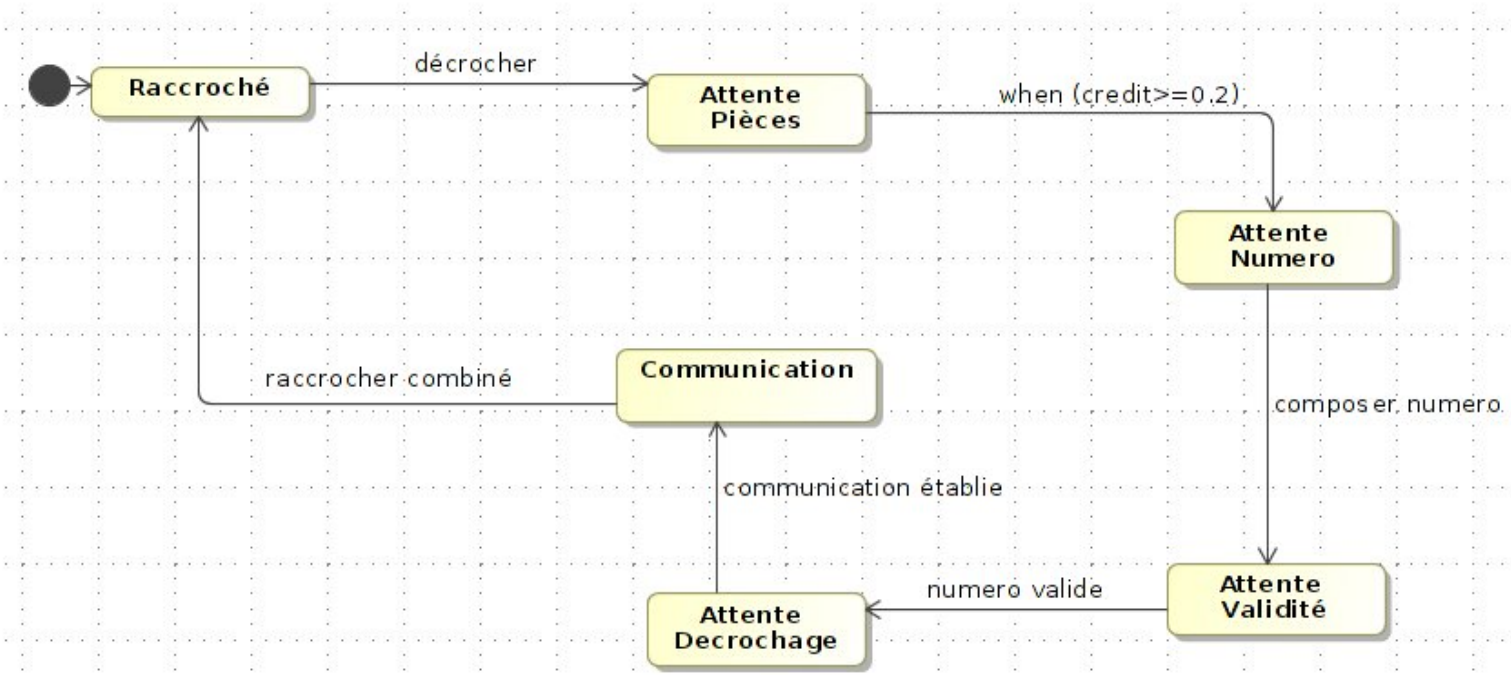
Les diagrammes d'états - L'état orthogonal

Permet de mettre en oeuvre des régions concurrentes



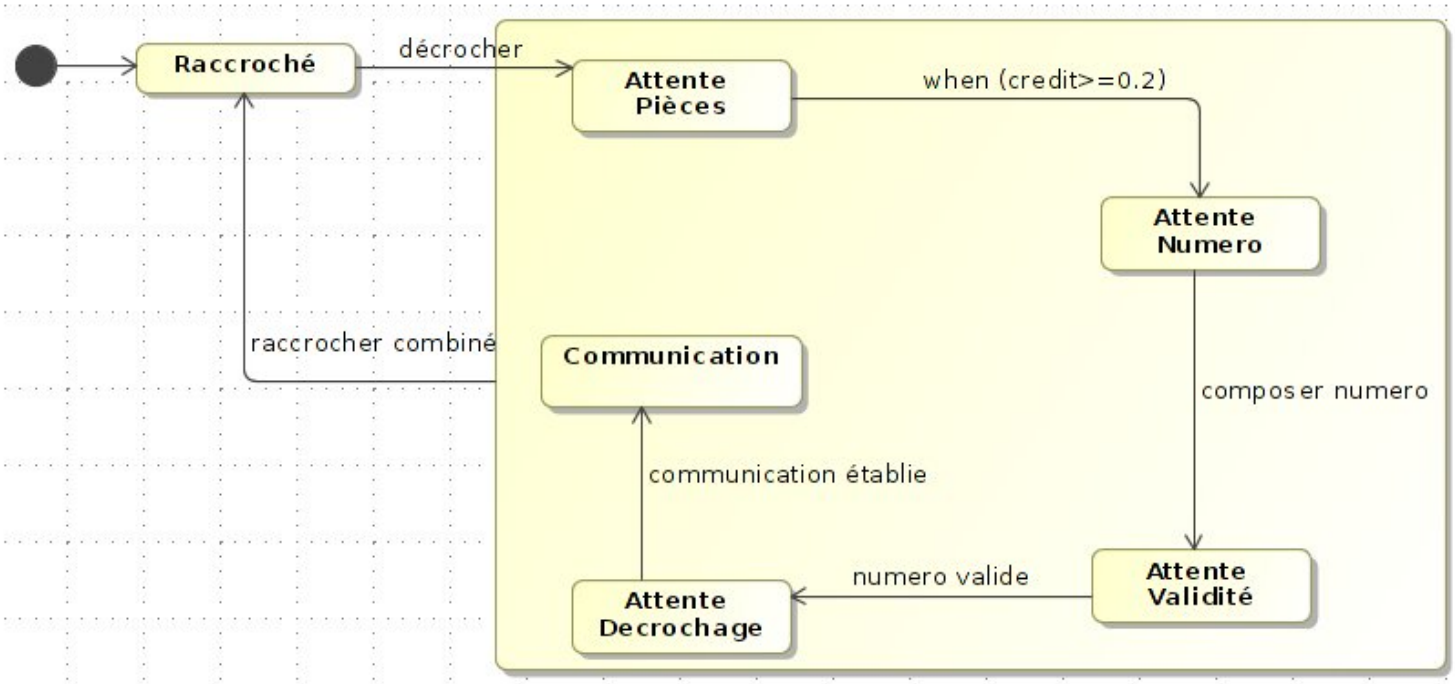
Les diagrammes d'états - Exemple de la cabine téléphonique

- réception de signal
- événement interne when(credit >= 0.2)



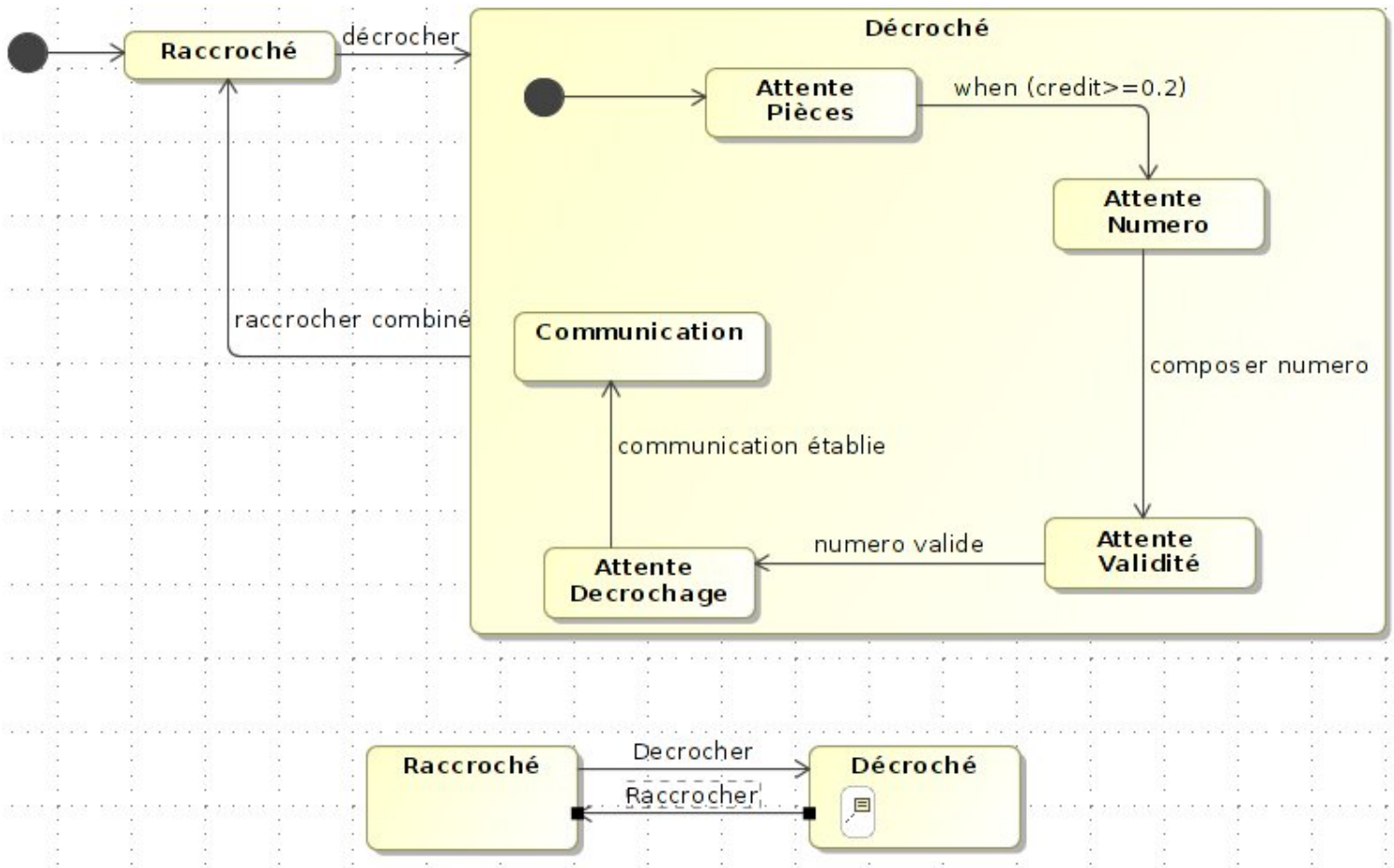
Les diagrammes d'états - État composite

- on peut raccrocher le combiné à tout moment



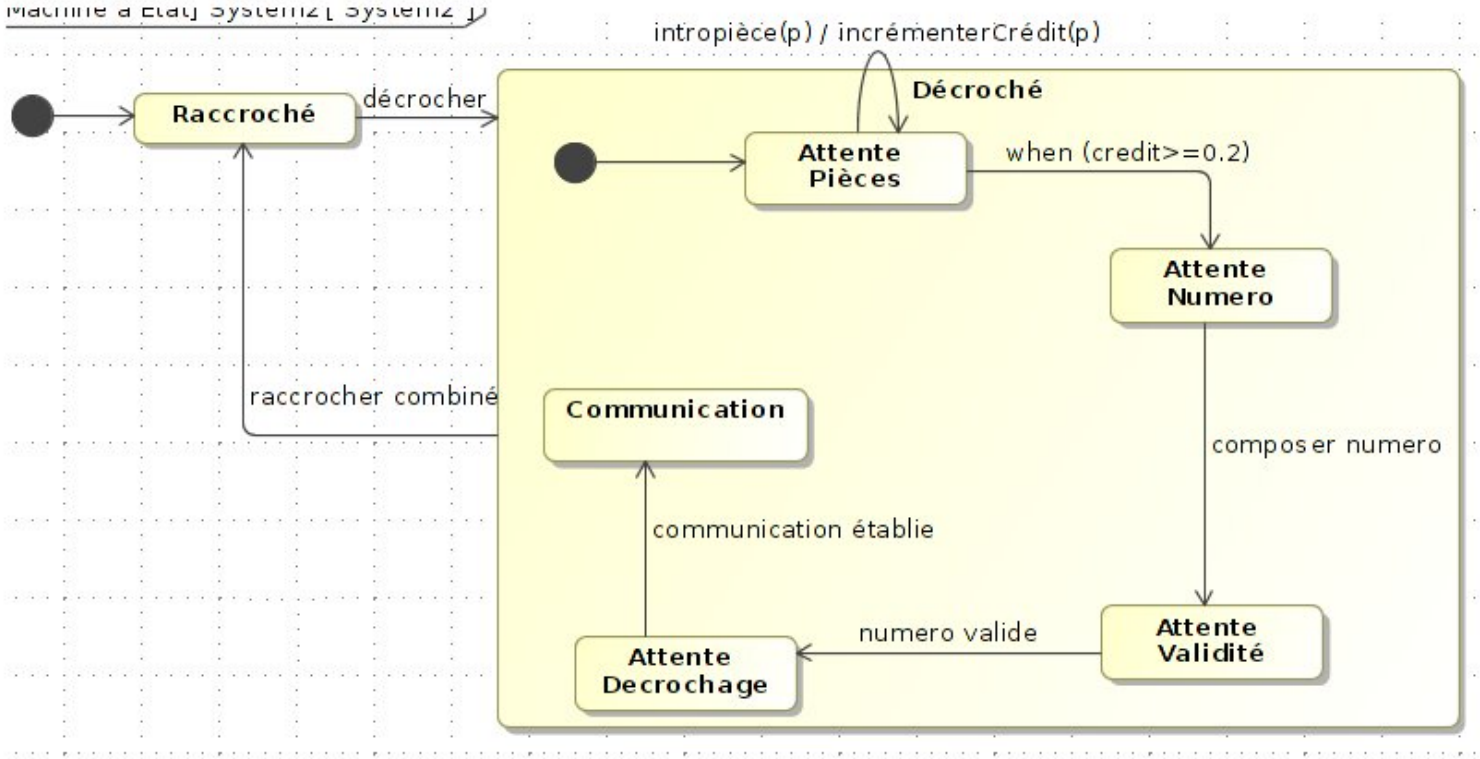
Les diagrammes d'états - État composite

- Découper le diagramme d'états en deux niveaux
  - un premier niveau ne faisant apparaître que les états raccroché et décroché
  - un second niveau correspondant à la décomposition de décroché



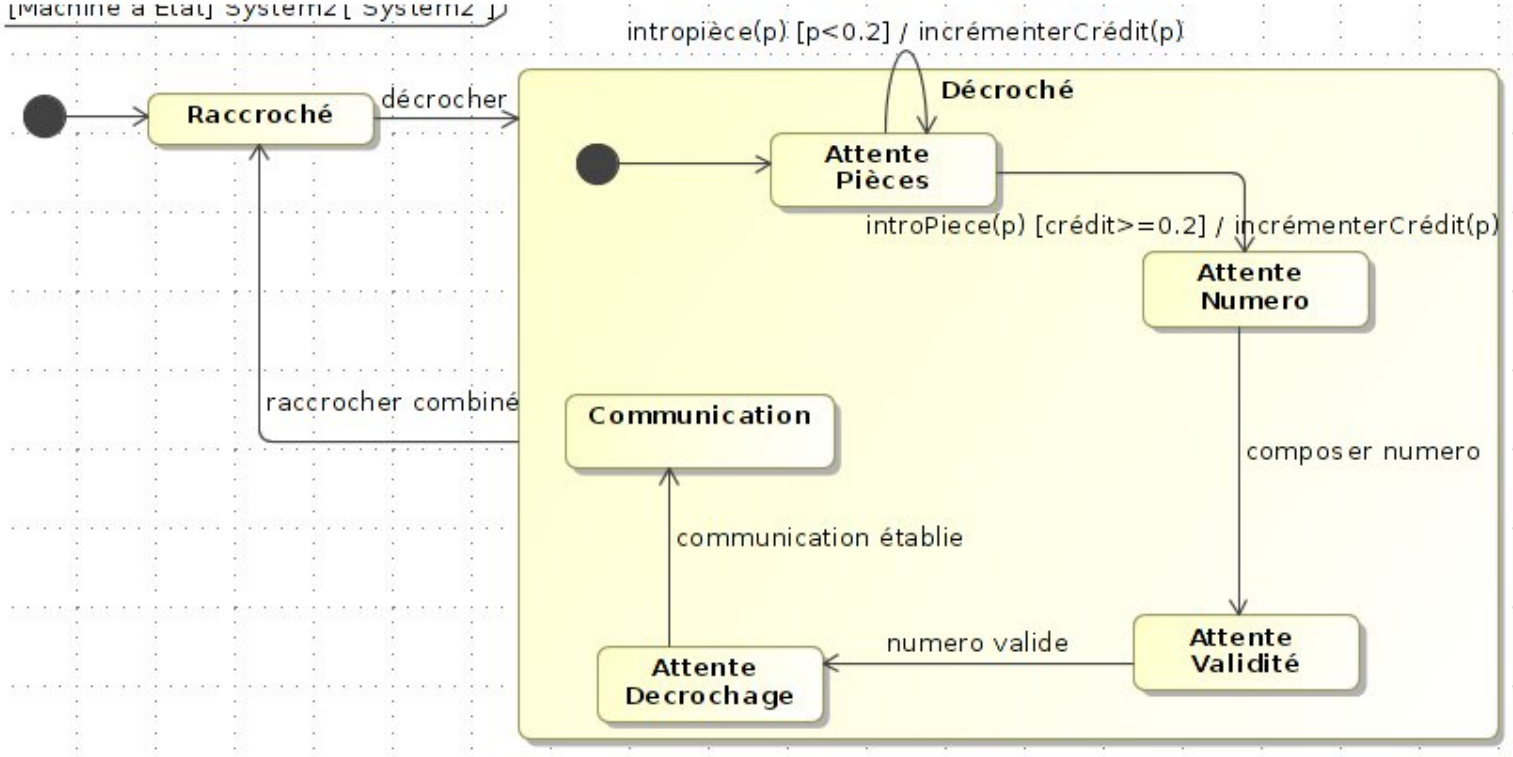
Les diagrammes d'états - Action, puis test

- Introduire des pièces jusqu'au crédit suffisant.



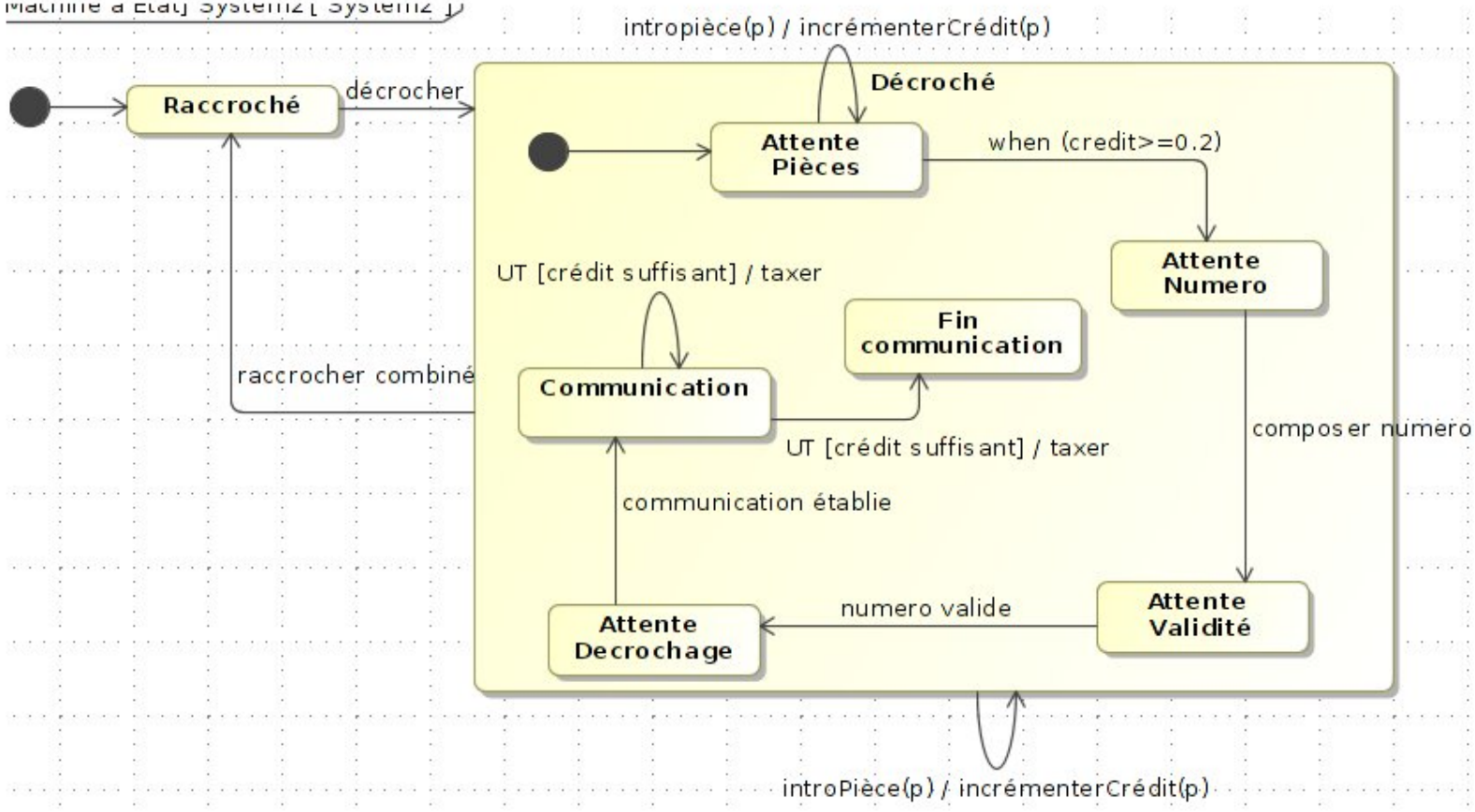
Les diagrammes d'états - Incorrect

- Lorsque l'événement déclencheur se produit la condition est testée. Ssi la condition est évaluée à vrai, la transition est déclenchée et l'effet associé est réalisé ensuite.



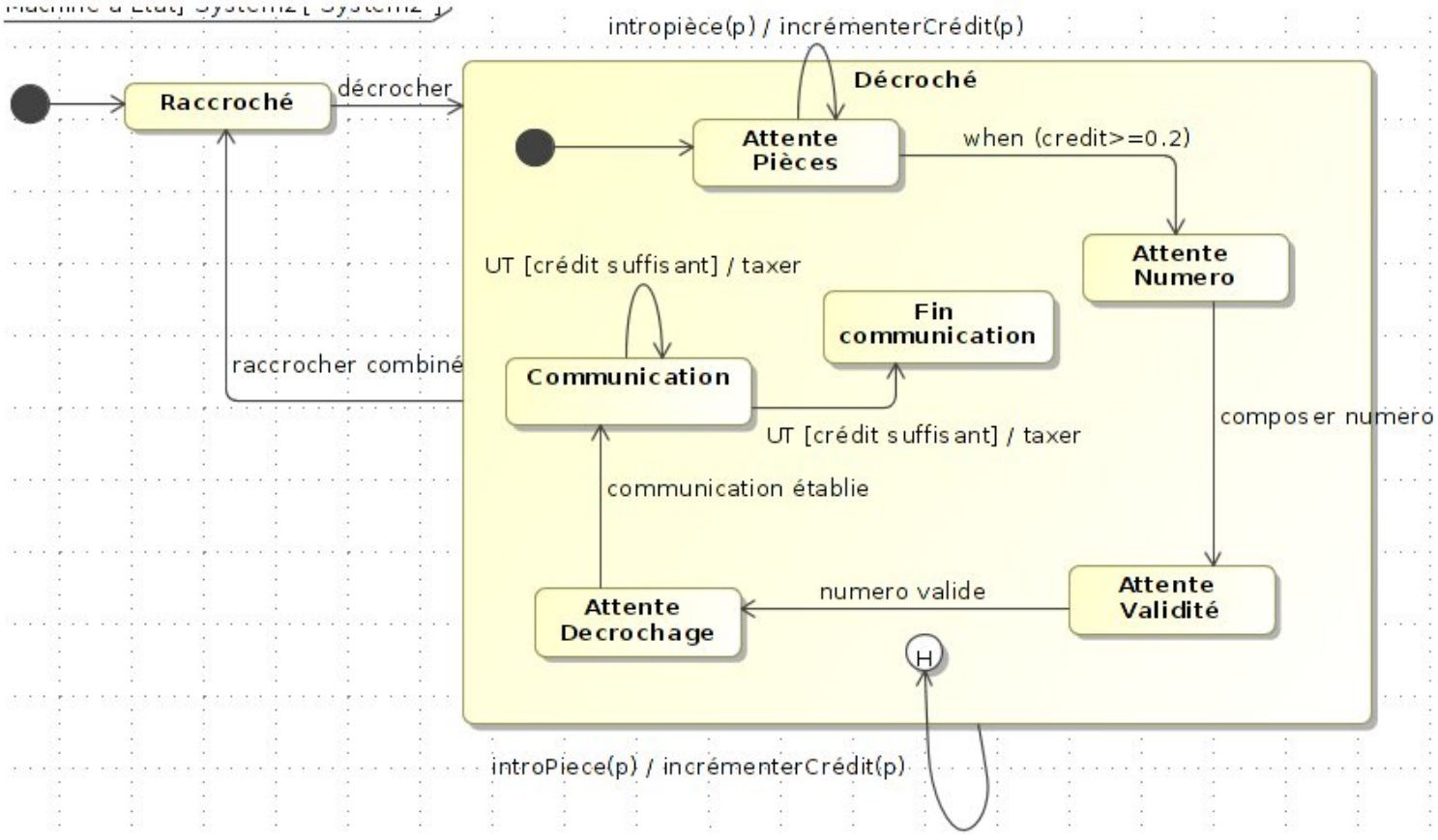
Les diagrammes d'états - Transition propre

- On peut introduire des pièces pendant une communication



Les diagrammes d'états - Pseudo-état History

- Permet à un état composite de se souvenir du dernier sous-état qui était actif avant une transition sortante

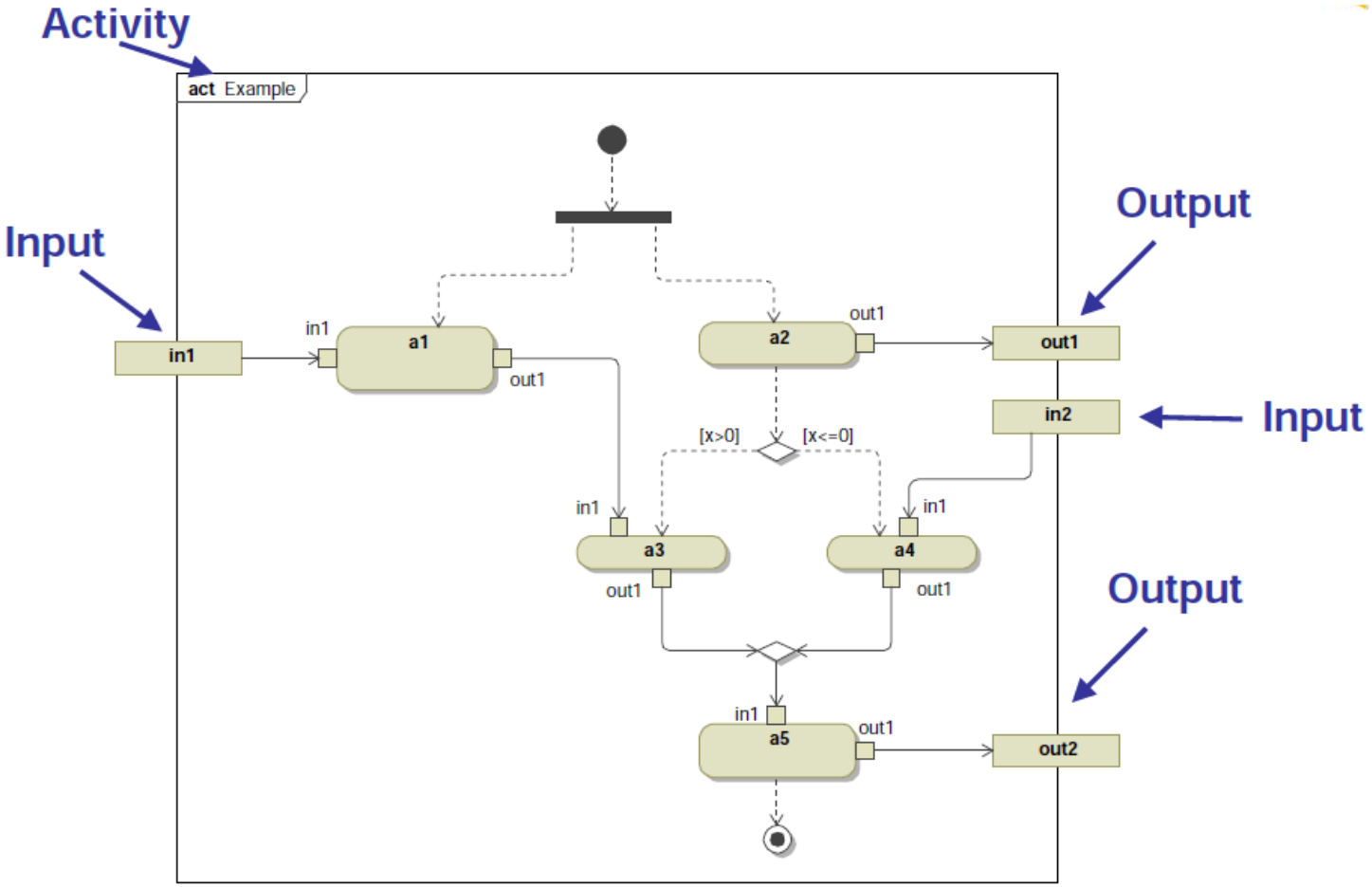


Programme

Les diagrammes de comportements - Activité

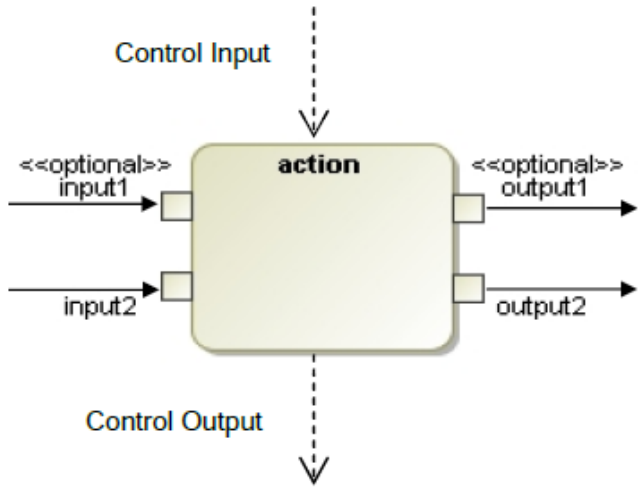
Les Diagrammes d'activité

Une activité spécifie la transformation des entrées vers les sorties à travers une séquence contrôlée d'actions



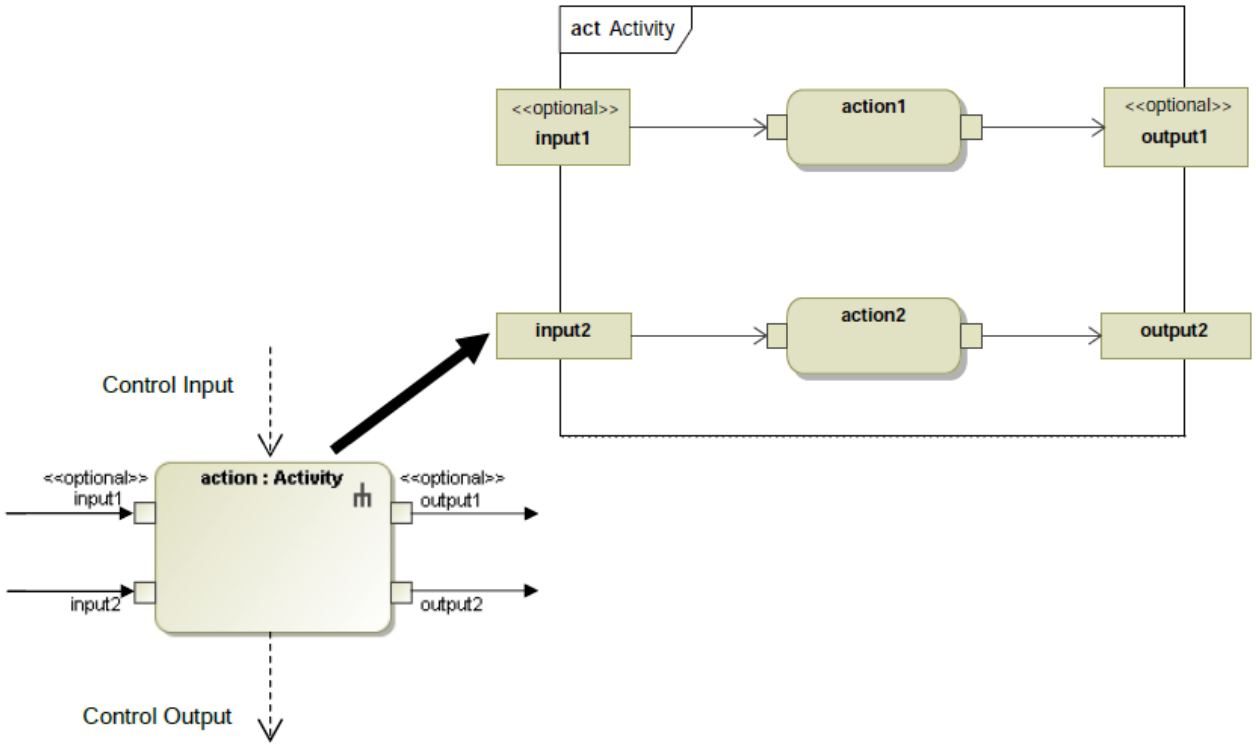
Les Diagrammes d'activité - Processus de flux mesures de contrôle et de données

- Deux types de flux
  - Objet / Donnée
  - Contrôle
- Unité de flux est appelé un *jeton* (consommé & produit par les actions)



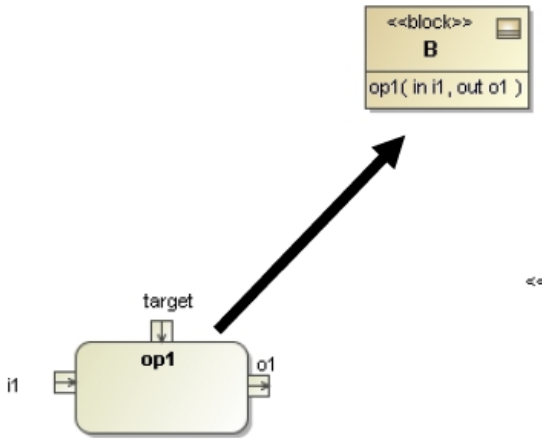
Les actions s'exécutent lorsque un jeton est disponible sur toutes les entrées de contrôle et les entrées requises

Une action peut invoquer une autre activité

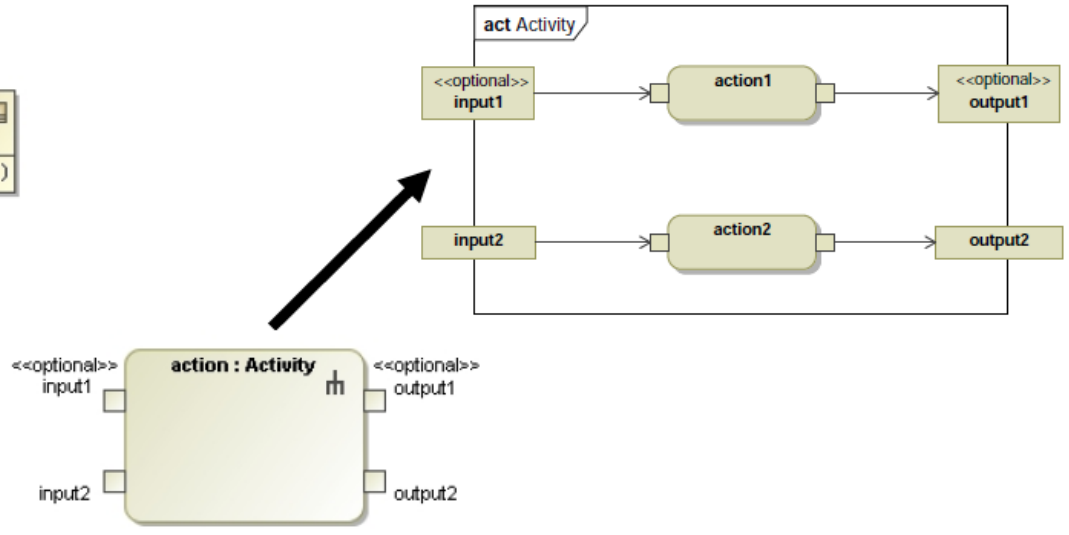


L'activité est invoquée lorsque l'action commence son exécution

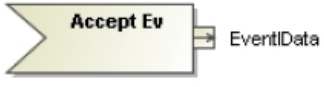
Les types d'actions



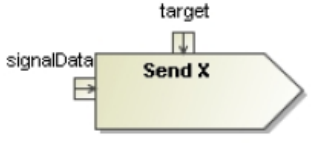
Call Operation Action
(can call leaf level function)



Call Behavior Action

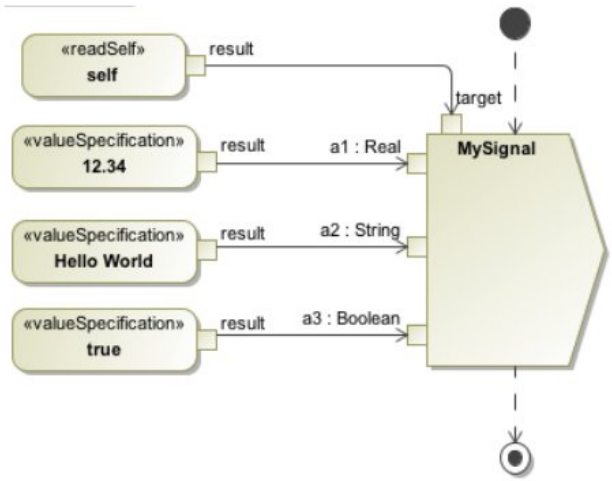


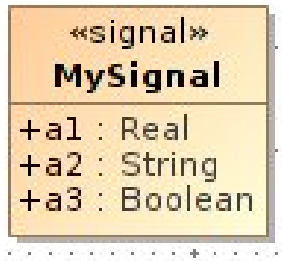
Accept Event Action
(Event Data Pin often elided)

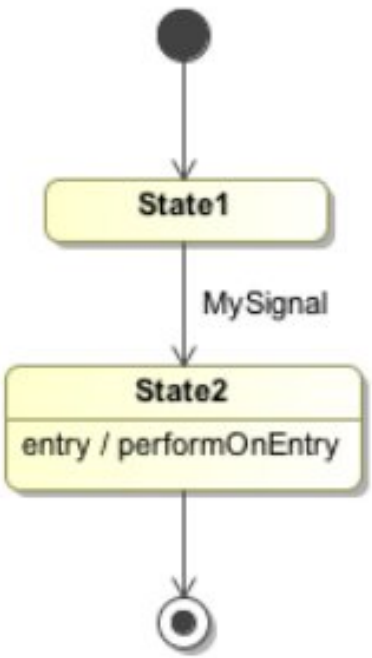


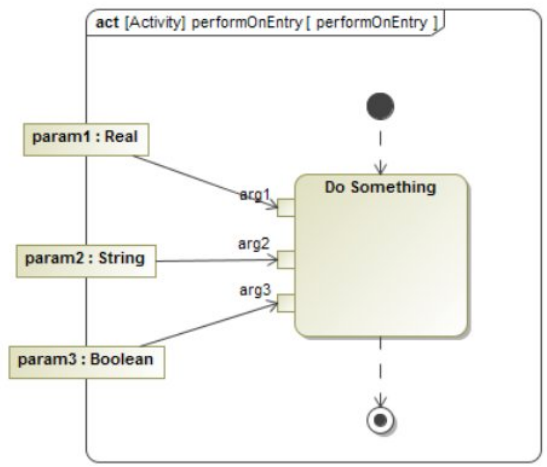
Send Signal Action
(Pins often elided)

Envoie de signal avec donnée

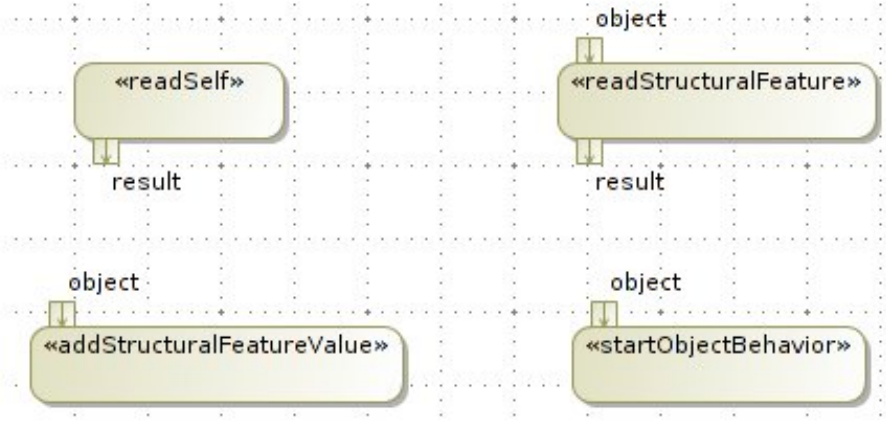




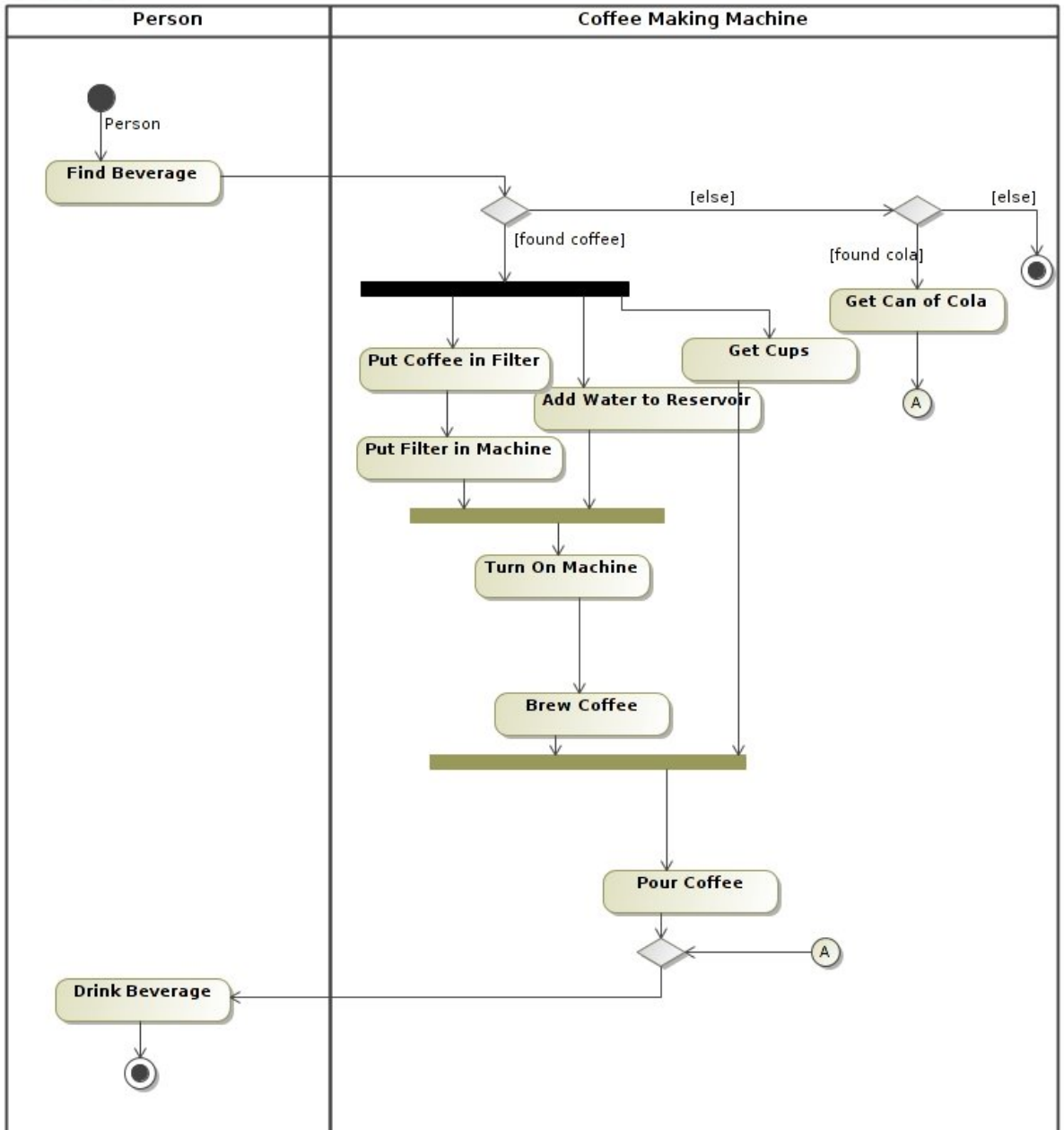




Autres actions



Swimlane - Allocation d'actions sur des blocks



Programme

Les diagrammes de comportements - Cas d'utilisation

Cas d'utilisation

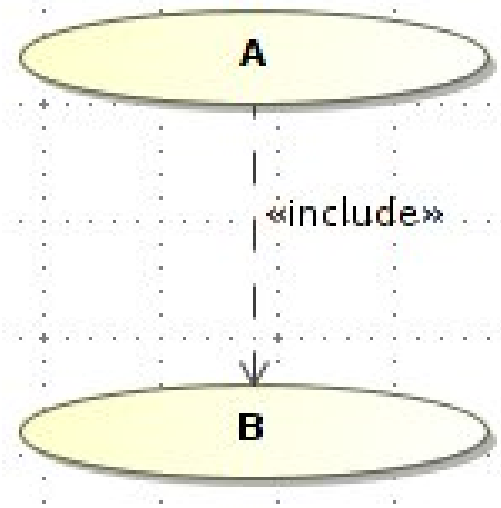
Les cas d'utilisation sont un moyen de spécifier les usages attendus d'un système. Typiquement ils sont utilisés pour cerner ou raffiner **les exigences fonctionnelles** d'un système, c'est-à-dire les fonctionnalités que le système doit remplir. Les concepts clés d'un tel diagramme sont les cas d'utilisation, les acteurs et les sujets (subject en anglais). Le sujet est le système considéré sur lequel les cas d'utilisation s'appliquent. Les utilisateurs et autres systèmes qui interagissent avec le sujet sont représentés comme des acteurs. Un acteur représente un ensemble cohérent de rôles que peuvent jouer des utilisateurs d'un système lorsqu'ils interagissent avec les cas d'utilisation.

Les relations

- Association : entre un acteur et un use case
- Inclusion : entre deux cas d'utilisation, décrit une fonctionnalité commune qui doit être incluse dans un cas d'utilisation
- Extension : met en relation un cas d'utilisation "étendant" et un cas d'utilisation étendu et spécifie comment et quand le comportement du cas d'utilisation étendant peut être inséré dans le comportement du cas d'utilisation étendu. Un point d'extension permet de spécifier la condition sous laquelle le comportement sera étendu.
- Généralisation : entre deux acteurs ou entre deux cas d'utilisation, implémente la notion d'héritage

Inclusion

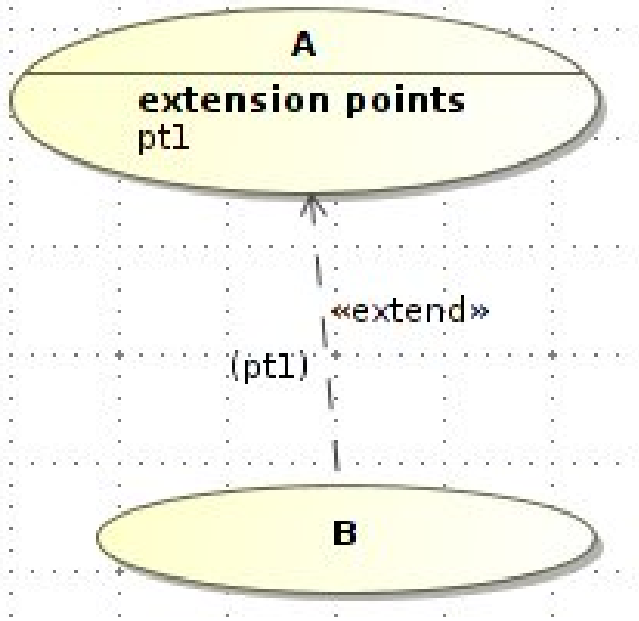
Soit un cas d'utilisation A qui inclut le cas d'utilisation B



- une instance de A va engendrer une instance de B et l'exécuter
- A dépend de B
- B n'existe pas tout seul et A n'existe pas sans B

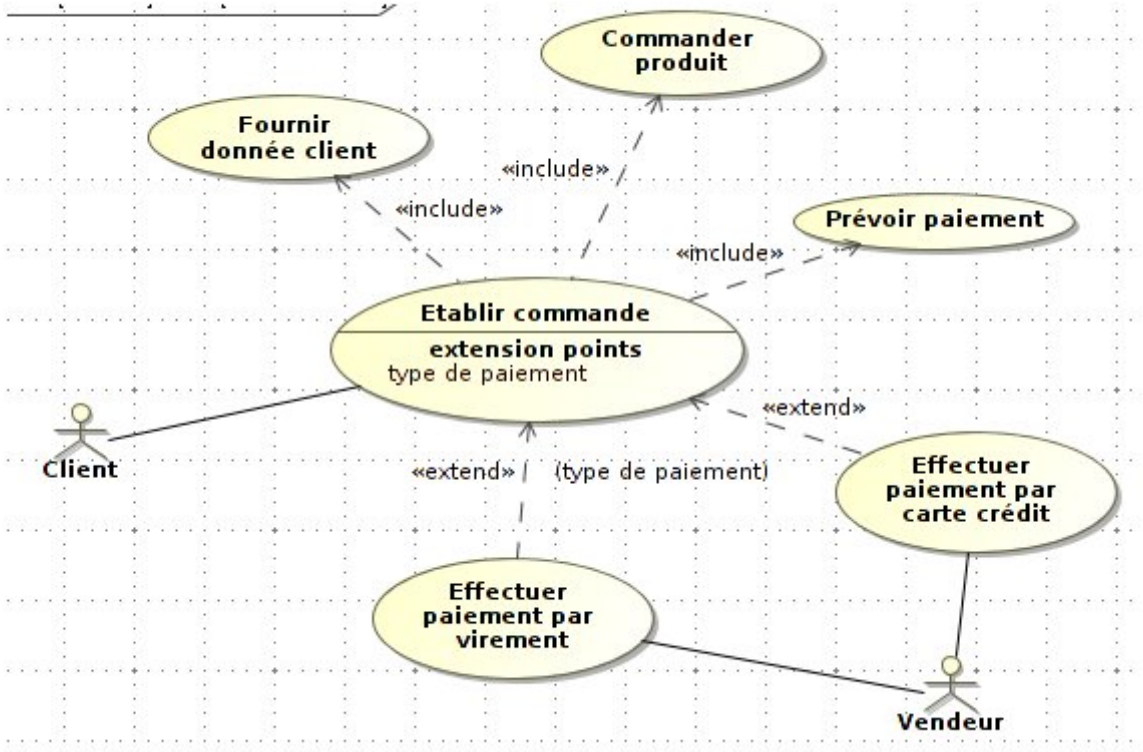
Extension

Soit un cas d'utilisation B qui étend le cas d'utilisation A

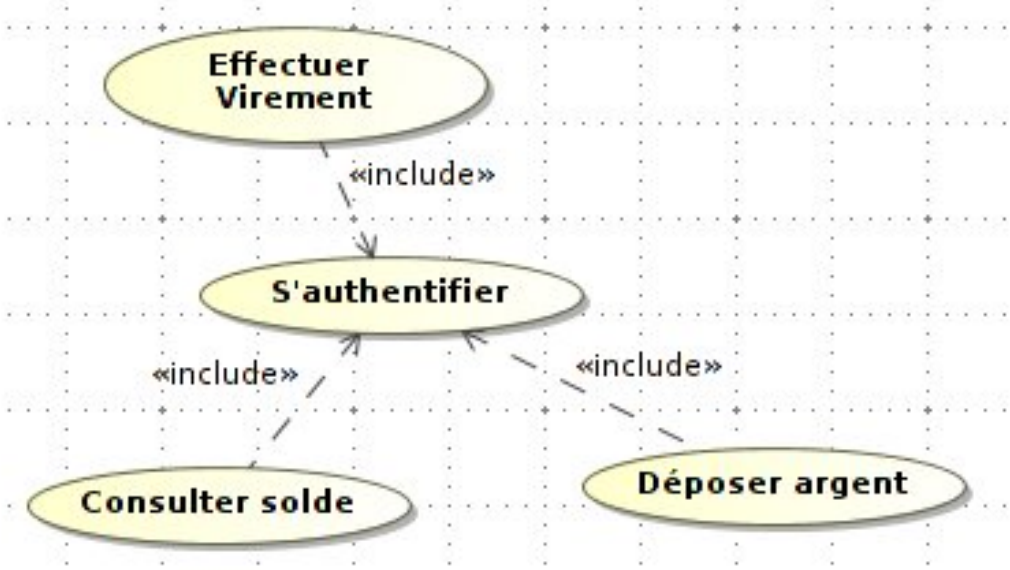


- une instance de A va éventuellement engendrer une instance de B et l'exécuter sous certaine conditions, B sait ou s'insérer dans A
- B dépend de A et non l'inverse
- B n'existe pas tout seul et A peut exister sans B (B est un cas particulier de A)

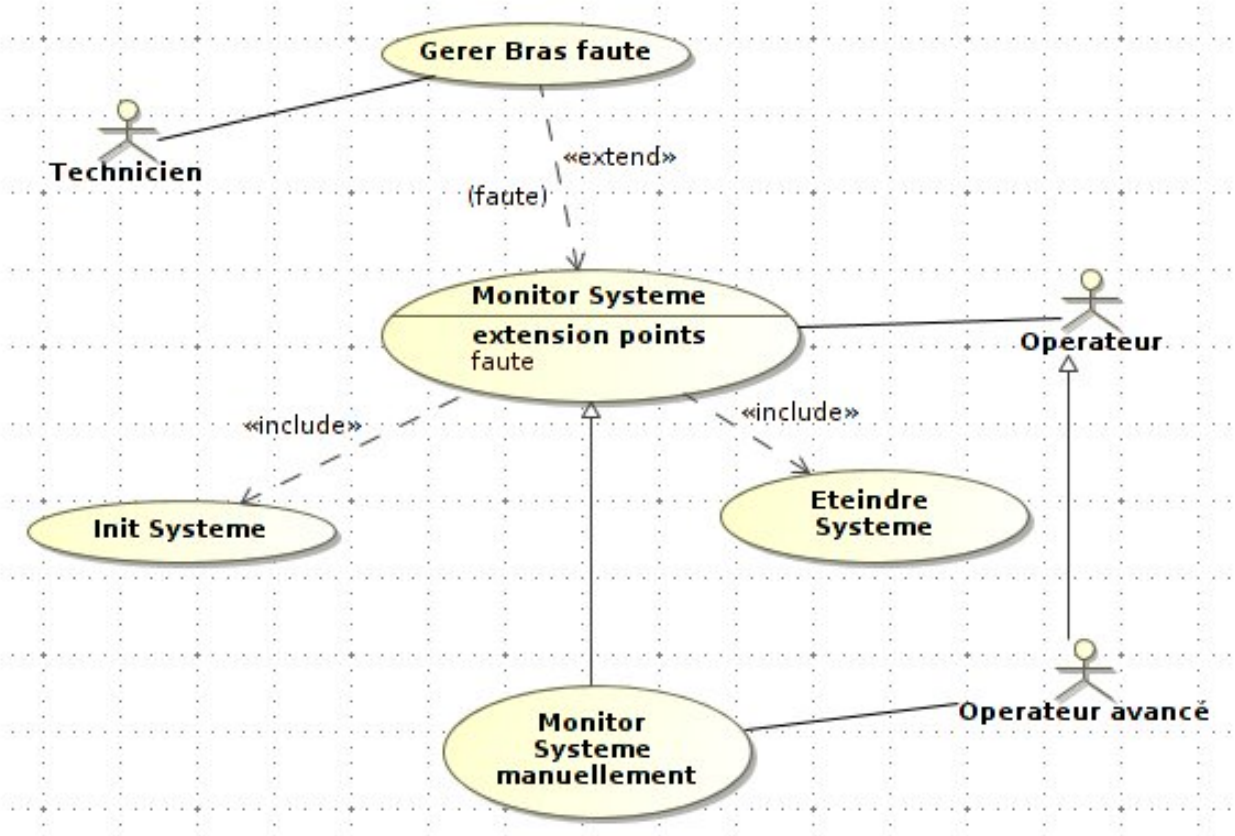
Exemple 1


Commander produit
Fournir donnée client
«include»
«include»
Prévoir paiement
«include»
Etablir commande
extension points
type de paiement
«extend»
Client
«extend»
(type de paiement)
Effectuer paiement par carte crédit
Effectuer paiement par virement
Vendeur

Exemple 2


Effectuer Virement
«include»
S'authentifier
«include»
«include»
Consulter solde
Déposer argent

Exemple 3

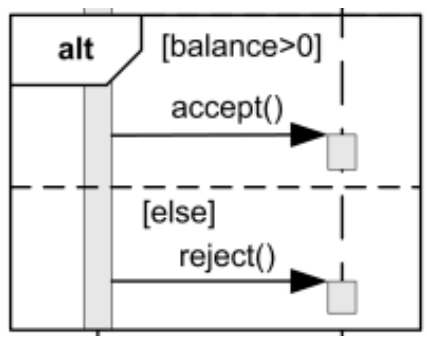

Gerer Bras faute
Technicien
«extend»
(faute)
Monitor Systeme
extension points
faute
Operateur
«include»
«include»
Init Systeme
Eteindre
Systeme
Monitor
Systeme
manuellement
Operateur avancé

Programme

Les diagrammes de comportements - Interaction (Diagramme de séquence)

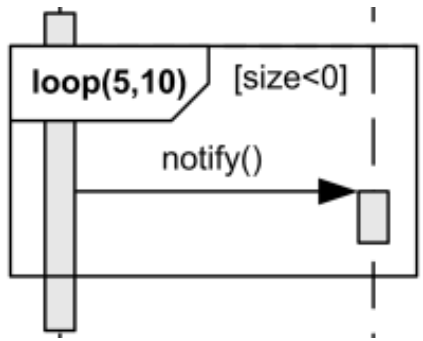
Alternative

alt exprime la possibilité de choisir différents comportements. Ceci est réalisé par le choix d'un unique opérande d'interaction en fonction des contraintes d'interaction.



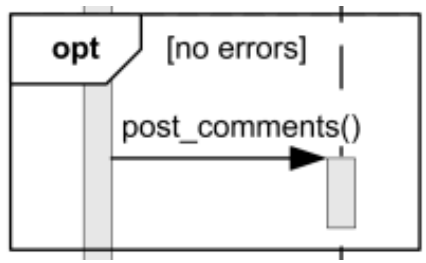
Boucle

loop représente une boucle. L'opérande d'interaction sera répété un certain nombre de fois.



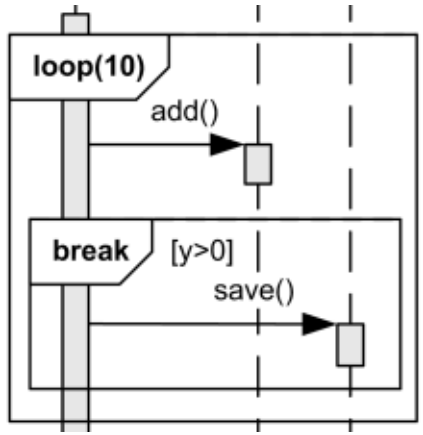
Optionnelle

opt représente un choix dans le comportement où soit seul l'opérande d'interaction contenu dans le fragment combiné s'exécute, soit rien ne se produit.



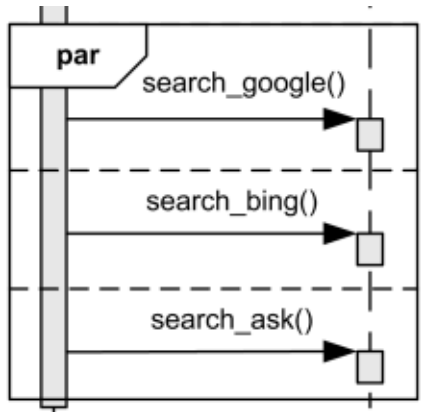
Break

break représente un scénario d'arrêt qui est exécuté à la place du reste du fragment d'interaction englobant.



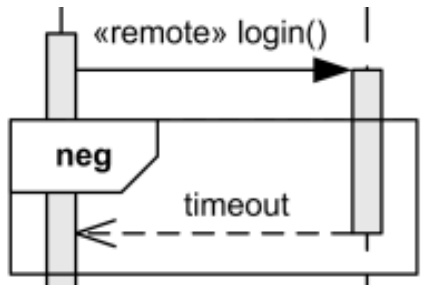
Parallèle

par désigne un interclassement (ou entrelacement) entre les comportements des opérandes. Les occurrences des événements des divers opérandes d'interaction peuvent être entrelacées de toutes les façons tant que l'ordre imposé par chaque opérande est préservé.



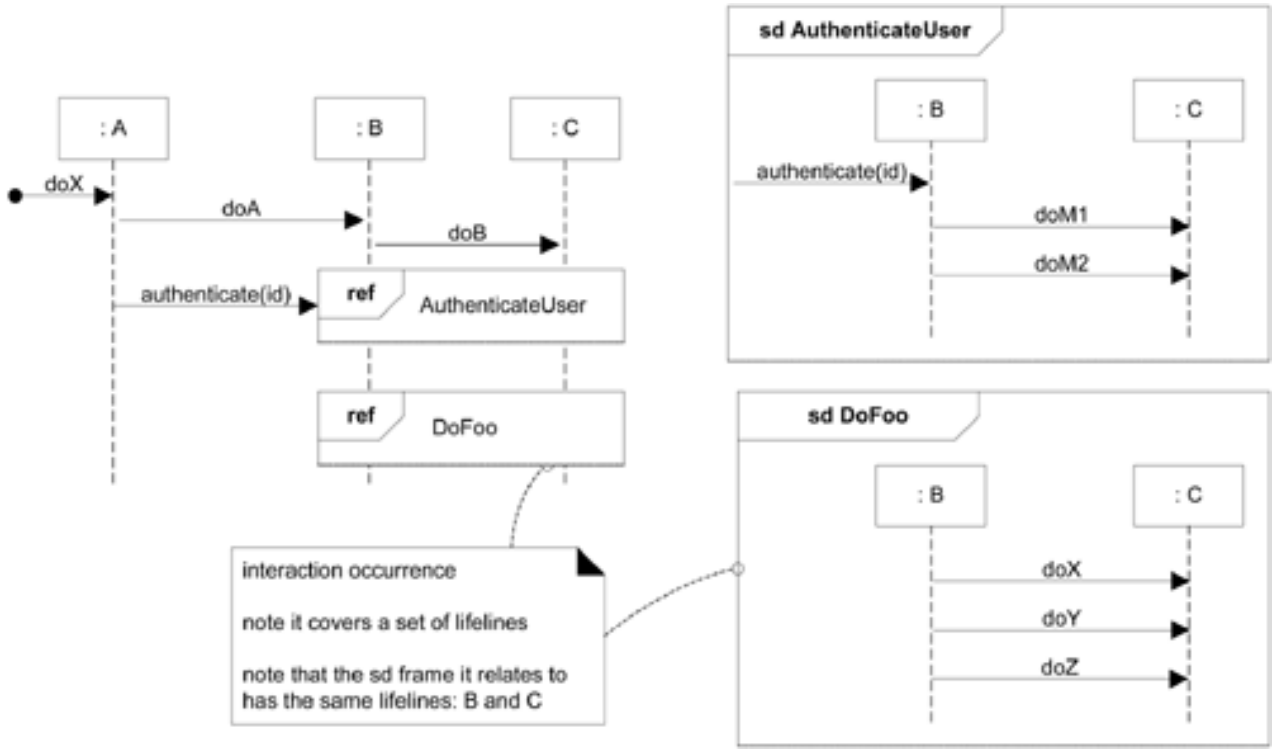
Scénario redouté

neg définit des traces invalides ou négatives. Les traces négatives sont des traces qui surviennent lorsque le système renvoie une erreur.



Référence

C'est un moyen de copier le contenu de l'interaction référencée à l'endroit de l'occurrence d'interaction afin de favoriser la réutilisation et de gérer la complexité.

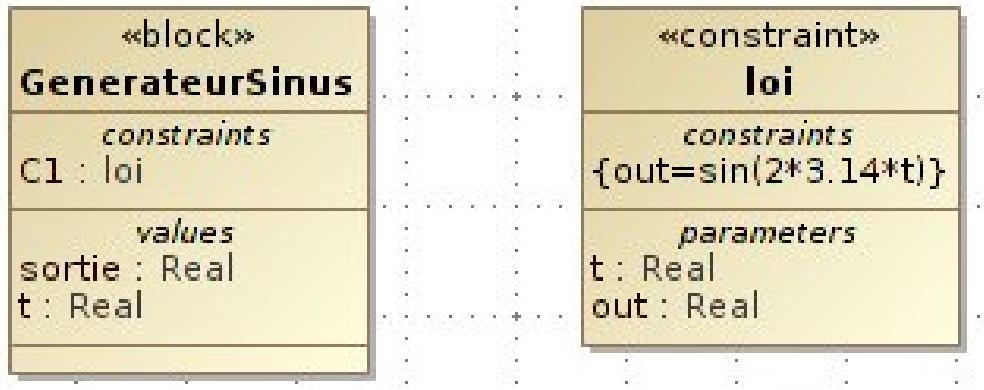


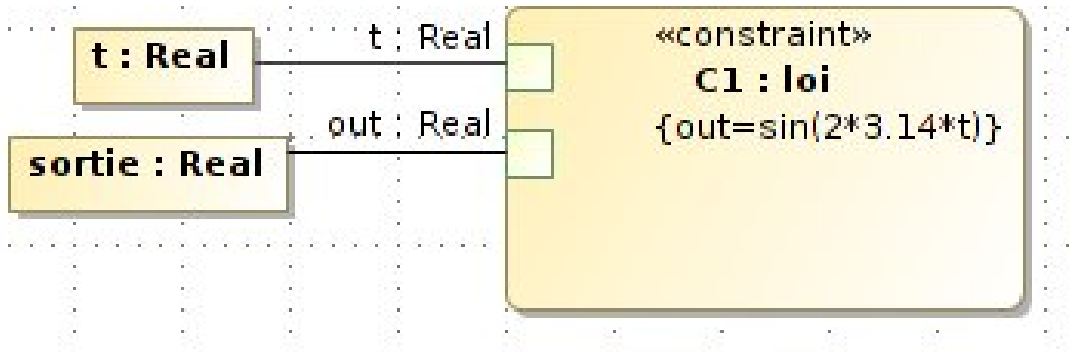
Programme

Les diagrammes de structure - Diagramme paramétrique

Contrainte

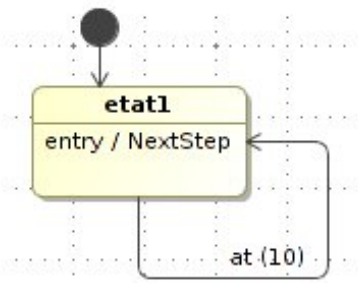
Un diagramme paramétrique inclut l'utilisation de block de contrainte pour contraindre les propriétés/paramètres d'un autre block à travers des équations ou des lois. L'état d'un système peut être spécifié en terme de valeurs des propriétés.

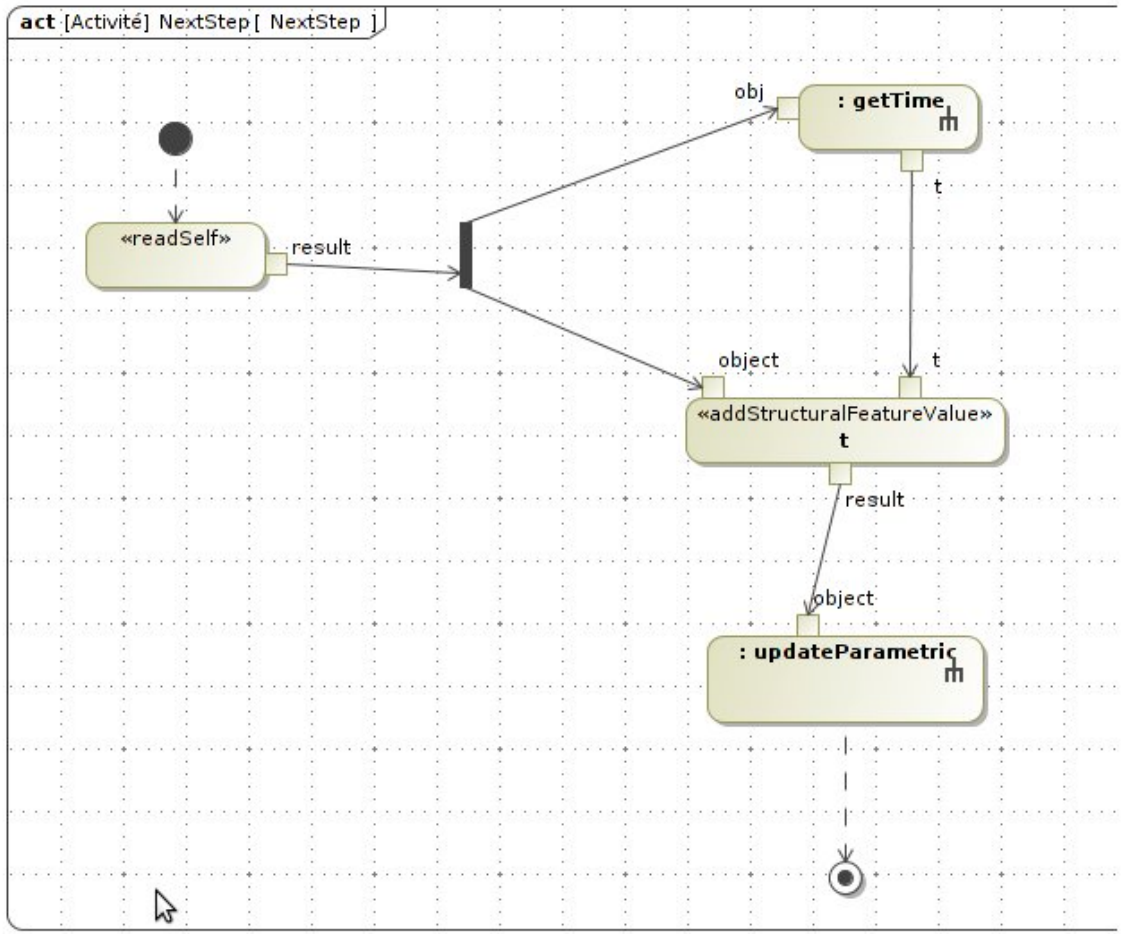




Changement d'état

Un changement d'état pourra conduire au re-calcul d'un ensemble d'équation ou de loi.





Programme

Les diagrammes de structure - Diagramme de blocks (bdd)

Compartiments d'un block

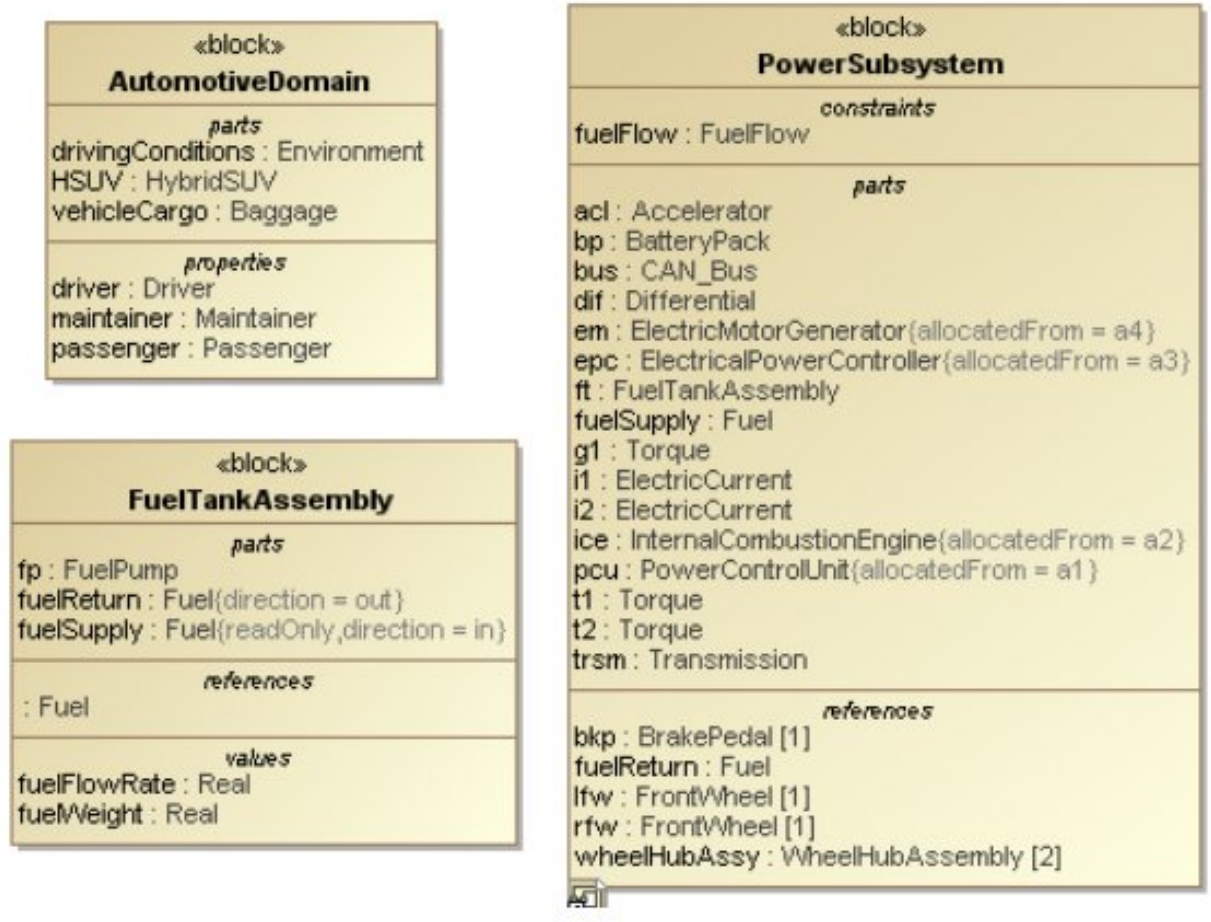
- Divers compartiments décrivent les caractéristiques du block
  - *Properties (parts, references, values, ports)*
  - *Operations*
  - *Constraints*
  - Allocations de ou vers d'autres éléments du modèle (ex. activités)
  - Exigences satisfaites par le *block*

Types de propriétés

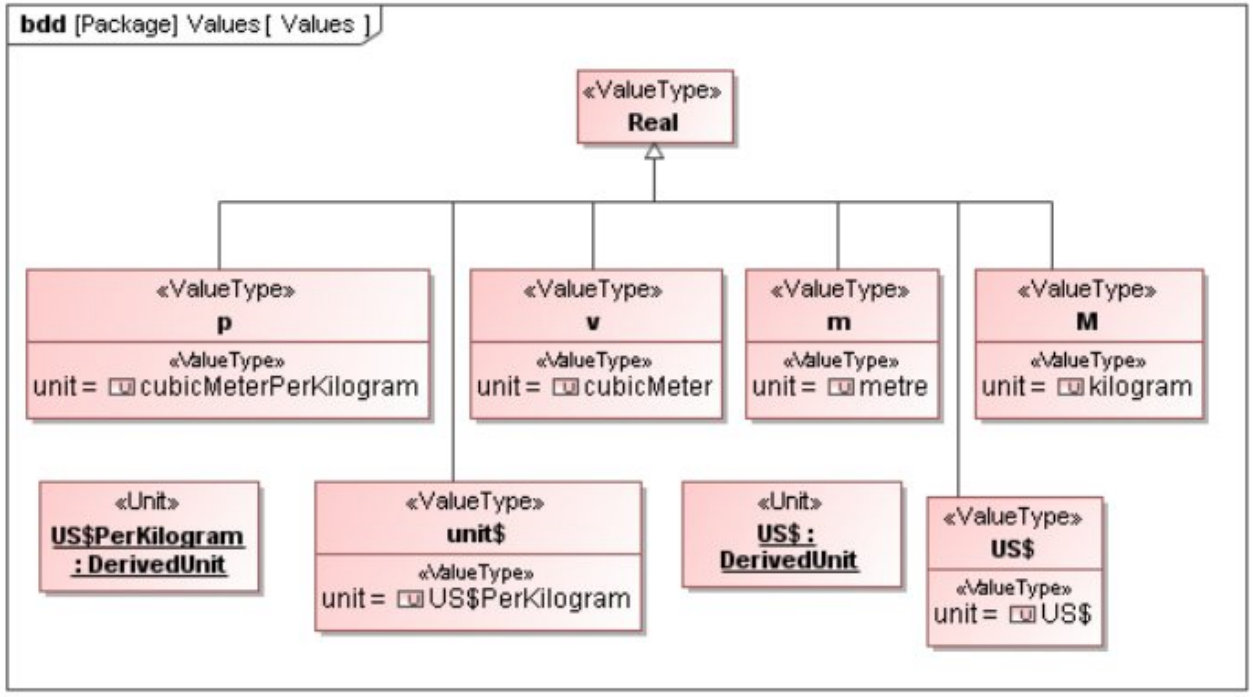
Une *propriété* est une caractéristique structurelle d'un block

- *Part property* (typé par un *block*)
  - Utilisation d'un block dans un contexte d'encapsulation (composition) d'un block
  - Exemple - right-front :wheel
- *Reference property* (typé par un *block*)
  - Une *Part* qui n'appartient pas à la composition d'un *block*
  - Exemple - agrégation ou association
- *Value property* (typé par une *value type*)
  - Une propriété quantifiable par une unité, une dimension
  - Exemple - cubicmeter

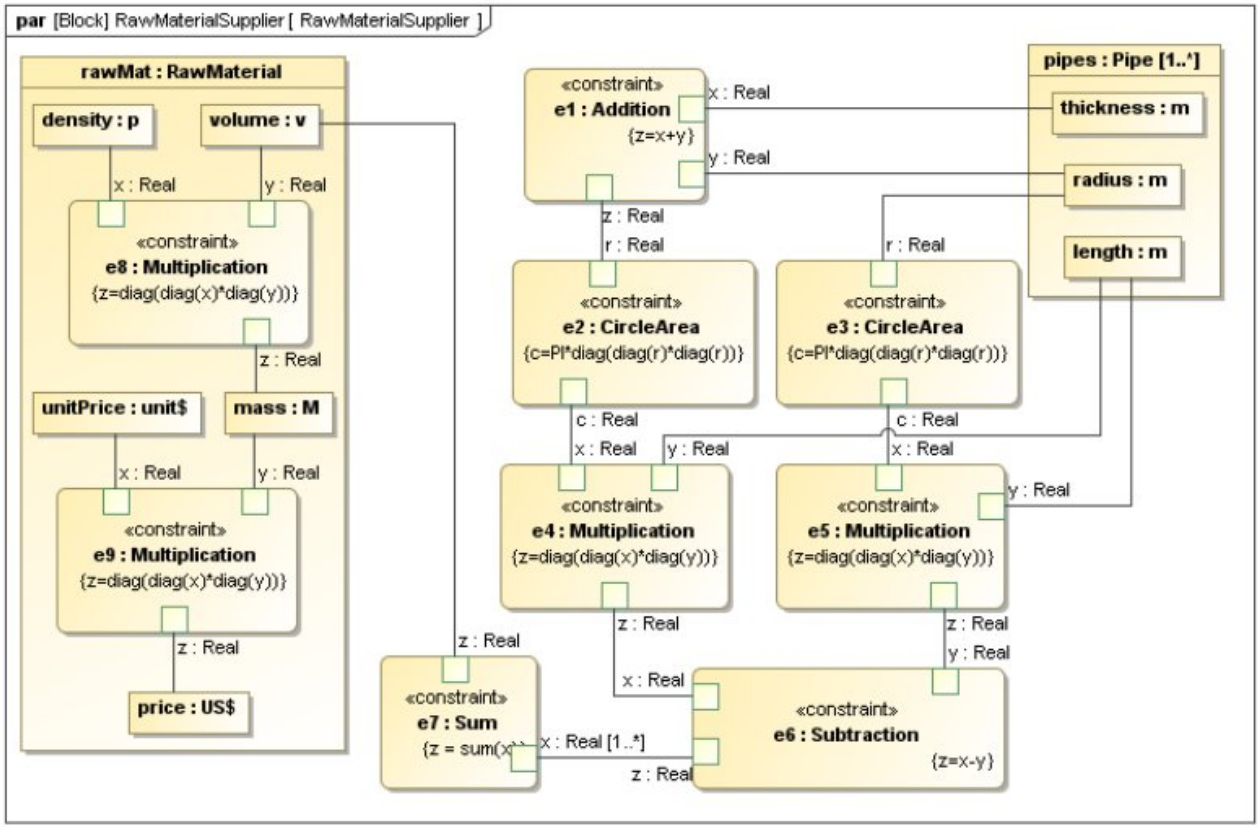
Compartiments d'un block



Value Type


bdd [Package] Values [Values]
«ValueType»
Real
«ValueType»
p
«ValueType»
unit = cubicMeterPerKilogram
«ValueType»
v
«ValueType»
unit = cubicMeter
«ValueType»
m
«ValueType»
unit = metre
«ValueType»
M
«ValueType»
unit = kilogram
«Unit»
US$PerKilogram
: DerivedUnit
«ValueType»
unit$
«ValueType»
unit = US$PerKilogram
«Unit»
US$:
DerivedUnit
«ValueType»
US$
«ValueType»
unit = US$

Utilisation des types de valeurs définies

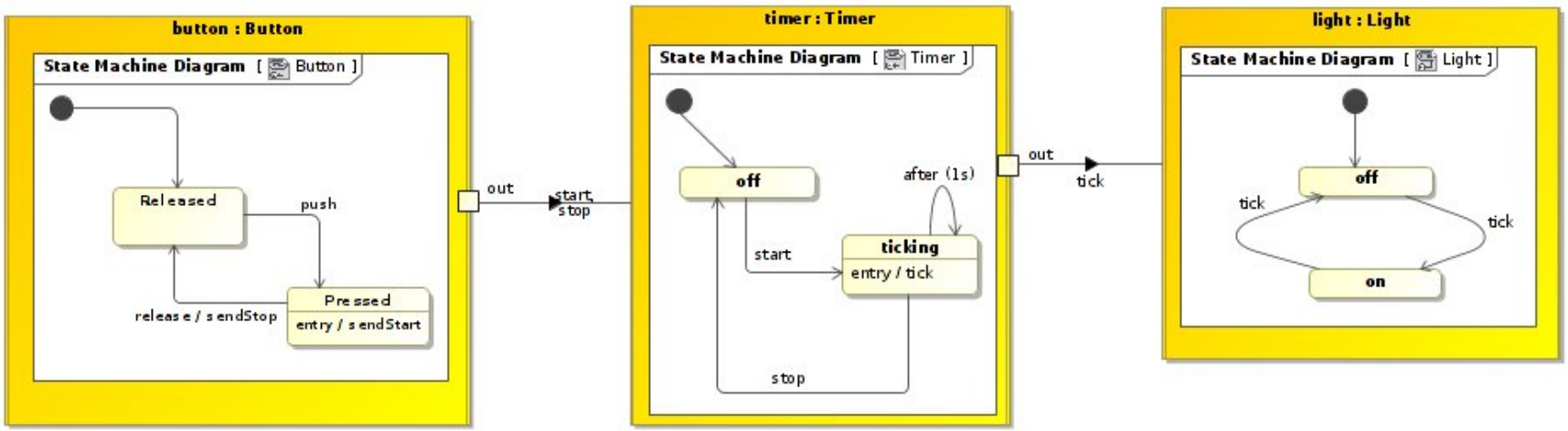


Programme

Les diagrammes de structure - Diagramme de description interne de blocks (ibd)

Diagramme de description interne de block

Représente la structure interne du block en terme de propriétés et de connections entre ces propriétés. Un block inclut les valeurs, les parts et les références, et les ports.



Programme

Les diagrammes d'exigences

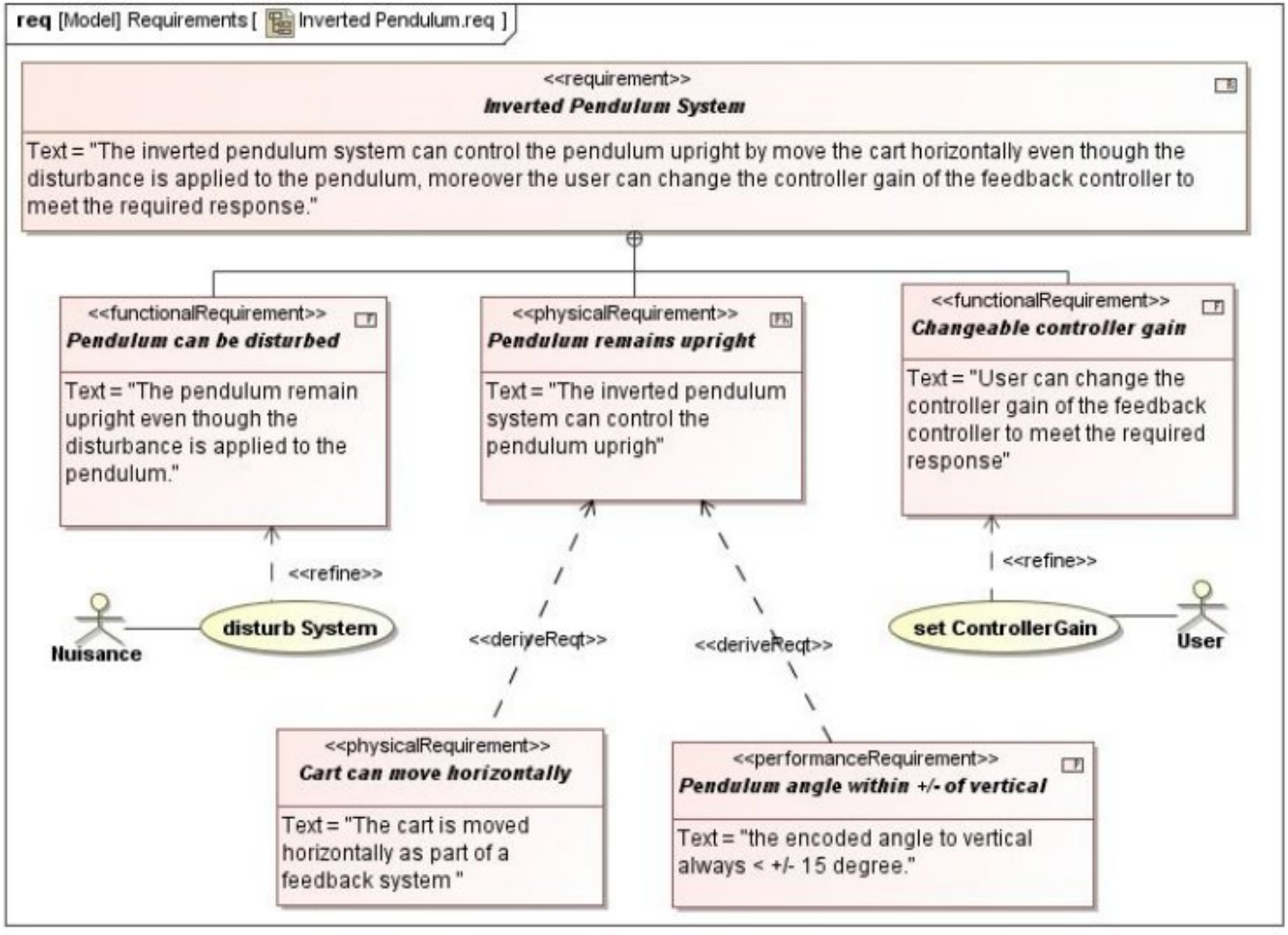
Types d'Exigences

- Fonctionnelle : une exigence qui spécifie un comportement que le système ou une partie du système doit réaliser
- Interface : une exigence qui spécifie les ports pour connecter le systèmes et les parties du système
- Performance : une exigence qui peut mesurer quantitativement l'étendu de satisfaction d'une condition ou d'une fonctionnalité
- Physique : une exigence qui spécifie des caractéristiques ou des contraintes physiques
- Business
- Usability

Relations entre Exigences et autres éléments du modèle

- trace : dépendance générale entre une exigence et tout autre élément du modèle
- satisfy : dépendance entre une exigence et un élément du modèle qui remplit cette exigence (généralement un block satisfait une exigence alors qu'une activity est allouée à un block)
- verify : dépendance entre une exigence et un cas de test (un cas de test est de type behavior)
- derive : dépendance entre deux exigences
- refine : dépendance entre une exigence et un use case (décrit par un diagramme de séquence par exemple ou une machine à état (pour décrire les modes opérationnels par exemple).

Exemple


req [Model] Requirements [Inverted Pendulum.req]
<<requirement>>
Inverted Pendulum System
Text = "The inverted pendulum system can control the pendulum upright by move the cart horizontally even though the disturbance is applied to the pendulum, moreover the user can change the controller gain of the feedback controller to meet the required response."
<<functionalRequirement>>
Pendulum can be disturbed
Text = "The pendulum remain upright even though the disturbance is applied to the pendulum."
<<physicalRequirement>>
Pendulum remains upright
Text = "The inverted pendulum system can control the pendulum uprigh"
<<functionalRequirement>>
Changeable controller gain
Text = "User can change the controller gain of the feedback controller to meet the required response"
<<refine>>
<<refine>>
Nuisance
disturb System
<<deriveReqt>>
<<deriveReqt>>
set ControllerGain
User
<<physicalRequirement>>
Cart can move horizontally
Text = "The cart is moved horizontally as part of a feedback system "
<<performanceRequirement>>
Pendulum angle within +/- of vertical
Text = "the encoded angle to vertical always < +/- 15 degree."