Etude de Cahiers des Charges Industriels à l'aide du Langage UML

John Klein

Université de Lille1

Plan

- Votre reponsable vous nomme en charge d'un projet industriel **Navig**.
- Ce projet consiste en le développement d'automates validant l'accès à un transport en commun. Un usager abonné reçoit une carte personelle contenant un émetteur RFID. L'automate doit lire le numéro d'utilisateur sur la carte et si l'abonnement est valide, il laisse l'usager accéder au service.
- Votre entreprise est un prestataire de service et votre client une société de transport d'une grande agglomération.
- Un audit des besoins du client a été mené au préalable permettant l'établissement d'un cahier des charges.
- Ce document de 200 pages vous est livré par votre responsable, à vous de jouer !

- Votre reponsable vous nomme en charge d'un projet industriel **Navig**.
- Ce projet consiste en le développement d'automates validant l'accès à un transport en commun. Un usager abonné reçoit une carte personelle contenant un émetteur RFID. L'automate doit lire le numéro d'utilisateur sur la carte et si l'abonnement est valide, il laisse l'usager accéder au service.
- Votre entreprise est un prestataire de service et votre client une société de transport d'une grande agglomération.
- Un audit des besoins du client a été mené au préalable permettant l'établissement d'un cahier des charges.
- Ce document de 200 pages vous est livré par votre responsable, à vous de jouer !

- Votre reponsable vous nomme en charge d'un projet industriel **Navig**.
- Ce projet consiste en le développement d'automates validant l'accès à un transport en commun. Un usager abonné reçoit une carte personelle contenant un émetteur RFID. L'automate doit lire le numéro d'utilisateur sur la carte et si l'abonnement est valide, il laisse l'usager accéder au service.
- Votre entreprise est un prestataire de service et votre client une société de transport d'une grande agglomération.
- Un audit des besoins du client a été mené au préalable permettant l'établissement d'un cahier des charges.
- Ce document de 200 pages vous est livré par votre responsable, à vous de jouer !

- Votre reponsable vous nomme en charge d'un projet industriel **Navig**.
- Ce projet consiste en le développement d'automates validant l'accès à un transport en commun. Un usager abonné reçoit une carte personelle contenant un émetteur RFID. L'automate doit lire le numéro d'utilisateur sur la carte et si l'abonnement est valide, il laisse l'usager accéder au service.
- Votre entreprise est un prestataire de service et votre client une société de transport d'une grande agglomération.
- Un audit des besoins du client a été mené au préalable permettant l'établissement d'un cahier des charges.
- Ce document de 200 pages vous est livré par votre responsable, à vous de jouer !

- Votre reponsable vous nomme en charge d'un projet industriel **Navig**.
- Ce projet consiste en le développement d'automates validant l'accès à un transport en commun. Un usager abonné reçoit une carte personelle contenant un émetteur RFID. L'automate doit lire le numéro d'utilisateur sur la carte et si l'abonnement est valide, il laisse l'usager accéder au service.
- Votre entreprise est un prestataire de service et votre client une société de transport d'une grande agglomération.
- Un audit des besoins du client a été mené au préalable permettant l'établissement d'un cahier des charges.
- Ce document de 200 pages vous est livré par votre responsable, à vous de jouer !

Que faire ?

- Se décourager ? Déprimer ? ...
- Spécifier signifie définir, exprimer clairement selon une norme.
- On spécifie un maximum de choses dans un projet : les besoins, les composants de la solution, les données, les intervenants. Le fruit de la spécification est un ensemble de diagrammes et de textes qui forment un document.
- Modéliser signifie offrir une (ou des) représentation(s) traduisant le fonctionnement du système (ex : équations, fonctions de transfert, diagramme). Plus la modélisation est efficace, plus le document est clair, concis, facile à lire pour tous.

Que faire ?

- Rester zen et utiliser une méthode de spécification/modélisation.
- Spécifier signifie définir, exprimer clairement selon une norme.
- On spécifie un maximum de choses dans un projet : les besoins, les composants de la solution, les données, les intervenants. Le fruit de la spécification est un ensemble de diagrammes et de textes qui forment un document.
- Modéliser signifie offrir une (ou des) représentation(s) traduisant le fonctionnement du système (ex : équations, fonctions de transfert, diagramme). Plus la modélisation est efficace, plus le document est clair, concis, facile à lire pour tous.

Que faire ?

- Spécifier signifie définir, exprimer clairement selon une norme.
- On spécifie un maximum de choses dans un projet : les besoins, les composants de la solution, les données, les intervenants. Le fruit de la spécification est un ensemble de diagrammes et de textes qui forment un document.
- Modéliser signifie offrir une (ou des) représentation(s) traduisant le fonctionnement du système (ex : équations, fonctions de transfert, diagramme). Plus la modélisation est efficace, plus le document est clair, concis, facile à lire pour tous.

Que faire ?

- Spécifier signifie définir, exprimer clairement selon une norme.
- On spécifie un maximum de choses dans un projet : les besoins, les composants de la solution, les données, les intervenants. Le fruit de la spécification est un ensemble de diagrammes et de textes qui forment un document.
- Modéliser signifie offrir une (ou des) représentation(s) traduisant le fonctionnement du système (ex : équations, fonctions de transfert, diagramme). Plus la modélisation est efficace, plus le document est clair, concis, facile à lire pour tous.

Que faire ?

- Spécifier signifie définir, exprimer clairement selon une norme.
- On spécifie un maximum de choses dans un projet : les besoins, les composants de la solution, les données, les intervenants. Le fruit de la spécification est un ensemble de diagrammes et de textes qui forment un document.
- Modéliser signifie offrir une (ou des) représentation(s) traduisant le fonctionnement du système (ex : équations, fonctions de transfert, diagramme). Plus la modélisation est efficace, plus le document est clair, concis, facile à lire pour tous.

Pourquoi modéliser ?

- pour fragmenter le problème en sous-systèmes plus faciles à analyser.
- pour rationaliser le processus et éviter d'oublier certains aspects. Revenir sur la modélisation après avoir réalisé un prototype est très couteux.
- pour normaliser le développement du système et mieux collaborer avec l'équipe et les partenaires.
- pour éviter la surcharge verbeuse du cahier des charges. Un modèle est clair et compréhensible par tous les intervenants.

Quelle méthode choisir ?

- Chaque entreprise a, en général, une méthode de modélisation qui lui est propre pour des raisons historiques ou autres.
- Dans ce cours nous n'étudierons pas une méthode mais un langage de modélisation : UML.

Pourquoi un langage ?

- Un des buts d'UML est d'uniformiser les modèles de projet, mais il est impensable de faire changer de méthodes les entreprises.
- Un langage permet d'utiliser n'importe quelle méthode, mais d'exprimer les diagrammes de ces méthodes avec un seul langage compréhensible par tous.
- Par exemple, on peut reformuler tout SA-RT en langage UML tout en gardant l'esprit SA-RT.
- UML facilite donc l'inter-opérabilité de systèmes conçus avec des méthodes différentes.

Si UML est un langage alors ...

TABLE : Passer d'un langage à un autre.

Français	UML
mot	symbole
phrase	association de symboles
paragraphe	diagramme
rapport	document

- L'ensemble des symboles forme un dictionnaire.
- Les règles d'association entre symboles forment une syntaxe.

Pourquoi UML ?

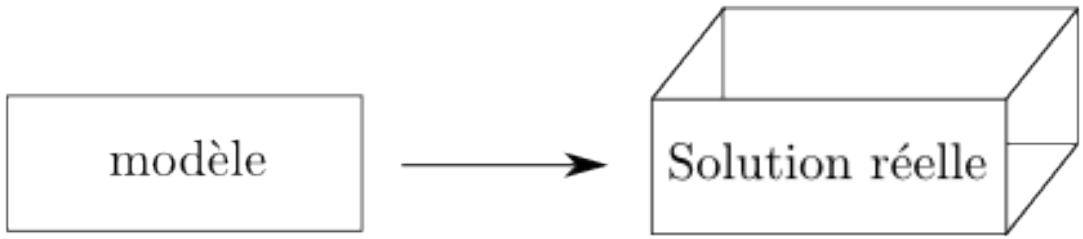
- UML est le fruit de la réflexion des plus grands spécialistes de la modélisation. Il est « la norme ».
- UML permet de modéliser :
  - des systèmes informatiques (ex : site web)
  - des systèmes industriels (ex : projet **navig**)
  - des systèmes abstraits (ex : sécurité d'un aéroport)
- UML est un langage formel : le dictionnaire de symboles qui le compose et la sémantique de ces symboles ne permettent pas de formulation ambigue.
- UML est très complet : peu de chance qu'un aspect de votre système ne puisse être capté par une expression UML. Même si c'était le cas, UML est extensible. Vous pouvez rajouter des mots au dictionnaire, ou même des diagrammes.

Comment utiliser UML ?

- Doit-on tout modéliser de A à Z en UML ? C'est peu probable. Trop de formalisme dans votre modélisation est parfois contre-productif. La plupart des entreprises préfèrent utiliser des diagrammes UML en appui d'un texte.
- Si l'entreprise a déjà sa méthode, vous devez l'utiliser et traduire les schémas en UML.
- Si l'entreprise ne possède pas de méthode ou souhaite en changer, il existe une approche générale qui consiste à examiner le modèle sous différentes vues.

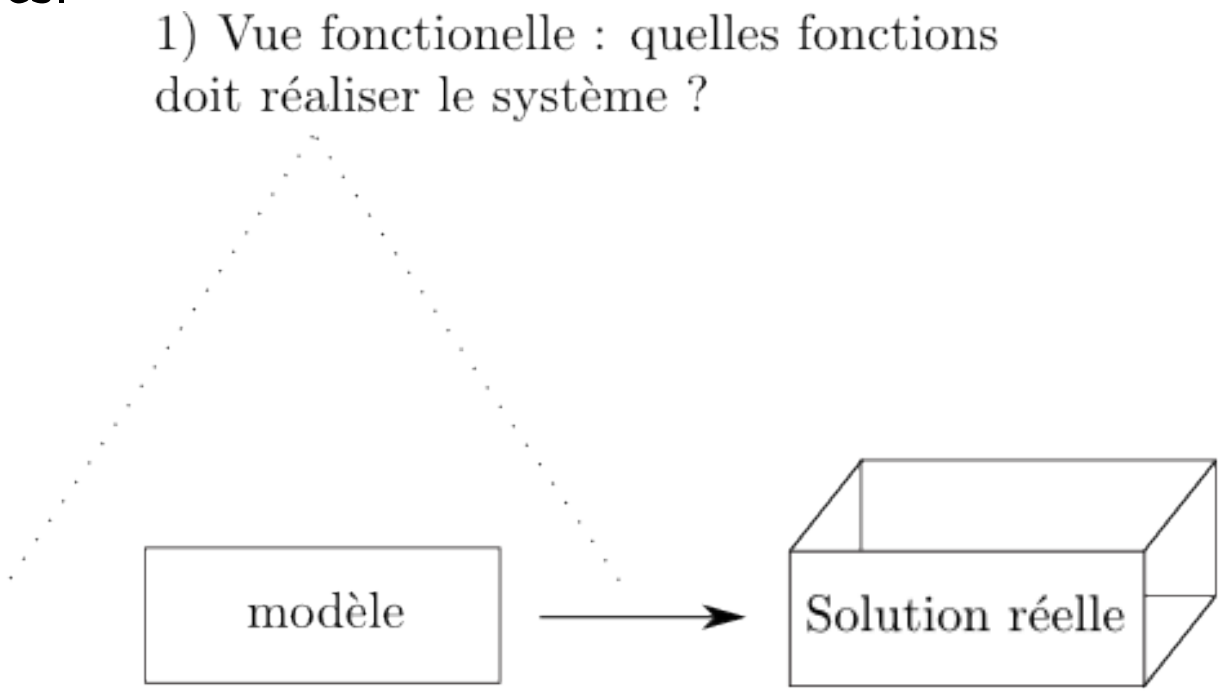
Modéliser en UML à l'aide des vues :

- Le modèle du système est inaccessible : impossible de tout capturer en un seul schéma sans rien oublier. Seules des vues particulières du modèle sont accessibles.



Modéliser en UML à l'aide des vues :

- Le modèle du système est inaccessible : impossible de tout capturer en un seul schéma sans rien oublier. Seules des vues particulières du modèle sont accessibles.

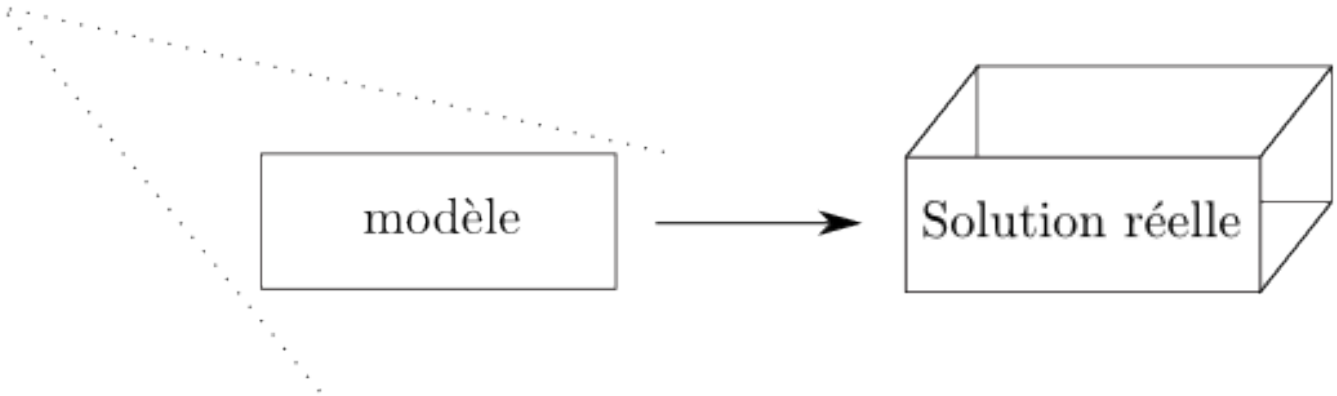


Modéliser en UML à l'aide des vues :

- Le modèle du système est inaccessible : impossible de tout capturer en un seul schéma sans rien oublier. Seules des vues particulières du modèle sont accessibles.

1) Vue fonctionelle : quelles fonctions doit réaliser le système ?

2) Vue structurelle : quels composants pour le système ? Quelle architecture ?

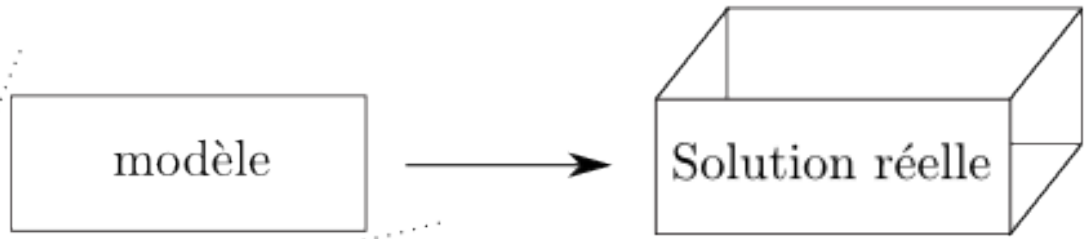


Modéliser en UML à l'aide des vues :

- Le modèle du système est inaccessible : impossible de tout capturer en un seul schéma sans rien oublier. Seules des vues particulières du modèle sont accessibles.

1) Vue fonctionelle : quelles fonctions doit réaliser le système ?

2) Vue structurelle : quels composants pour le système ? Quelle architecture ?



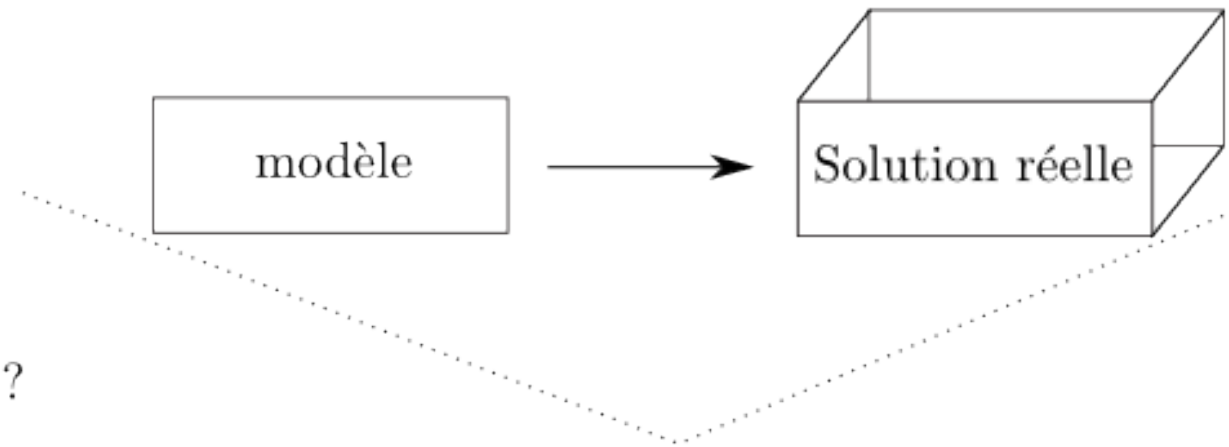
3) Vue dynamique : quelles séquences de vie du système ?

Modéliser en UML à l'aide des vues :

- Le modèle du système est inaccessible : impossible de tout capturer en un seul schéma sans rien oublier. Seules des vues particulières du modèle sont accessibles.

1) Vue fonctionelle : quelles fonctions doit réaliser le système ?

2) Vue structurelle : quels composants pour le système ? Quelle architecture ?



3) Vue dynamique : quelles séquences de vie du système ?

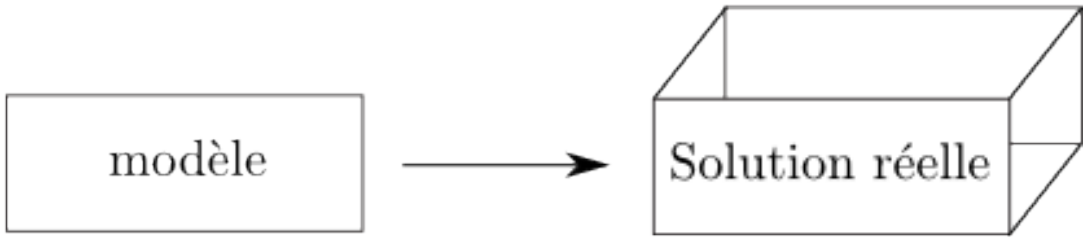
4) Vue physique : comment mettre en oeuvre le système ?

Modéliser en UML à l'aide des vues :

- Le modèle du système est inaccessible : impossible de tout capturer en un seul schéma sans rien oublier. Seules des vues particulières du modèle sont accessibles.

1) Vue fonctionelle :
diagramme de cas d'utilisation

2) Vue structurelle :
diagramme de classe et d'objets
diagramme de composants



3) Vue dynamique :
diagramme d'activité
diagramme de machines état
diagramme de sequence

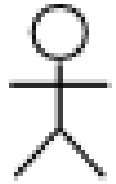
4) Vue physique :
diagramme de déploiement

et après ?

- Appliquer la méthode itérative :
  - simuler l'utilisation du système,
  - dégager de nouvelles exigences, ou des exigences superflues,
  - corriger le modèle et recommencer jusqu'à convergence du modèle.
- UML dans se version 2.0 a été pensé pour pouvoir être compilé. Des équipes de recherche essaient de faire en sorte que des machines puissent communiquer directement en UML. Un jour peut être, programmer un système informatique se résumera à piocher dans une bibliothèque écrite en UML.

Diagramme de cas d'utilisation : symboles

- l'acteur : c'est une entité extérieure au système et inter-agissant avec lui. Typiquement des utilisateurs humains, des sous-systèmes. Une bonne manière d'identifier un acteur est de se demander si on peut intervenir sur ce dernier.



- le cas d'utilisation : c'est une fonction particulière que doit réaliser le système. Vous identifiez ces cas en lisant le cahier des charges.

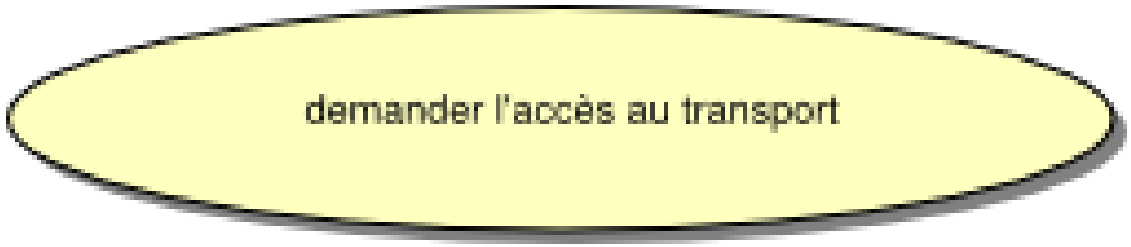


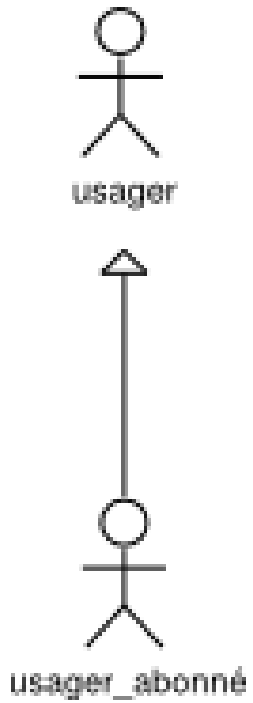
Diagramme de cas d'utilisation : syntaxe entre symboles

- entre un acteur et un cas d'utilisation : on parle de ligne de communication pour ce type de liaison.

Il est possible d'orienter ces liaisons quand un sens de communication est privilégié mais cela n'a rien d'obligatoire.
Plusieurs acteurs peuvent intervenir pour un même cas d'utilisation.

Diagramme de cas d'utilisation : syntaxe entre symboles

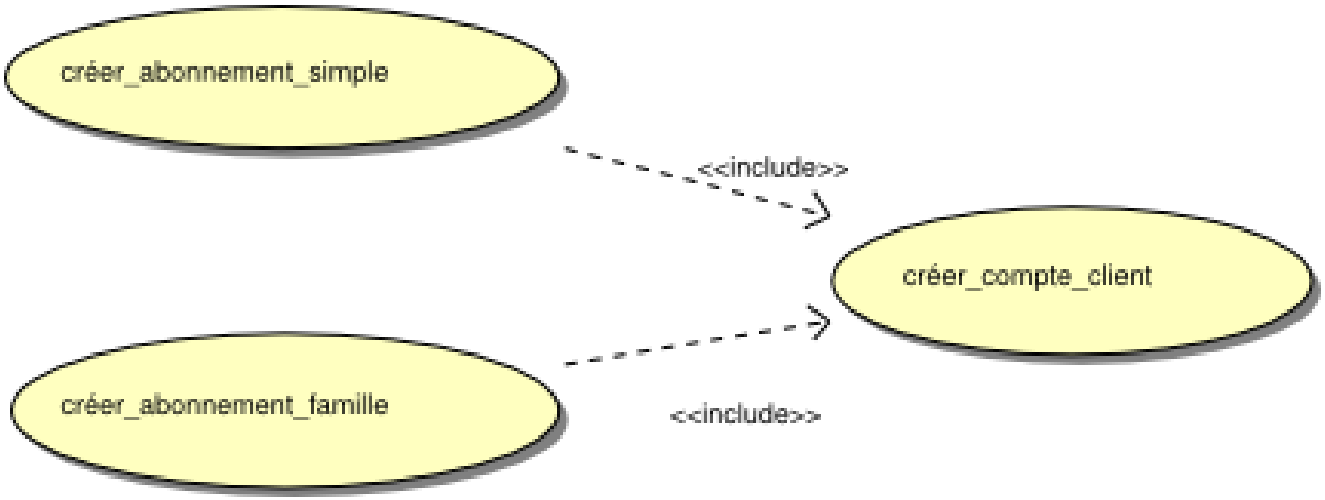
- entre deux acteurs : la seule liaison possible est la généralisation (héritage).



Un usager abonné est un type particulier d'usager. Ils n'utiliseront pas les services de la même manière mais tous les attributs d'un usager se retrouvent dans les attributs d'un usager abonné.

Diagramme de cas d'utilisation : syntaxe entre symboles

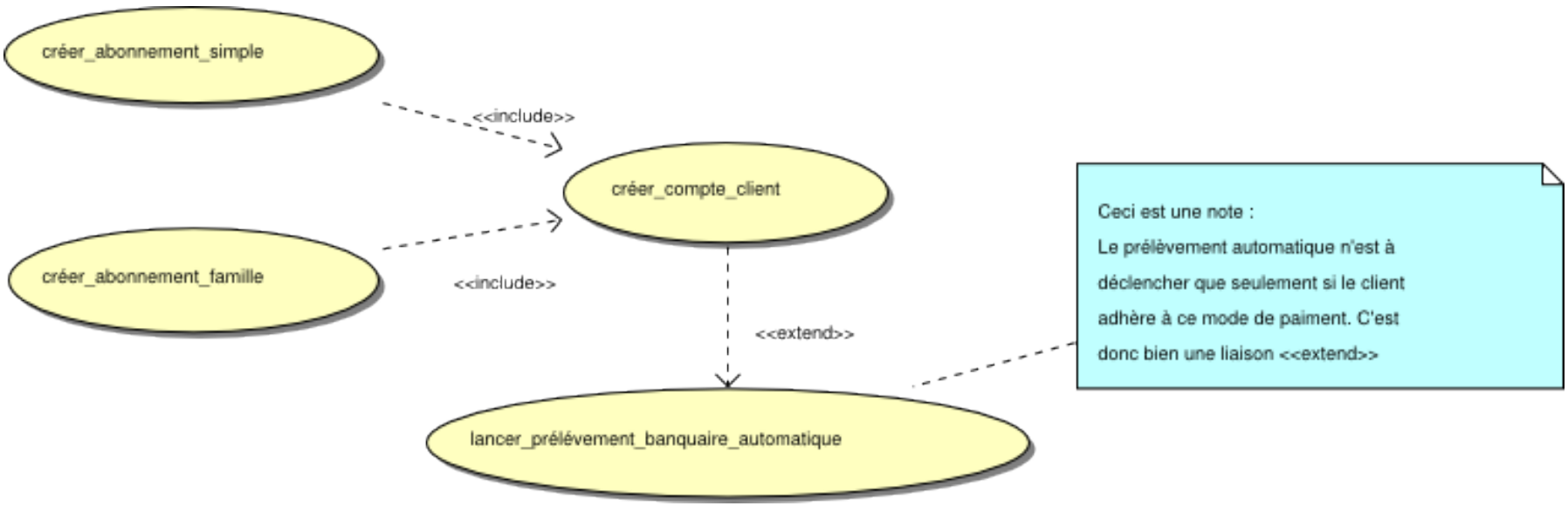
- entre deux cas d'utilisation : la liaison la plus répandue est celle d'inclusion. Deux cas d'utilisation ont en commun une même procédure. On crée alors un troisième cas correspondant à cette procédure et les deux cas initiaux sont reliés à ce nouveau cas par une liaison « include ».



Cela permet d'éviter la redondance dans la gestion des cas : que ce soit pour un abonnement simple ou famille, vous avez de toutes façons besoin de créer un compte client !

Diagramme de cas d'utilisation : syntaxe entre symboles

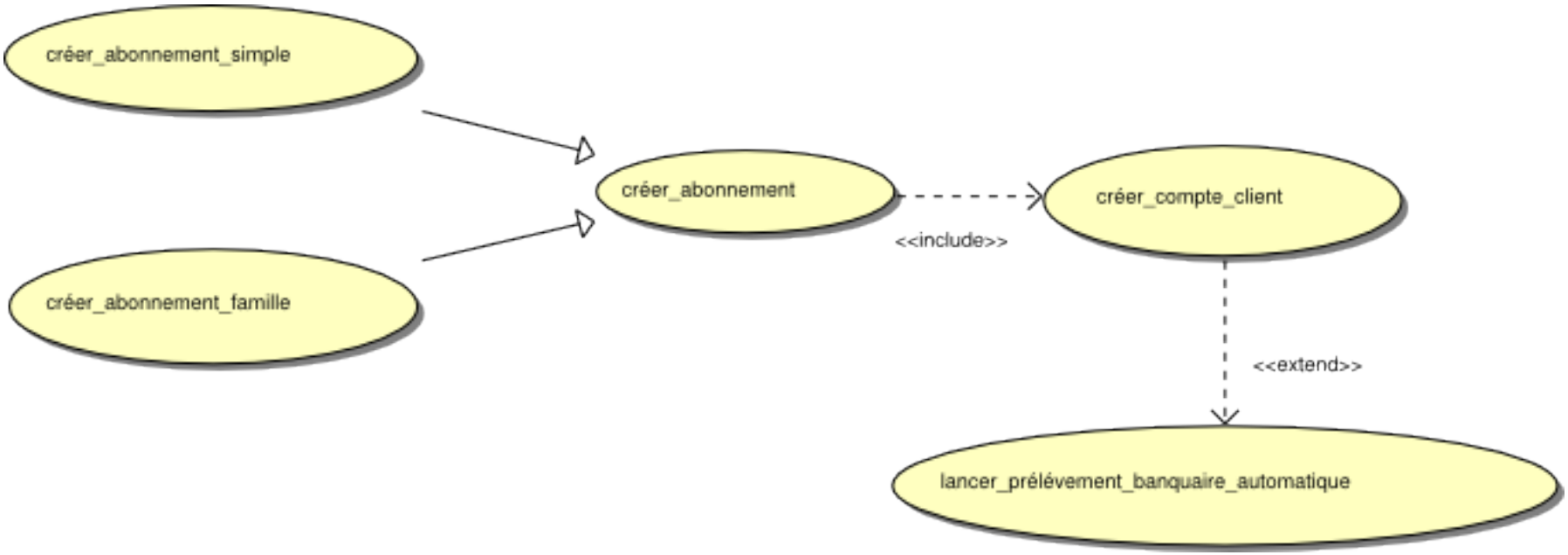
- entre deux cas d'utilisation : quand un cas est inclus dans un autre mais que son usage est falcultatif, UML note alors cette liaison ≪ extend ≫.



Attention ! Cette liaison, malgré sa similitude avec le ≪ extend ≫ de Java, ne concerne pas l'héritage !

Diagramme de cas d'utilisation : syntaxe entre symboles

- entre deux cas d'utilisation : la relation d'héritage existe en effet entre cas d'utilisation et elle est représentée par le symbole habituel.



créer_abonnement_simple et *créer_abonnement_famille* sont deux cas particuliers de création d'abonnement. On ne fait hériter du cas *créer_compte_client* car on estime que la création d'abonnement nécessite plus d'étapes que la création d'un compte.

Diagramme de cas d'utilisation : diagramme détaillé

- Le diagramme de cas d'utilisation donne un aperçu de la façon dont sera utilisé le système et des fonctions qu'il doit remplir. Il est cependant important de détailler chaque cas d'utilisation. En général, ces détails sont fournis sous forme d'un tableau :

TABLE : Lancer_prélèvement_banquaire_automatique

Détail	
Exigence associée	Pouvoir régler par prélèvement banquaire automatique
Objectif dans le contexte	Un nouveau client s'abonne et on lui propose le prélèvement automatique comme mode de paiement.
Précondition	Le client doit en faire la demande explicite et fournir un RIB
Condition de succès	Sa banque autorise le prélèvement.
Condition d'échec	Le numéro IBAN ne correspond à aucun compte La banque refuse le prélèvement.
Acteurs principaux	Le client, le commercial ou le serveur
Acteurs secondaires	La banque du client
Déclencheur	Le commercial ou le serveur

Diagramme de cas d'utilisation : diagramme détaillé

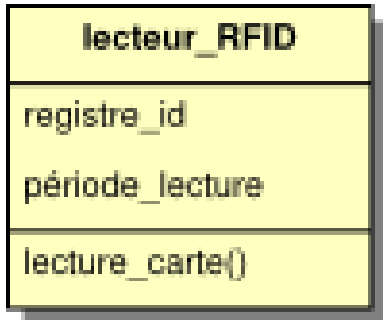
- Le diagramme de cas d'utilisation donne un aperçu de la façon dont sera utilisé le système et des fonctions qu'il doit remplir. Il est cependant important de détailler chaque d'utilisation. En général, ces détails sont fournis sous forme de tableau :

TABLE : Lancer_prélèvement_banquaire_automatique

Détail	
Flux principal (5 étapes)	1. Le commercial rentre le numéro IBAN fournit dans le RIB dans l'application 2. Le commercial appuie sur le bouton "envoi de demande automatique de prélèvement". 3. La demande est imprimée et envoyée à l'adresse personelle du client. 4. Le client remplit la demande et dépose le document à sa banque. 5. Un transfert d'argent correspondant au coût de l'abonnement est constaté au 1er du mois sur le compte de la société en provenance du compte du client.
Extensions	5.1 En cas de non versement, l'abonnement est suspendu. (alternative si l'étape 5 échoue.)

Diagramme de classes : symboles

- la classe : le cas d'utilisation capture le fonctionnement du système face aux éxigences. Une classe décrit un objet nécessaire à la satisfaction des exigences

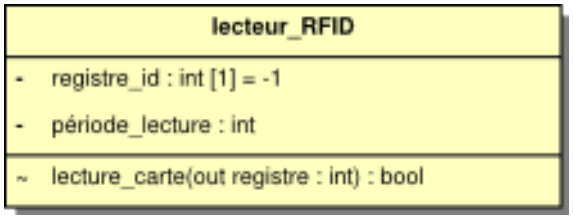


- Une classe représente de manière abstraite l'objet en opposition avec un objet concret qui est une instance de la classe ex : classe = système d'exploitation, instance = Ubuntu 10.1.
- Une classe possède :
  - un nom (Lecteur_RFID),
  - un ou plusieurs attributs (registre_ID,période_lecture),
  - une ou plusieurs opérations (lecture_carte).

Diagramme de classes : symboles

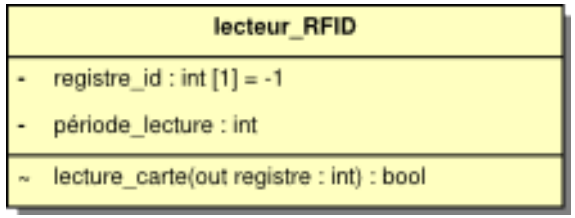
- La façon dont les classes s'imbriquent les unes aux autres capture la structure du système.
- Il est très rare que la structure puisse être entièrement décrite sur un seul diagramme de classe.
- Un ensemble de classes attrayant à un même objectif peuvent être regroupées en paquetage.

Diagramme de classes : classes détaillées



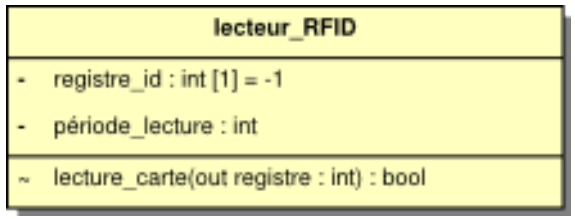
- De même que dans un langage orienté objet, les attributs et les méthodes se voient attribuées une visibilité :
  - $+$: accès public, toute classe a accès à cet élément,
  - $\#$: accès protégé, seules les classes héritant ont accès à cet élément,
  - ˜ : accès de paquetage, seules les classes du paquetage ont accès à cet élément,
  - - : accès privé, seule la classe elle-même peut accéder à cet élément.

Diagramme de classes : classes détaillées



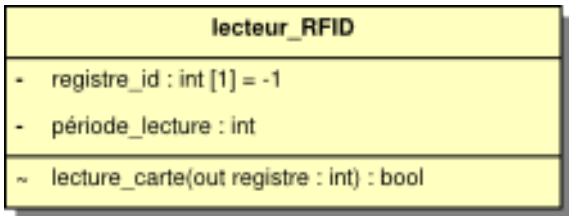
- On définit également le type de chaque élément après le nom et précédé du symbole « : ».

Diagramme de classes : classes détaillées



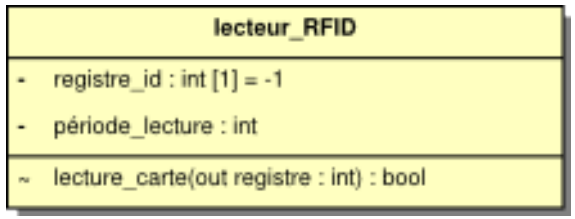
- Pour un attribut, on peut également spécifier sa multiplicité, comme ici pour l'attribut registre dont la multiplicité est 1. Un attribut avec une multiplicité supérieure à 1 peut être vu comme un tableau. La multiplicité est entourée de crochets « [.] ».

Diagramme de classes : classes détaillées



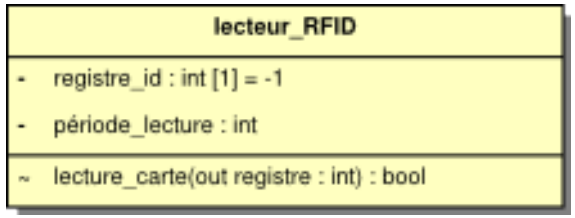
- Pour un attribut, ou un paramètre de méthode, on peut préciser une valeur par défaut. Cette valeur est placée en toute fin et est précédée de $\ll = \gg$.

Diagramme de classes : classes détaillées



- Pour un paramètre de méthode, on peut préciser sa direction : ≪ in ≫ pour un paramètre d'entrée et ≪ out ≫ pour un paramètre de sortie.

Diagramme de classes : classes détaillées



- Tous ces détails prennent beaucoup de place et en général, la définition détaillée d'une classe est renvoyée en annexe. Sur le diagramme de classe, on préfère se limiter au nom de classe et éventuellement des attributs et des méthodes pour plus de lisibilité.

Diagramme de classes : syntaxe

- une première façon de relier deux classes est l'héritage ou généralisation (comme pour les cas d'utilisation ou les acteurs). Tous les attributs et les méthodes de la classe mère sont présents chez la classe fille.

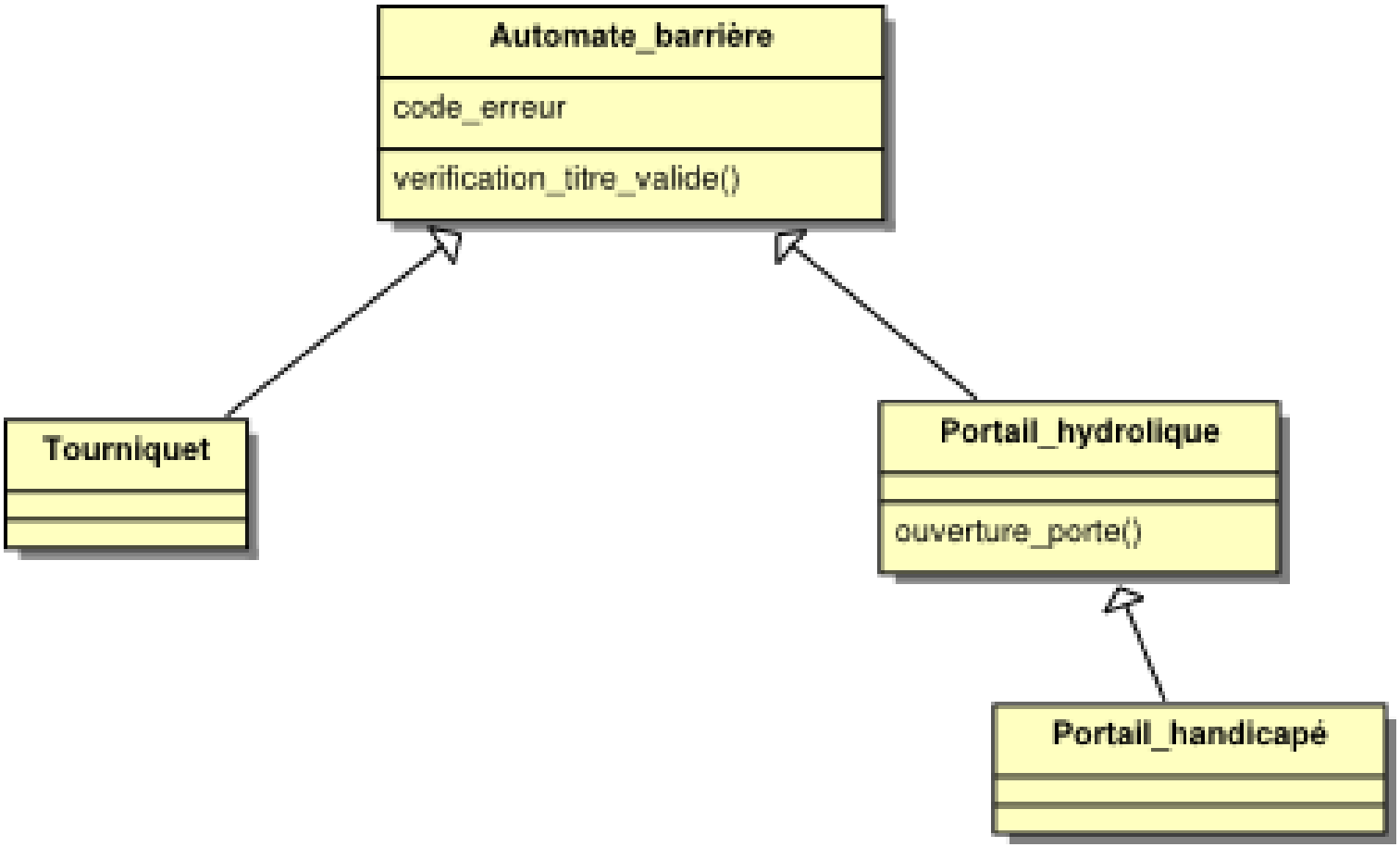


Diagramme de classes : syntaxe

- Comme en langage orienté objet, on retrouve la notion d'interface. Une interface est une forme de spécification. Elle oblige une classe qui la réalise, a posséder certaines fonctionalités.
- Elle peut se voir comme une « certification ». Puisque la classe *Portail_hydrolique* réalise l'interface *Norme_sécurité*, *Portail_hydrolique* est certifiée conforme par rapport à *Norme_sécurité*.

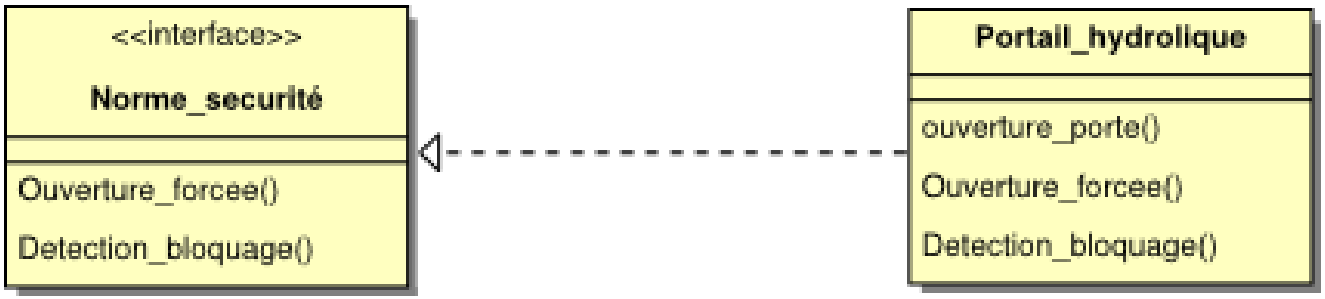


Diagramme de classes : syntaxe

- Quand une classe contient, une fois instanciée, un ou plusieurs éléments d'une autre, ces classes sont liées par une relation d'agrégation.
- Quand de plus, la destruction d'un objet de la classe principale implique la destruction de celles qui le compose, on utilise une relation de composition (composition plus fort qu'aggrégation).

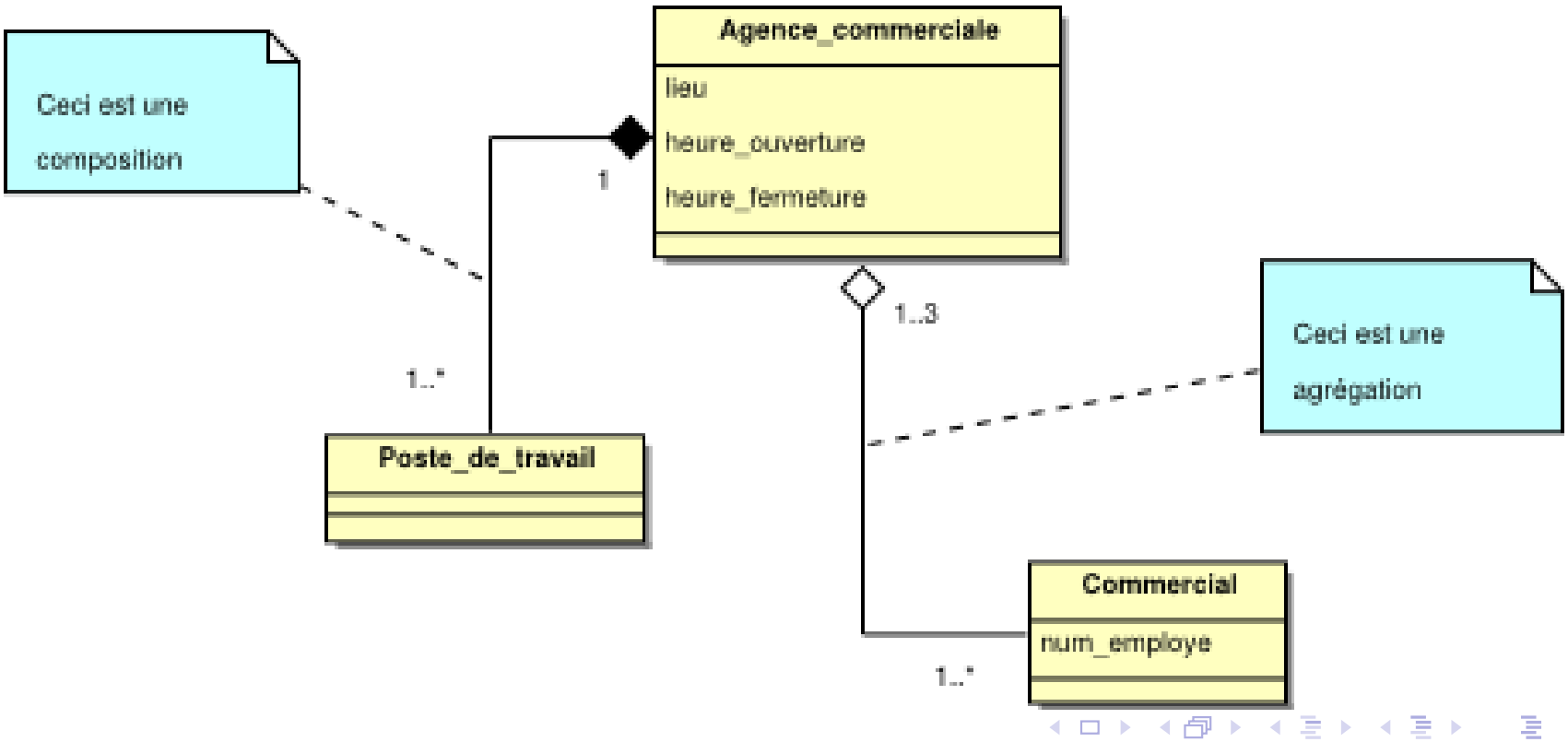


Diagramme de classes : syntaxe

- Quand deux classes sont amenées à collaborer mais que ce qui les lie ne correspond pas à de la composition ou de l'agrégation, on utilise la relation d'assossiation.
- La composition et l'agrégation sont des formes particulières d'association.

Diagramme de classes : syntaxe

- Quand une classe peut être amenée à fonctionner avec un objet d'une autre classe, on utilise la relation de dépendance. C'est une forme plus faible de liaison que l'association.
- Contrairement à l'association, l'orientation est obligatoire pour une dépendance.
- L'intérêt d'une dépendance est de souligner, que deux entités doivent être active au même moment pour que tout fonctionne correctement.

Diagramme d'objets :

- Le diagramme d'objet contient des instances de classes, appelés objets et des instances d'associations appelés liens.

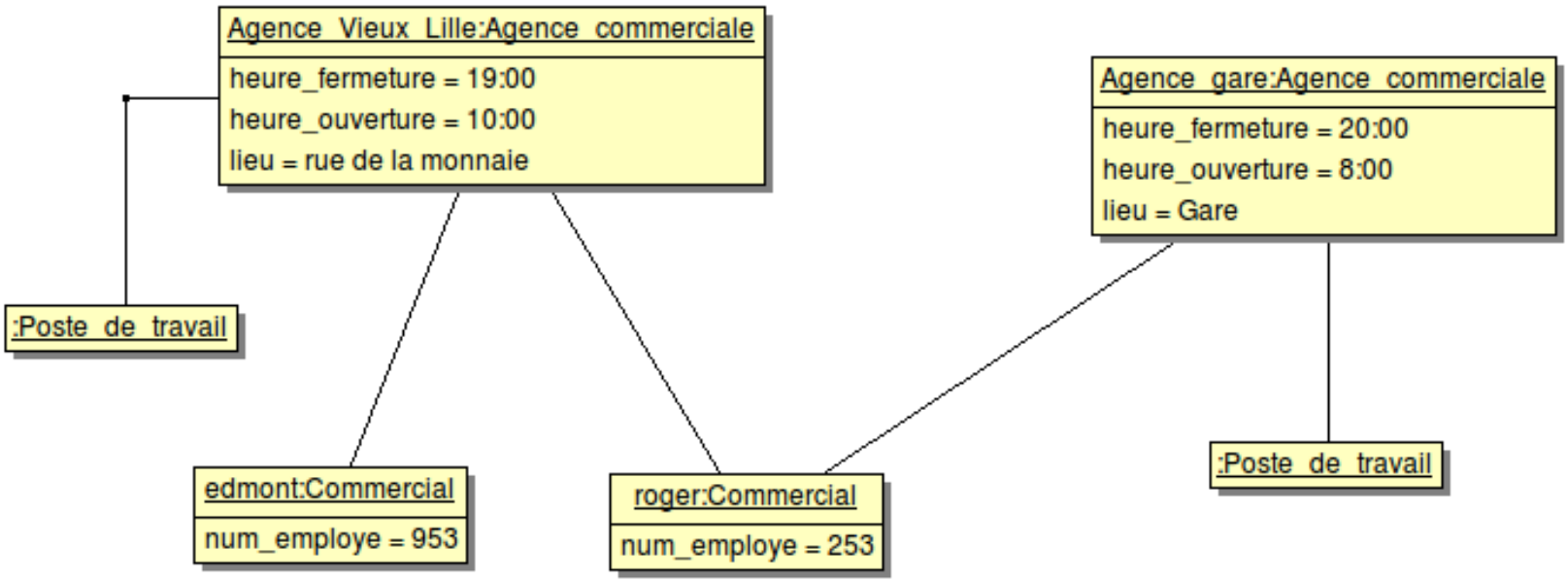


Diagramme de composants :

- Le diagramme de composants peut être réalisé avant ou après les diagrammes de classes.
- Son but est de décrire l'architecture du système de manière plus globale qu'un diagramme de classe.
- Il appartient donc bien à la vue structurelle.

Diagramme de composants : symbole

- Le diagramme de composants contient un seul symbole : le composant.
- Un composant est un élément du système qui encapsule une ou plusieurs classes.
- Il est pensé pour être ré-utilisable dans un autre projet ou remplaçable pour la maintenance.

Diagramme de composants : symbole

- Le diagramme de composants contient un seul symbole : le composant.
- Un composant est un élément du système qui encapsule une ou plusieurs classes.
- Il est pensé pour être ré-utilisable dans un autre projet ou remplaçable pour la maintenance.

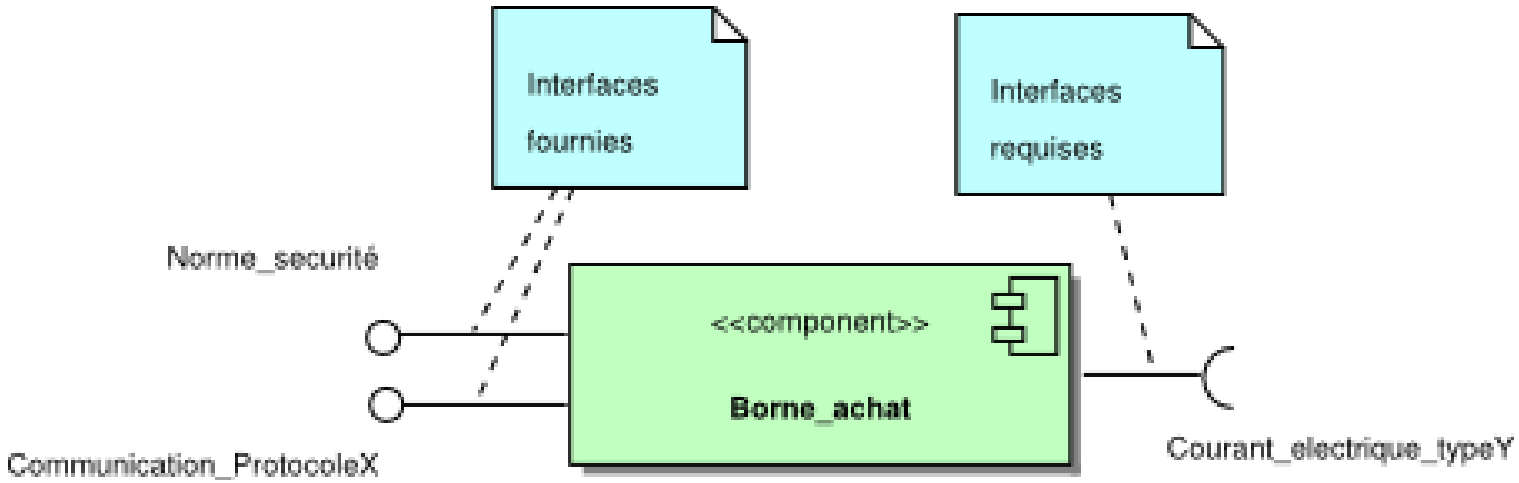


Diagramme de composants : symbole

- Pour être ré-utilisable, un composant doit être normalisé. C'est pourquoi il est important de représenter les interfaces qu'un composant réalise et les interfaces dont il dépend.
- On rappelle que quand une classe ou un composant réalise une interface, il s'engage à respecter une norme.
- Pour le remplacer un jour, il suffira de trouver un autre composant compatible avec les mêmes interfaces.

Diagramme de composants : syntaxe

- La syntaxe des diagrammes de composants est très simple : il suffit de les emboîter !

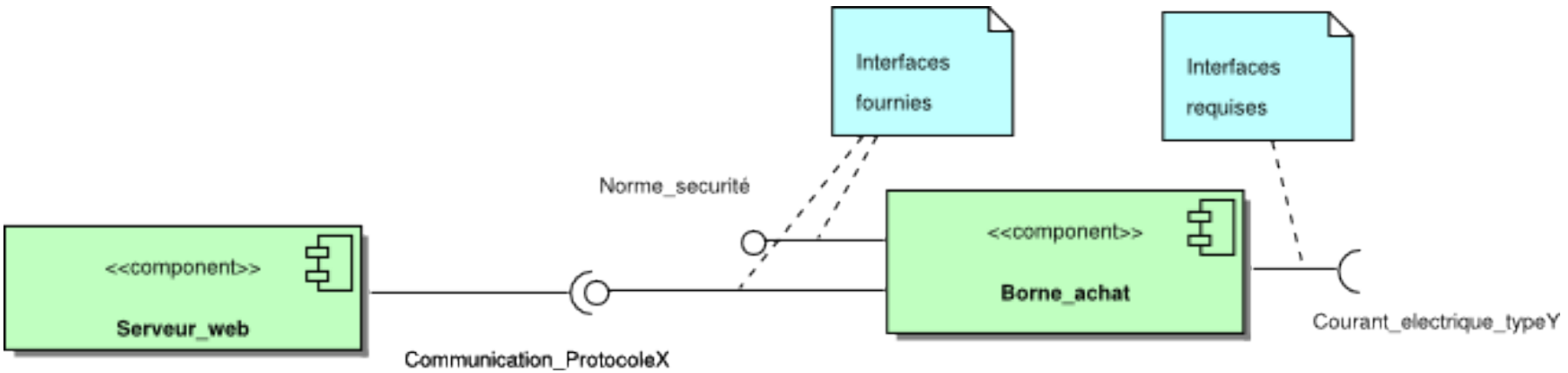


Diagramme de composants : remarque

- Vous pouvez également préciser les classes qui réalisent le composant, c'est à dire celles qui sont nécessaires à sa réalisation.

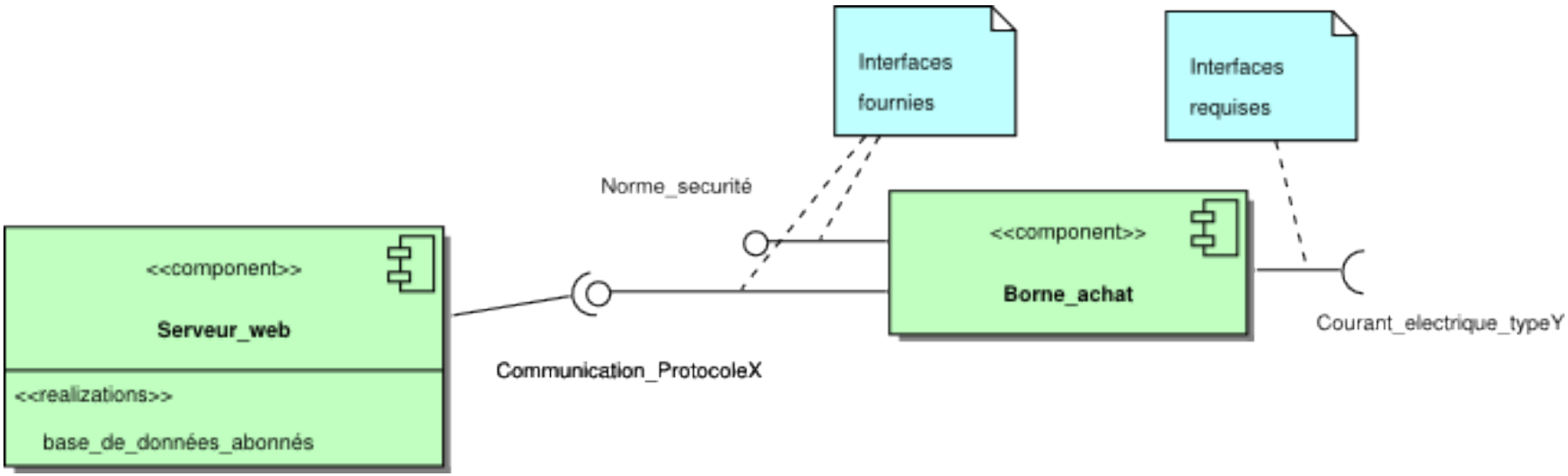


Diagramme d'activité :

- Le diagramme d'activité décrit une séquence que doit réaliser le système, c'est pourquoi il appartient à la vue dynamique (aspect temporel).
- Si le diagramme de cas d'utilisation répond à la question « que doit faire le système ? », le diagramme d'activité répond à « comment le système remplit ces fonctions ? ».
- Le diagramme d'activité vient en complément de la fiche détaillée d'un cas d'utilisation. On a vu que cette dernière contient un « flux principal ». C'est ce flux qui est exprimé à travers le diagramme d'activité.
- Le diagramme d'activité est dans l'esprit « flow chart », il est donc le plus facile à comprendre pour un interlocuteur non-initité à UML. On pourra comparer ce diagramme à un Grafcet même si il n'y a pas d'équivalence réelle entre les deux.

Diagramme d'activité : symboles

- Le symbole le plus important d'un diagramme d'activité est l'action :

- Pour démarrer le diagramme, on utilise le symbole de noeud initial :

- Et pour finir, le noeud final :

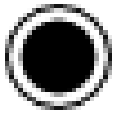
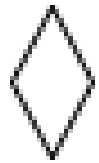


Diagramme d'activité : symboles

- Quand on souhaite que le diagramme contienne plusieurs branches mutuellement exclusives, on utilise le symbole de décision/confluence :



C'est l'équivalent de la divergence/convergence en OU en Grafcet.

- Quand on souhaite que le diagramme contienne plusiseurs branches qui s'exécutent en parallèle, on utilise le symbole de fourche/jonction :

C'est l'équivalent de la divergence/convergence en ET en Grafcet.

Diagramme d'activité : syntaxe

- Il existe une seule liaison entre symboles d'un diagramme d'activité : la transition, symbolisée par une flèche.
- Exemple : traduction du flux principal du cas d'utilisation ≪ Lancer_prélèvement_banquaire_automatique ≫ en diagramme d'activité :

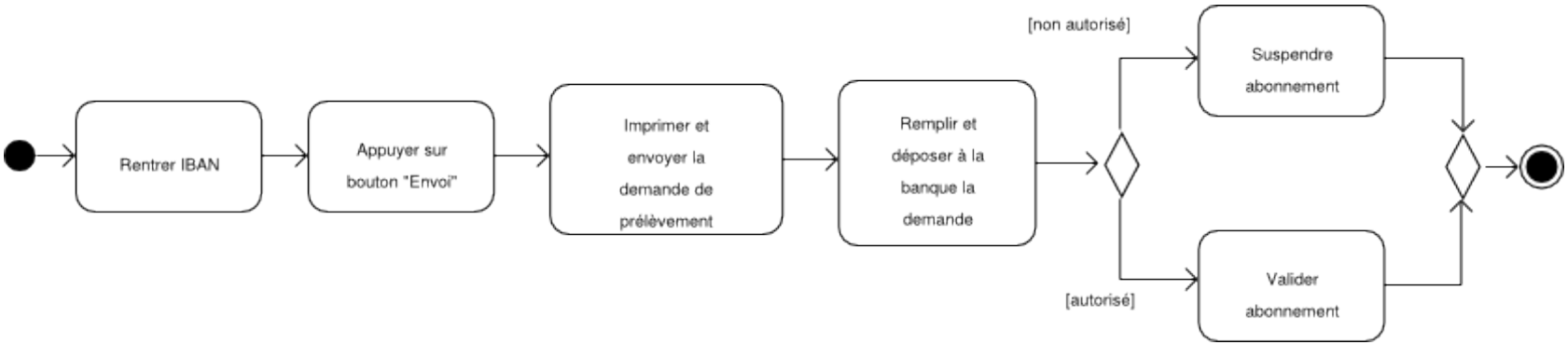
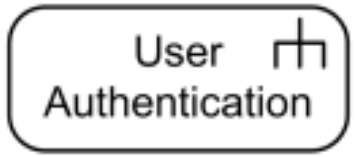


Diagramme d'activité : éléments supplémentaires

- Sur le diagramme précédent, on note la présence de conditions relatives aux deux cas exclusifs du symbole de décision. Ces conditions sont exprimées en langage OCL (Object Constraint Language). C'est le langage permettant d'exprimer contraintes et opérations arithmétiques de base, le plus couramment employé avec UML, mais vous pouvez choisir un autre langage.
- Quand une action est en réalité un appel à une activité subordonnée, on rajoute un symbole « rateau » :



- On peut également préciser la nature d'objets en entrée ou sortie d'une action à l'aide de « pins » :

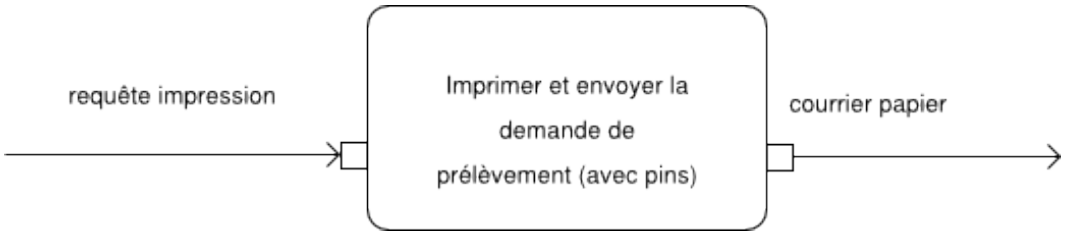


Diagramme d'activité : comparaison avec Grafcet

- Il est possible de ré-exprimer un Grafcet avec un diagramme d'activité à condition de s'astreindre à conditionner chaque transition (même quand cette condition est toujours vraie).
- Néanmoins, l'esprit de Grafcet reste différent et il n'est pas conseillé de procéder ainsi. Par exemple la notion d'étape ne correspond pas vraiment à celle d'action. Mieux vaut étendre UML en y incorporant Grafcet.

Diagramme de séquence : généralités

- Toujours dans la vue dynamique, le diagramme de séquence permet de décrire une partie du fonctionnement interne du système.
- Il capture l'interaction entre différentes parties du système en fonction du temps. Il répond à la question ≪ qui fait quoi ? ≫.
- Il y a des redondances par rapport au diagramme d'activité mais un diagramme de séquence est plus complet, et donc moins synthétique.

Diagramme de séquence : symboles et syntaxe

- Il existe un seul symbole dans ce diagramme, mais de nombreuses interactions possibles. Ce symbole est une partie du système.

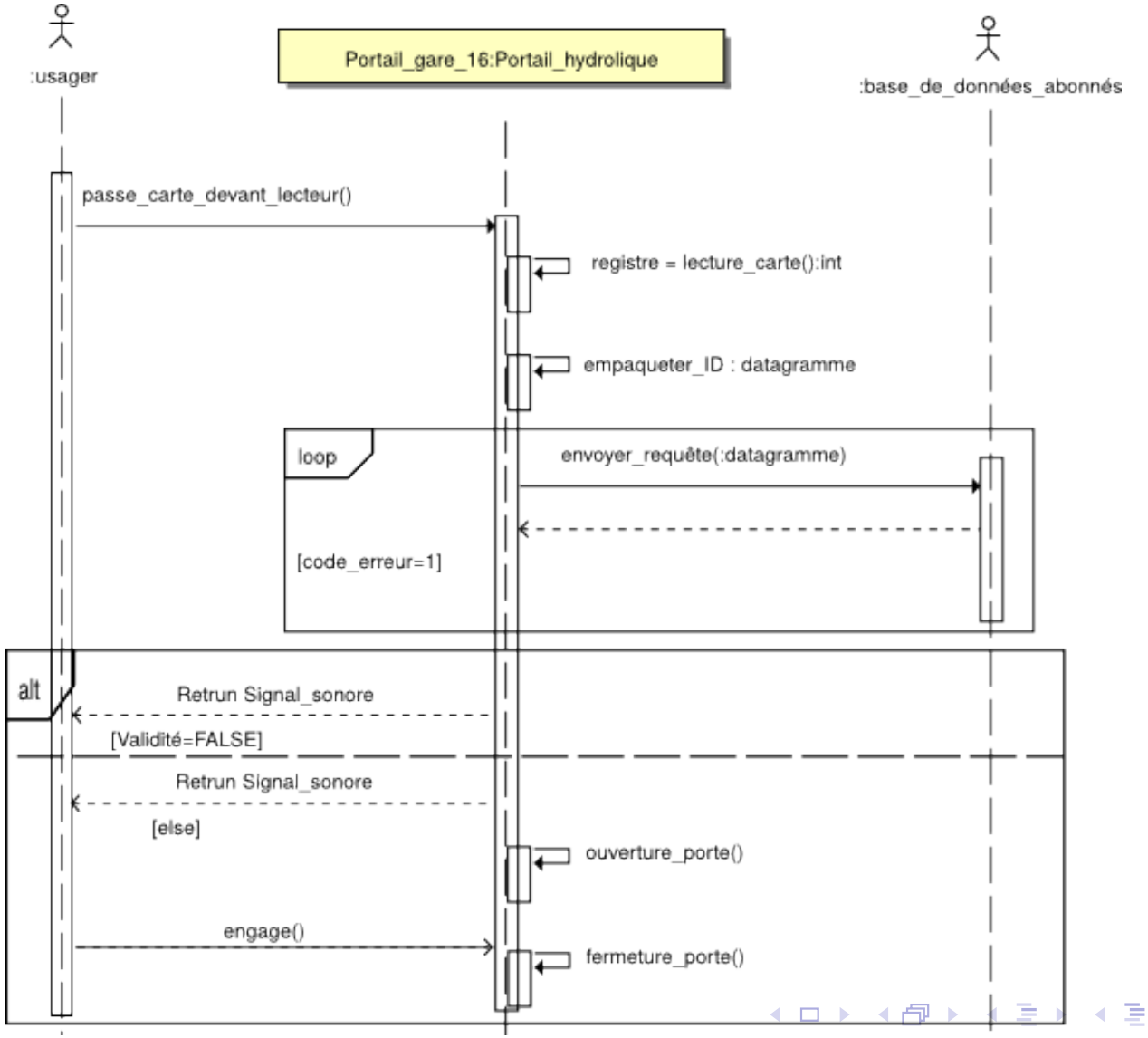


Diagramme de séquence : symboles et syntaxe

- Le plus souvent il s'agit d'un acteur issu d'un diagramme de cas d'utilisation ou d'une instance de classe.

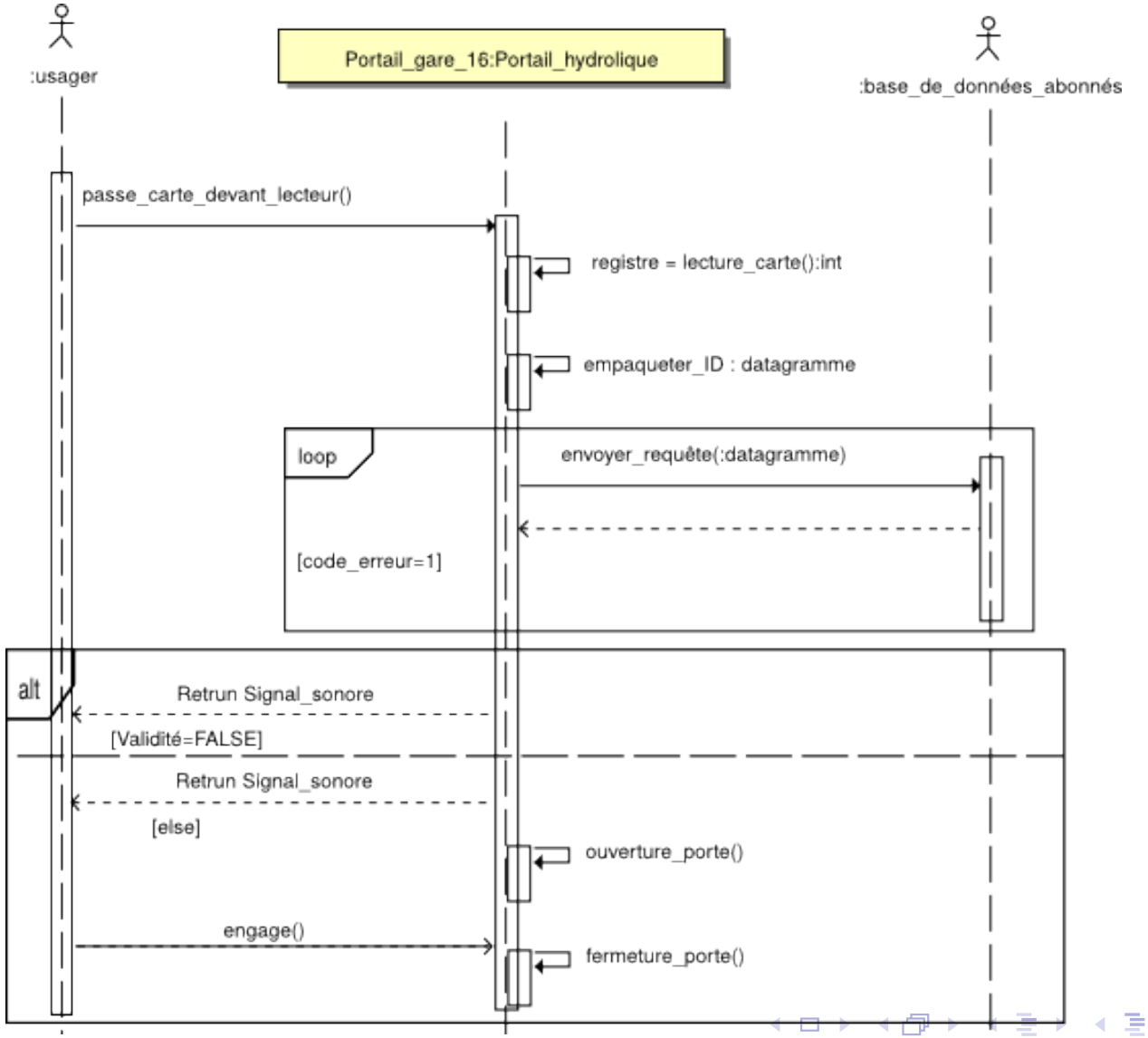


Diagramme de séquence : symboles et syntaxe

- Chaque partie possède une ligne de vie verticale et le temps s'écoule de haut en bas.

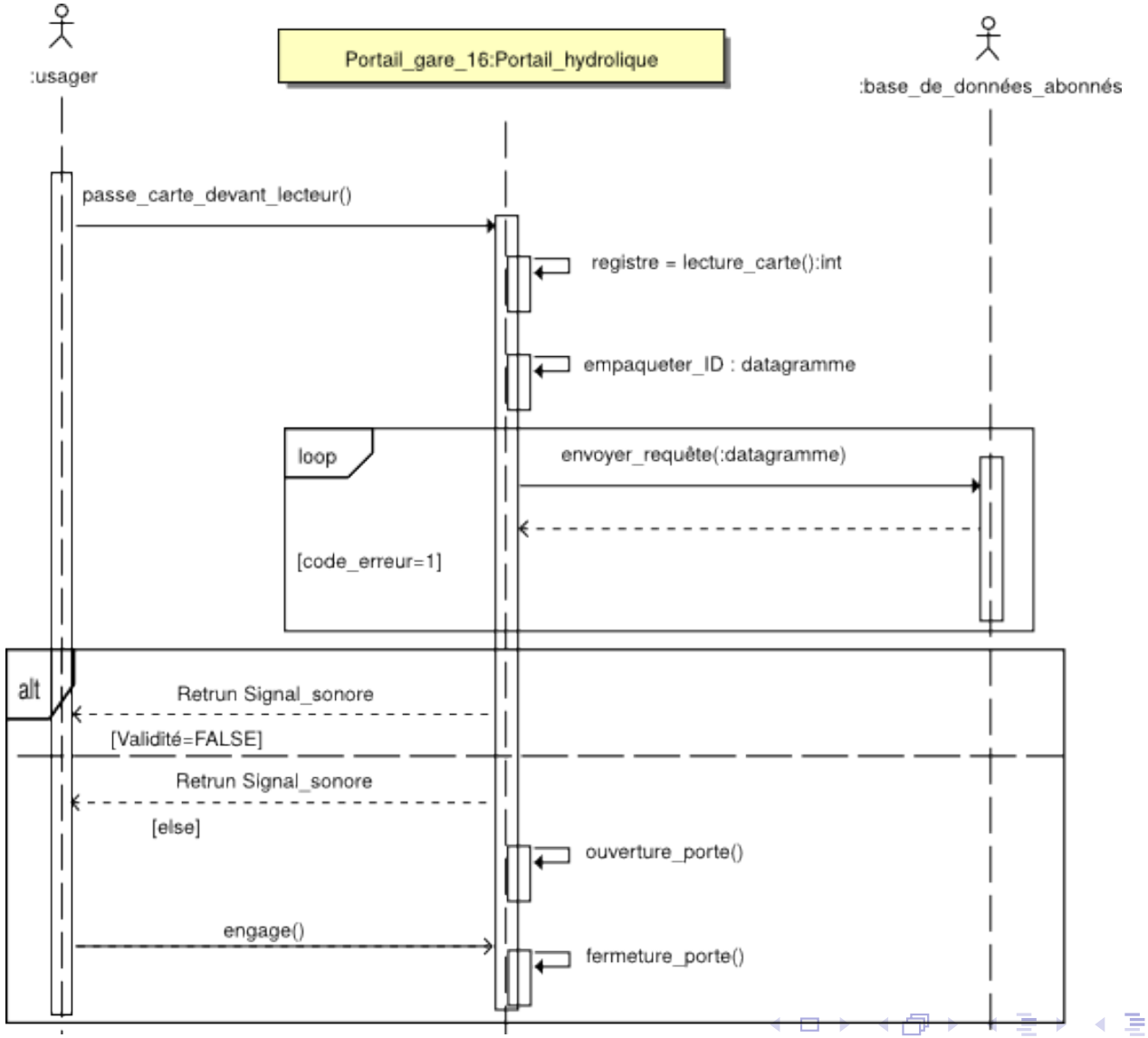


Diagramme de séquence : symboles et syntaxe

- Quand la ligne de vie devient rectangulaire, la partie est active.

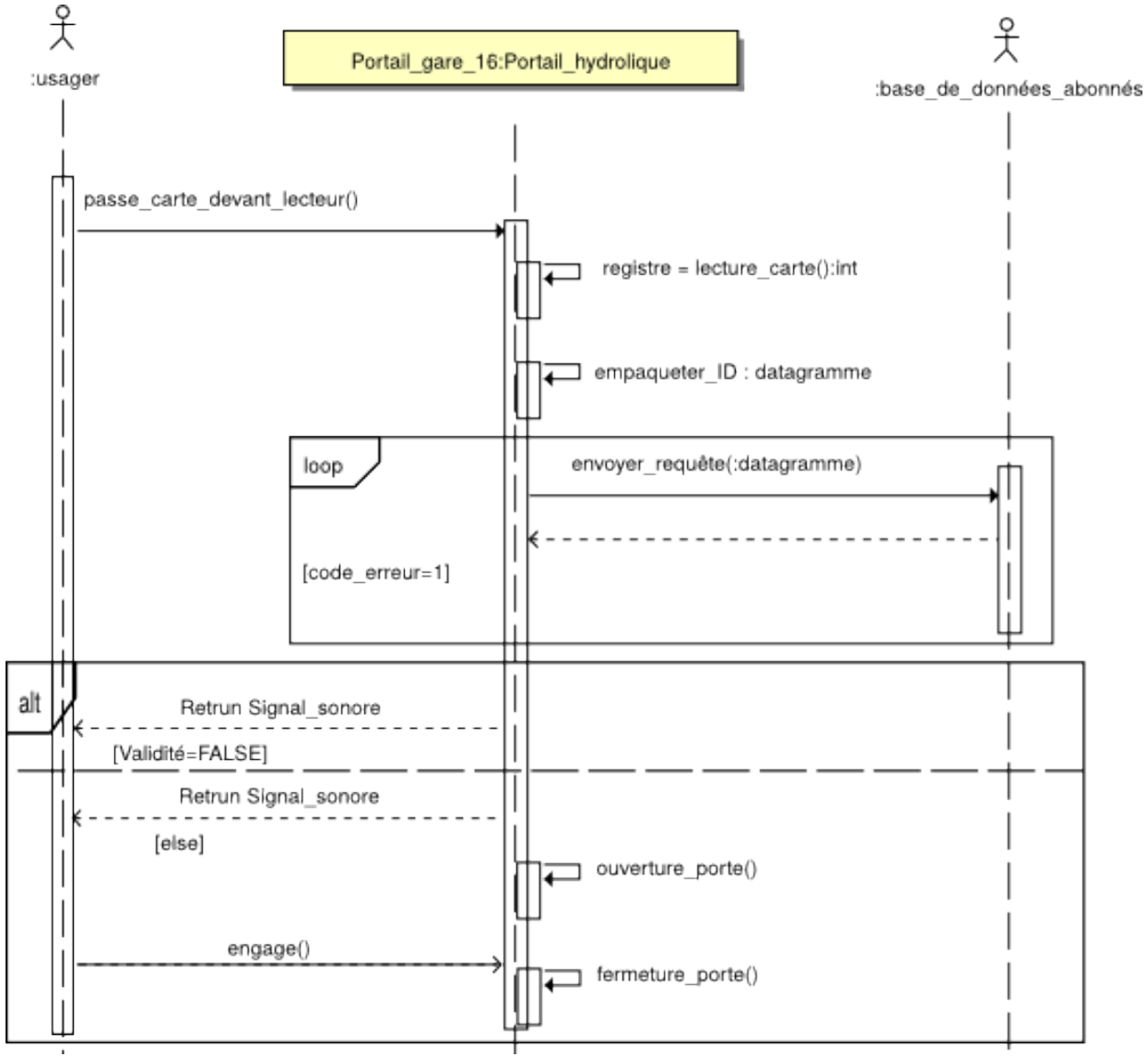


Diagramme de séquence : symboles et syntaxe

- La liaison la plus fréquemment utilisée entre parties est le signal ou message.

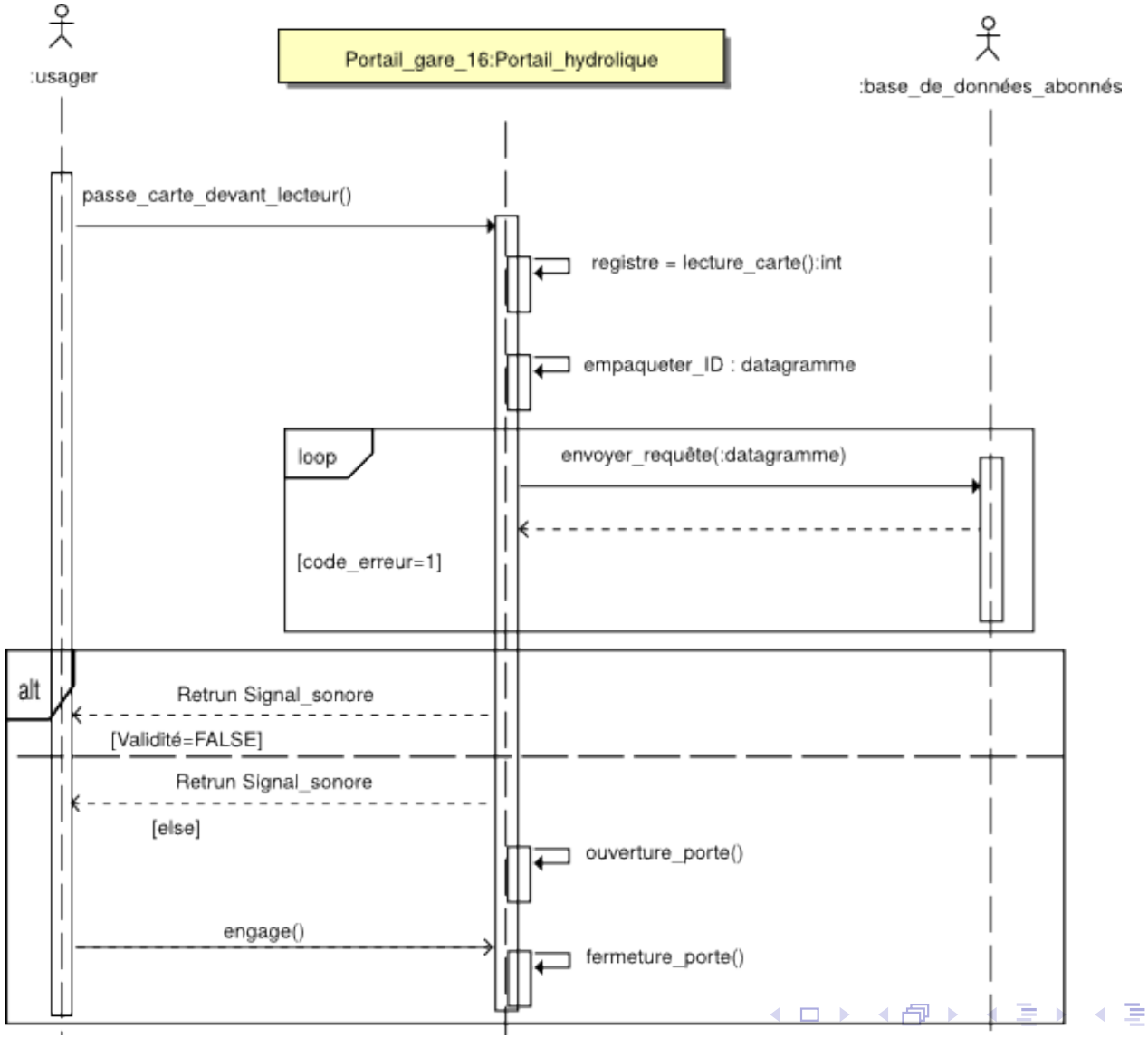


Diagramme de séquence : symboles et syntaxe

- Un message est de nature événementiel mais il peut convoyer des informations nécessaires au traitement.

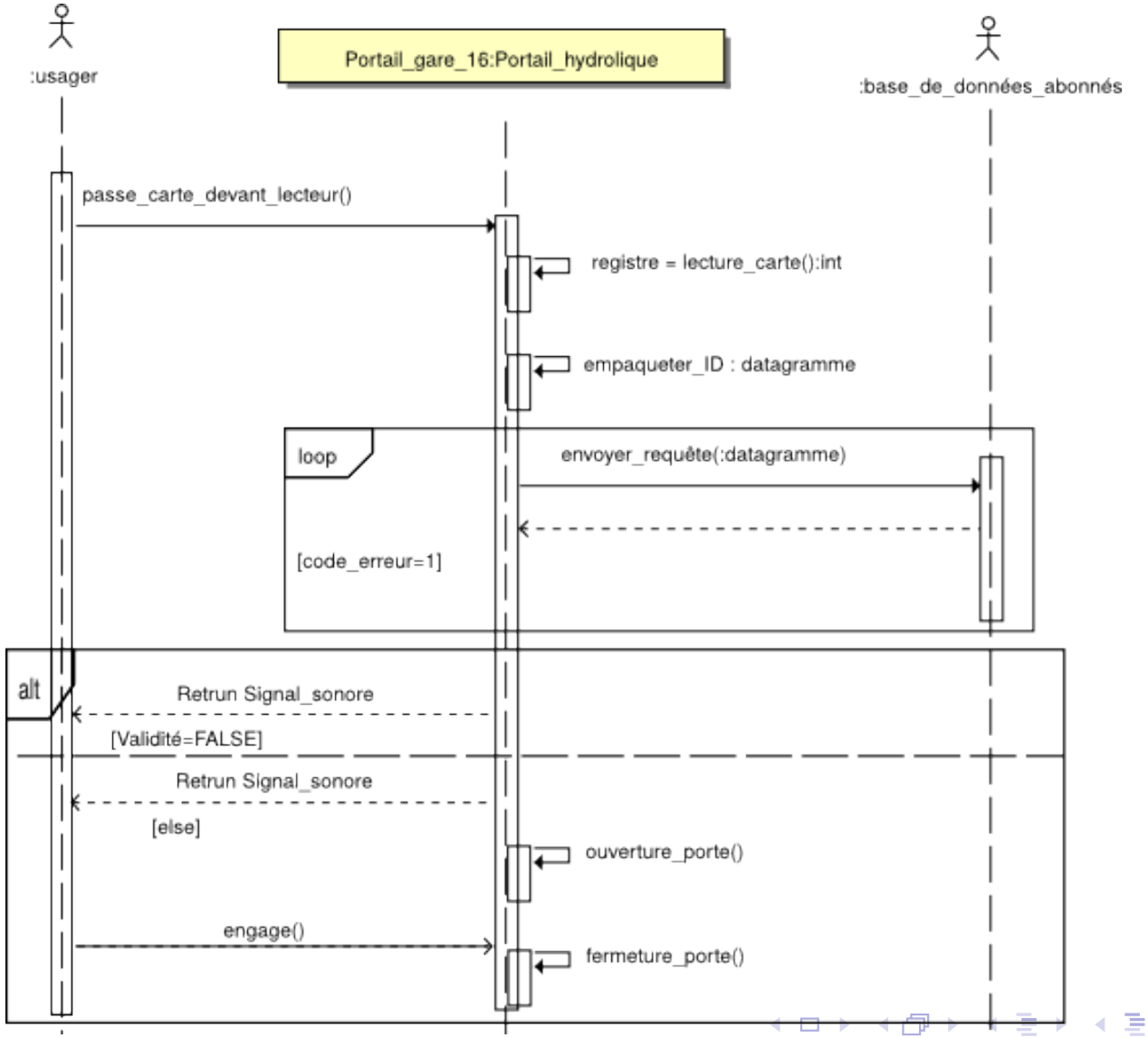


Diagramme de séquence : symboles et syntaxe

- Quand un message est synchrone la partie émettrice est bloquée jusqu'à réception du message de retour, qui précise qu'un résultat est atteint.

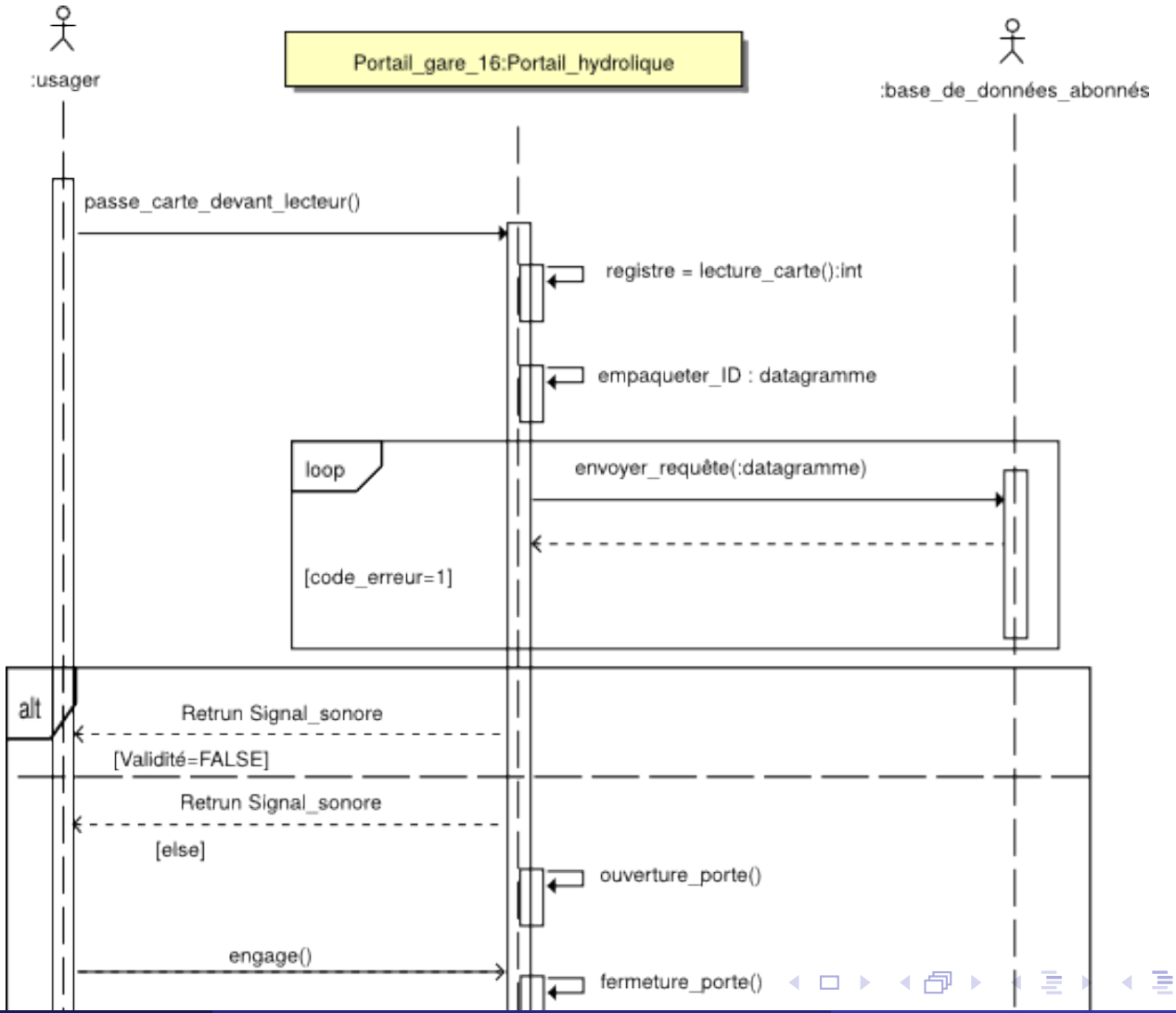


Diagramme de séquence : symboles et syntaxe

- Quand un message est asynchrone la partie émettrice continue son exécution sans attendre d'acquittement.

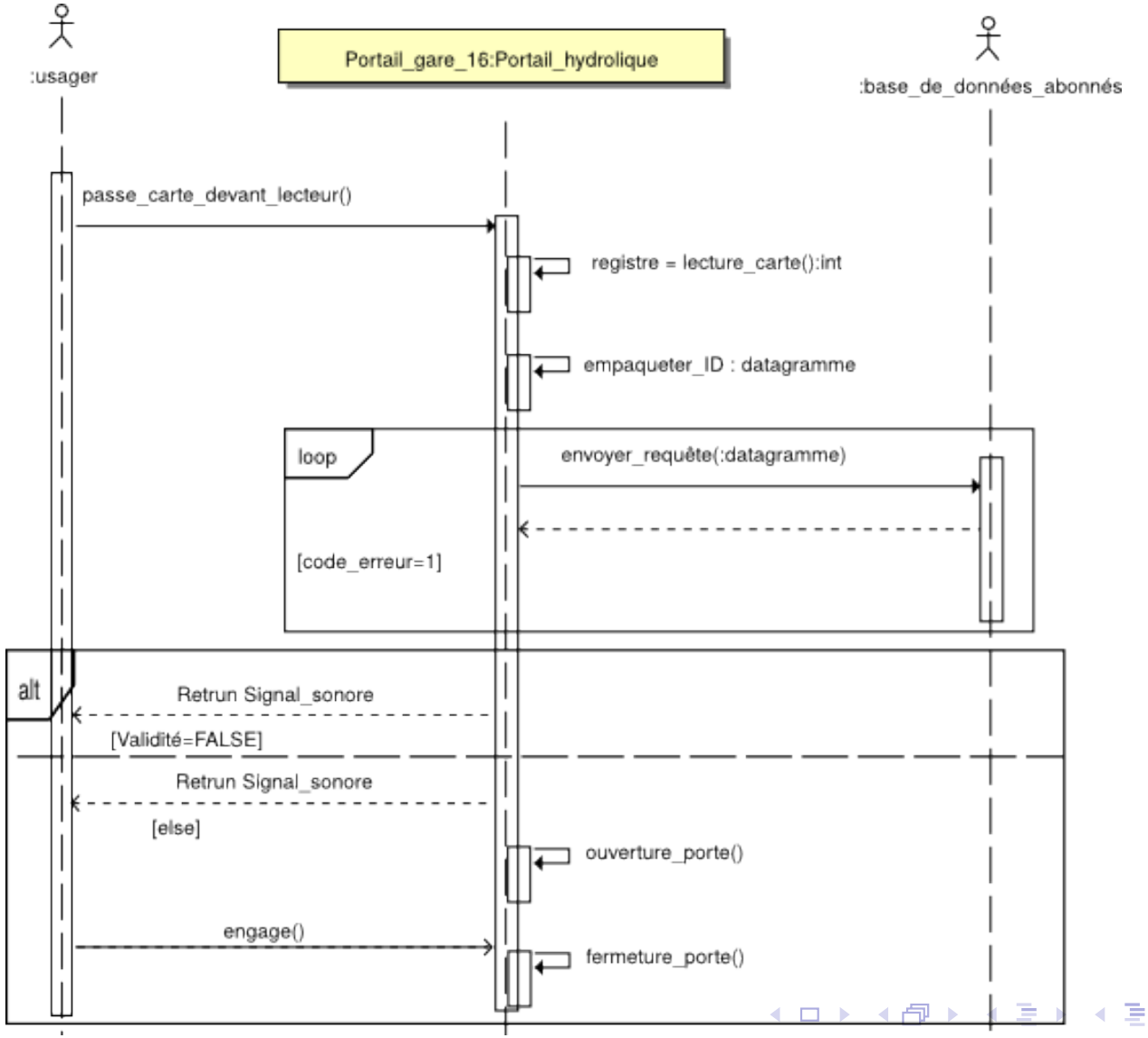


Diagramme de séquence : symboles et syntaxe

- Quand une partie doit exécuter une fonction interne on utilise un message réfléxif.

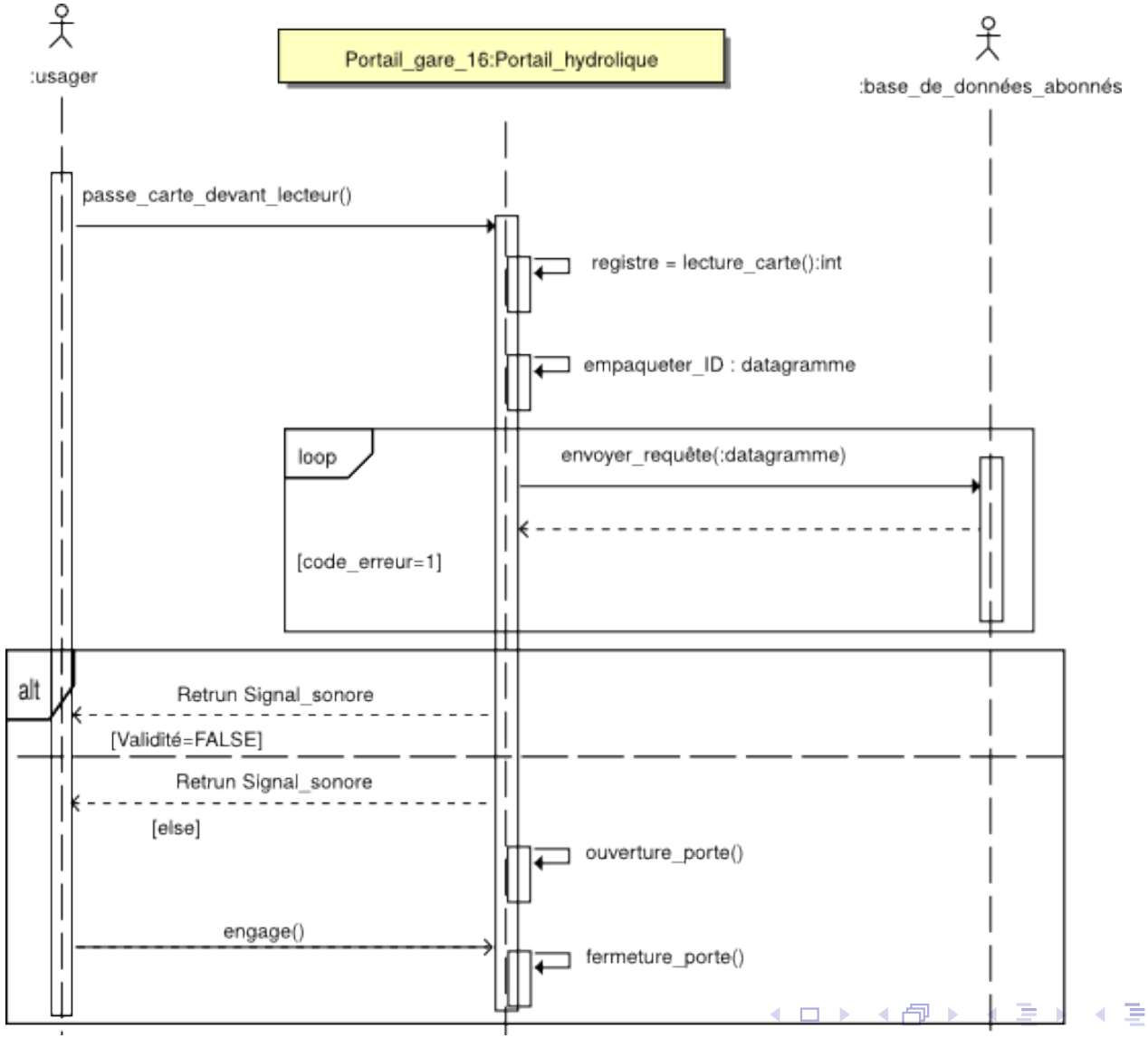


Diagramme de séquence : symboles et syntaxe

- On peut spécifier qu'une partie de la séquence est à répéter en boucle à l'aide d'un opérateur de fragment « loop ».

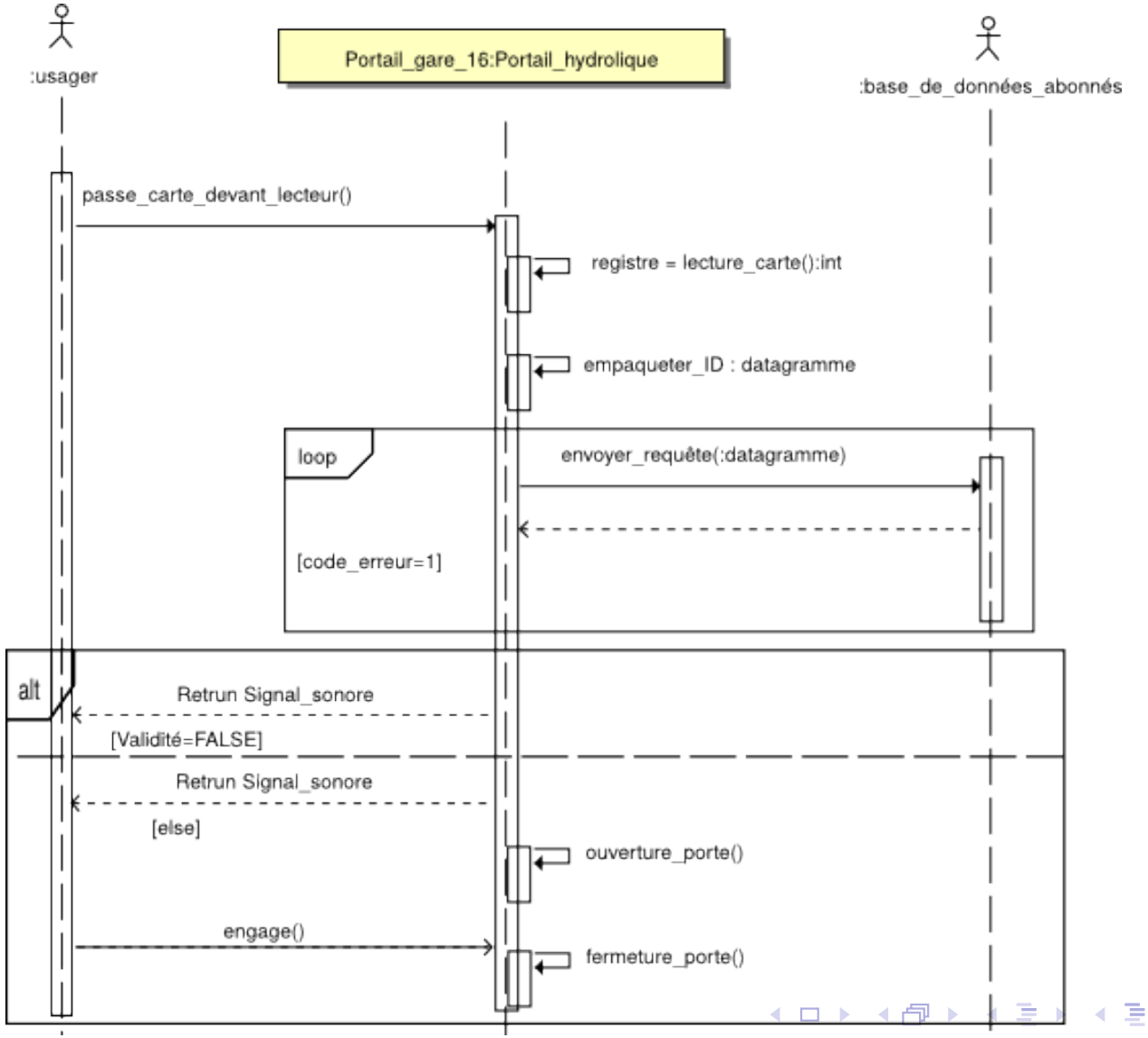


Diagramme de séquence : symboles et syntaxe

- De même, on spécifie qu'une partie de la séquence est optionelle à l'aide de l'opérateur de fragment « alt » ou « opt ».

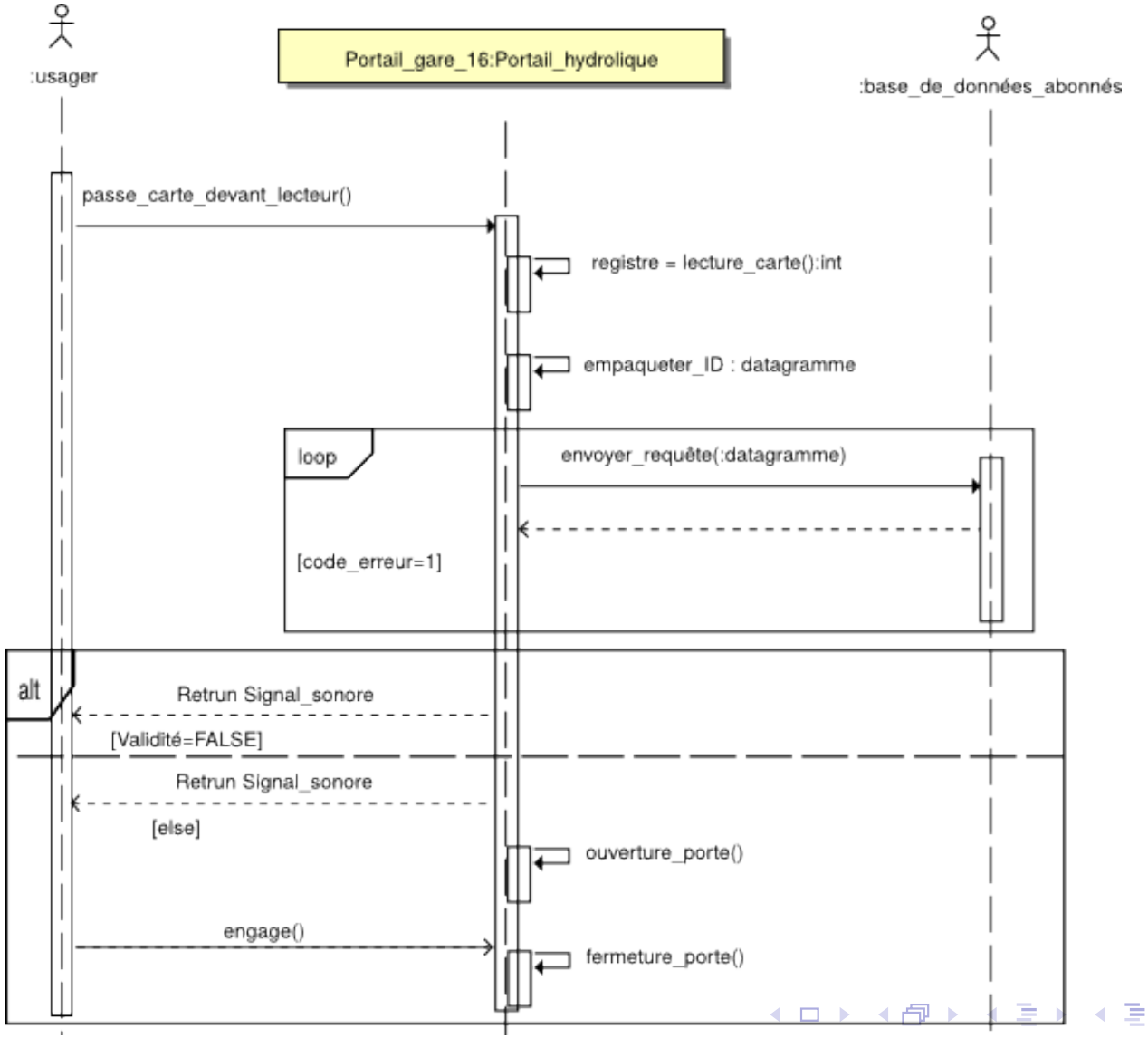


Diagramme de séquence : éléments complémentaires

- Il existe d'autres types de signaux comme ceux début et de fin de vie.

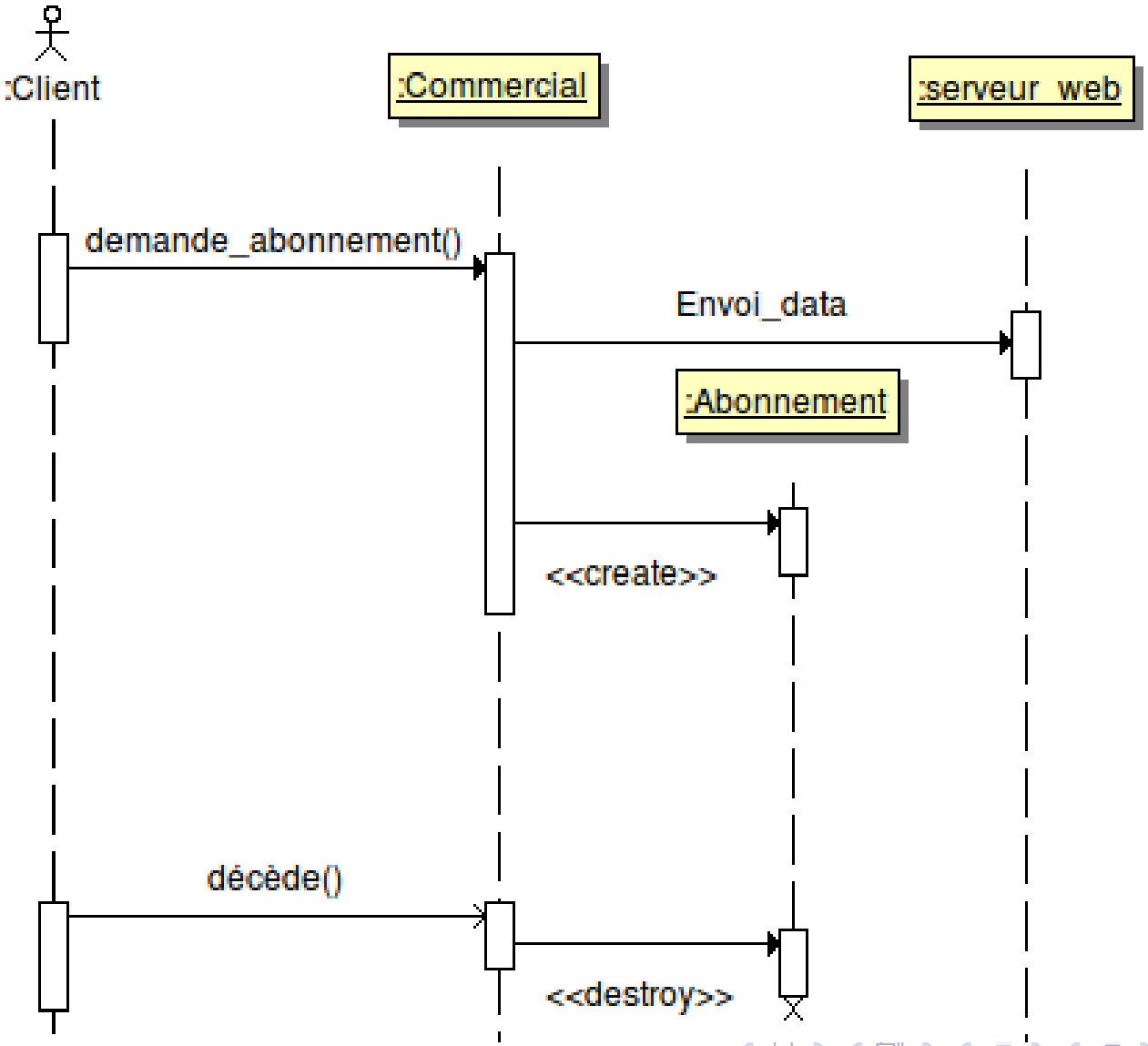


Diagramme de séquence : éléments complémentaires

- L'instance résultant d'une création ne se trouve pas au même niveau que celles présentes dès le départ.

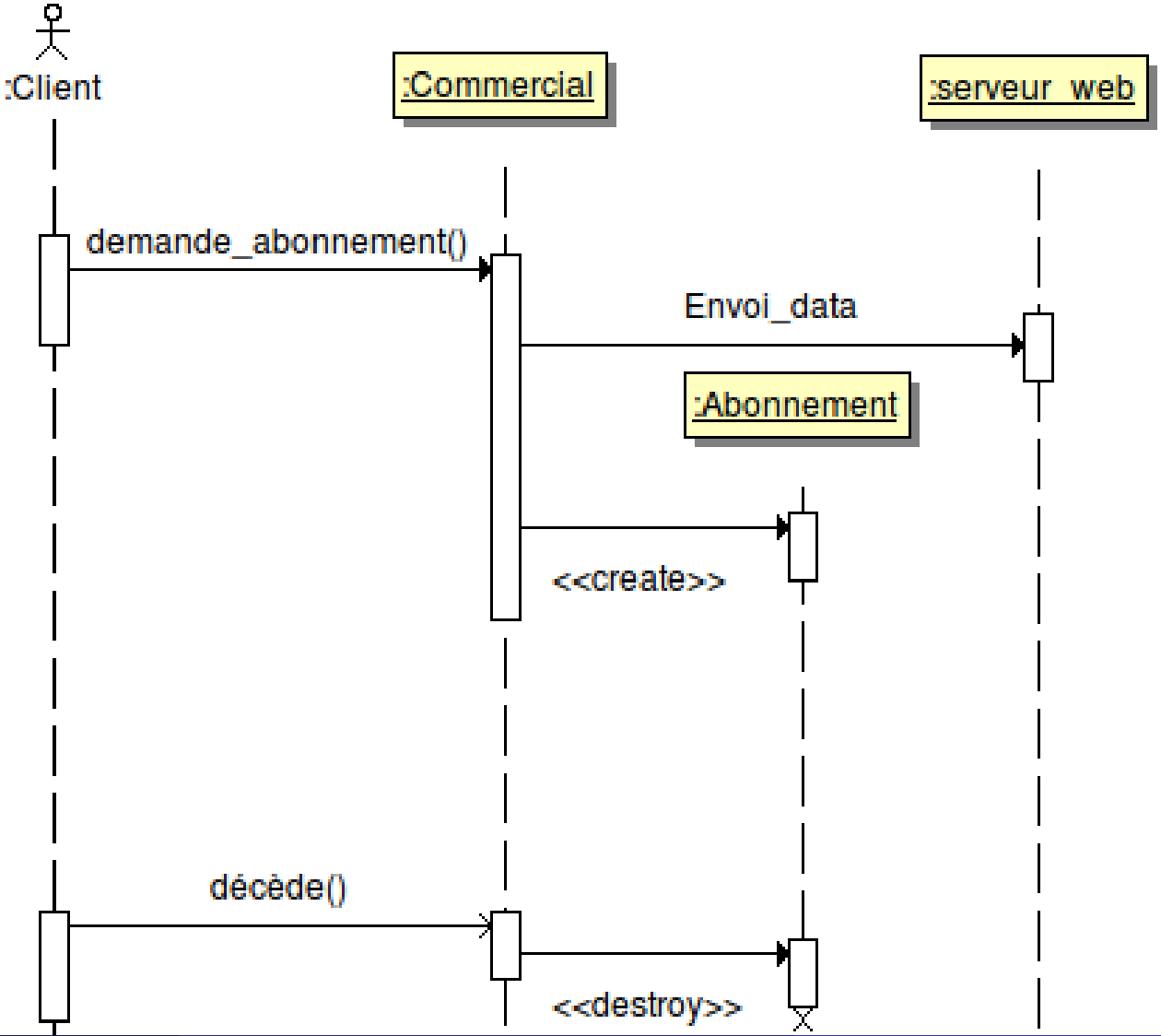


Diagramme de communication : Généralités

- Le diagramme de communication reprend les mêmes informations que le diagramme de séquence mais avec une autre perspective.
- Le diagramme de séquence a un défaut : l'interaction entre parties est confinée à une seule dimension (sens horizontal).
- Quand les interactions sont nombreuses, il devient nécessaire de les réprésenter en 2D en n'utilisant plus l'axe vertical pour le temps. C'est ce que propose le diagramme de communication.
- Il faut néanmoins alors numéroter les messages pour que la séquence soit lisible.

Diagramme de communication : Exemple

Equivalent du deuxième diagramme de séquence

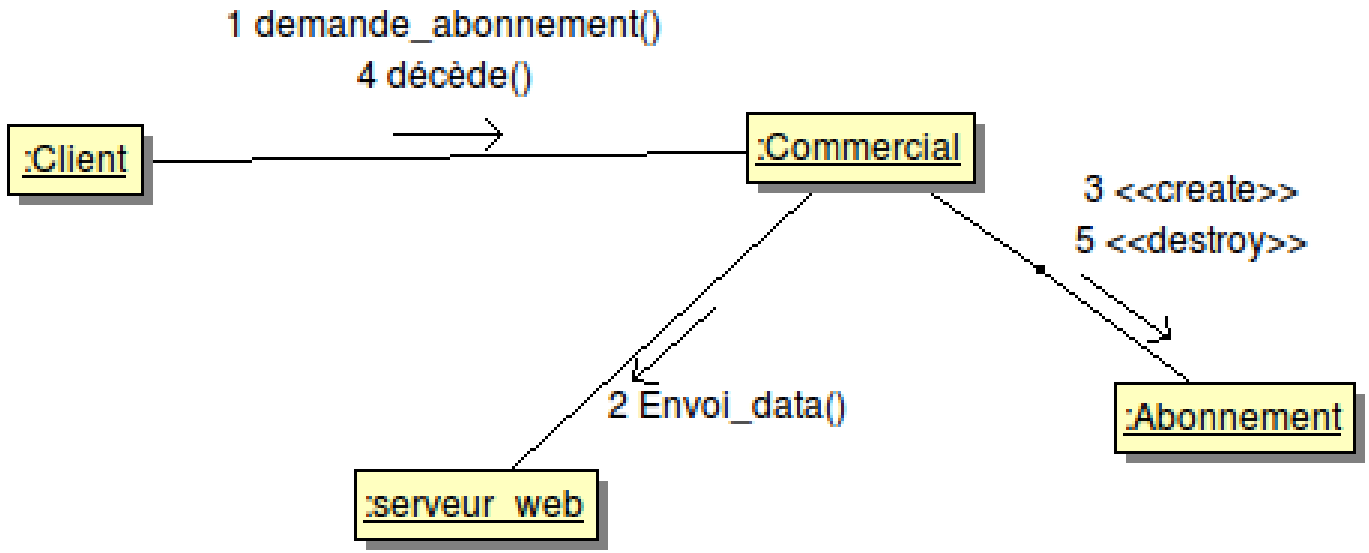


Diagramme de machine à états : Généralités

- Quand on se focalise sur une partie intervenant dans un diagramme de séquence, on peut vouloir détailler de quel état à quel état elle passe.
- Le diagramme de machine à états permet de faire cela en utilisant un dictionnaire et une syntaxe proche des diagrammes d'activité.
- Attention ! La logique est inversée par rapport à un diagramme d'activité : pour passer d'un état à l'autre, on réalise souvent une action ! Autrement dit, ici les actions deviennent des transitions.

Diagramme de machine à états : symboles et syntaxe

- Tous les symboles dans ce diagramme sont des états ou pseudo-états de l'objet.
- Le dictionnaire de symboles est le même que pour un diagramme d'activité mais le sens des symboles est orienté « état ».

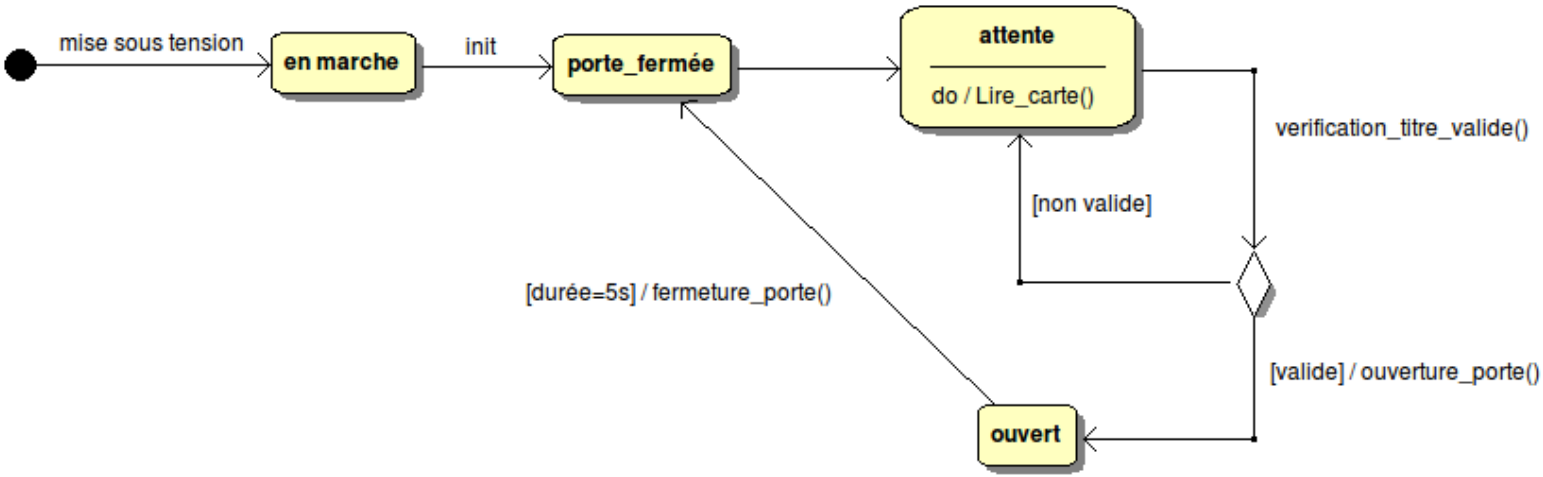


Diagramme de machine à états : symboles et syntaxe

- Les transitions correspondent à des actions mais ces actions peuvent être conditionnées.

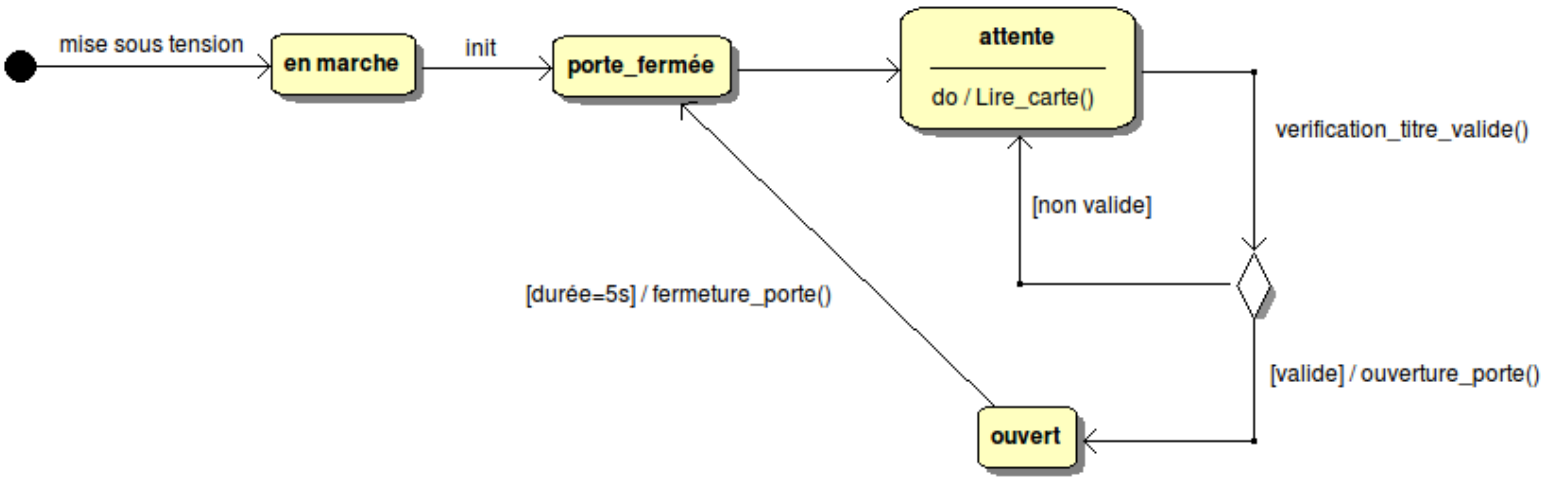


Diagramme de machine à états : symboles et syntaxe

- On peut trouver des actions voire des transitions inhérentes à un état, comme pour l'état ≪ attente ≫.

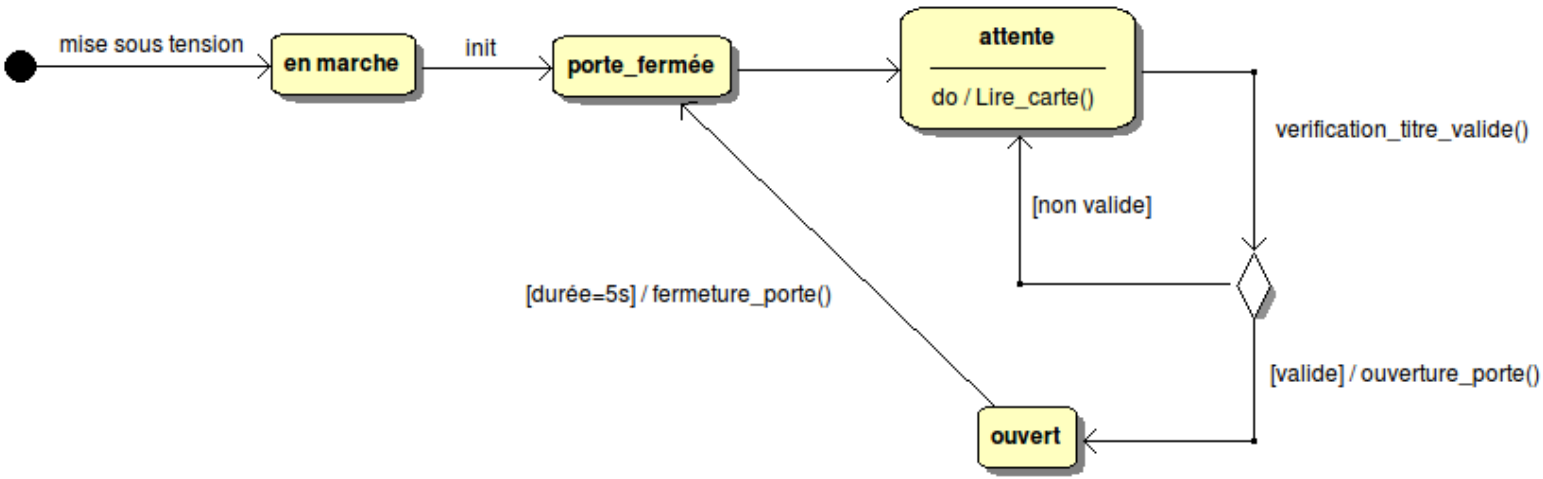


Diagramme de machine à états : symboles et syntaxe

- Pour les mêmes raisons qu'évoquées dans la présentation du diagramme d'activités, ce diagramme ne possède pas une rigueur équivalente au Grafcet.

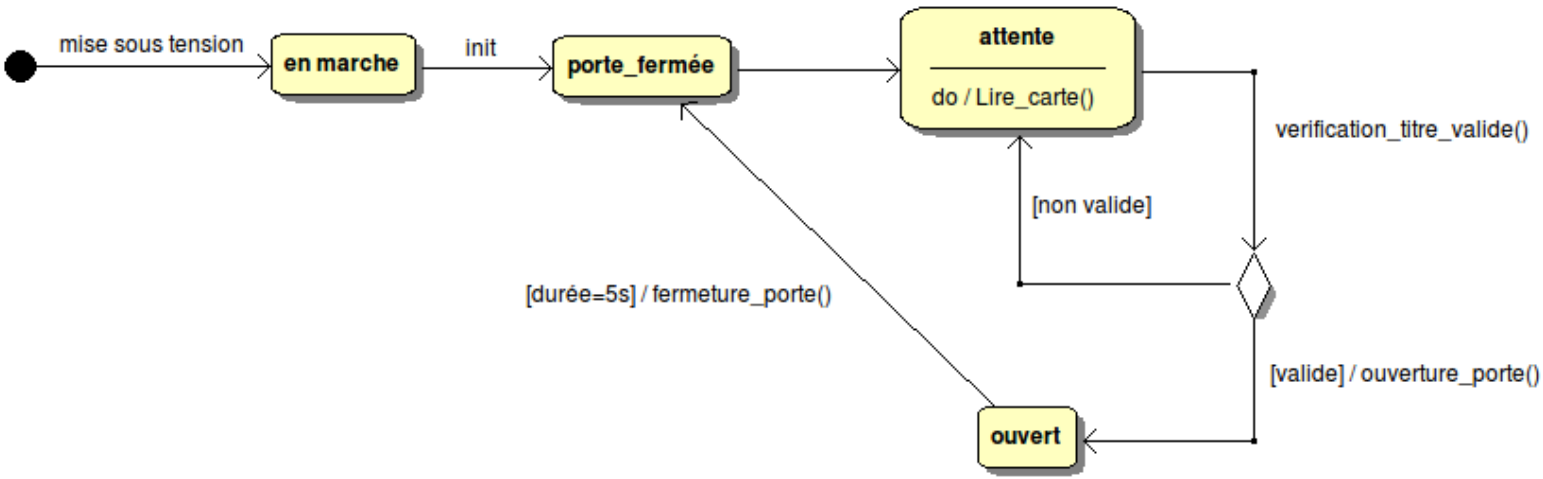


Diagramme de déploiement : Généralités

- Ce diagramme est surtout utile en fin de projet car il permet de préparer la phase de réalisation.
- Néanmoins, il n'est pas forcément à réaliser en dernier. En effet, vous pouvez faire une esquisse grossière au départ et l'affiner au fur et à mesure.
- En UML, il est important de faire évoluer tous les diagrammes conjointement pour prendre en compte tous les aspects.
- Le but du diagramme de déploiement est de lister les ressources matérielles nécessaires à la réalisation physique du projet.
- Il précise comment ces ressources s'articulent entre elles.
- Il précise également quels composants s'intègrent à quelle ressource.

Diagramme de déploiement : symbole

- Le diagramme de déploiement contient un seul symbole : le noeud.
- Le noeud est une ressource matérielle sur laquelle est déployée un composant.

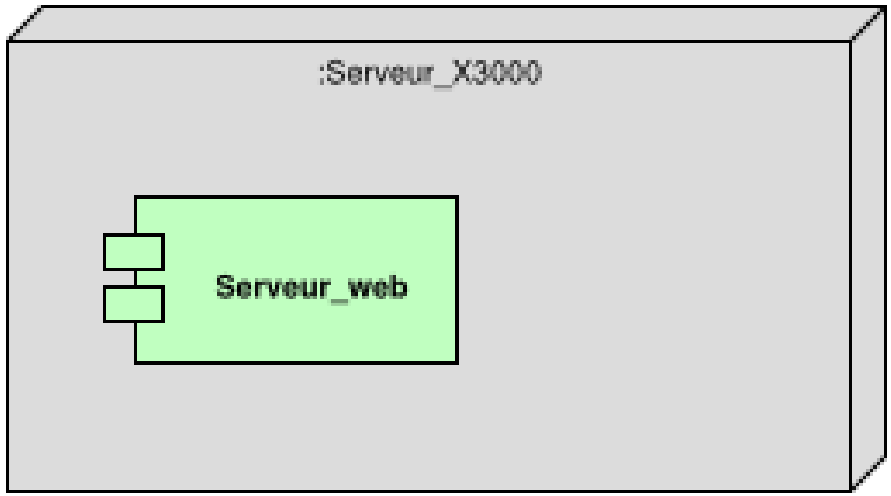


Diagramme de déploiement : syntaxe

- Une première syntaxe possible entre noeuds est l'encapsulation, mais il convient de préciser leurs natures via des stéreotypes pour plus de clarté.

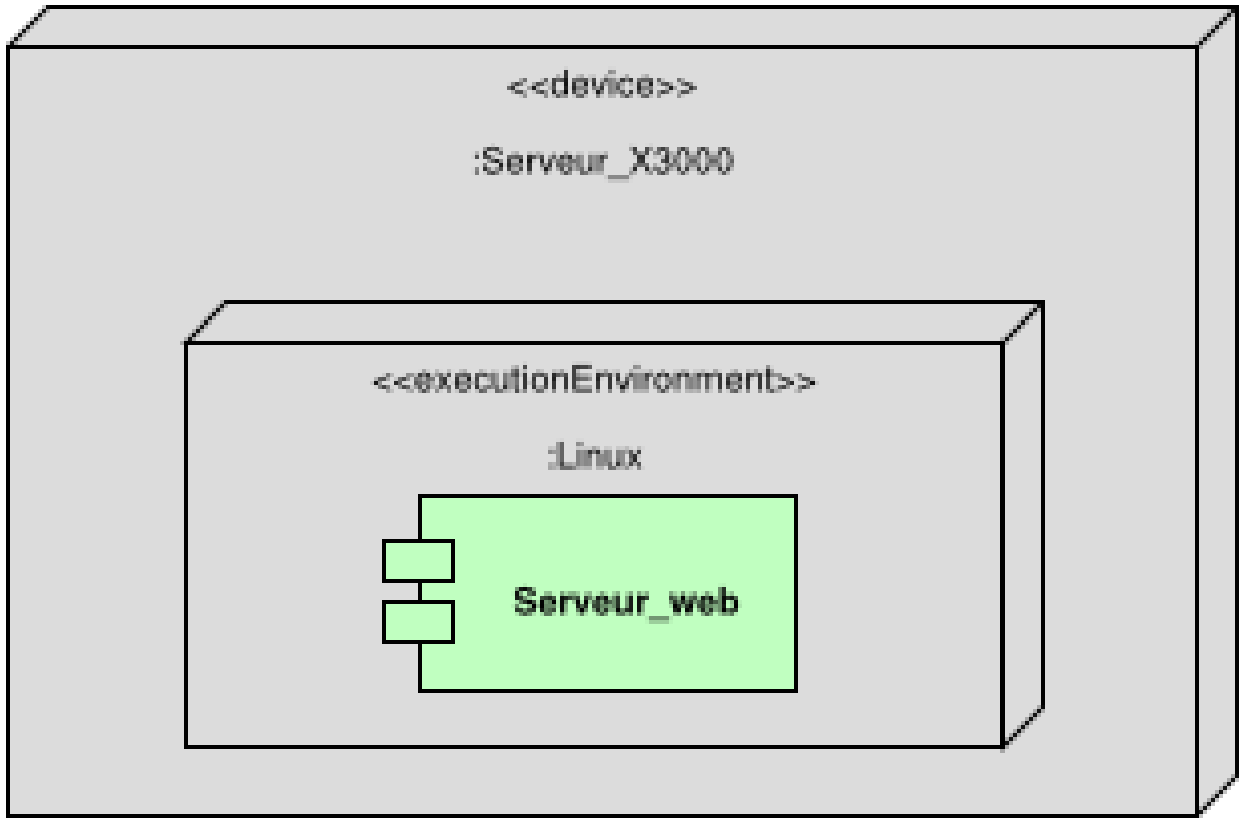


Diagramme de déploiement : syntaxe

- Quand on relie deux noeuds, c'est à dire deux ressources matérielles, il est normal que la liaison soit elle aussi physique. On la nomme en UML chemin de communication.
- Pour préciser sa nature, si le nom ne suffit pas, on peut la stéreotyper.

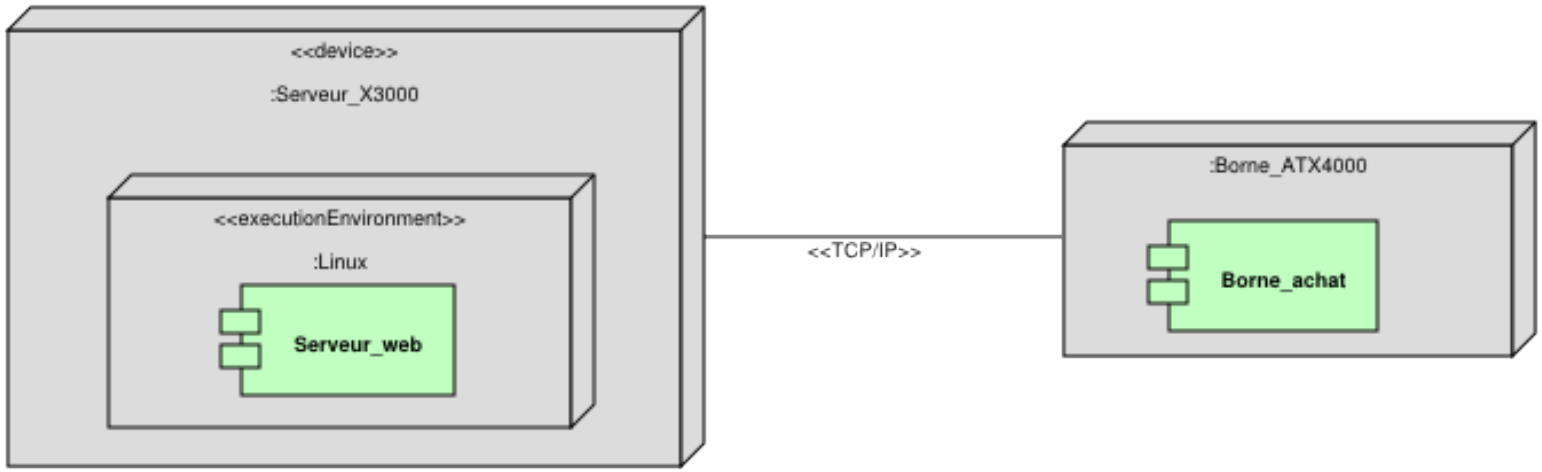
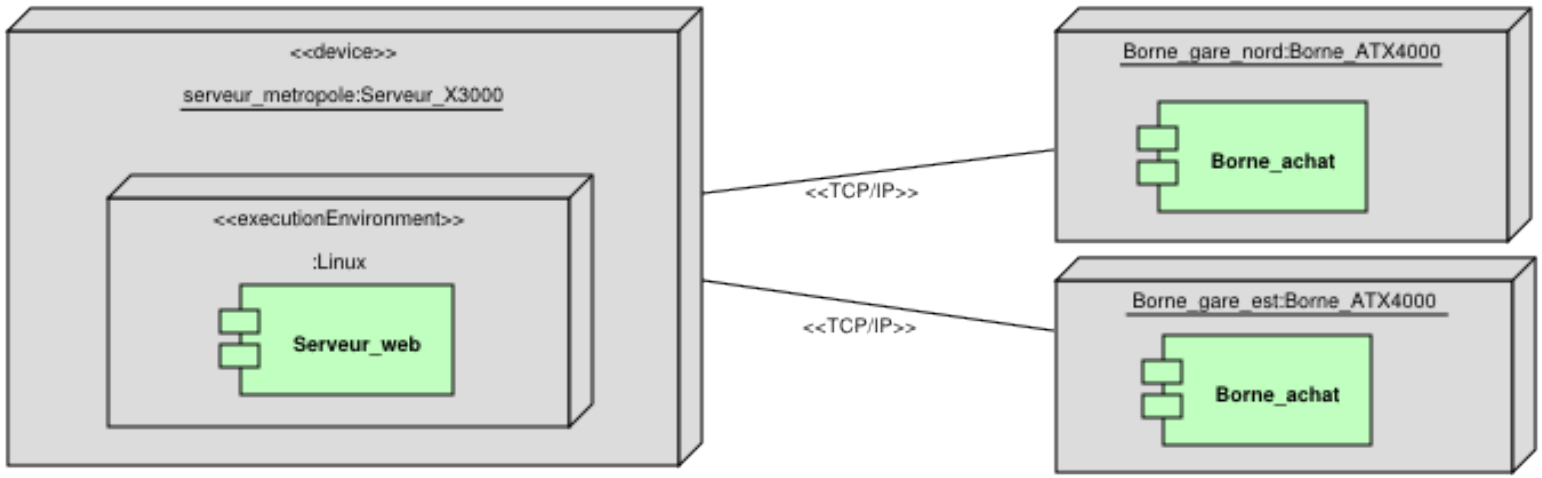


Diagramme de déploiement : éléments supplémentaires

- Parfois, les noeuds sont représentés avec des artéfacts au lieu de composants en leur sein. Un artéfact peut être vu comme la matérialisation d'un composant. Par exemple pour un composant logiciel, l'artéfact est le fichier exécutable.
- Il est possible d'instancier des noeuds pour décrire le déploiement du système dans une situation particulière :



Diagrammes non étudiés :

- Le diagramme de chronométrage (vue dynamique) : il vient en appui d'un diagramme de séquence et de machine à état. L'évolution de l'état de chaque partie y est représentée à la manière d'un chronogramme. Le temps nécessaire aux transitions y est indiqué (intéressant pour temps réel).
- Le diagramme global d'interaction (vue structurelle) : grosso modo, c'est un diagramme d'activité dans lequel les actions sont remplacées par des diagrammes de séquence. C'est un diagramme de haut niveau destiné à regrouper des interactions qui décrivent une certaine séquence. Chaque interaction est un diagramme de séquence.

notions non étudiés :

- Les structures composites (vue structurelle) : Elles permettent de donner plus de détails sur les éléments qui composent une classe. Dans quelques cas particuliers, les relations d'agrégation ou de composition laissent en effet une ambiguïté.
- Le paquetage (vue structurelle) : un paquetage ressemble à un composant : c'est un élément de niveau élevé dans la hiérarchie et il contient plusieurs classes (à la façon d'une poupée russe). À la différence d'un composant, il n'est pas pensé pour être interchangeable mais pour structurer des classes ayant un objectif, une fonctionalité commune. Il est possible de définir ensuite des diagrammes de paquetage pour spécifier les dépendances entre paquetages et également pour rappeler dans quels cas d'utilisation ils interviennent. C'est un diagramme de très haut niveau qui peut à lui seul résumer une grande partie du système.

éléments bibliographiques :

- théorie : Introduction à UML 2, Russ Miles & Kim Hamilton, O'Reilly 2006.
- pratique : UML 2 Entraînez-vous à la modélisation, Laurent Debrauwer & Naouel Karam, ENI collection *les tp informatiques* 2009.