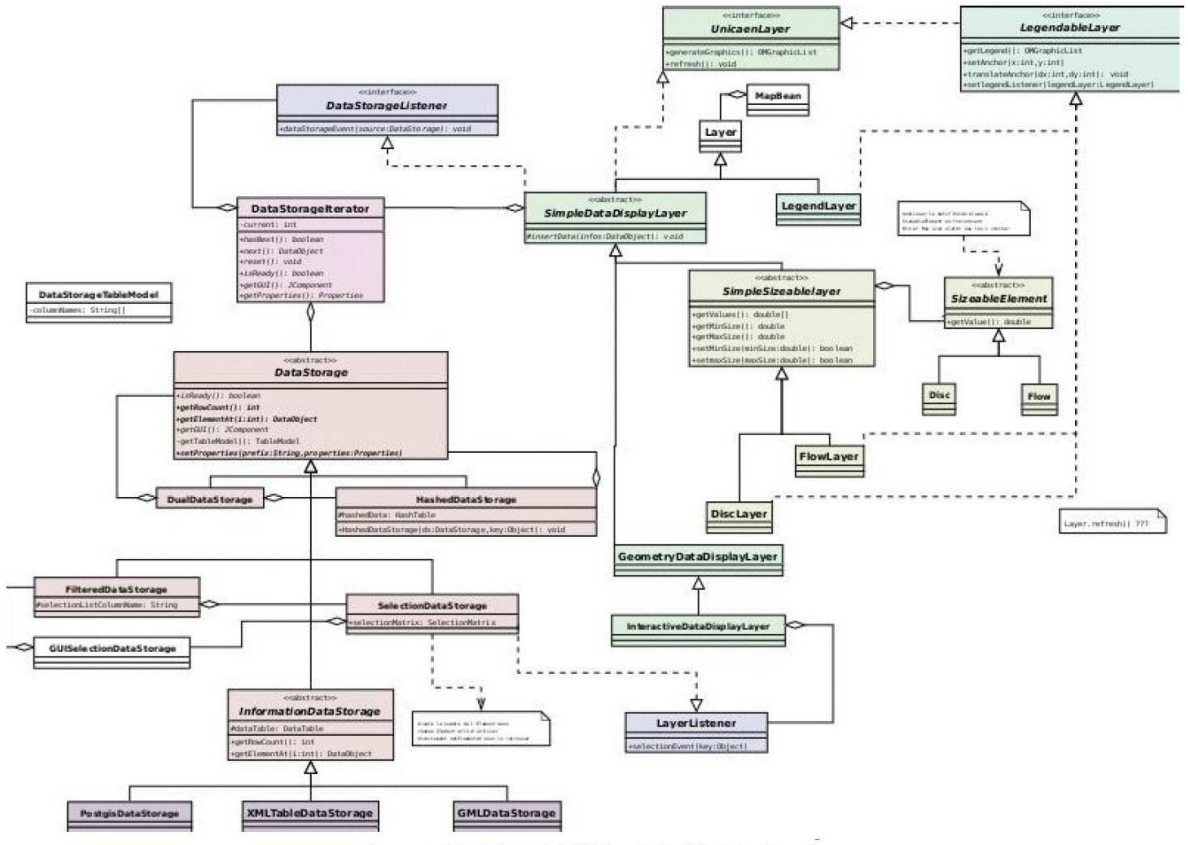


Langage SYSML et UML



Sommaire

Analyse fonctionnelle système Page N°3

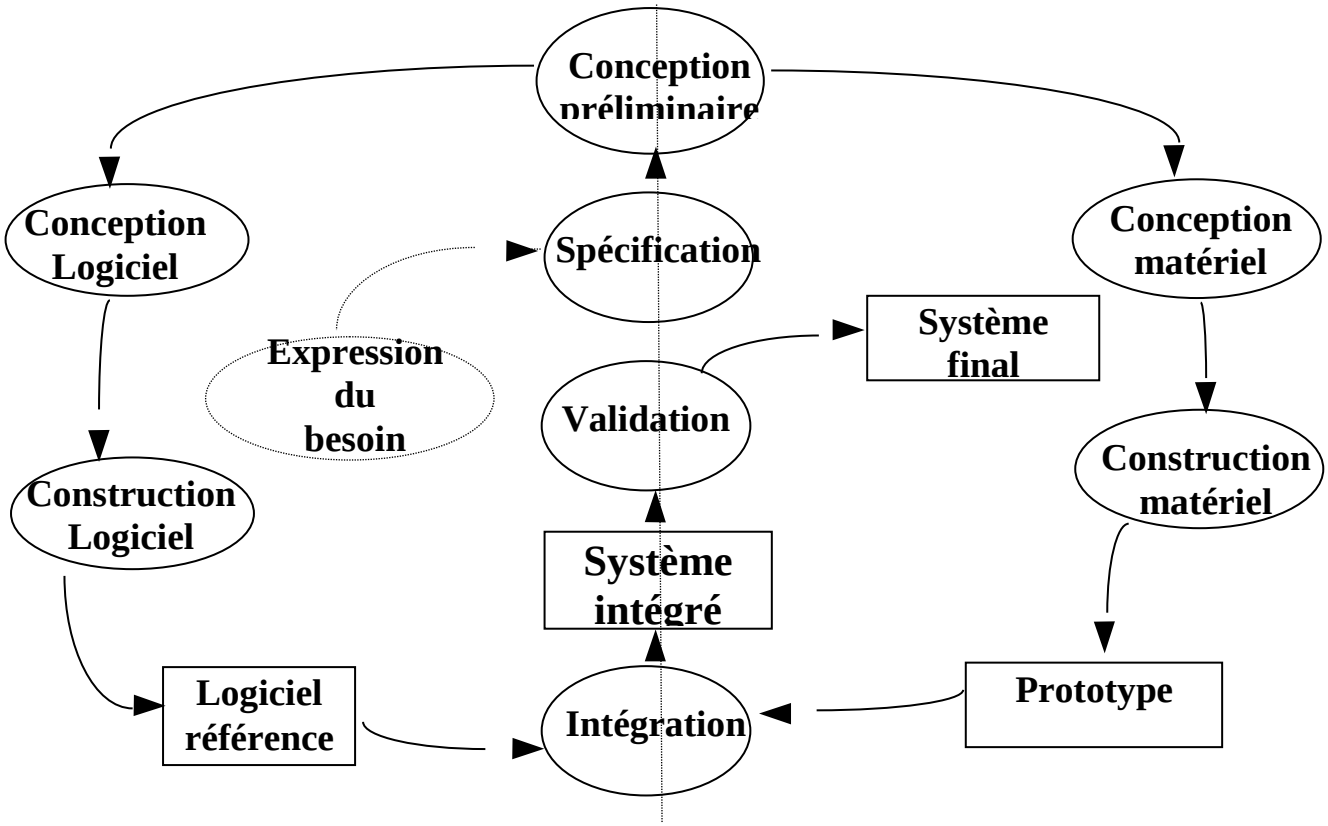
1) Présentation du langage SysML et UML Page N°6
2) Diagramme de cas d'utilisation Page N°9
3) Diagramme de séquence ou d'interactions Page N°13
4) Diagramme d'exigences Page N°20
5) Diagramme de définition de blocs Page N°24
6) Diagramme de bloc interne Page N°29
7) Diagramme d'états transitions Page N°32
8) Diagramme de classe Page N°35
9) Diagramme de déploiement Page N°40
10) Une démarche SysML Page N°43

La conduite de projet
Méthode et spécification

1) Introduction

Le but de ce polycopié est d'exposer des principes et des méthodes de spécification et de conception globale des systèmes électroniques.

2) Développement d'un système électronique



La spécification d'un système est l'expression technique du besoin auquel il doit répondre, c'est une traduction de celui-ci en une représentation, qui permet de décrire le comportement et l'environnement du système, sans faire appel à des considérations informatiques ou électronique. La spécification d'un système a pour but de répondre à la question **"que faire ?"**.

La conception consiste à proposer une architecture informatique ou électronique globale du système et une façon de procéder pour la construire. La conception a pour but de répondre à la question **"comment faire ?"**.

La construction du système se fait suivant les procédés définis dans la conception détaillée.

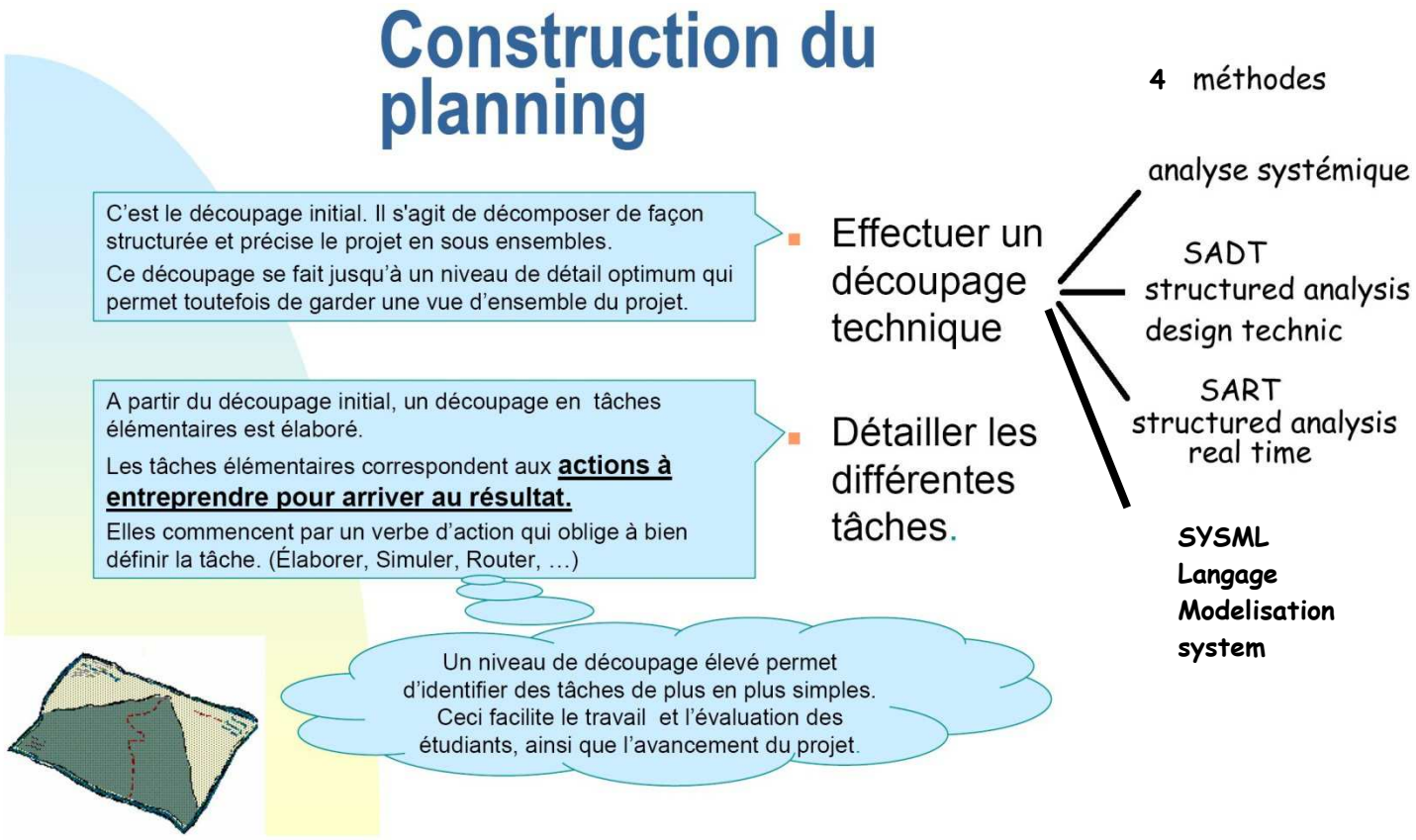
- Pour le logiciel: Codage, test unitaire, intégration du logiciel
- Pour le matériel: Fabrication du prototype, tests et mesures.

A l'issue de la construction, un logiciel de référence et un prototype matériel du système sont produits.

L'intégration est la phase de mise en oeuvre du logiciel de référence et du prototype du matériel avec le processus à conduire.

La validation est la phase consistant à établir que le produit est conforme au besoin exprimé par le client.

3) Planification d'un projet



Construction du planning

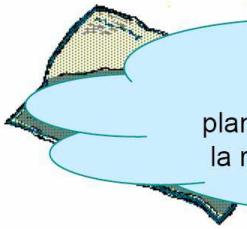
- Définir une logique de déroulement entre les tâches.

Il s'agit notamment de bien identifier les tâches indépendantes des tâches qui ne pourront être menées à bien que lorsque d'autres seront achevées.

Ceci permet d'anticiper des situations de blocage (ex: « liaison radio » mise au point d'un récepteur en l'absence d'un émetteur opérationnel !)

- Estimer la durée des tâches

La durée choisie doit être à la fois réaliste et raisonnable. Elle est basée le plus souvent sur les expériences passées et sur des situations voisines.



Des erreurs sur les estimations de durée sont inévitables, elles ne rendent pas pour autant la planification inopérante dans le pilotage du projet. C'est la réaction par rapport à d'éventuels décalages qui est importante.

Description fonctionnelle par SYSML ou UML

1) Présentation

L'UML est un outil de conception utilisé principalement en informatique pour la modélisation des systèmes informatiques orientés objet. SysMl est un dérivé du langage UML mais pour la modélisation et la description des systèmes techniques pluritechniques.

- UML : Unified Modeling Language

(Langage de modélisation unifié)

- SysML : Systems Modeling Language

(Langage de modélisation de systèmes)

L'UML ou le SysML sont "souples", on peut les utiliser pour décrire tout ou partie d'un système, d'un point de vue structurel (statique), d'interactions ou comportemental.
Les diagrammes SysML, le plus souvent, sont liés entre eux (interconnectés) et ont leurs descriptions propres. Ils peuvent remplacer la plupart des autres outils de description auparavant utilisés (Grafcet, Fast, SADT, etc).

Ces diagrammes comportent trois aspects :

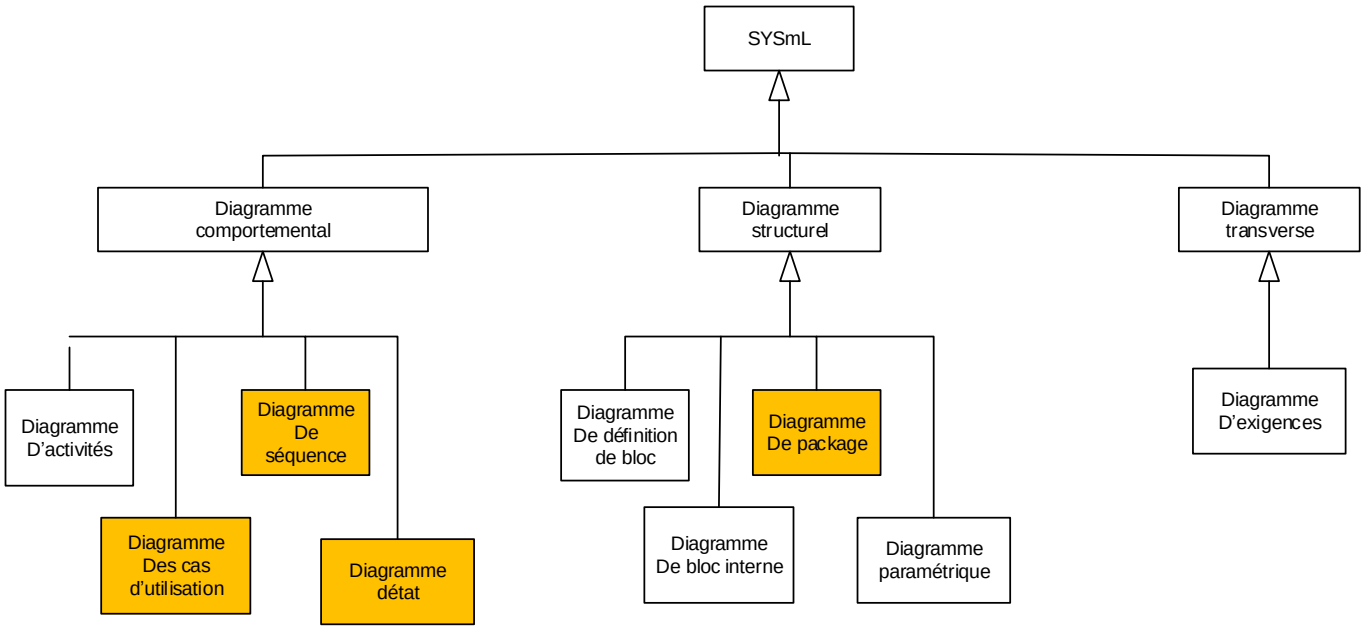
- Aspects comportementaux

Diagrammes fonctionnels (que doit faire le système ?) et diagrammes dynamiques (comment le système doit-il se comporter ?) :

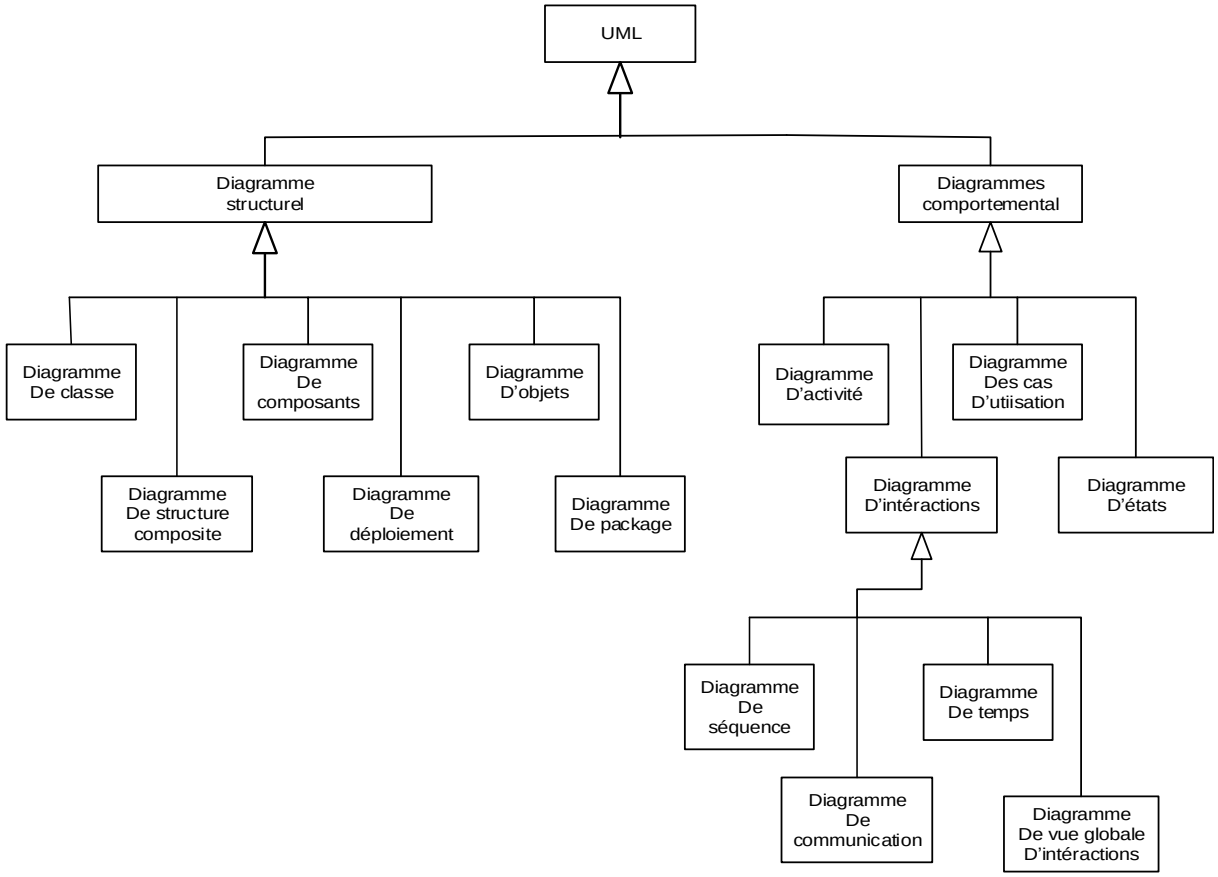
- Aspects structurels

diagrammes statiques (comment le système est-il construit ?) :

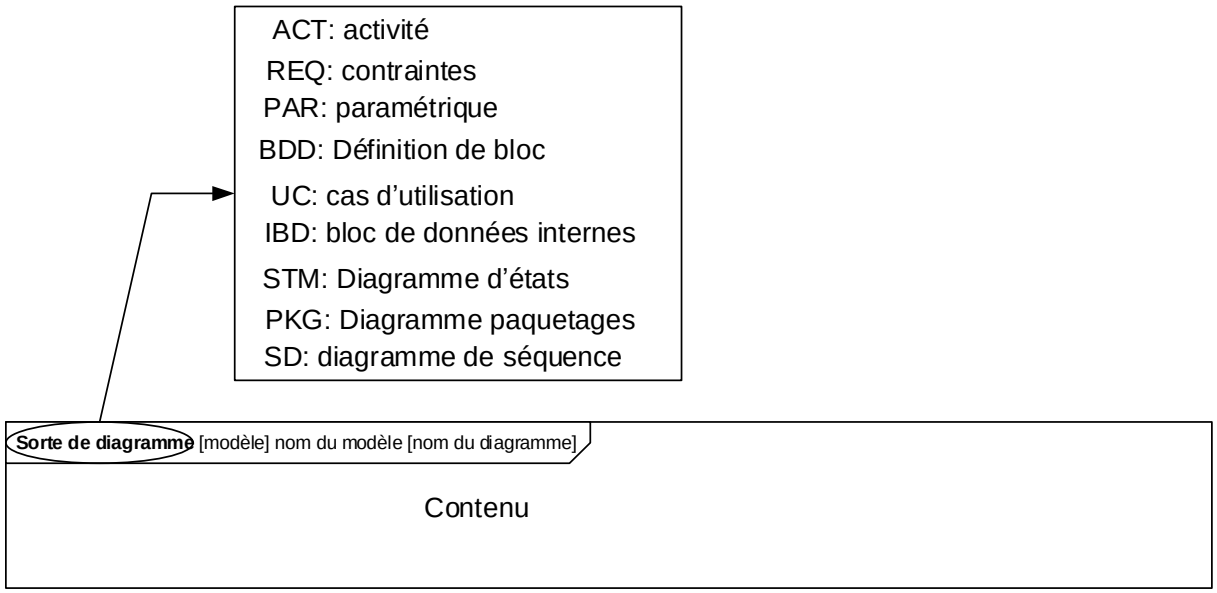
- aspect transverse : le diagramme d'exigences (montre les exigences du système et leurs relations) ;



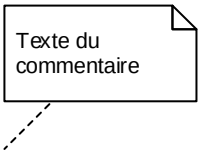
Sur gras : Diagramme commun UML et SYSML



Tous les diagrammes SysML sont constitués d'un cartouche. Les cartouches de diagramme fournissent un contexte visible pour les diagrammes. Ils permettent de spécifier le type de diagramme SysML, le type de l'élément concerné, l'élément concerné, et le nom du diagramme.



SysML permet d'utiliser des notes graphiques sur tous les types de diagrammes (sorte de commentaire).



Le chapitre suivant présente le diagramme de cas d'utilisation, qui fournit une description de haut niveau des fonctionnalités du système.

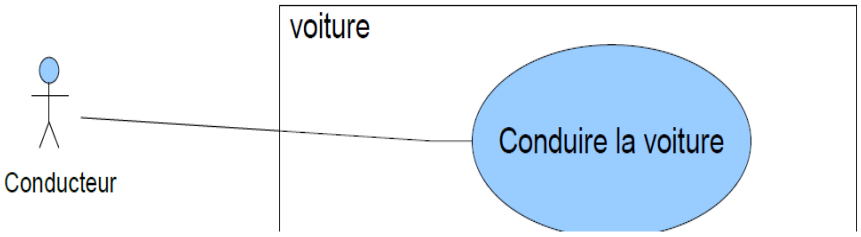
Le diagramme de cas d'utilisations décrit la captation du besoin exprimé par le client. Pour valider le besoin, le valoriser et chercher des solutions.
Nous allons décrire dans les chapitres suivants les diagrammes UML ou SYSML prépondérants dans le cadre d'une démarche de gestion de projet. Cette démarche s'inscrira dans le cadre d'une spécification pour le génie logiciel (UML) ou système (SysML).

2) Diagramme de cas d'utilisation

2.1) Présentation

Ce diagramme est une représentation des **fonctionnalités** du système **visible de l'extérieur**

Il montre les interactions fonctionnelles entre les acteurs et le système.
Il délimite précisément le système, décrit ce que fera le système sans spécifier comment (et non ce que fera l'utilisateur).
Un cas d'utilisation se représente par une **ellipse** contenant l'intitulé du cas sous forme d'un **verbe à l'infinitif** et d'un **complément.**
Cette ellipse est reliée par un trait continu à un acteur symbolisé par un petit bonhomme avec son rôle inscrit dessous.
L'acteur peut être une personne, un objet ou un processus qui interagit avec le système. Les cas d'utilisation sont dans un cadre représentant la **frontière du système**, les acteurs sont à l'extérieur de ce cadre.

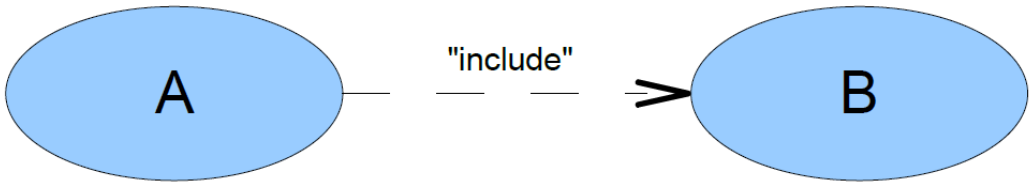


Un cas d'utilisation (use case, ou UC) représente un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable intéressant pour un acteur particulier. Chaque cas d'utilisation spécifie un comportement attendu du système considéré comme un tout, sans imposer le mode de réalisation de ce comportement. Il permet de décrire ce que le futur système devra faire, sans spécifier comment il le fera.
Un cas d'utilisation doit être relié à au moins un acteur.

2.2) Relations entre les cas d'utilisation

a "include"

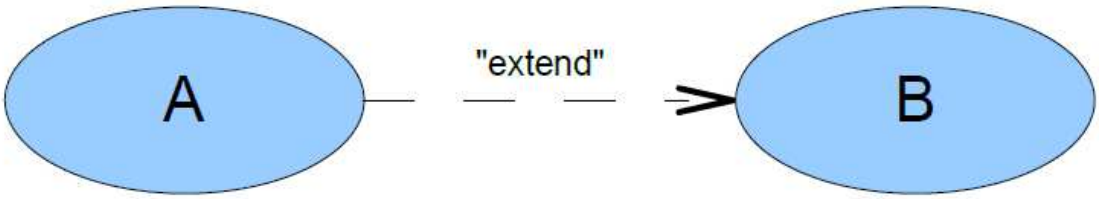
Un cas A inclut un cas B si B est **obligatoirement** utilisé quand A est utilisé. Le cas d'utilisation de base en incorpore **explicitement** un autre, de façon **obligatoire**,



b "extend"

Un cas A étend un cas B si A est **éventuellement** (pas toujours) utilisé quand B est utilisé. Extension «extend» : Le cas d'utilisation de base en incorpore **implicitement** un autre, de façon **optionnelle**,

La relation extend indique une possibilité, **un complément** possible.



c) généralisation

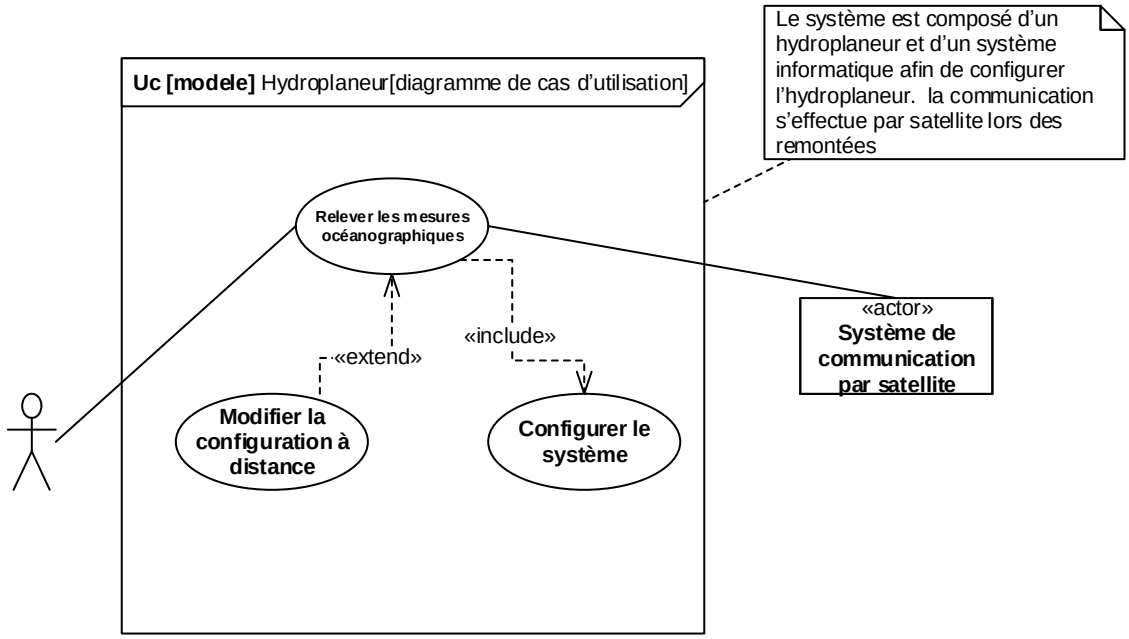
Un cas A est une généralisation d'un cas B si B est un cas particulier de A.



B hérite de A

Généralisation/spécialisation (flèche blanche): les cas d'utilisation ou acteurs descendants **héritent** de la description de leur parent commun.

Les **acteurs principaux** sont représentés **à gauche** des cas d'utilisation, et les **acteurs secondaires à droite**. Un acteur **non humain** est représenté par **un rectangle**.



Une erreur fréquente concernant les cas d'utilisation consiste à vouloir descendre trop bas du point de vue microscopique. Un cas d'utilisation représente un ensemble de séquences d'actions réalisées par le système, et le lien entre ces séquences d'actions est précisément l'objectif métier de l'acteur. Le cas d'utilisation ne doit donc pas se réduire systématiquement à une seule séquence, et encore moins à une simple action.

Limitez à 20 le nombre de vos cas d'utilisation de base (en dehors des cas inclus, spécialisés, ou des extensions). Avec cette limite arbitraire, on reste synthétique concis en offrant une bonne lisibilité.

Les acteurs candidats sont systématiquement :

- les utilisateurs humains directs : faites donc en sorte d'identifier tous les profils possibles, sans oublier l'administrateur, l'opérateur de maintenance, etc. ;

On peut simplifier un diagramme en regroupant les cas d'utilisation dans un paquetage qui serait identifié par un cas d'utilisation principal (on évite alors de trop alourdir le diagramme)

- les autres systèmes connexes qui interagissent aussi directement avec le système étudié, souvent par le biais de protocoles

bidirectionnels.

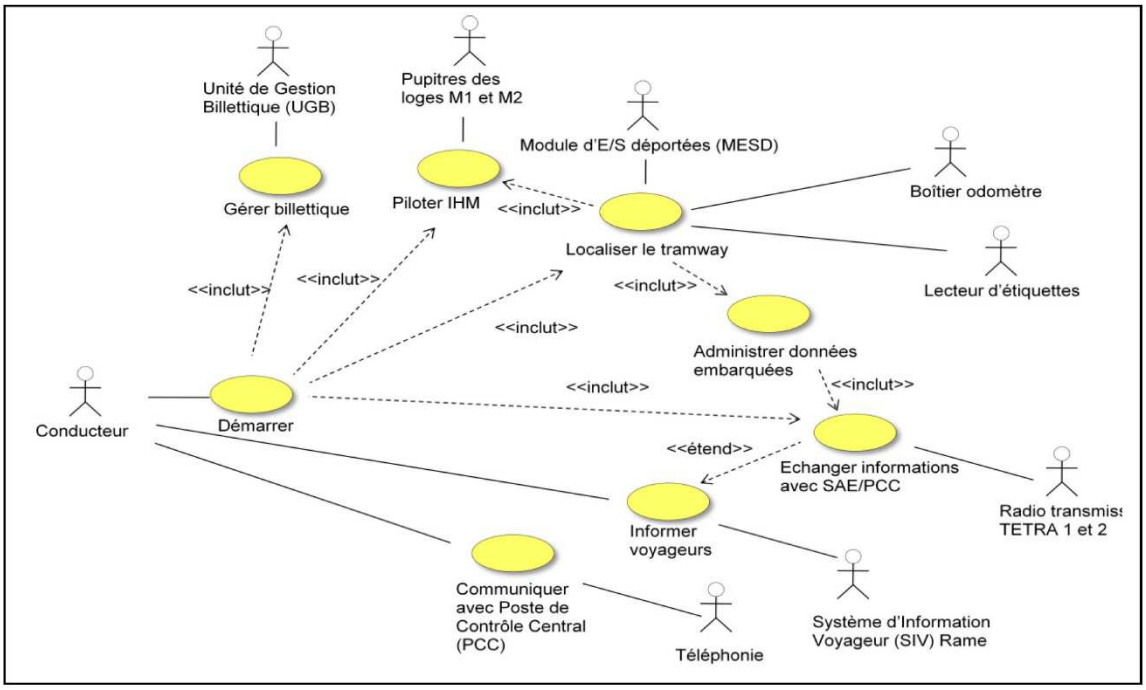


Diagramme de cas d'utilisation du système d'exploitation embarqué d'un tramway

Nous appelons acteurs principaux ceux pour qui le cas d'utilisation produit un résultat observable. Par opposition, nous qualifions d'acteurs secondaires les autres participants du cas d'utilisation. Les acteurs secondaires sont souvent sollicités pour des informations complémentaires ; ils peuvent uniquement consulter ou informer le système lors de l'exécution du cas d'utilisation. Une bonne pratique *consiste à* faire figurer les acteurs principaux à gauche des cas d'utilisation, et les acteurs secondaires à droite.

On peut également associer à chaque cas d'utilisation un ou plusieurs diagrammes de séquence comme expliqué dans le chapitre qui suit.

Pour documenter les cas d'utilisation, la description textuelle est indispensable, En revanche, le texte présente des désavantages puisqu'il est difficile de montrer comment les enchaînements se succèdent, ou à quel moment les acteurs secondaires sont sollicités.

Afin de faciliter la lecture du diagramme on peut regrouper les cas d'utilisation dans un paquetage

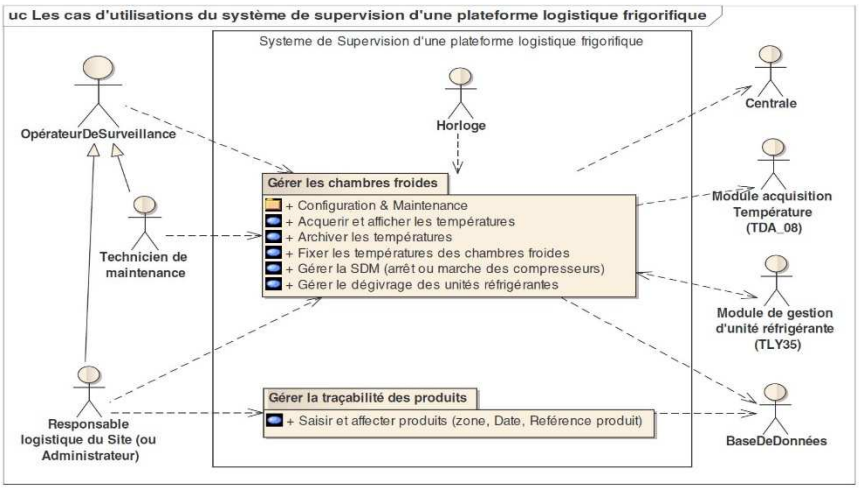


Figure 4 : Paquetages des cas d'utilisation

2.3) Le vocabulaire usuel du diagramme des cas d'utilisation

Choisir, développer, participer, gérer, consulter, identifier, chronométrer, mesurer, relever, modifier, configurer, surveiller , évacuer, permettre, déplacer, identifier, superviser, charger, manipuler, visualiser, effectuer, commander, affecter, paramétrer, démarrer, arrêter, fixer, archiver, saisir, préparer, anticiper, planifier, calculer,

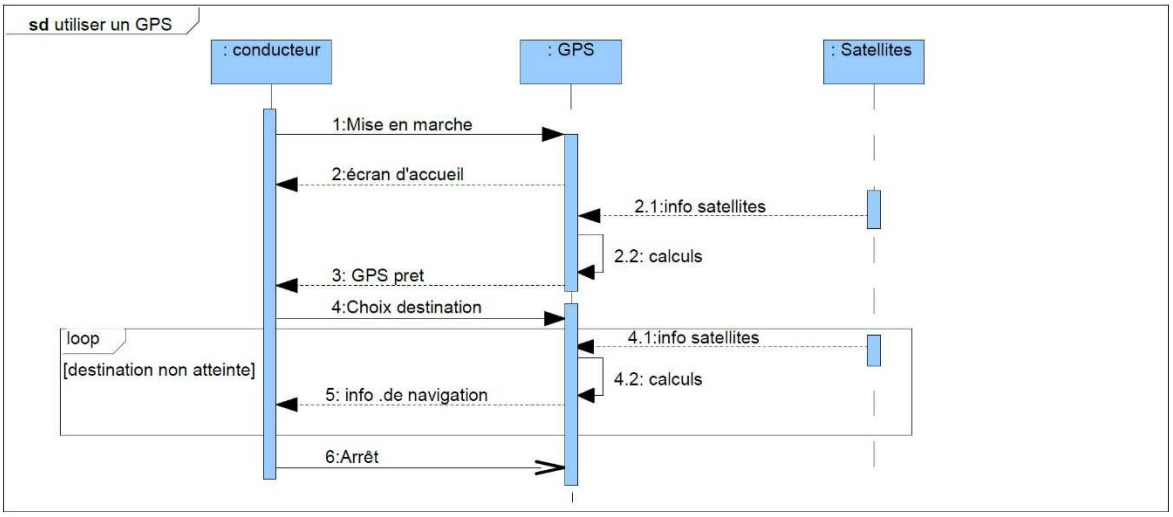
3) Diagramme de séquence ou d'interactions

3.1) Définitions

Ce diagramme décrit la **chronologie** les interactions (messages) entre acteurs et objets correspondant à un cas d'utilisation.

Il décrit chronologiquement les échanges de messages au sein d'un système

Chaque élément du système est représenté par un rectangle doté d'une **ligne de vie** verticale. Un rectangle étroit vertical

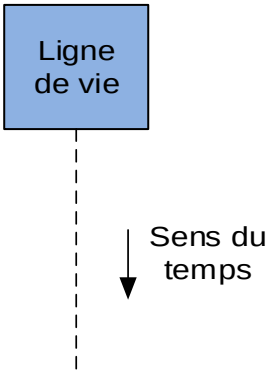


Un objet est matérialisé par un rectangle et une barre verticale appelée ligne de vie.

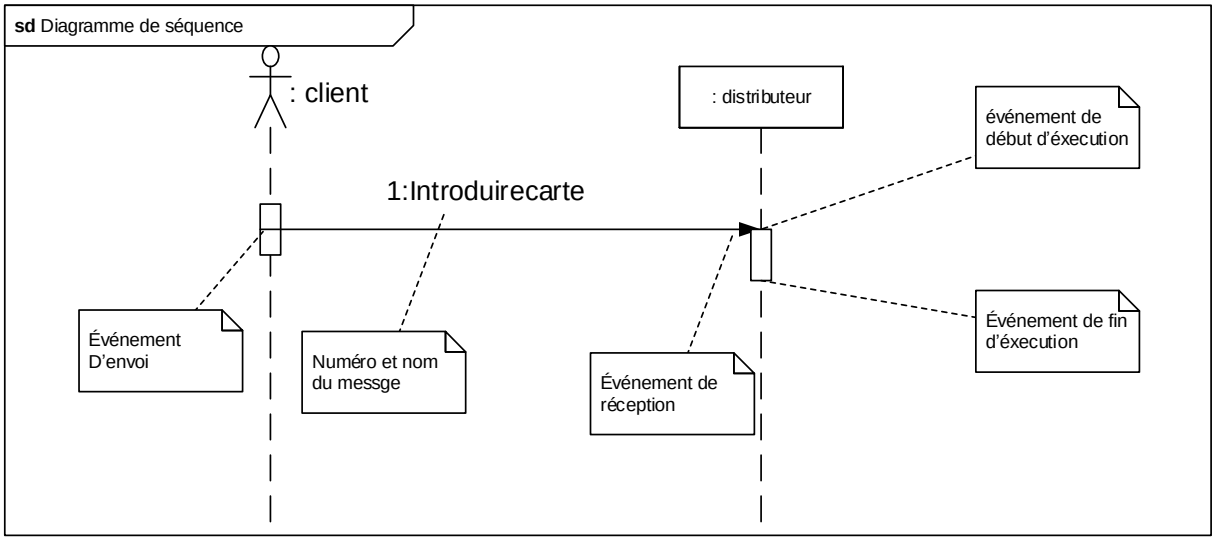


3.2) Les lignes de vie:

Elles représentent un élément participant à la communication. Elles peuvent être actives ou inactives (bandes verticales). Le diagramme de séquence montre **la séquence verticale** des messages passés entre éléments (**lignes de vie**) au sein d'une interaction. Elle est représentée graphiquement par une ligne verticale en pointillés.



Les objets communiquent en échangeant des messages **(appel de méthodes en POO)** représentés au moyen de lignes horizontales, orientées de l'émetteur du message vers le destinataire.



Les diagrammes de séquence permettent également de représenter les périodes d'activités des objets. Une période d'activité correspond au temps pendant lequel un objet effectue une action. Les périodes d'activité se représentent par des bandes rectangulaires placées sur les lignes de vie. Le début et la fin d'une bande correspondent respectivement au début et à la fin d'une période d'activité.

3.3) Types de messages:

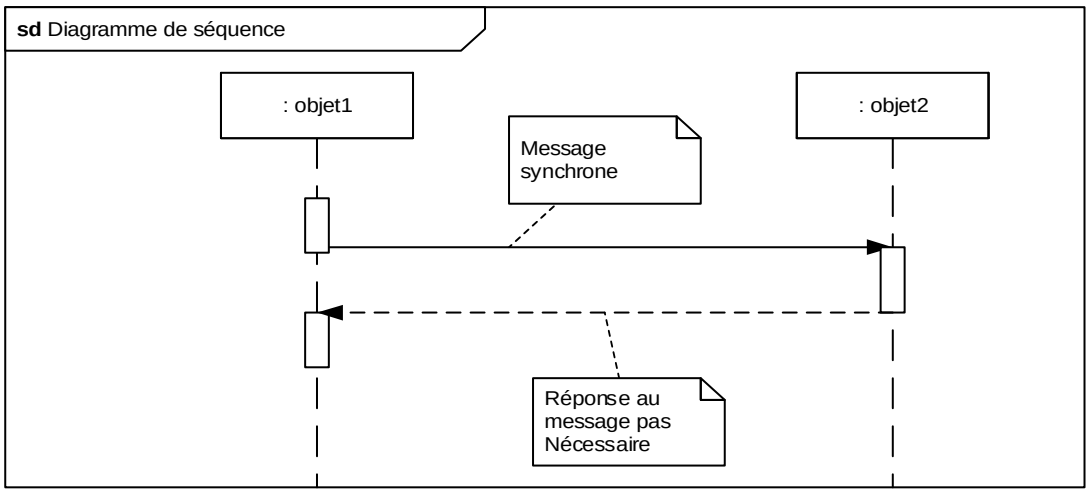
Message synchrone (émetteur bloqué en attente de réponse en POO appel d'une méthode)

Message asynchrone (événements matériels logiciels, ou interruptions).

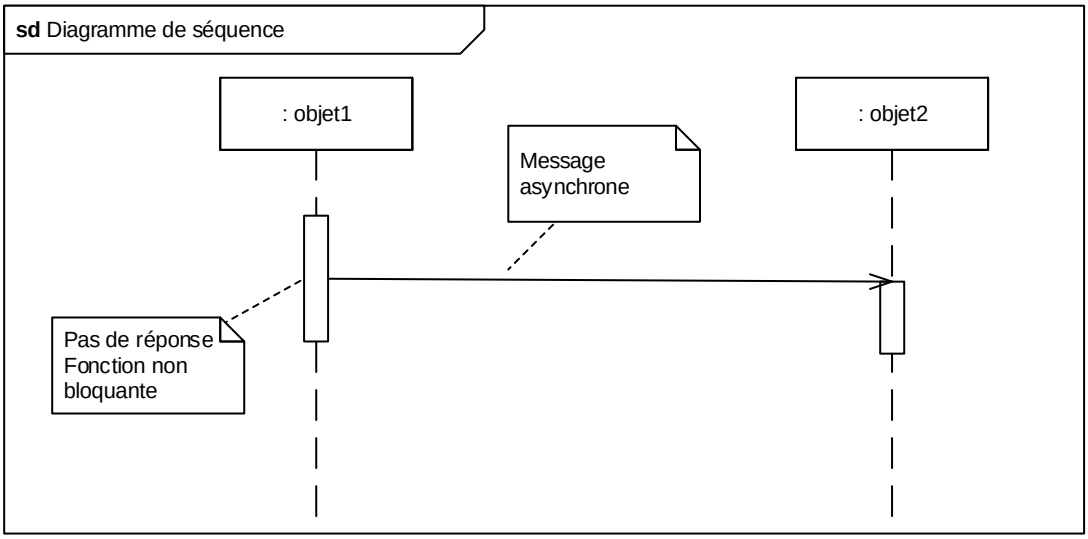
Message réflexif (comportement interne, appel d'une méthode interne à l'objet en POO)

type	signification	représentation
asynchrone	message sans réponse attendue	
synchrone en émission	message qui attend une réponse	
synchrone en réception	Réponse à un message	

La flèche pointillée représente un retour. Cela signifie que le message en question est le résultat direct du message précédent.
La flèche qui boucle (message réflexif) permet de représenter un comportement interne.
Un message synchrone (émetteur bloqué en attente de réponse) est représenté par une flèche pleine, alors qu'un message asynchrone est représenté par une flèche évidée.

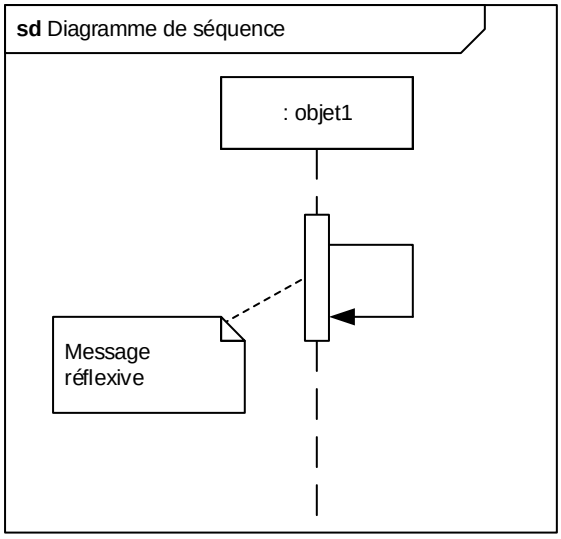


Les **envois asynchrones** pour lesquels l'émetteur n'est pas bloqué et peut continuer son exécution.



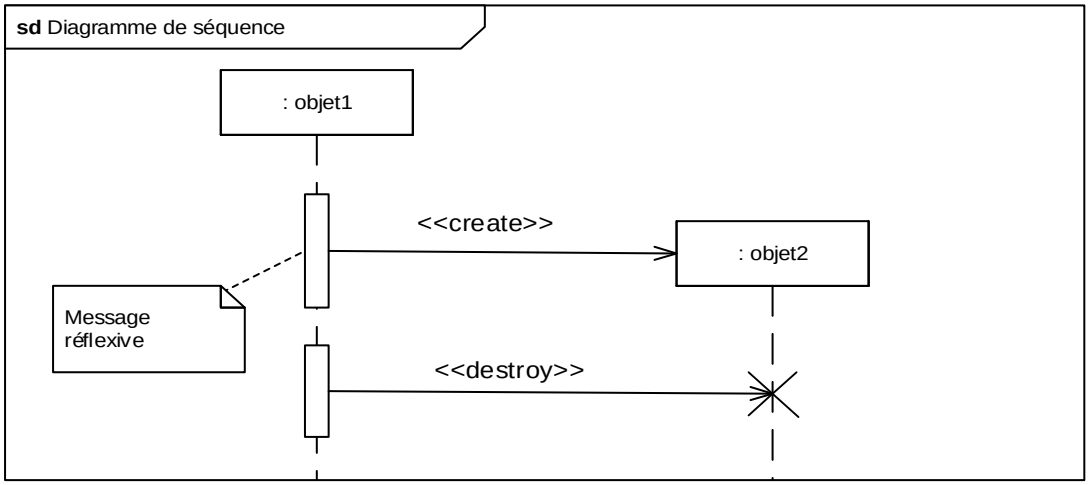
Les messages asynchrones correspondent à des interruptions, des événements. Ils peuvent être utilisés par exemple sur des machines différentes, socket, bus de terrain …

Un objet peut également s'envoyer un message (message réflexive). Cette situation se représente par une flèche qui boucle sur la ligne de vie de l'objet. Cette construction peut indiquer un point d'entrée dans une activité de plus bas niveau.



3.4) Création et destruction d'un objet

La création d'un objet se représente en faisant pointer le message de création sur le rectangle qui symbolise l'objet créé.
La destruction est indiquée par la fin de la ligne de vie et par une lettre x

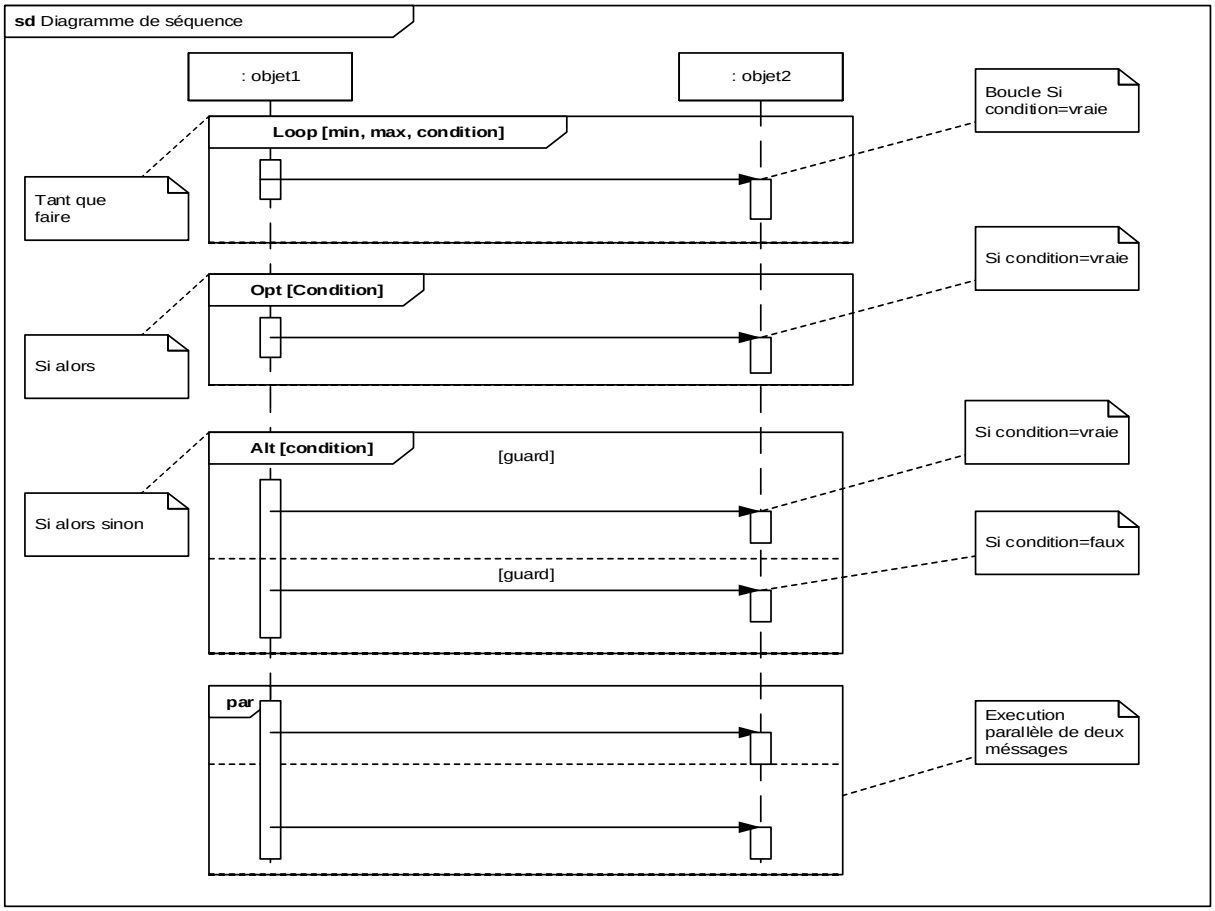


3.5) Les opérateurs

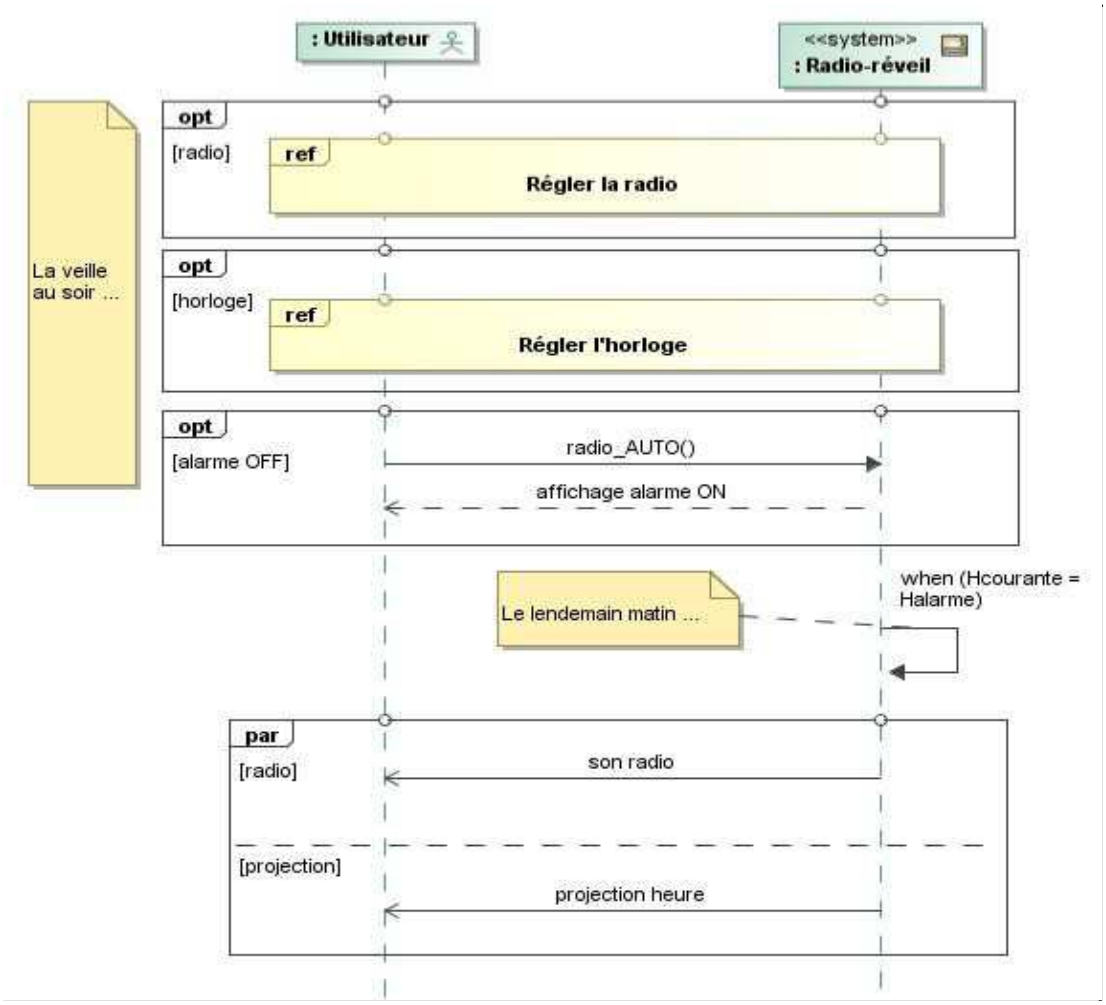
Il est possible d'ajouter des **opérateurs** sur des sous-parties de la séquence.

Les principaux opérateurs sont:

- ✓ Opt,optionnel: Le bloc s'exécute que si la condition fournie est vraie.
- ✓ Loop,boucle:Le bloc peut s'exécuter plusieurs fois.
- ✓ Alt, fragment alternatif : condition vraie suivie de sinon condition fausse.



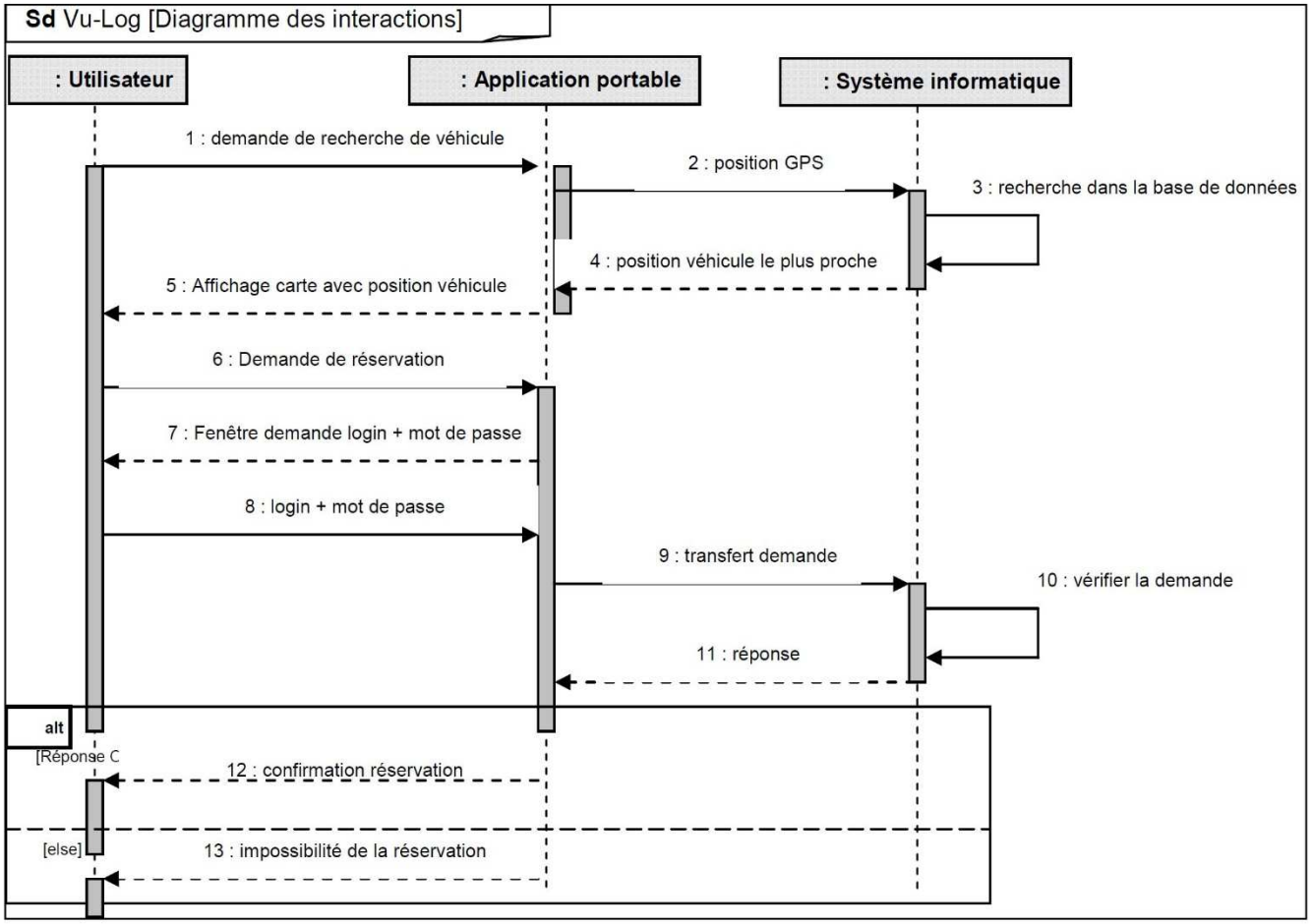
Un diagramme de séquence peut en référencer un autre grâce à un second type de rectangle avec le mot-clé ref. Cette notation est très pratique pour modulariser les diagrammes de séquence, et créer des hyperliens graphiques exploitables par les outils de modélisation.



3.6) Diagramme de séquence « système »

Pour les messages propres à un cas d'utilisation, les diagrammes de séquence « système » montrent non seulement les acteurs externes qui interagissent directement avec le système, mais également ce système (en tant que boîte noire) et les événements système déclenchés par les acteurs. L'ordre chronologique se déroule vers le bas et l'ordre des messages doit suivre la séquence décrite dans le cas d'utilisation.

Nous utilisons le terme de diagramme de séquence « système » pour souligner le fait que dans ce type de diagramme de séquence, nous considérons le système entier comme une boîte noire et le représentons par une seule ligne de vie. Le comportement du système est décrit vu de l'extérieur, sans préjuger de comment il le réalisera.



4) Diagramme d'exigences

4.1) Présentation

Le diagramme d'exigences synthétise en langage SysML le cahier des charges.

Il nous informe sur les exigences auxquelles doit satisfaire le système. Il permet une organisation hiérarchique des exigences et l'association avec les éléments du modèle

Une exigence permet de spécifier une capacité ou une contrainte qui doit être satisfaite par un système.

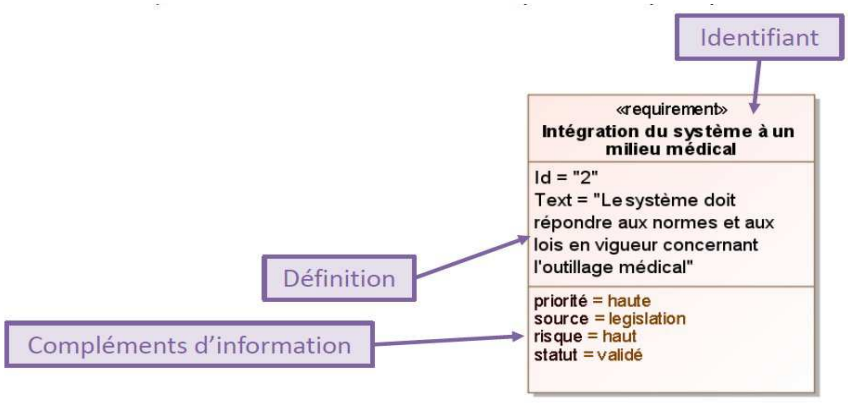
Elle peut spécifier une fonction que le système devra réaliser ou une condition de performance, de fiabilité, de sécurité, technique, physique, de fiabilité, d'ergonomie, d'esthétisme…**etc**.

Les exigences servent à établir un contrat entre le client et les réalisateurs du futur système.

C'est un diagramme fonctionnel. Il décrit les exigences du cahier des charges fonctionnel. Une exigence exprime une capacité ou une contrainte à satisfaire par un système.

Les deux propriétés de base d'une exigence sont :

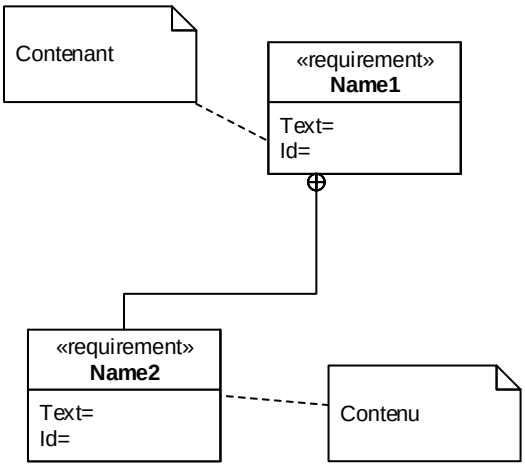
- un identifiant unique (permettant ensuite de gérer la traçabilité avec l'architecture, etc.) ;
- un texte descriptif.



4.2) Les relations entre bloc d'exigences

Les exigences peuvent être reliées entre elles par des relations de contenance, de raffinement ou de dérivation :

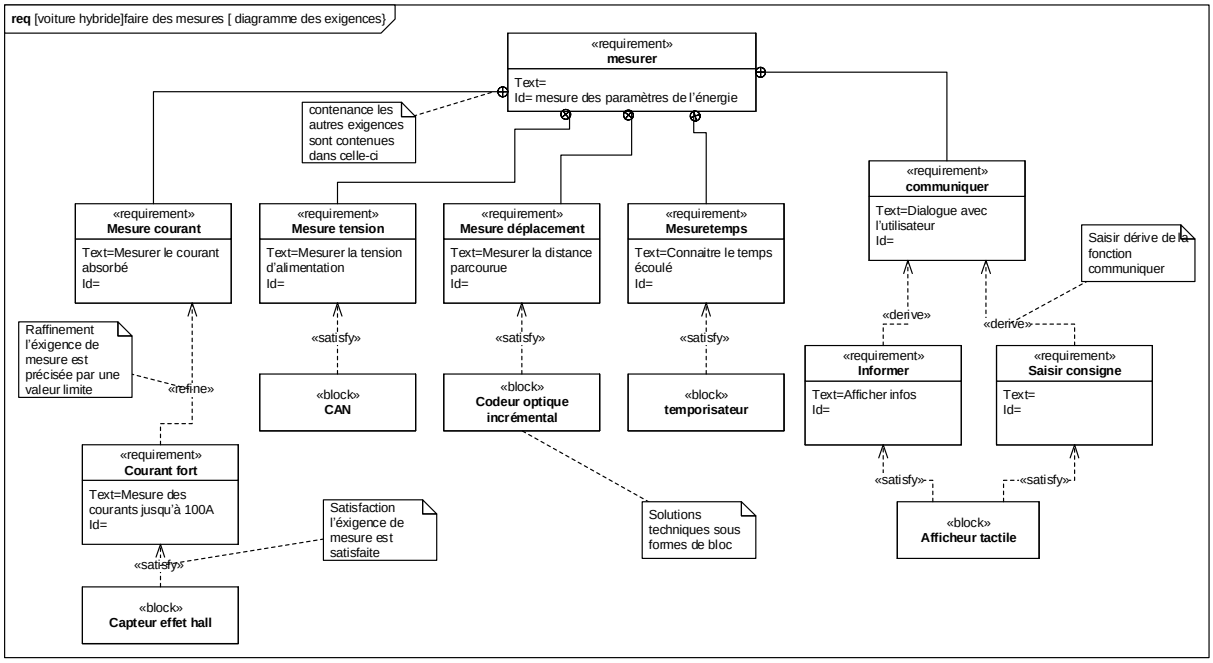
a) la contenance (ligne terminée par un cercle contenant une croix du côté du conteneur) permet de décomposer une exigence composite en plusieurs exigences unitaires, plus faciles ensuite à tracer vis-à-vis de l'architecture ou des tests. Elle permet de décomposer une **exigence** en plusieurs **sous-exigences**



b) La dérivation (« deriveReqt ») consiste à relier des exigences de niveaux différents, par exemple des exigences d'architecture. Elle consiste à relier des exigences de niveaux différents, par exemple des exigences système à des exigences de niveau sous-système avec un lien de parenté ou de filiation.

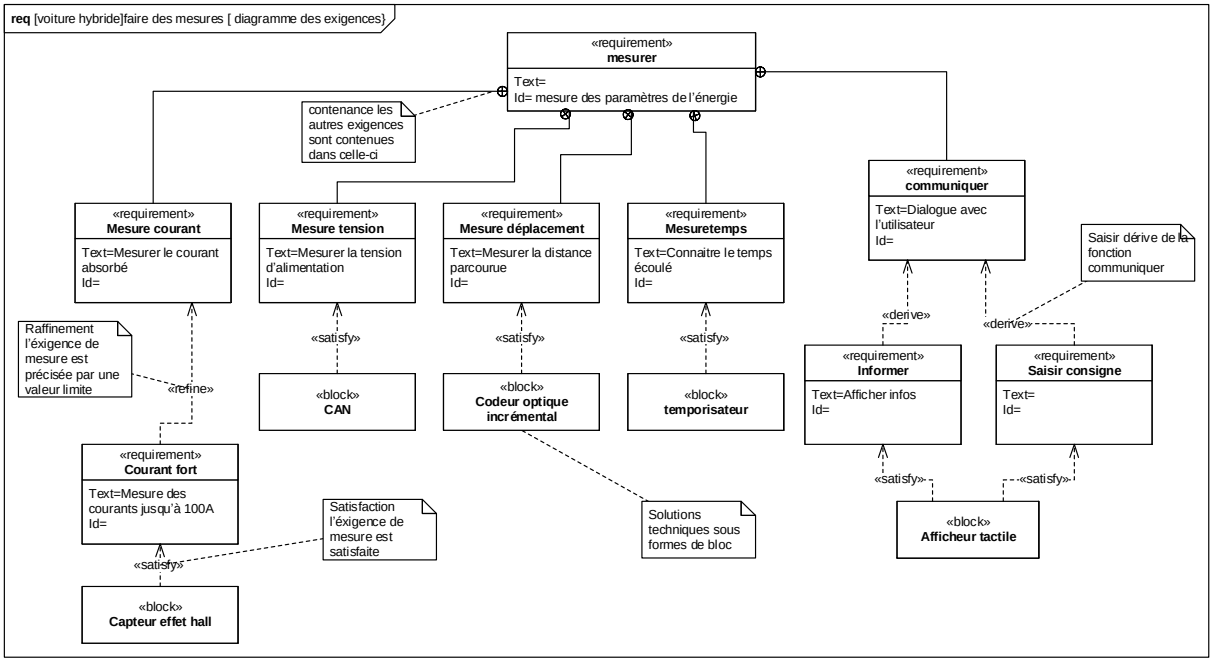
«deriveReqt»





c) Le raffinement («refine»): permet de rajouter des précisions, par exemple de données quantitatives

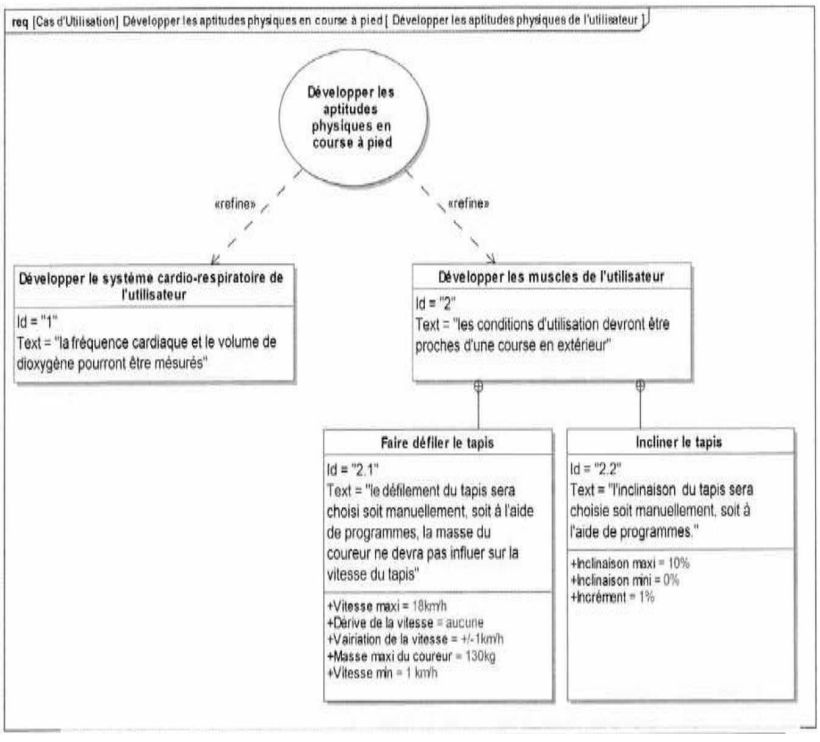
« refine »

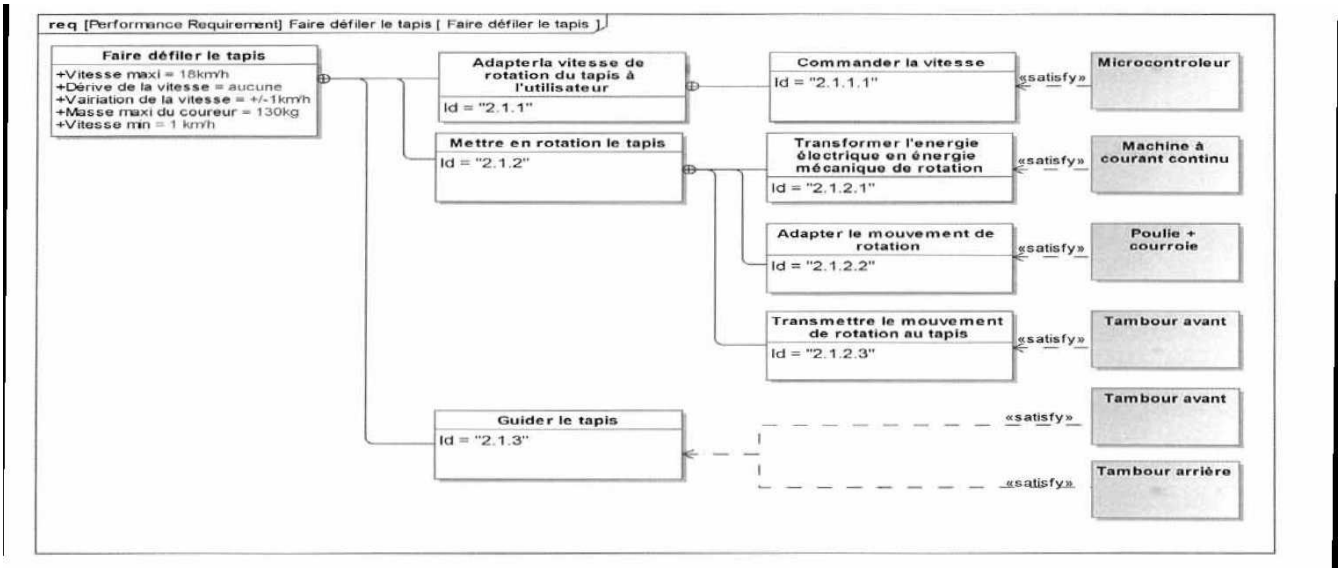


d) exigence – cas de test : « verify ».

e) exigence – bloc d'architecture : « satisfy » ;

Un diagramme d'exigences peut aboutir sur une solution technique avec un diagramme de bloc qui satisfait l'exigence comme sur l'exemple ci-dessus ou ci-dessous. Dans le cas où le nombre d'exigences est important on peut décomposer le diagramme en plusieurs sous-diagramme comme un tapis roulant




req [Performance Requirement] Faire défiler le tapis [Faire défiler le tapis]
Faire défiler le tapis
+Vitesse maxi = 18km/h
+Dérive de la vitesse = aucune
+Vairiation de la vitesse = +/-1km/h
+Masse maxi du coureur = 130kg
+Vitesse min = 1 km/h
Adapterla vitesse de rotation du tapis à l'utilisateur
Id = "2.1.1"
Mettre en rotation le tapis
Id = "2.1.2"
Guider le tapis
Id = "2.1.3"
Commander la vitesse
Id = "2.1.1.1"
Transformer l'energie électrique en énergie mécanique de rotation
Id = "2.1.2.1"
Adapter le mouvement de rotation
Id = "2.1.2.2"
Transmettre le mouvement de rotation au tapis
Id = "2.1.2.3"
«satisfy»
Microcontroleur
Machine à courant continu
Poulie + courroie
Tambour avant
Tambour avant
Tambour arrière

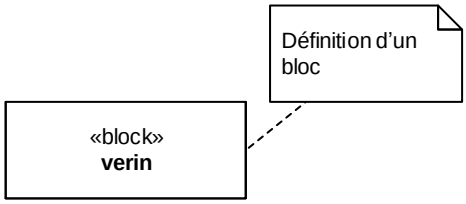
5) Le diagramme de définition de blocs

5.1) Présentation

La structure d'un système est exprimée sur un Diagramme de Définition de Bloc à l'aide de blocs, il peut représenter divers éléments (système, sous-système, composant élémentaire, flot,…).

Un bloc pour représenter des entités physiques, mais aussi des entités logiques ou conceptuelles.

Un bloc est représenté graphiquement par un rectangle découpé en compartiments. Le nom du bloc apparaît tout en haut dans la définition du bloc, et constitue l'unique compartiment obligatoire.

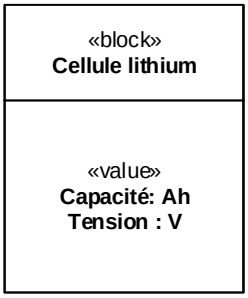


Le mot-clé « block » apparaît par défaut, sauf si nous définissons de nouveaux mots-clés tels que « system » ou « subsystem

On a différentes zones en dessous de la définition du bloc. Ce sont des compartiments qui possèdent des labels indiquant ce qu'ils contiennent.

Les plus connus sont :

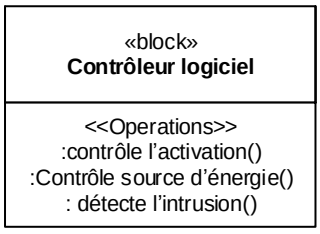
- **Les valeurs** (value properties) sont utilisées pour modéliser les caractéristiques quantitatives des blocs (domaine de valeur, dimension et unité optionnelles). Il est très important de bien définir les types de ces valeurs en termes de value types réutilisables.



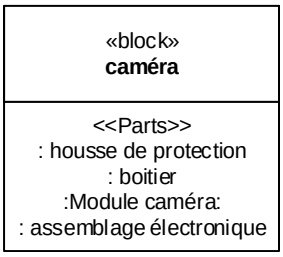
- **Les attributs** qui représentent des propriétés qui caractérisent ce bloc.

- **Les opérations** qui représentent ce que l'on peut demander au bloc. Les opérations sont montrées dans un compartiment supplémentaire avec leur signature (nom, paramètres et type de retour) :

  nom_opération (liste de paramètres) type_retour



Si ce bloc est décomposé en un ensemble de parts (partie d'un bloc) dans un autre diagramme (diagramme de bloc interne), la multitude de parts est automatiquement chargées (traçabilité) dans un label noté parts

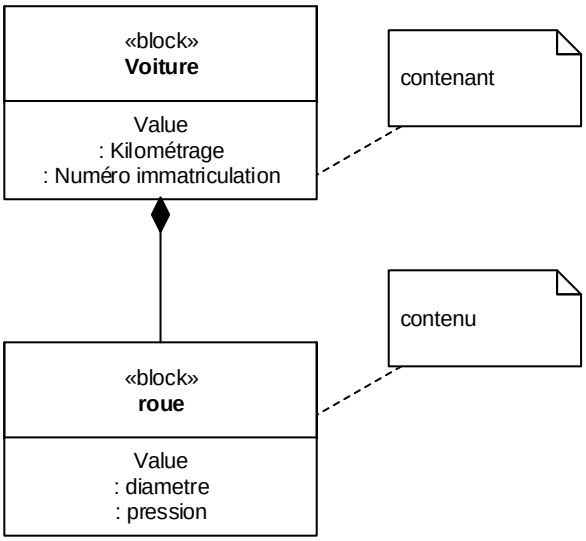


Il peut représenter un système complet, un sous-système ou un composant élémentaire. Les blocs sont décomposables et peuvent posséder un comportement. Le diagramme de définition de blocs (block definition diagram ou bdd) décrit la hiérarchie du système et les classifications système/composant.

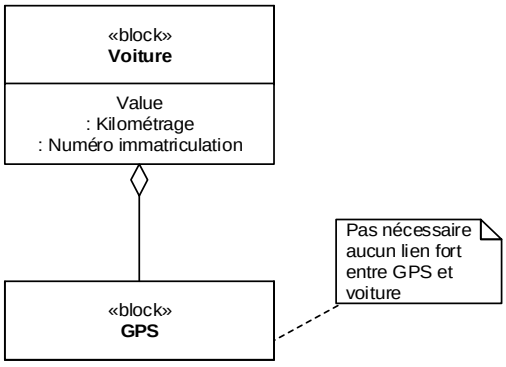
5.2) Les relation entre les blocs

Ces blocs sont reliés entre eux par des relations: ◦Composition, Agrégation, Association …..

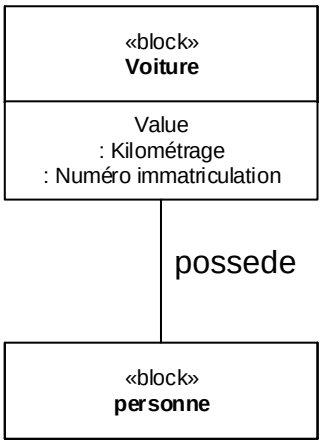
a) **Composition** La relation de composition existe lorsqu'un bloc englobe ses parties. Côté du tout indiqué par un losange plein.



b) **Agrégation** Similaire à la composition, cependant, les parties ne sont pas nécessaires au tout. Côté du tout indiqué par un losange blanc.

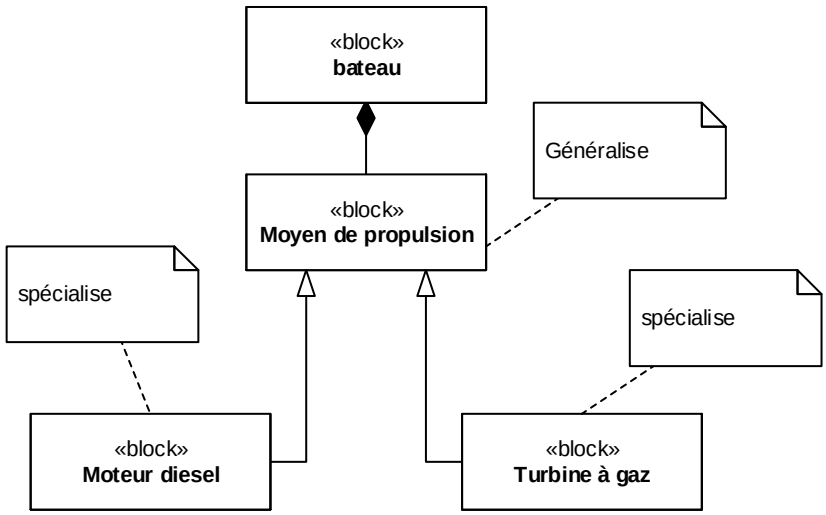


c) **L'association** est une relation n'impliquant pas de contenance mais une relation d'égal à egal. Une association représente une relation sémantique durable entre deux blocs. , une association entre blocs est par défaut bidirectionnelle. Les compositions et les agrégations sont des cas particuliers d'association.

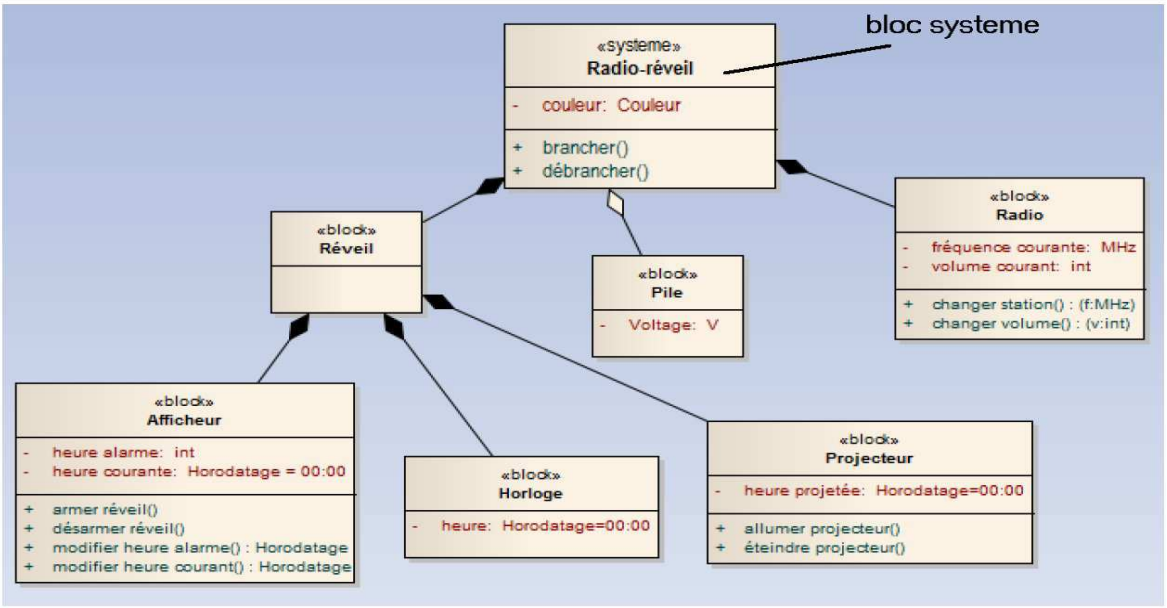


d) **La généralisation**: Il est souvent intéressant de factoriser des propriétés communes (valeurs, parties, etc.).

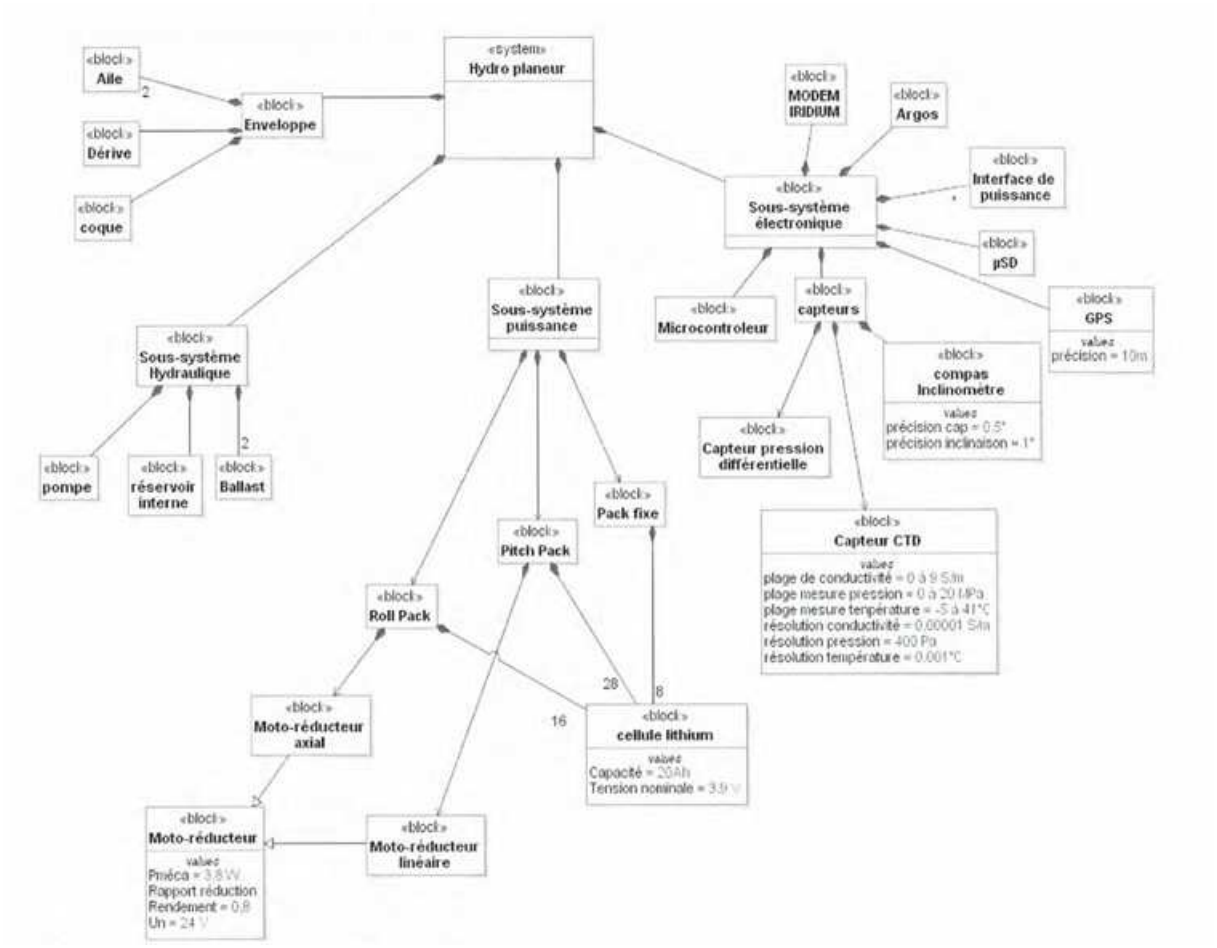
Ainsi, les blocs spécialisés «héritent» des propriétés du bloc généralisé et peuvent comporter des propriétés spécifiques supplémentaires.



5.3) Diagramme de bloc d'un radio-réveil



5.4) Diagramme de bloc d'un hydro-planeur

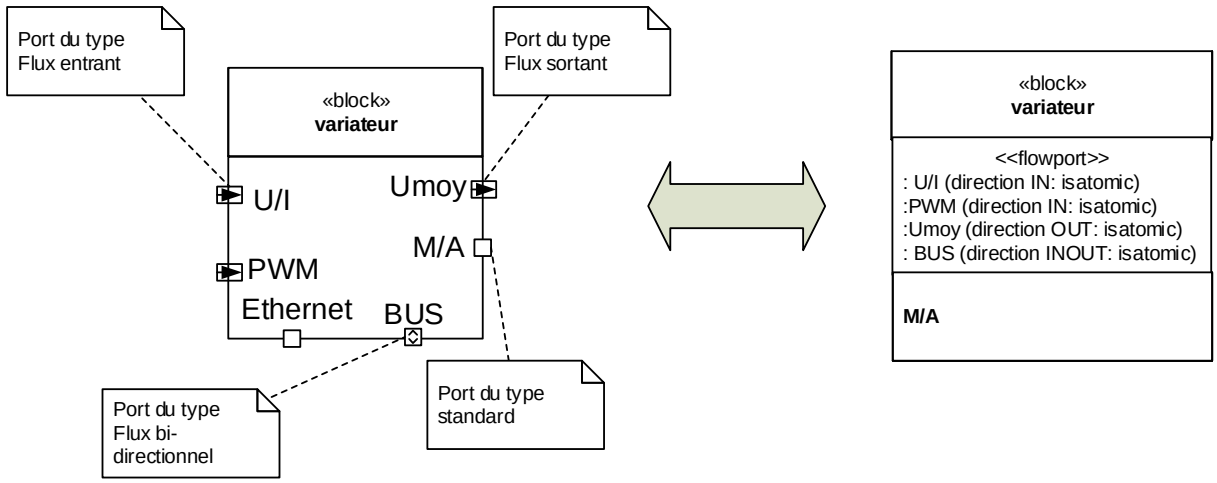


Le bloc permet de décrire également les flux qui circulent à travers un système.

Les ports sur les blocs représentent entrées et sorties de flux.

Ces flux peuvent être de toute nature :

- Matière fluide, gaz, objet...
- Energie Electrique, mécanique...
- Information Logique, analogique...



Les ports faisant partie de la définition des blocs, il est tout à fait possible de les faire apparaître dès le bdd. Dans ce cas, ils peuvent être représentés soit graphiquement comme précédemment, soit sous la forme d'un compartiment supplémentaire.

L'intérêt du diagramme suivant le diagramme de bloc interne consiste là encore à pouvoir décrire les connexions entre les ports de différents blocs au moyen de connecteurs, alors qu'un bloc du diagramme de définition de blocs ne permet que de définir les ports sans les connecter.

6) Le diagramme de bloc interne

6.1) Présentation

Ce diagramme permet de détailler un bloc et de montrer les flux internes entre les différents blocs appelés des parts

Les flux sont clairement identifiés avec un sens de circulation :

- Des flux d'information
- Des flux d'énergies (électrique et mécanique)
- Des flux de matière (gaz et liquides)

Le diagramme de bloc interne permet également de décrire la logique de connexion, de services et de flux entre parts grâce au concept de « port ».

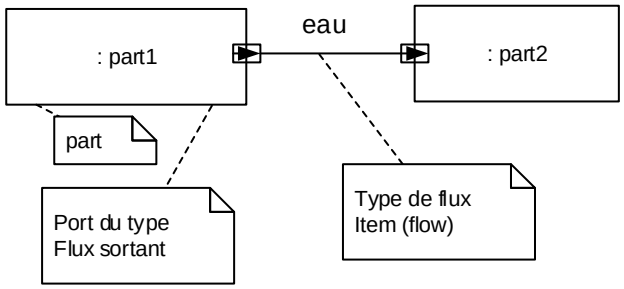
6.2) Les flow port

Les ports définissent les points d'interaction offerts (provided) et requis (required) entre les parts. Un part peut avoir plusieurs ports qui spécifient des points d'interaction différents. Les ports peuvent être de deux natures :

- ✓ flux (flow port) : ce type de port autorise la circulation de flux physiques entre les parts. La nature de ce qui peut circuler va des fluides aux données, en passant par l'énergie ;
- ✓ standard : ce type de port autorise la description de services logiques entre les parts, au moyen d'interfaces regroupant des opérations.

Les flow port sont soit atomiques (un seul flux), soit composites (agrégation de flux de natures différentes).

La direction étant simplement indiquée par une flèche à l'intérieur du carré représentant le port.



Les éléments de flux (item flows) permettent de décrire ce qui circule réellement sur les connecteurs, alors que les flow ports définissent ce qui peut circuler

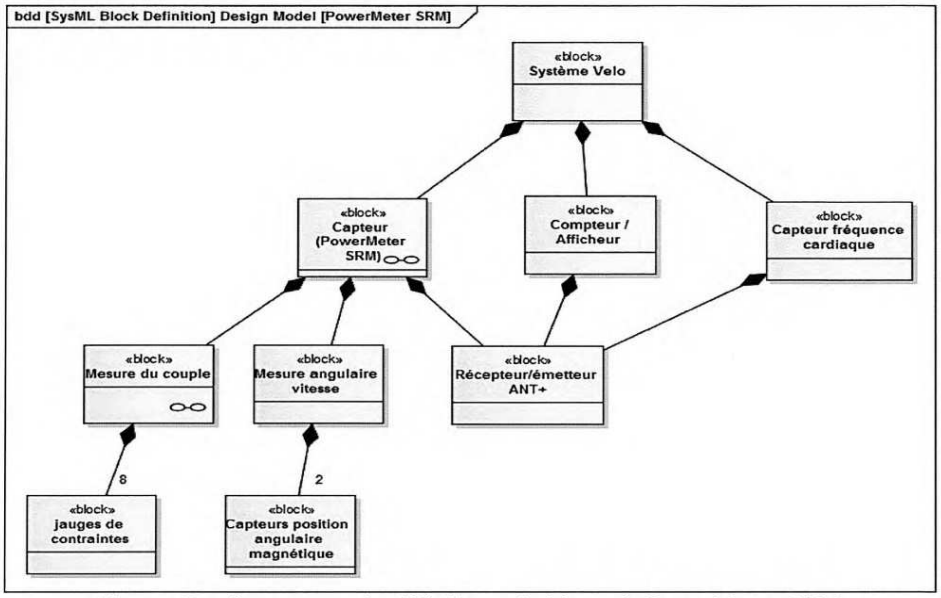


Figure 10 : diagramme de définitions de blocs du PowerMeter SRM

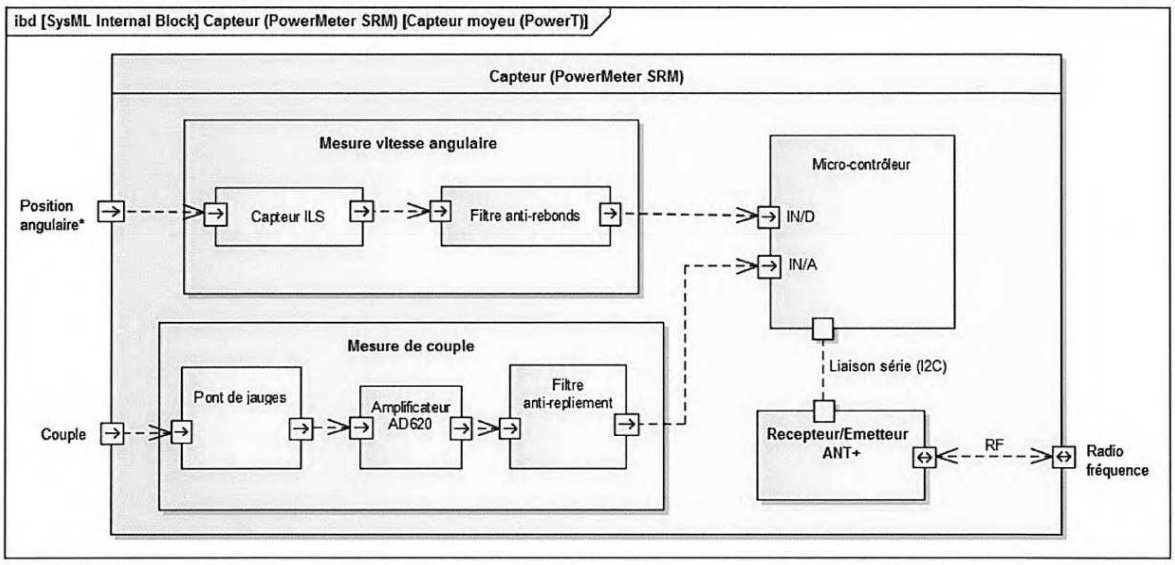


Figure 11 : diagramme de blocs interne du PowerMeter SRM

Le vocabulaire usuel des diagrammes de bloc en électronique

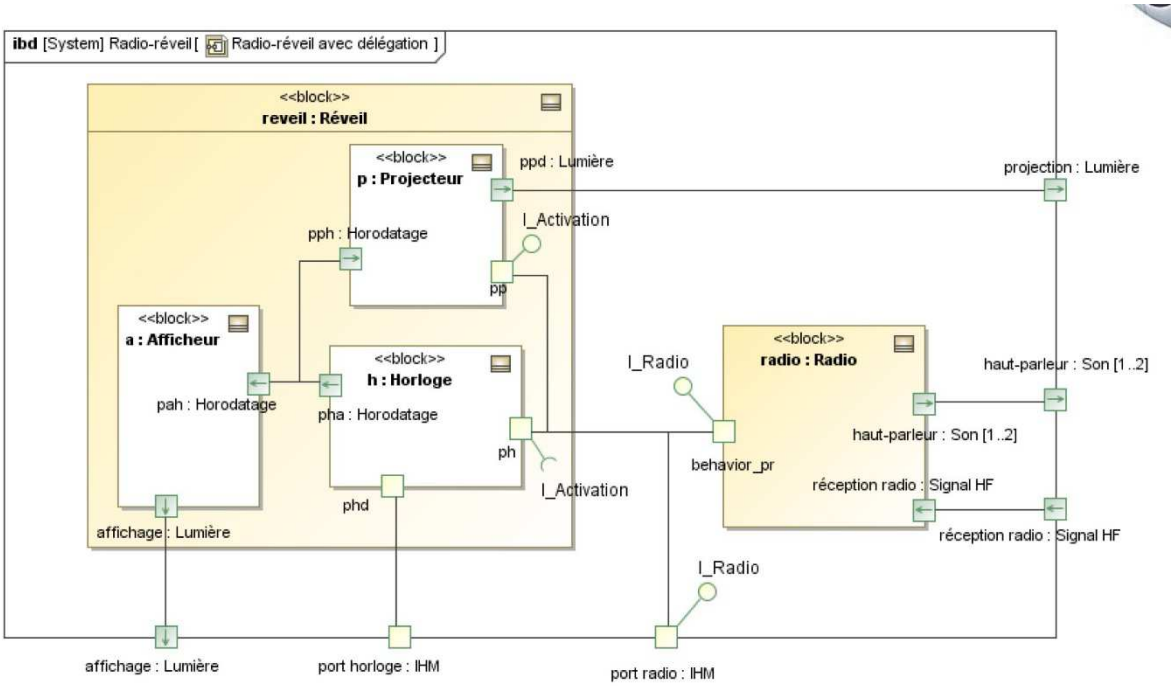
On donne ci-dessous une liste non exhaustive du vocabulaire rencontré dans les diagrammes

traiter, mémoriser, acquérir, surveiller, adapter, contrôler, limiter,enclencher,activer,normaliser,générer,moduler,amplifier,commuter,convertir,démoduler,détecter,coupler,calculer,comparer,intégrer,compter,produire,multiplier ,visualiser,déplacer,capter,décoder,multiplexer, démultiplexer, transduction ,réguler.

6.3) Les interfaces

Pour décrire un comportement basé sur l'invocation de services, le port standard est tout à fait adapté. Mais au lieu d'assigner directement des opérations aux ports, il est plus intéressantde les regrouper en ensembles cohérents **appelés interfaces**.

Une interface est un ensemble d'opérations abstraites constituant une sorte de contrat qui devra être réalisé par un ou plusieurs blocs. Elle est représentée par le symbole d'un cercle.



On peut remarquer que les blocs peuvent encore être décomposés en parts par d'autres diagramme de blocs internes

7) Diagramme d'états transition

7.1) Présentation

Les diagrammes d'états-transitions d'UML décrivent le comportement interne d'un objet à l'aide d'un automate à états finis. Ils présentent les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie en réaction à des événements discrets (de type signaux, invocations de méthode).
Les diagrammes d'états-transitions permettent de décrire les changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs.
· Un état se caractérise par sa durée et sa stabilité, il représente une conjonction instantanée des valeurs des attributs d'un objet.
· Une transition représente le passage instantané d'un état vers un autre.
· Une transition est déclenchée par un événement. En d'autres termes : c'est l'arrivée d'un événement qui conditionne la transition.
· Les transitions peuvent aussi être automatiques, lorsqu'on ne spécifie pas l'événement qui la déclenche.
· En plus de spécifier un événement précis, il est aussi possible de conditionner une transition, à l'aide de "gardes" : il s'agit d'expressions booléennes, exprimées en langage naturel (et encadrées de crochets).

7.2) L'État

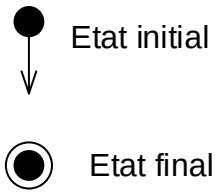
7.2.1) État dans un diagramme d'états-transitions

Le nom de l'état peut être spécifié dans le rectangle



Un objet peut passer par une série d'états pendant sa durée de vie. Un état représente une période dans la vie d'un objet pendant laquelle ce dernier attend un événement ou accomplit une activité. La configuration de l'état actif de l'objet est le jeu des états (élémentaires) qui sont actifs à un instant donné.

7.2.2) État initial et final



7.3) Les événements et les messages

• Un événement est un stimulus envoyé à un objet ;
• l'objet recevant un événement peut réagir en changeant son état (en exécutant une opération) et/ou en émettant un autre événement ;
• un objet placé dans un état donné attend l'occurrence d'un événement pour changer d'état ;
• un événement n'a pas de durée contrairement à un état ;
• un événement peut être émis par un objet du système ou provenir de l'extérieur du système (clic de souris par exemple) ;
• un message est un événement émis par un objet ;
. Une garde est une condition booléenne qui permet de valider une transition

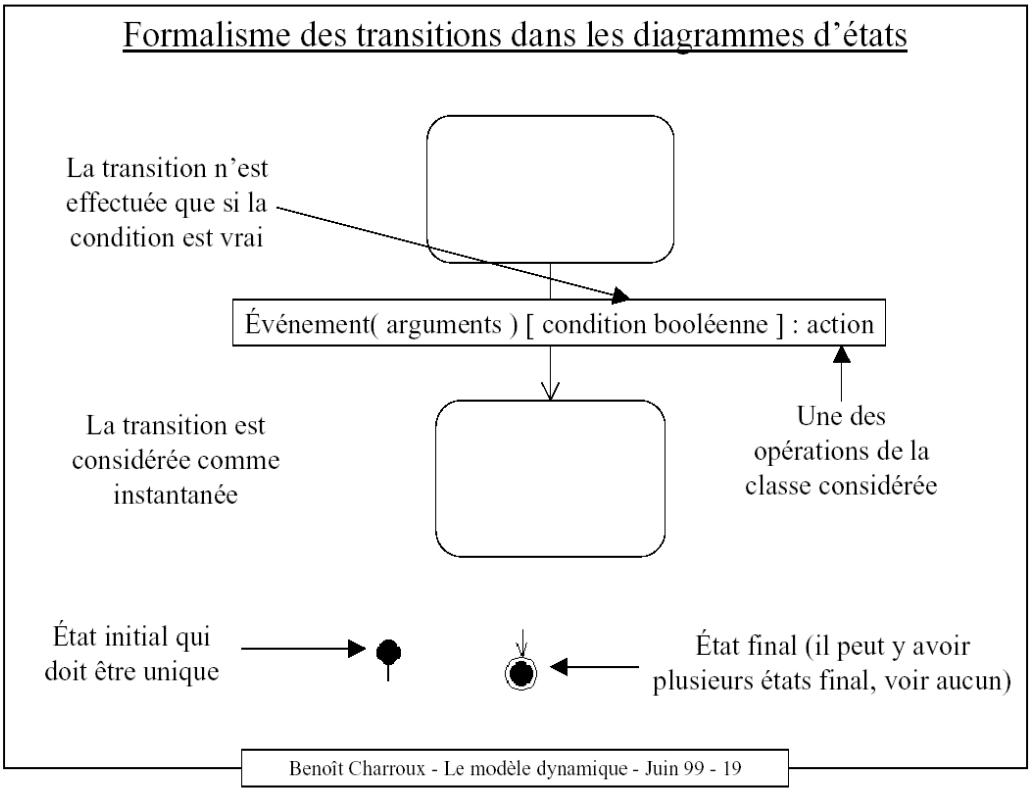
5 types d 'événements sont distingués :

. changement d 'une condition booléenne :
. réception d 'un signal explicite envoyé par un autre **objet**
. demande d 'opération faite par un autre objet :
. épuisement d 'un délai temporel : **after(expression)**

. survenance d'une date, timer : **when(expression)**

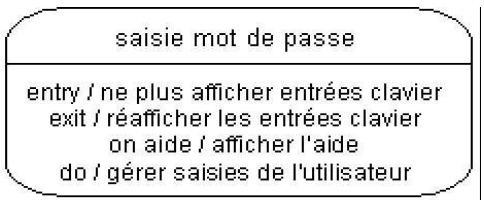
7.4) Une transition

Une transition est le passage d'un objet d'un état à un autre dû à un événement.



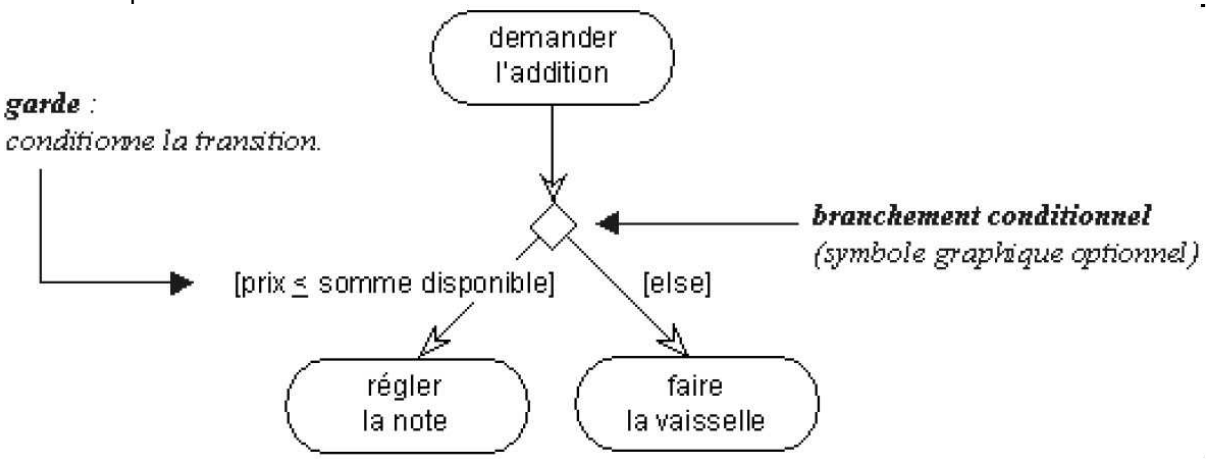
7.5) Action dans un état

On peut aussi associer une action à l'événement qui déclenche une transition.
La syntaxe est alors la suivante : événement / action
Ceci exprime que la transition (déclenchée par l'événement cité) entraîne l'exécution de l'action spécifiée sur l'objet, à l'entrée du nouvel état.
Exemple : il pleut / ouvrir parapluie
Une action correspond à une opération disponible dans l'objet dont on représente les états.
Les actions propres à un état peuvent aussi être documentées directement à l'intérieur de l'état.
UML définit un certain nombre de champs qui permettent de décrire les actions dans un état :
entry / action : action exécutée à l'entrée de l'état
exit / action : action exécutée à la sortie de l'état
on événement / action : action exécutée à chaque fois que l'événement cité survient
do / action : action récurrente ou significative, exécutée dans l'état

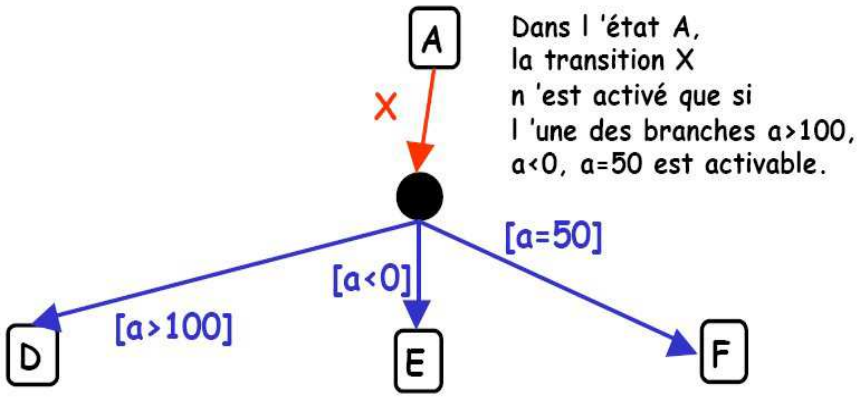


7.6) Point de décision ou branchement conditionnel

Les gardes situées après le point de décision sont évaluées au moment ou il est atteint.
Une fois le point de décision atteint au moins un chemin doit être franchissable.

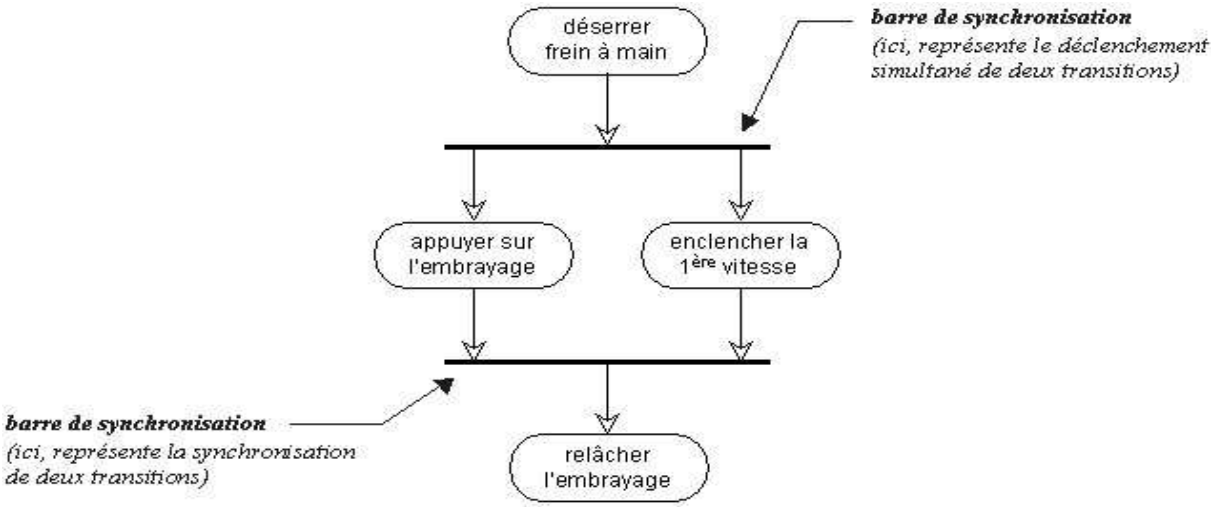


7.7) Le point de jonction Statique



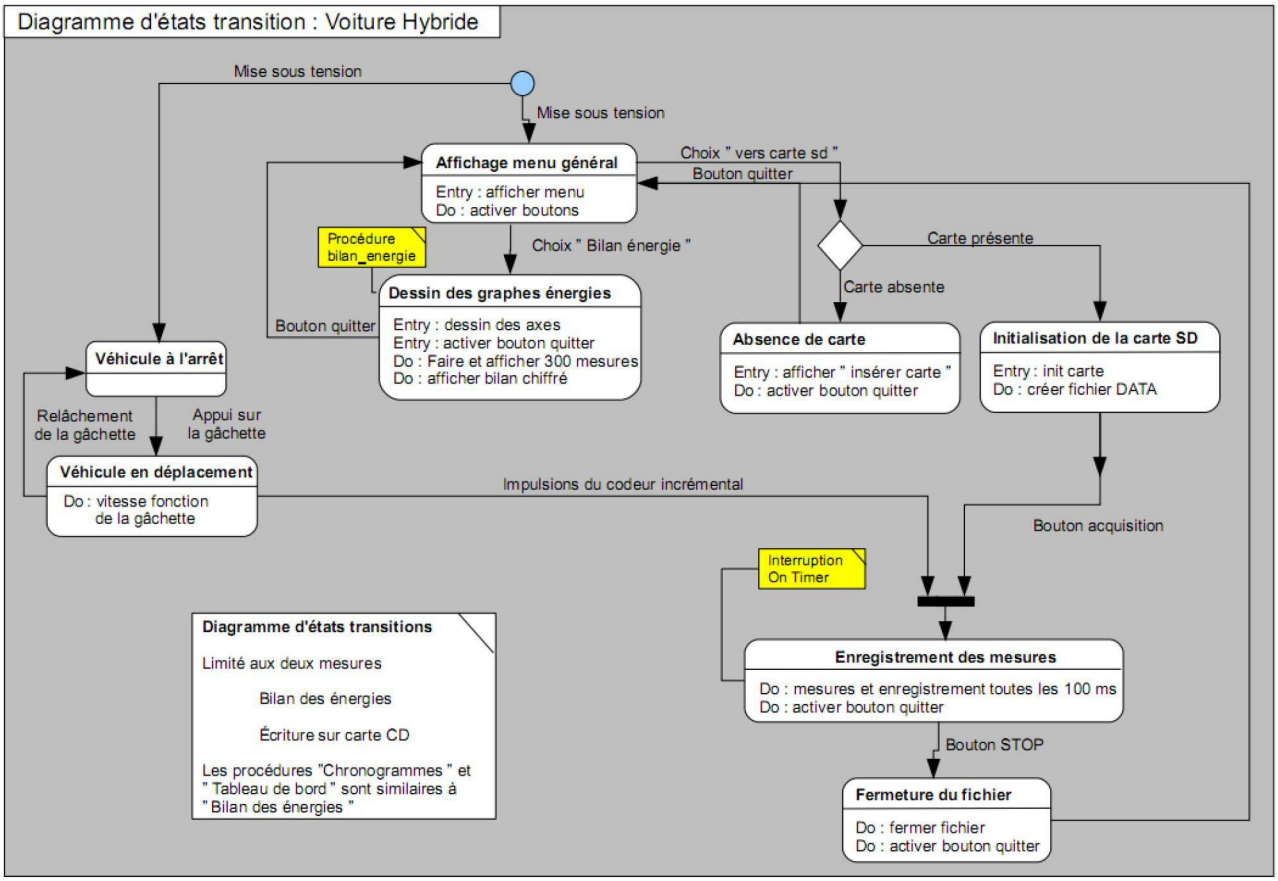
7.8) Les barres de synchronisation

Les transitions qui partent d'une barre de synchronisation ont lieu en même temps.
On ne franchit une barre de synchronisation qu'après réalisation de toutes les transitions qui s'y rattachent.



Les diagrammes d'états-transitions permettent de décrire efficacement les mécanismes concurrents grâce à l'utilisation des barres de synchronisation.

7.9) Exemple de diagramme d'état



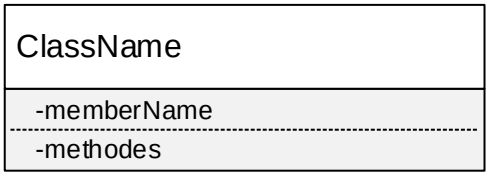
8) Diagramme de classe

8.1) Introduction

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet, il est le seul obligatoire lors d'une telle modélisation.
Alors que le diagramme de cas d'utilisation montre un système du point de vue des acteurs, le diagramme de classes en montre la structure interne. Il permet **de fournir une représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas d'utilisation**. Il est important de noter qu'un même objet peut très bien intervenir dans la réalisation de plusieurs cas d'utilisation. Les cas d'utilisation ne réalisent donc pas une partition des classes du diagramme de classes. Un diagramme de classes n'est donc pas adapté (sauf cas particulier) pour détailler, décomposer, ou illustrer la réalisation d'un cas d'utilisation particulier.
Il s'agit d'une vue statique car on ne tient pas compte du facteur temporel dans le comportement du système.
Une classe est la description formelle d'un ensemble d'objets ayant une sémantique et des propriétés communes. Un objet est une instance d'une classe. C'est une entité discrète dotée d'une identité, d'un état et d'un comportement que l'on peut invoquer. Les objets sont des éléments individuels d'un système en cours d'exécution.

8.2) Représentation graphique

Une classe est un classeur. Elle est représentée par un rectangle divisé en trois à cinq compartiments

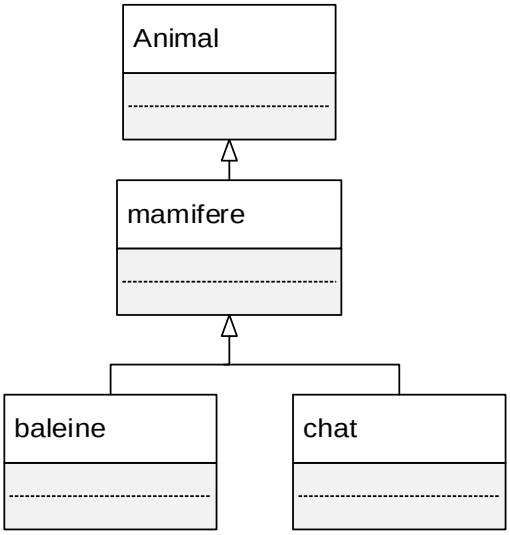


Le premier indique le nom de la classe, le deuxième ses attributs ou ses membres et le troisième ses opérations ou ses méthodes.
Il existe quatre visibilités prédéfinies.
public ou + **:** tout élément qui peut voir le conteneur peut également voir l'élément indiqué.
protected ou # : seul un élément situé dans le conteneur ou un de ses descendants peut voir l'élément indiqué.
private ou - : seul un élément situé dans le conteneur peut voir l'élément.

8.3) Relation entre classes

8.3.1) L'héritage

Dans le langage UML, ainsi que dans la plupart des langages objet, cette relation de généralisation se traduit par le concept d'héritage. On parle également de relation d'héritage. Ainsi, l'héritage permet la classification des objets.
Le symbole utilisé pour la relation d'héritage ou de généralisation est une flèche avec un trait plein dont la pointe est un triangle fermé désignant le cas le plus général.




```
Class fille : mere
{
Public:
…...
Private:
…...
};
```

8.3.2) La composition et l'agrégation

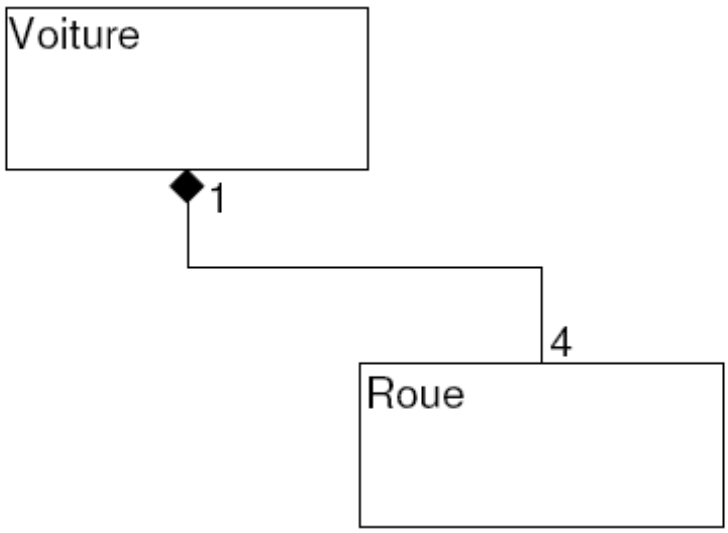
a) La composition

La composition consiste à utiliser un objet comme attribut d'un autre objet
C'est une agrégation implémentée par valeur
Par exemple, la classe *Voiture* contiendra quatre objets de la classe *Roue*:

```
Class roue
{
Public:
…...
Private:
…...
};
```

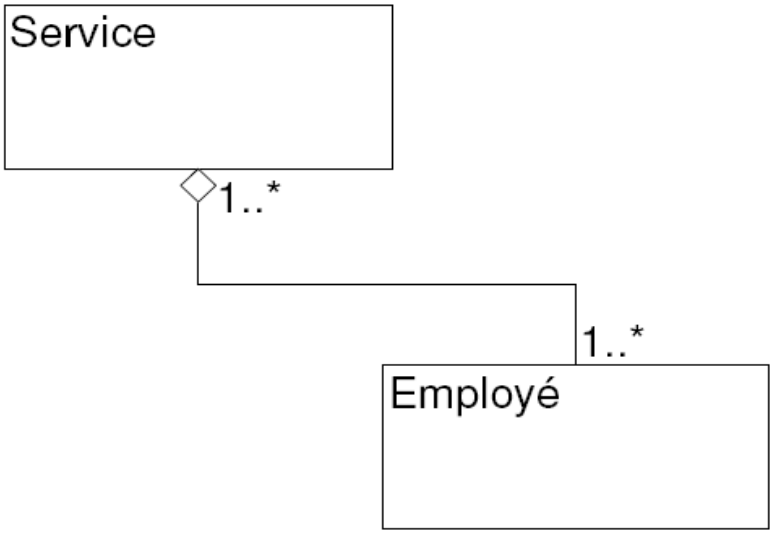
```
Class voiture
{
Public:
…...
Private:
Roue roueavantdroite
……..
};
```

On dit que A et B sont reliés par une relation de **composition**, c'est-à-dire que B est composé de A.
Il s'agit d'une relation forte: si B est détruit, A disparaît aussi



b) L'agrégation

L'agrégation **consiste essentiellement en une utilisation d'un objet comme faisant partie d'un autre objet**
- Contrairement à la composition, où l'objet inclus disparaît si l'objet englobant est détruit, dans une agrégation cet objet ne disparaît pas lorsque l'objet englobant est détruit
- La manière habituelle d'implémenter l'agrégation **en C++ est par l'utilisation de Pointeurs**



- Définition de la classe:

```
class Service
{
  ...
private:
  string nom_;
  Employe* receptionniste_;

}
```

- Constructeur:

```
Service::Service(string nom)
 : nom_(nom),
   receptionniste_(NULL)
{
}
```

- Méthode pour associer le réceptionniste:

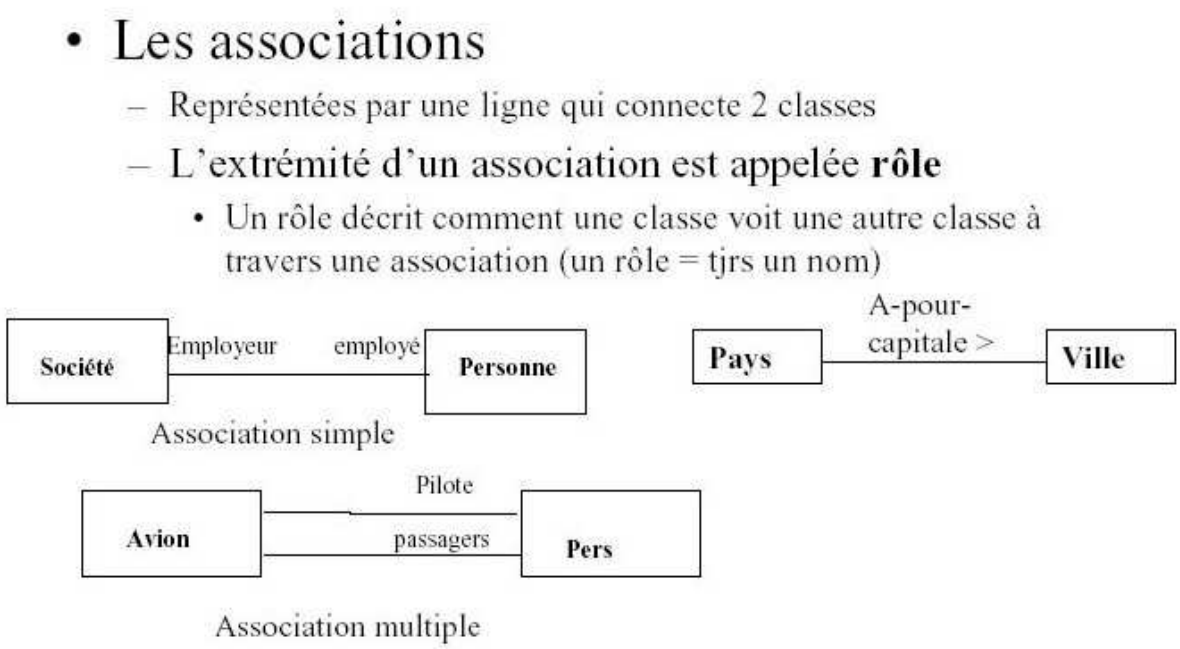
```
Service::set_receptionniste(Employe* employe)
{
  receptionniste_ = employe;
}
```

```
int main()
{
  ...
  Service expedition("Expeditions");
  Employe* michel = new Employe("Michel Gagnon", 50000.0);
  expedition.set_receptionniste(michel);
  ...
}
```

Pour qu'un objet de la classe Service soit réellement intéressant, il nous faut une méthode pour lui associer un employé comme réceptionniste. Ceci suppose qu'un objet dynamique de la classe Employé a déjà été créé auparavant et que la méthode fera pointer son pointeur sur cet objet.
Lorsqu'on fait une agrégation par référence, la référence doit absolument être initialisée lors de la création de l'objet. L'objet doit donc exister déjà lorsqu'on crée une instance de la classe englobante.

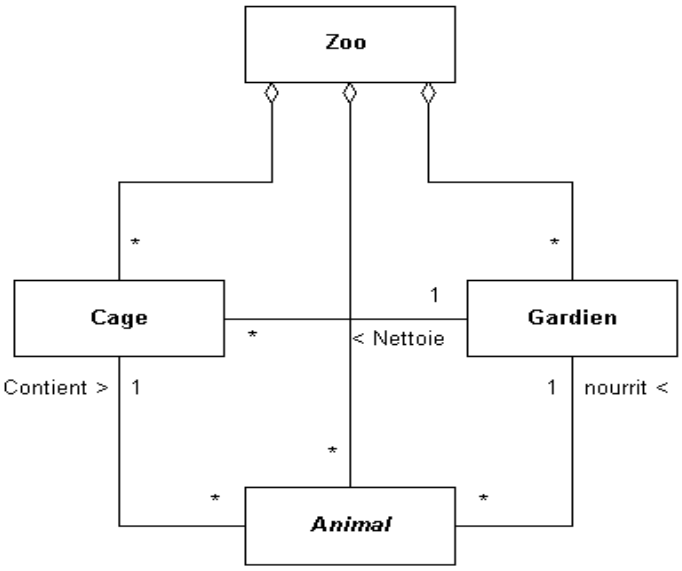
c) L'association

Une association est une relation entre deux classes (association binaire) ou plus (association n-aire), qui décrit les connexions structurelles entre leurs instances (**en général, cela se traduit en C++ par un appel de méthode**).



Considérons un exemple modélisant un Zoo. Le Zoo est composé de :

. Un ensemble de cages
. Un ensemble d'animaux
. Un ensemble de gardiens



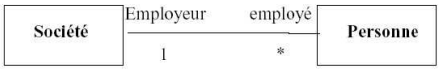
Ces relations étant, de toute évidence, de l'agrégation !
En revanche, un gardien doit s'occuper d'un certain nombre d'animaux (possédés par le Zoo) et nettoyer un certain nombre de cages (possédées par le Zoo). De la même manière, une cage regroupe certains animaux (possédés par le Zoo).

Ces dernières relations ne sont pas de l'agrégation (on considère habituellement qu'un même objet n'est pas agréable par plusieurs) mais plutôt des Associations. En fait, on rangera dans l'association tout type de relation un peu flou qui ne sera ni de l'agrégation, ni de l'héritage. La composition ou l'agrégation sont des cas particuliers de l'association
A l'instar de l'agrégation, on définit des cardinalités sur la relation d'association ainsi que des rôles. Par exemple, si nous considérons la relation entre les classes Cage et Gardien, nous pouvons lire :
"Un Gardien nettoie de 0 à n Cages / Une cage est nettoyée par 1 et un seul gardien".

8.4) La cardinalité

- Les associations
  - La **multiplicité** d'une association montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe

1	un et un seul
0..1	0 ou un
M..N	De M à N (entiers naturels)
* 0..*	De 0 à plusieurs
1..*	De 1 à plusieurs

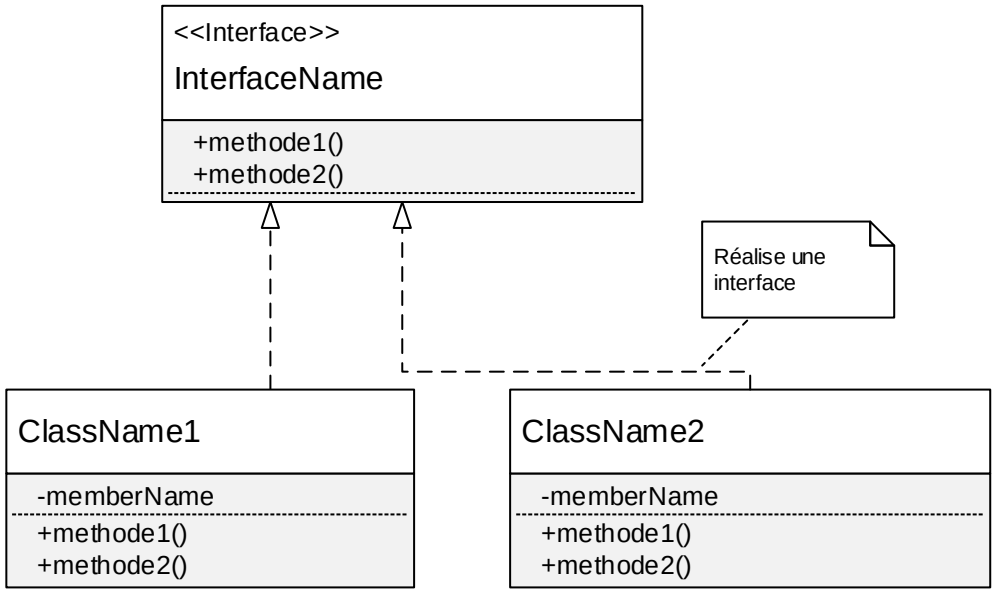


8.5) Choix entre association et agrégation

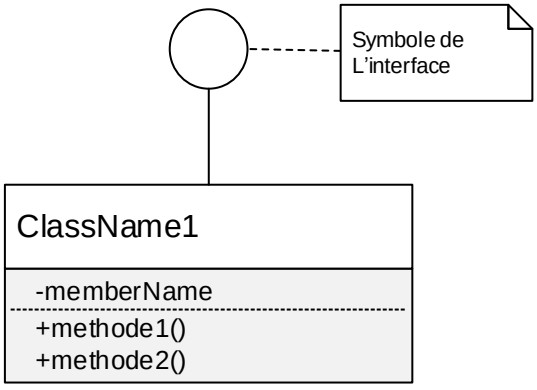
Il est parfois difficile de choisir entre association classique et agrégation. Pour cela, il faut s'interroger sur le cycle de vie des objets concernés.
Voici quelques questions qui vous permettront peut-être de trouver une réponse :
Les objets concernés sont-ils indépendants les uns des autres ? C'est à dire, la mort de l'un entraine-t-il la mort d'autres ? Si NON alors ASSOCIATION.
Y a-t-il des méthodes d'une classe qui s'appliqueraient aux autres ? Si oui, cette classe est le composé et sera reliée aux autres par une AGREGATION
Demandez-vous si cela paraît naturel de dire mon objet TITI est une partie de mon autre objet TOTO ? Si oui, TOTO est composé de TITI (composant), et nous avons là une AGREGATION.

8.6) Les interfaces

Une interface est une classe totalement abstraite, c'est-à-dire sans attribut et dont toutes les méthodes sont abstraites et publiques. Une telle classe ne contient aucun élément d'implantation des méthodes.
Graphiquement, elle est représentée comme une classe avec le stéréotype "interface». L'implantation des méthodes est réalisée par une ou plusieurs classes concrètes, sous-classes de l 'interface.
Dans ce cas, la relation d'héritage qui existe entre l'interface et une sous-classe d'implantation est appelée relation de réalisation. Graphiquement, elle est représentée par un trait pointille au lieu d'un trait plein pour une relation d'héritage entre deux classes.
La figure ci-dessous illustre cette représentation



On peut simplifier la représentation d'une interface par le schéma suivant sans préciser le nom de l'interface



Une classe peut être liée à plusieurs interfaces (héritage multiple)

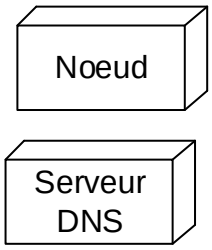
9) Le diagramme de déploiement

Le **diagramme de déploiement** permet de représenter l'architecture physique supportant l'exploitation du système. Cette architecture comprend des noeuds correspondant aux supports physiques (serveurs, routeurs…) ainsi que la répartition des composants logiciels (bibliothèques, exécutables, fichier…) appelés artefacts sur ces noeuds. C'est un véritable réseau constitué de noeuds et de connexions entre ces noeuds qui modélise cette architecture.

9.1) Noeud

Un **noeud** correspond à une ressource matérielle de traitement sur laquelle des artefacts seront mis en oeuvre pour l'exploitation du système. Les noeuds peuvent être interconnectés pour former un réseau d'éléments physiques.

Un noeud ou une instance de noeud se représente par un cube ou parallélépipède.

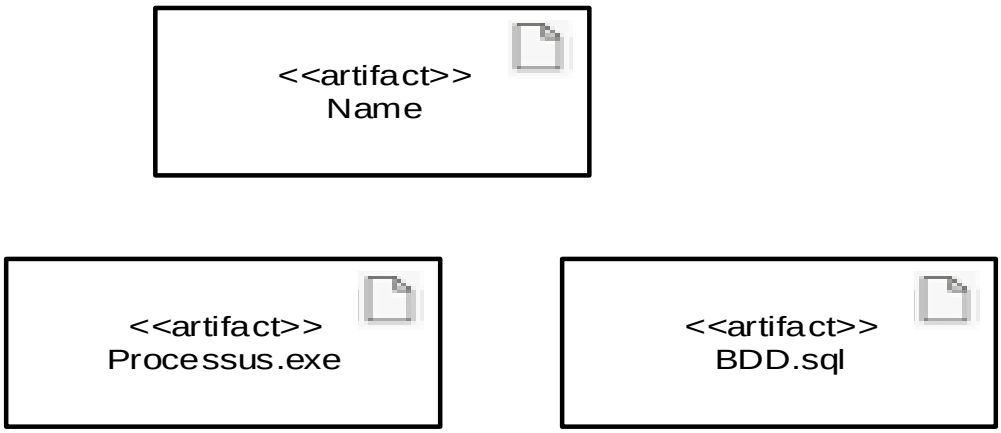


9.2) Artefact

Un **artefact** est la spécification d'un élément physique qui est utilisé ou produit par le processus de développement du logiciel ou par le déploiement du système. C'est donc un élément concret comme par exemple : un fichier, un exécutable ou une table d'une base de données.

Un artefact peut être relié à d'autres artefacts par notamment des liens de dépendance.

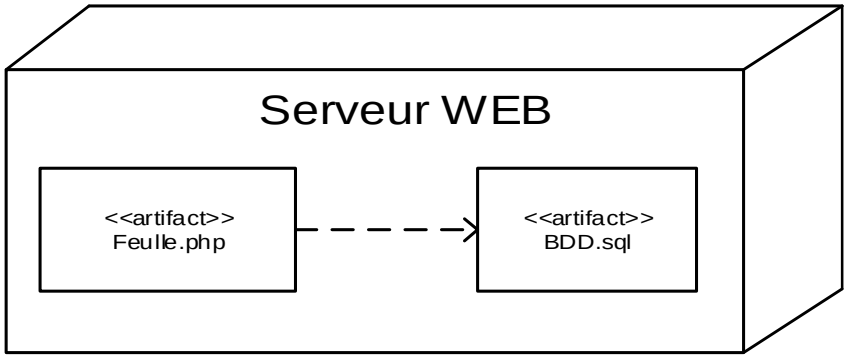
Un artefact se représente par un rectangle caractérisé par le mot-clé « artifact » et/ou une icône particulière dans le coin droit du rectangle.



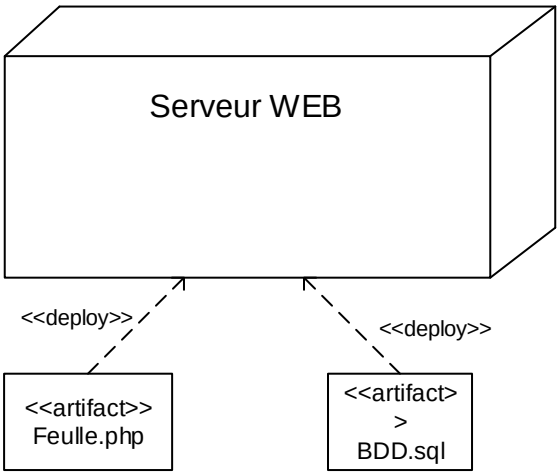
9.3) Liens entre un artefact et les autres éléments du diagramme

Il existe deux manières de représenter le lien entre un artefact et son nœud d'appartenance :

- **Représentation inclusive** – Dans cette représentation, un artefact est représenté à l'intérieur du noeud auquel il se situe physiquement.

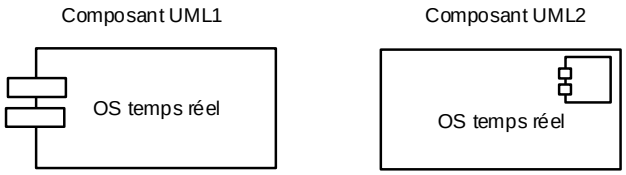


- **Représentation avec un lien de dépendance typé « deploy »** – Dans ce cas l'artefact est représenté à l'extérieur du noeud auquel il appartient avec un lien de dépendance entre l'artefact et le noeud typé avec le mot-clé « deploy ».



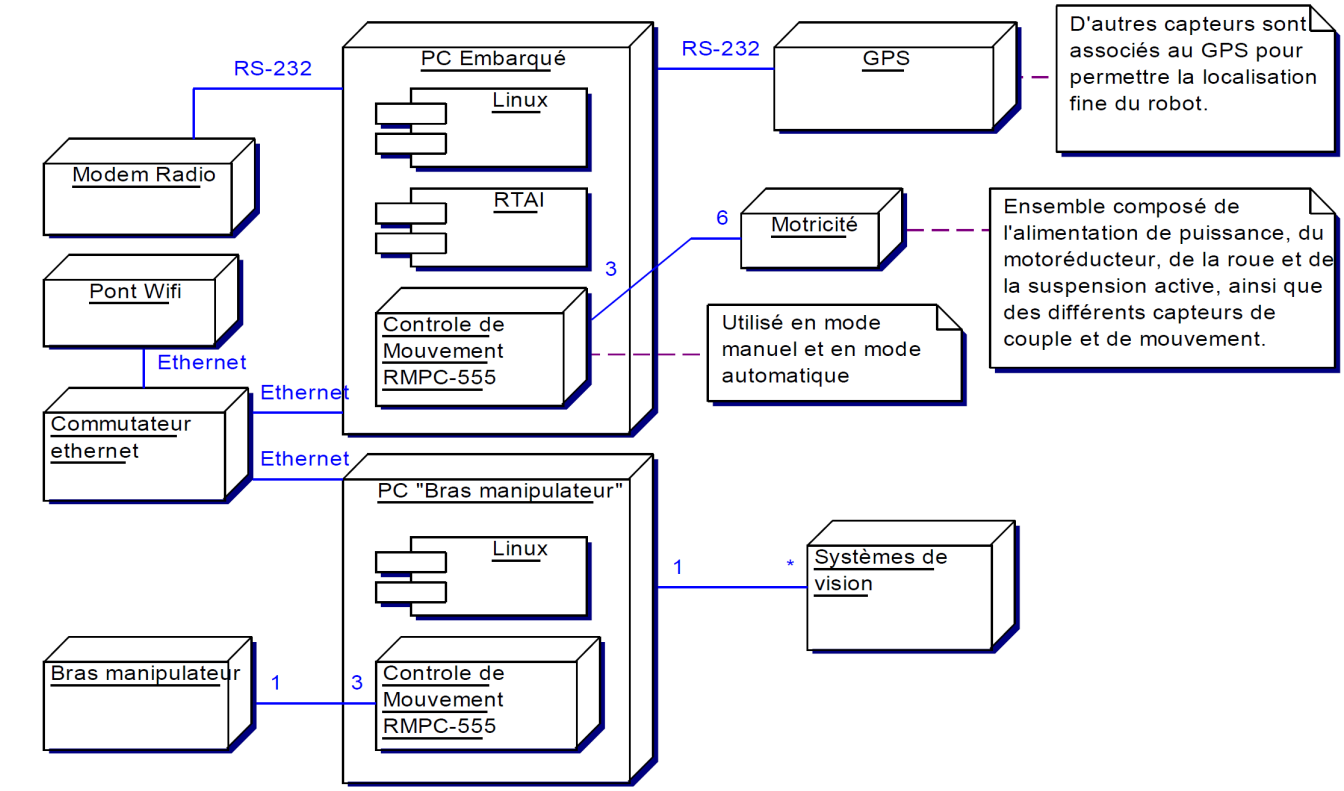
9.4) Le composant

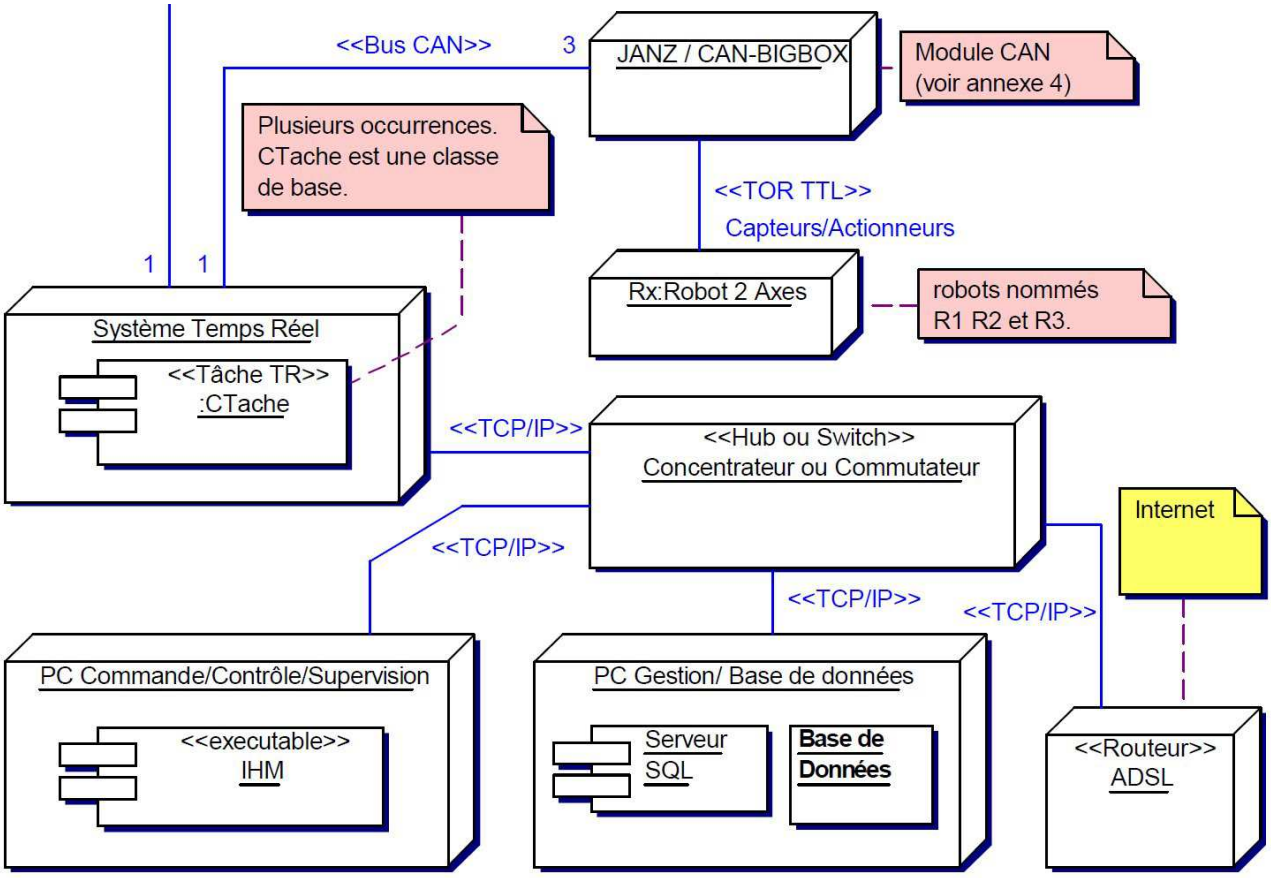
Un composant est une unité logicielle offrant des services au travers d'une ou de plusieurs interfaces. C'est une boite noire dont le contenu n'intéresse pas les clients. Un composant n'est pas forcément une classe. Une interface au sein d'un composant peut être implémentée à l'aide de langage de programmation non objets (langage C, assembleur). On peut regrouper plusieurs technologies de composants (OLE, COM, CORBA). Un composant qui utilise ces technologies bénéficie d'un standard, il est disponible sur étagère.



On peut retrouver le symbole du composant sur le diagramme de déploiement

9.5) Exemples de diagramme de déploiement

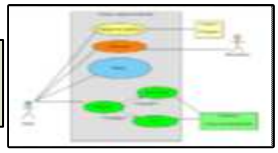




10) <u>Une démarche SYSML pour la gestion de projet</u>

Définir les cas d'utilisation de notre système ou sous-système

1



Pour chaque cas d'utilisation élaborer un diagramme d'exigence (fonction contraintes….)
établir sur les diagrammes d'exigence les blocs d'objets ou les solutions techniques

2



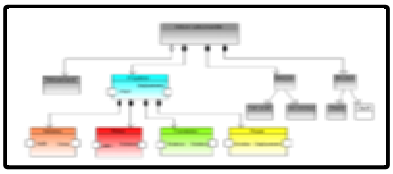
Elaborer si nécessaire un diagramme de séquence ou d'états avec le système (voir les objets) et les acteurs

3



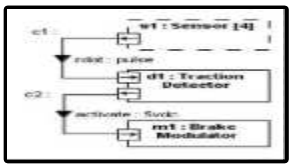
faire un ou des diagrammes de définition de bloc pour notre système ou sous système

4



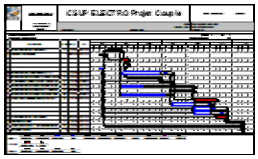
A partir du ou des diagrammes de définition de bloc établir les diagrammes de bloc interne (flux, ports ….)

5



A partir des diagrammes de bloc internes établir les différentes taches et le diagramme de Gantt

6



Conception, développement, test, intégration

VHDL
UML
Algorithmique
Programmation temps réél
Programmation orientée objet
schémas structurels
typon

<u>Le chapitre suivant va nous permettre de mettre en place cette démarche SYSML sur un sous-système baptisé simulateur de torpille</u>

La démarche SysML

Exemple de décomposition :

11) Conclusion sur le projet

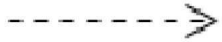
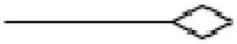
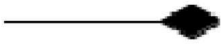
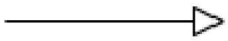
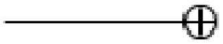
Clore le projet

Dans une entreprise cette phase est très importante, elle rentre dans la logique de capitalisation des expériences. Elle consiste à:

- •Rassembler tous les documents techniques et organisationnels relatifs au projet.
- •Analyser les « + » et les « - » du projet en terme de tenue des objectifs techniques.
- •Analyser les « + » et les « - » du projet en terme d'organisation.
- •Construire un plan d'amélioration pour la gestion des futurs projets.

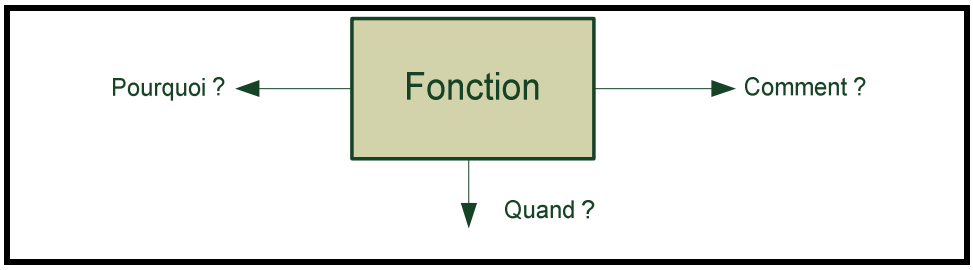
- Le bilan du projet.

Annexe les différentes relations

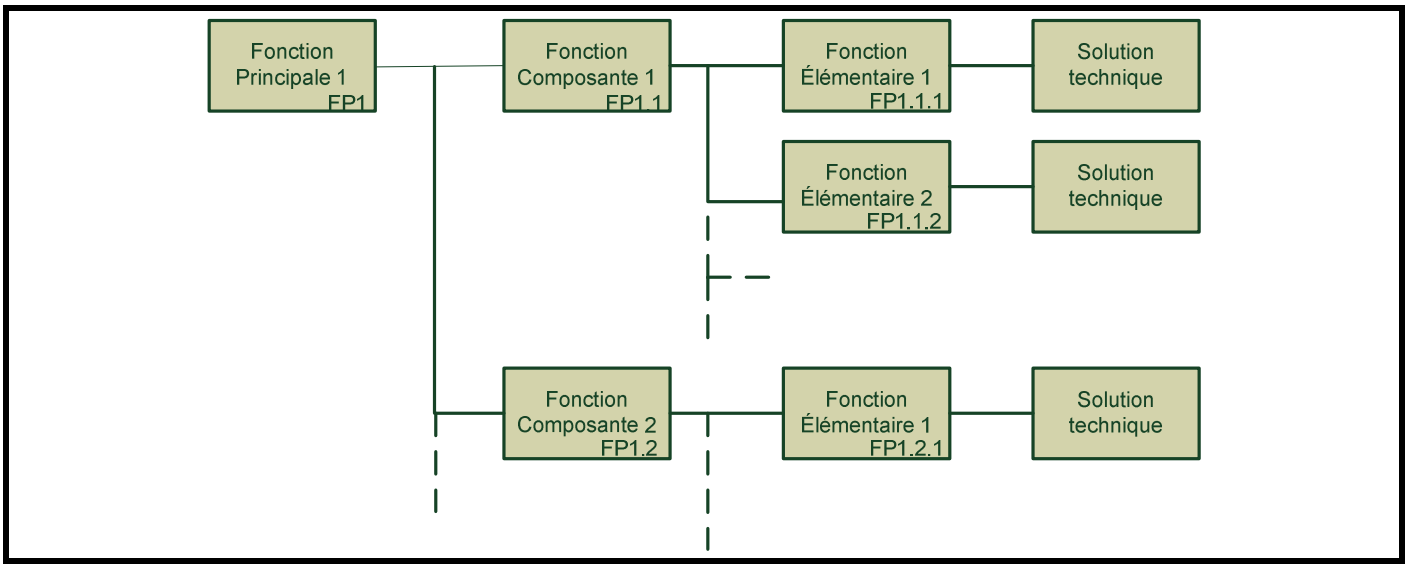
	**Association** : relation d'égal à égal entre 2 éléments ➢ A utilise B ➢ 2 diagrammes : cas d'utilisation, blocs
	**Dépendance** : 2 items indépendants mais dont l'un dépend de l'autre. ➢ A dépend de B ➢ 3 diagrammes : exigences, cas d'utilisation, blocs,
	**Agrégation** : un élément est une composante facultative d'un autre. ➢ A entre dans la composition de B (sans être indispensable) ➢ 2 diagrammes : exigences, blocs
	**Composition** : un élément est une composante obligatoire d'un autre. ➢ A entre dans la composition de B et lui est indispensable ➢ 2 diagrammes : exigences, blocs
	**Généralisation** : dépendance de type filiation entre 2 items ➢ A est une sorte de B ➢ 2 diagrammes : cas d'utilisation, blocs
	**Conteneur** : relation d'inclusion entre 2 items ➢ B contient A ➢ 3 diagrammes : exigences, cas d'utilisation, blocs,

Le diagramme FAST (Function analysis system technic)

Il se présente sous forme d'un arbre de fonctions partant de la fonction globale ou d'une fonction de service. Sa lecture est fondée sur une technique interrogative : Pourquoi cette fonction doit elle être assurée ? comment ? quand ?
Une représentation FAST peut se limiter à décrire la décomposition des fonctions (décomposition fonctionnelle) ou déboucher sur les solutions détaillées



Exemple de diagramme FAST :



Ce diagramme peut être remplacé par le diagramme d'exigences en SysML