Que va nous apporter le SysML?

Les 8 graphes SysML

Le diagramme de cas d'utilisations

Montrer les interactions entre les acteurs extérieurs au système et les fonctionnalités du système

Diagramme de cas d'utilisation seul

Le recyclage ne se justifie ici que dans le cas d'un travail sur l'eco-conception

Le diagramme SysML de séquence

Décrire chronologiquement les échanges de messages au sein d'un système

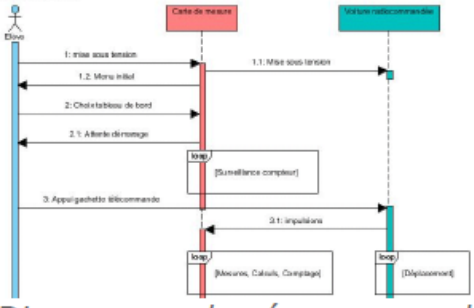


Diagramme de séquence seul

Le diagramme SysML d'états-transitions

Décrire le comportement interne d'un système par les changements d'états

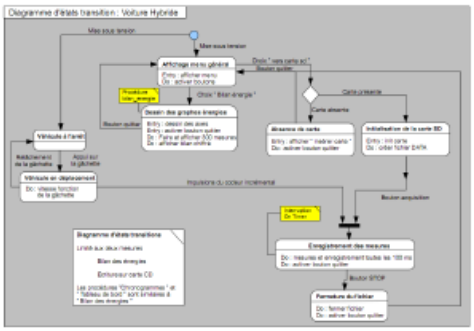


Diagramme d'états transitions de la Tamiya H-Cell

Le diagramme SysML d'activité

Décrire l'enchainement des actions pour une activité du système

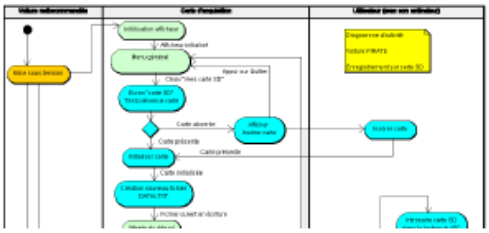


Diagramme d'activité seul

Le diagramme SysML de définition de bloc

Représenter un système sous forme de blocs hiérarchisés

Diagramme de définition de bloc seul

Le diagramme SysML de bloc interne

Montrer les liens entre les "ports" des différents blocs du système

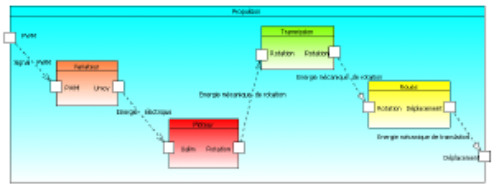


Diagramme de bloc interne seul

Le diagramme SysML des exigences

Montrer graphiquement les exigences auxquelles doit satisfaire un système

Diagramme des exigences seul

Le diagramme SysML paramétrique

Modéliser les différents blocs d'un système

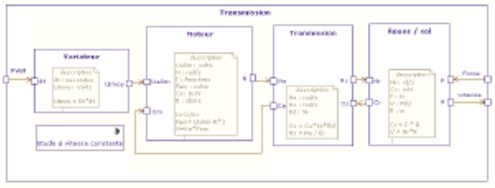


Diagramme paramétrique seul

Ces diagrammes ne sont pas indépendants et permettent d'associer les éléments de diagrammes différents. C'est l'un des points forts de ce type de langage. Il est ainsi possible de conserver la traçabilité des éléments dans les différents diagrammes, par exemple :

- lier une exigence avec des blocs pour établir le lien fonctions – solutions ;

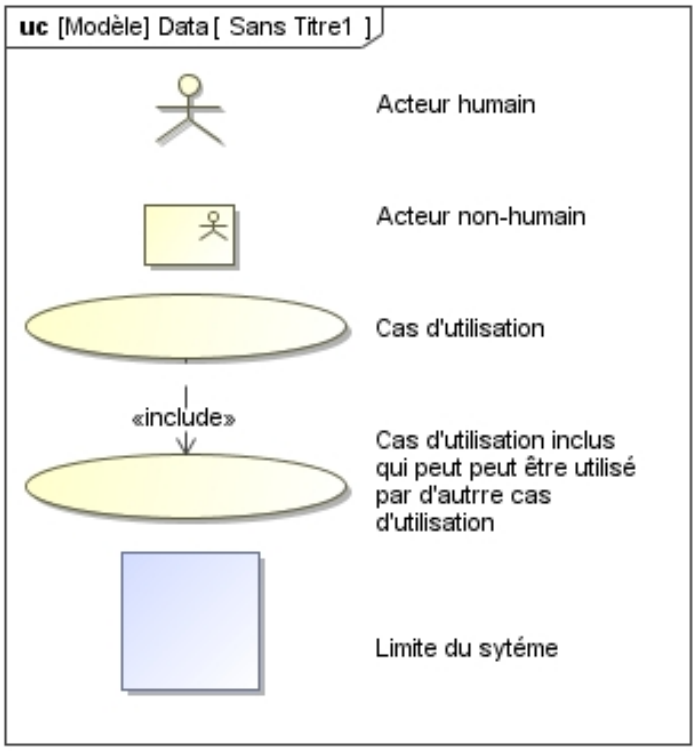
- lier des états avec les blocs pour établir le lien entre les actions et les composants qui les réalisent
- lier les cas d'utilisation avec les scénarii des diagrammes de séquences.

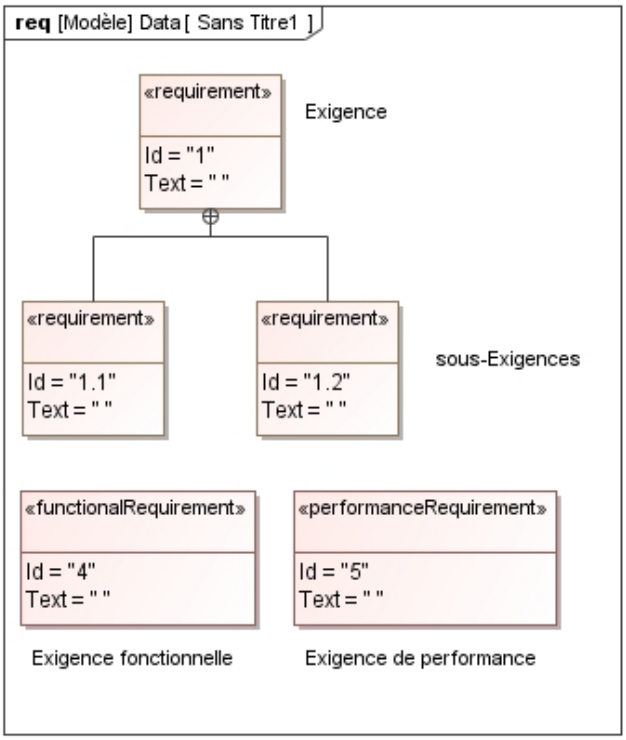
La syntaxe présentée dans le tableau suivant est nécessaire et suffisante pour aborder la description des constructions et des systèmes mécatroniques. La richesse et la finesse du langage SysML permettent une description d'une grande précision, mais peuvent également amener à des diagrammes très complexes **qui ne correspondent pas aux objectifs de l'enseignement STI2D.**

1) Diagrammes Fonctionnelles

Utilisation et acteurs extérieurs
Diagramme « Cas d'utilisation » :
(uc : use case diagram)

Spécifications du cahier des charges
Diagramme des exigences :
(**req: requirement diagram**)

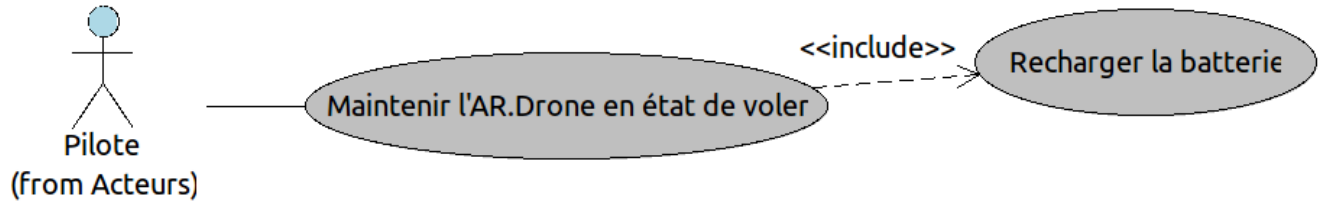




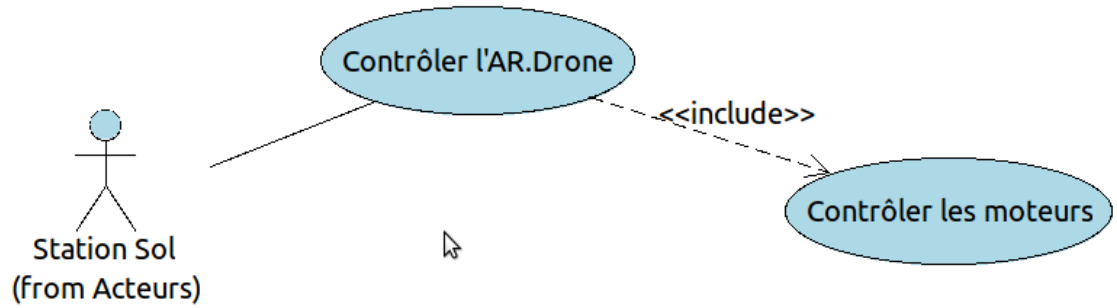
1.1) Diagramme de cas d'utilisation

Relations entre cas d'utilisation : l'inclusion

Pour une meilleure compréhension du système on est amené à décomposer un cas d'utilisation complexe en plusieurs cas simple, pour cela on utilise le stéréotype « include ». Il permet de signaler la relation entre les deux cas.



L'inclusion a un caractère obligatoire : Le cas « Maintenir l'AR.Drone en état de voler » inclut forcément le cas « Recharger la batterie ».



La Station-Sol envoie des commandes pour contrôler l'AR.Drone. Les commandes comme « Décollage », « Atterrissage », « Prendre de l'altitude », auront une influence directe sur la vitesse de rotation des hélices (le contrôle de l'AR.Drone passe par le contrôle individuel de la vitesse de rotation de ses quatre hélices entraînées par des moteurs brushless). On peut donc conclure que le cas « Contrôler l'AR.Drone » inclut le cas « Contrôler les moteurs ».

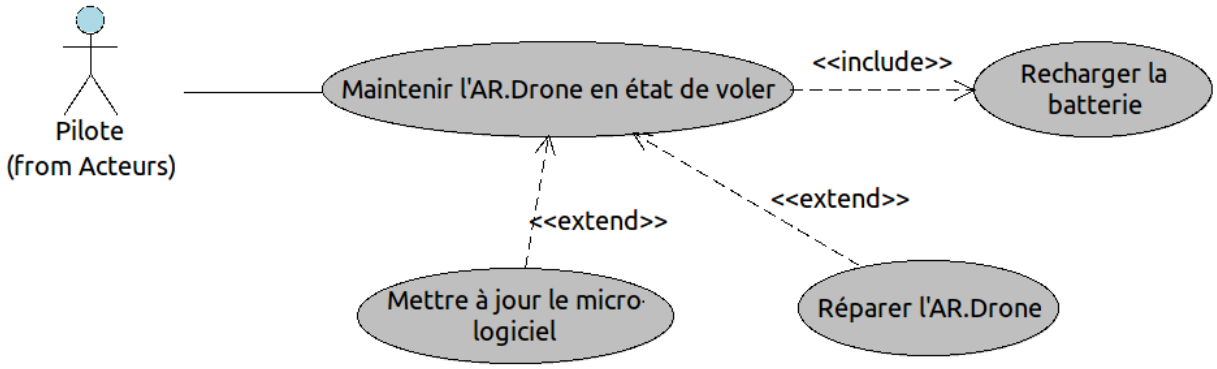
1.2) Relations entre cas d'utilisation : l'extension

On utilisera également la relation d'extension qui permet de signaler que, lors du déroulement d'un cas d'utilisation, un autre cas peut de se déclencher sous condition : c'est une extension possible du cas.

Pour ce type de relation, on utilise le stéréotype <<extend>>.

Contrairement à la relation d'inclusion qui a un caractère obligatoire, la relation d'extension signale que l'exécution du cas étendu est optionnelle.

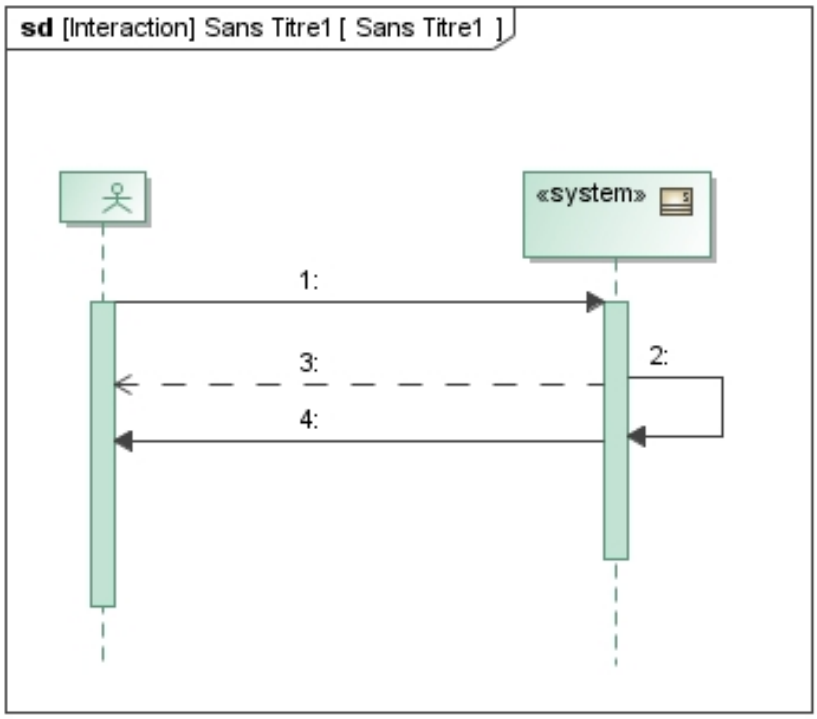
En reprenant l'exemple du cas « Maintenir l'AR.Drone en état de voler », on peut enrichir ce cas avec le cas « Réparer l'AR.Drone » qui se déroulera si l'AR.Drone chute et se casse, ainsi qu'avec le cas « Mettre à jour le micro-logiciel » si la Société Parrot décide de faire évoluer le logiciel interne de l'AR.Drone (correction de bug, ajout de nouvelles fonctionnalités, etc.).

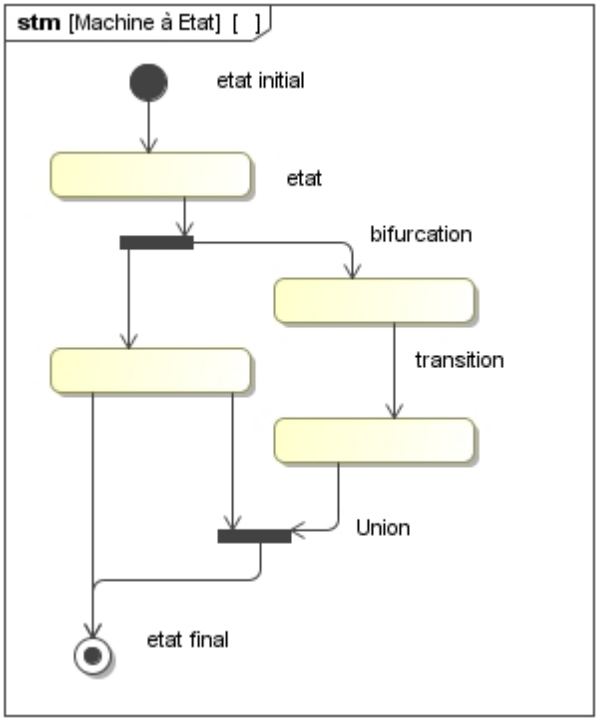


2) Diagrammes comportementaux

Diagramme de séquences : (sd : sequence diagram)
L' objectif est de décrire des scénario d'interactions

Diagramme d'états :
(stm : state machine diagram)
L'objectif est de décrire un comportement séquentiel



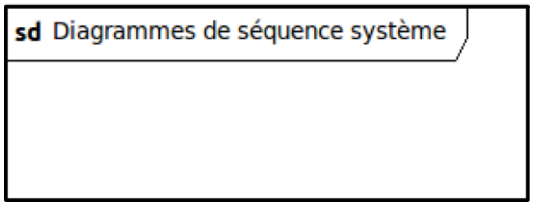
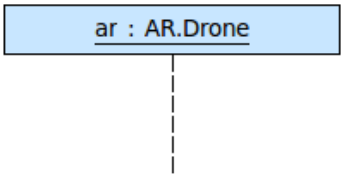
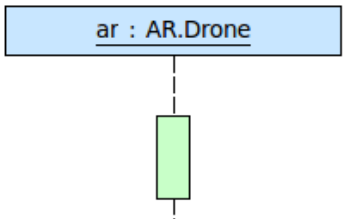


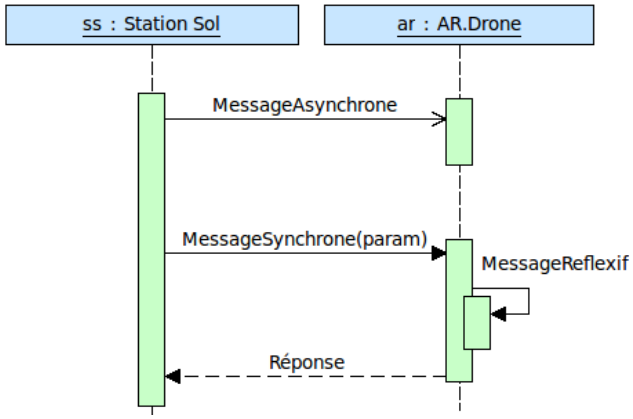
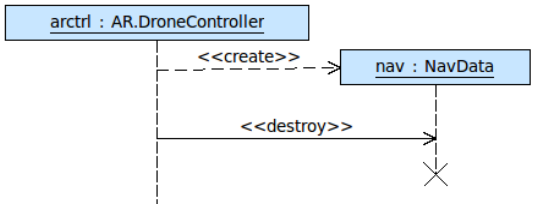
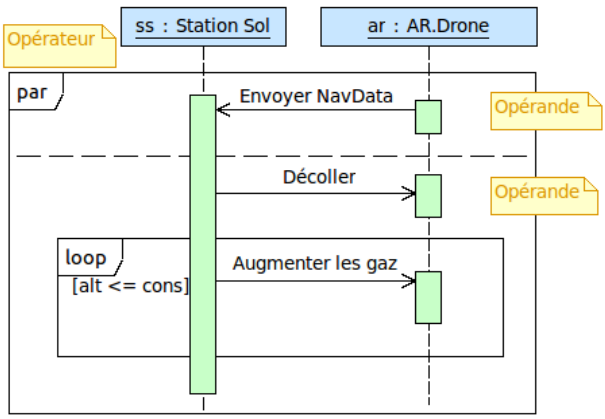
2.1) Diagramme de séquence

Le diagramme de séquence décrit les flots de contrôle entre acteurs et le système (point de vue « système ») ou entre les blocs qui composent le système.

Ce diagramme présente les messages émis ou reçus entre les éléments en interaction et où le temps est représenté le long de l'axe vertical (ligne de vie). Le diagramme de séquence peut représenter des situations complexes en utilisant des fragments combinés (boucle, option, etc.) et en décomposant une ligne de vie d'un bloc complexe en plusieurs lignes de vie de ses constituants.

Les principaux éléments graphiques utilisés dans les diagrammes de séquence :

Diagramme de séquence		Diagramme décrivant les interactions entre les instances de blocs ou de classes afin d'illustrer un scénario d 'un cas d'utilisation.
Ligne de vie		Une ligne de vie est une instance d'un bloc ou d'une classe participant à une interaction. Ex. : • ar : instance • AR.Drone : bloc Le temps se déroule vers le bas.
Activation		Permet de visualiser les moments pendant lesquels la ligne de vie est activée suite à l'émission ou la réception de messages.

<table>
<tr>
<td>Message</td>
<td> 
</td>
<td>Les messages représentent des communications entre instances (lignes de vie) : envoie d'un signal ou invocation d'une opération dont on attend un résultat.
Les messages sont donc :
• asynchrones (simple flèche) : l'émetteur n'attend pas la réponse du récepteur pour continuer sa tâche (métaphores : l'envoi d'un SMS, interruptions matérielles, etc.)
• synchrone (flèche pleine) : l'émetteur est bloqué le temps de l'exécution du traitement du message jusqu'à la réception d'une réponse (métaphores : un appel téléphonique). En programmation objet, les appels aux méthodes sont synchrones.
Les messages réflexifs permettre de modéliser les traitements internes.
On retrouve souvent en programmation, des messages qui permettent la création et la destruction d'instance de classe.</td>
</tr>
<tr>
<td></td>
<td> 
</td>
<td></td>
</tr>
<tr>
<td>Fragment combiné</td>
<td> 
</td>
<td>Les fragments combinés permettent de spécifier des interactions complexes.
Un fragment possède un opérateur d'interaction et un ou plusieurs opérandes .
Le choix entre plusieurs opérandes se fait à l'aide d'expressions booléennes entre crochet [ ] : les conditions de garde.
Les principaux opérateurs d'interaction utilisés sont :
• opt (option) : s'exécute si la condition de garde associée est vraie.
• alt (alternative) : l'opérande possédant la condition de garde vraie s'exécute.
• loop : l'exécute se fait en boucle selon la condition de garde.
• par (parallel) : les opérandes séparés par une ligne pointillées s'exécutent en parallèle.</td>
</tr>
</table>

Exemple :

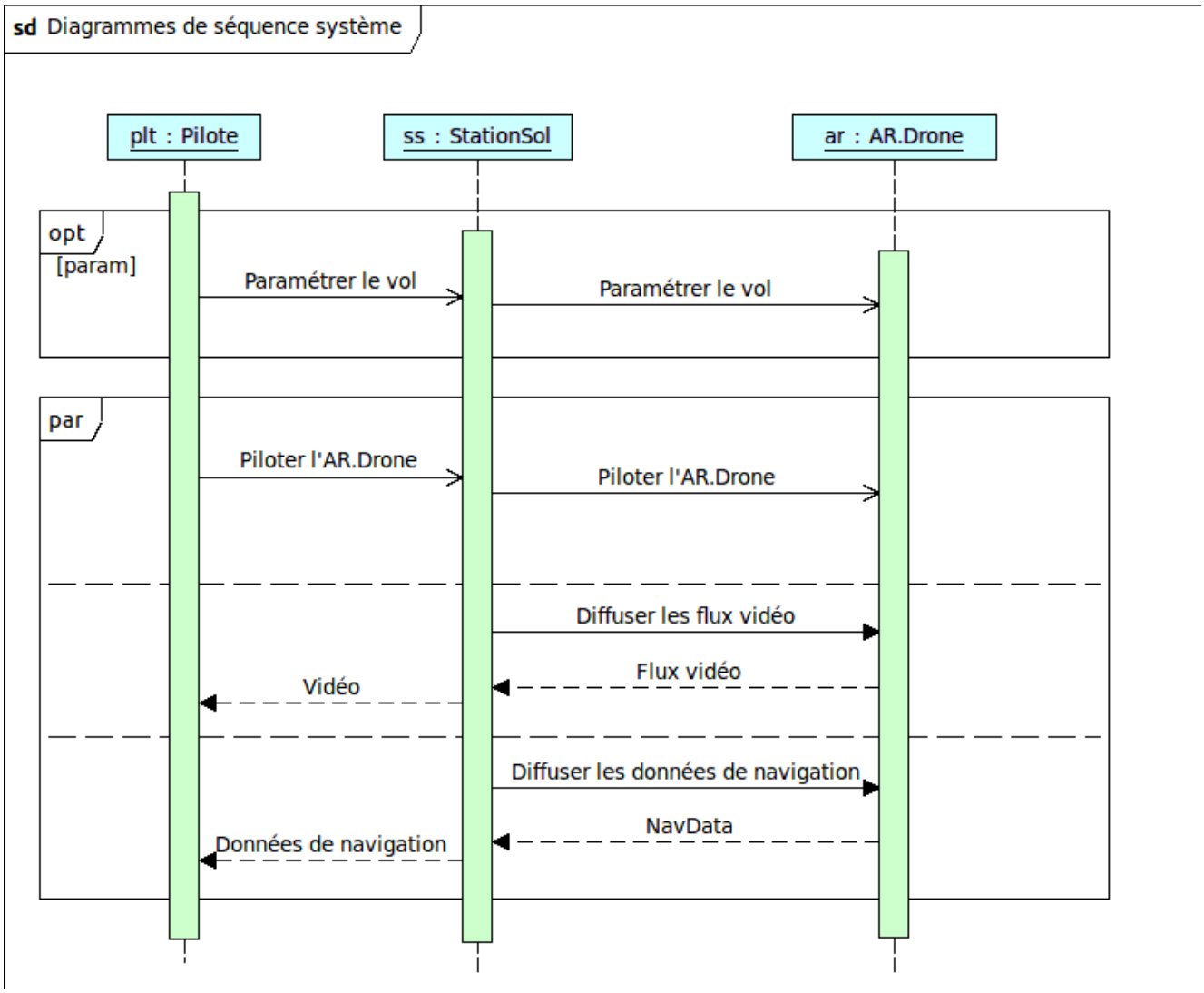


Diagramme de séquence système

Le diagramme de séquence montre que le pilote peut régler quelques paramètres de vol avant le décollage (fragment combiné **opt**), puis plusieurs fonctions s'exécutent en parallèle (fragment combiné **par**) :

- La Station-Sol affiche le flux vidéo des caméras de l'AR.Drone.
- Quelques informations issues des données de navigation reçues par la Station-Sol sont affichées (état de la batterie, état des moteurs, etc.)

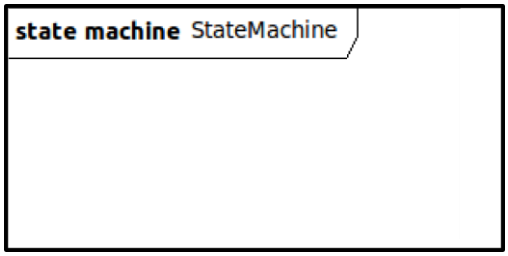
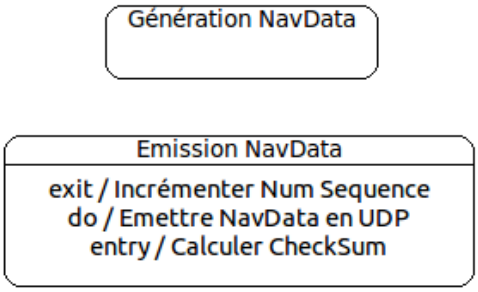
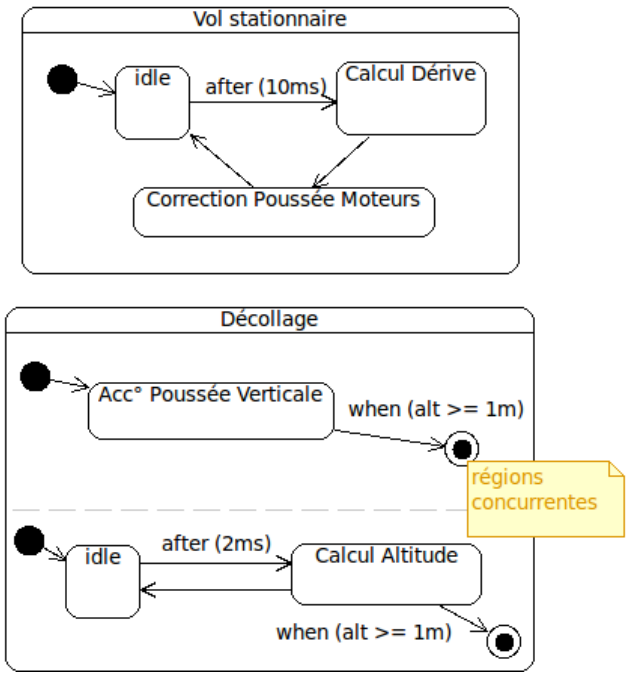
Le Pilote peut piloter l'AR.Drone grâce aux commandes disponibles sur la Station-Sol.

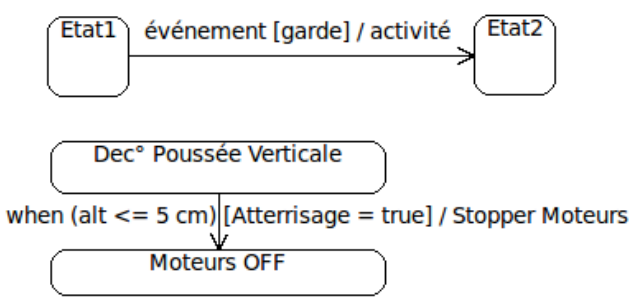
2.2) Diagrammes d'états

Certains comportements complexes d'un système sont modélisables au travers d'objets en termes d'états et de transitions. Il permet de voir l'évolution séquentielle du système et les activités déclenchées par une transition.
Les principaux éléments graphiques utilisés dans les diagrammes d'états :

Les principaux éléments graphiques utilisés dans les diagrammes d'états :

Les principaux éléments graphiques utilisés dans les diagrammes d'états :

Diagramme d'états		Diagramme décrivant les comportements internes d'une instance de blocs, de classes, d'un cas d 'utilisation ou d'une méthode de classe.
État simple		Un état représente la période de la vie d'un objet ou d'un composant pendant laquelle il accomplit une activité ou il attend un événement. On peut spécifier les activités déclenchées à l'entrée (**entry**), à la sortie (**exit**) de l'état. On peut aussi décrire l'activité principale de l'état (**do**).
État composite		Un état composite permet de modéliser un comportement interne d'un état par un ou plusieurs autres automates à états finis. Dans le cas ou plusieurs **régions** sont définies dans un état composite, les automates de chaque région sont concurrents (exécutés en parallèle).
État initial		État de départ (pour une classe, création d'une instance).
État final		Fin d'une machine à état (pour une classe, destruction d'une instance).

Transition		Une transition permet le changement d'état de la machine à états. Une transition peut être : • Automatique (flèche sans texte) : la fin de l'activité d'un état entraîne le passage à l'état suivant. • Sur événement (when, after, etc.) : l'occurrence de l'événement entraîne le passage à l'état suivant. • Sur événement gardé (ev [condition de garde]) : l'occurrence de l'événement ET la condition de garde VRAIE entraîne le passage à l'état suivant. Une activité peut être déclenchée lors d'une transition.

Exemple :

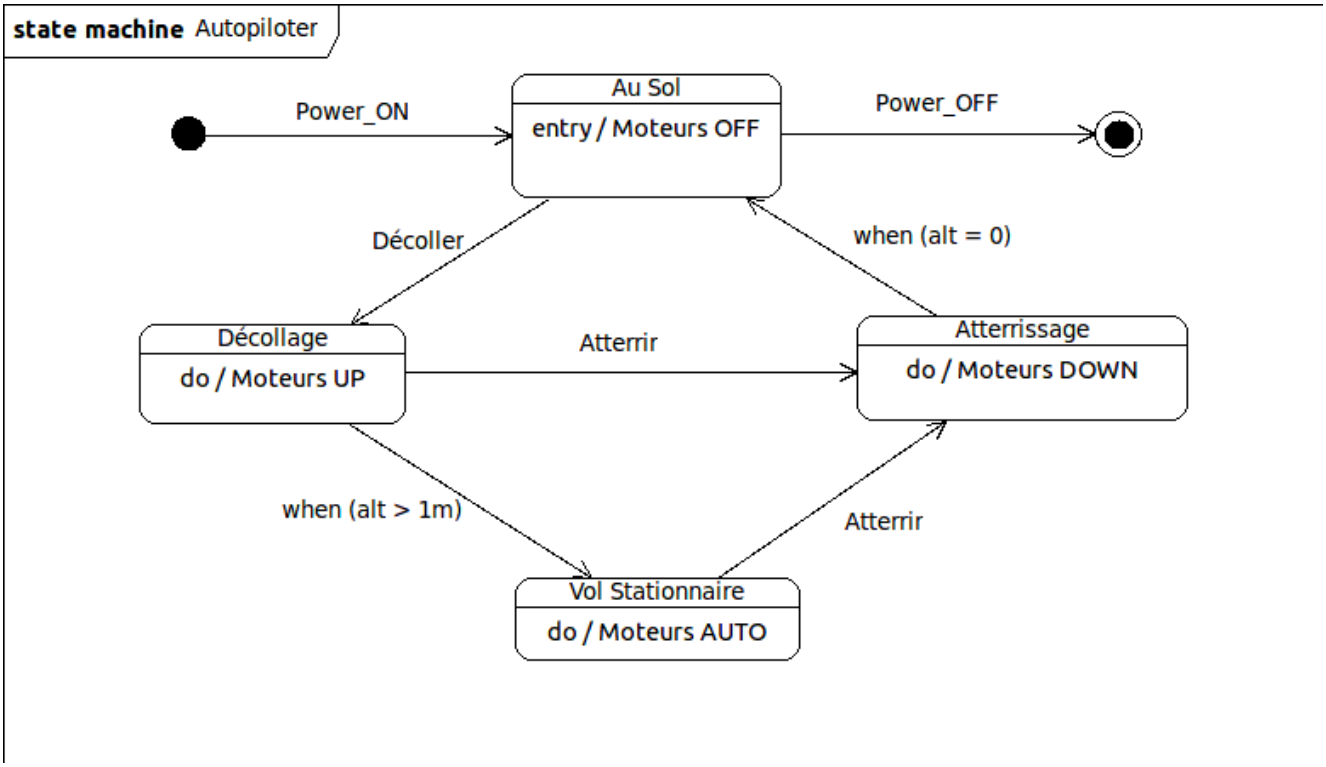


Fig. Diagramme d'états du CU "Autopiloter"

3) DIAGRAMME DES EXIGENCES

Une exigence indique des caractéristiques où des conditions qui doivent être satisfaites. Elle définit le niveau de performance qu'une fonctionnalité doit atteindre (sécurité, fiabilité, fidélité…..) . Ce diagramme permet de hiérarchiser et de décrire les exigences sous forme graphiques.

Les principaux éléments graphiques utilisés dans les diagrammes d'exigences :

Diagramme d'exigences

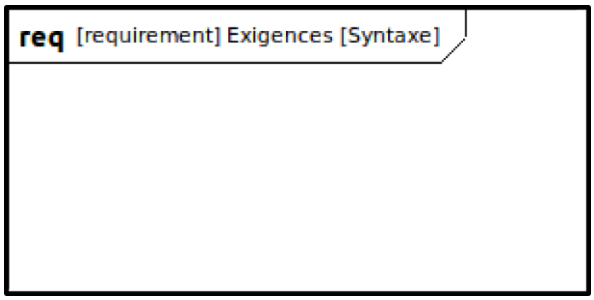
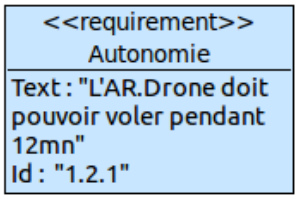


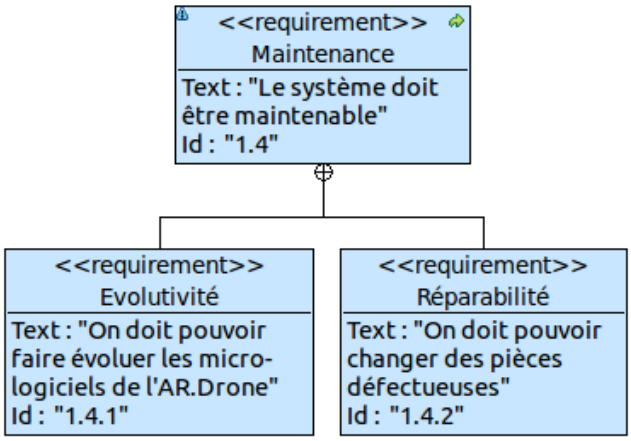
Diagramme décrivant les exigences d'un système.

Exigence



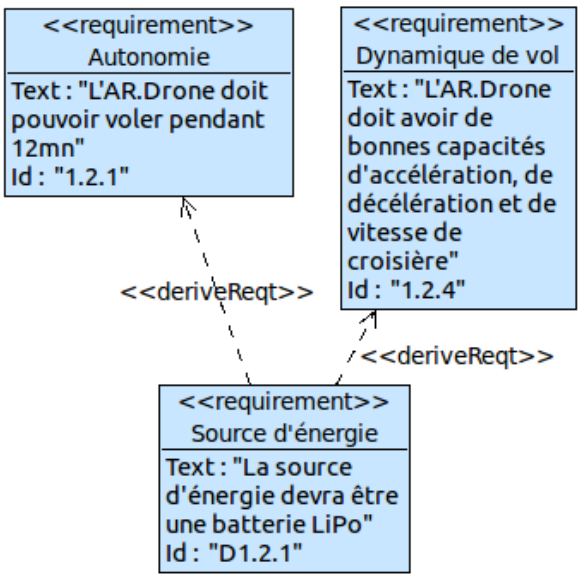
Une exigence possède un nom, une description sous la forme d'un texte et un identifiant unique permettant un référencement correct dans le système.

Relation de contenance



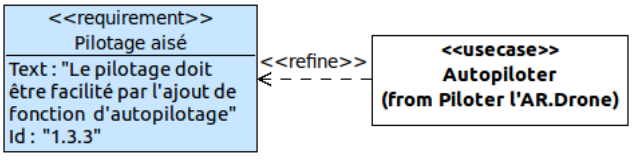
La relation de contenance permet de décomposer une exigence complexe en exigences unitaires plus faciles à relier aux autres éléments de modélisation (cas d'utilisation, blocs, etc.).

Relation de dérivation « deriveReqt »



Des exigences « métiers » peuvent se traduire en exigences « techniques » issues de choix de conception. La relation de dérivation permet de modéliser cela.

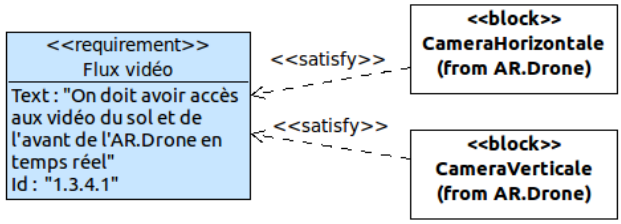
Relation de raffinement « refine »



La relation de raffinement peut être utile pour décrire comment un ou plusieurs éléments de la modélisation peuvent préciser une exigence (un cas d'utilisation, un

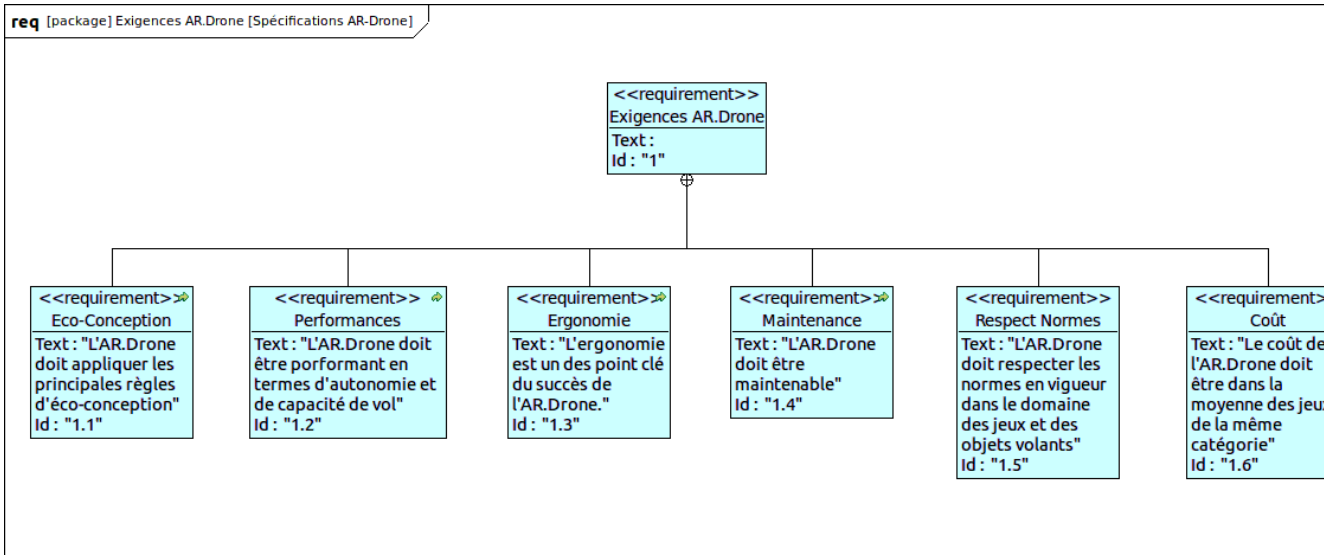
diagramme de séquence, etc.).

Relation de satisfaction « satisfy »



La relation de satisfaction décrit comment un élément d'architecture ou d'implémentation du modèle satisfait une exigence.

Exemples :



: Exigences AR.Drone

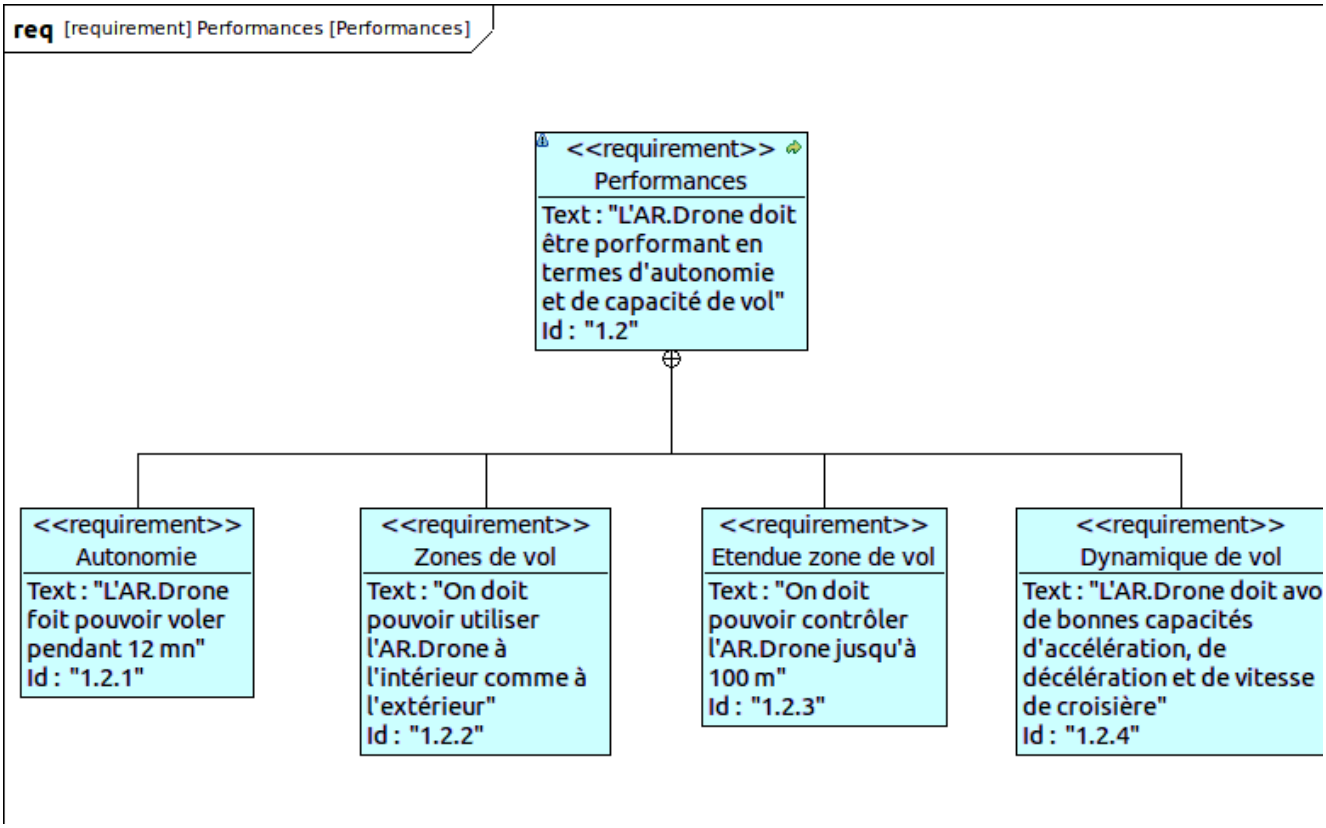
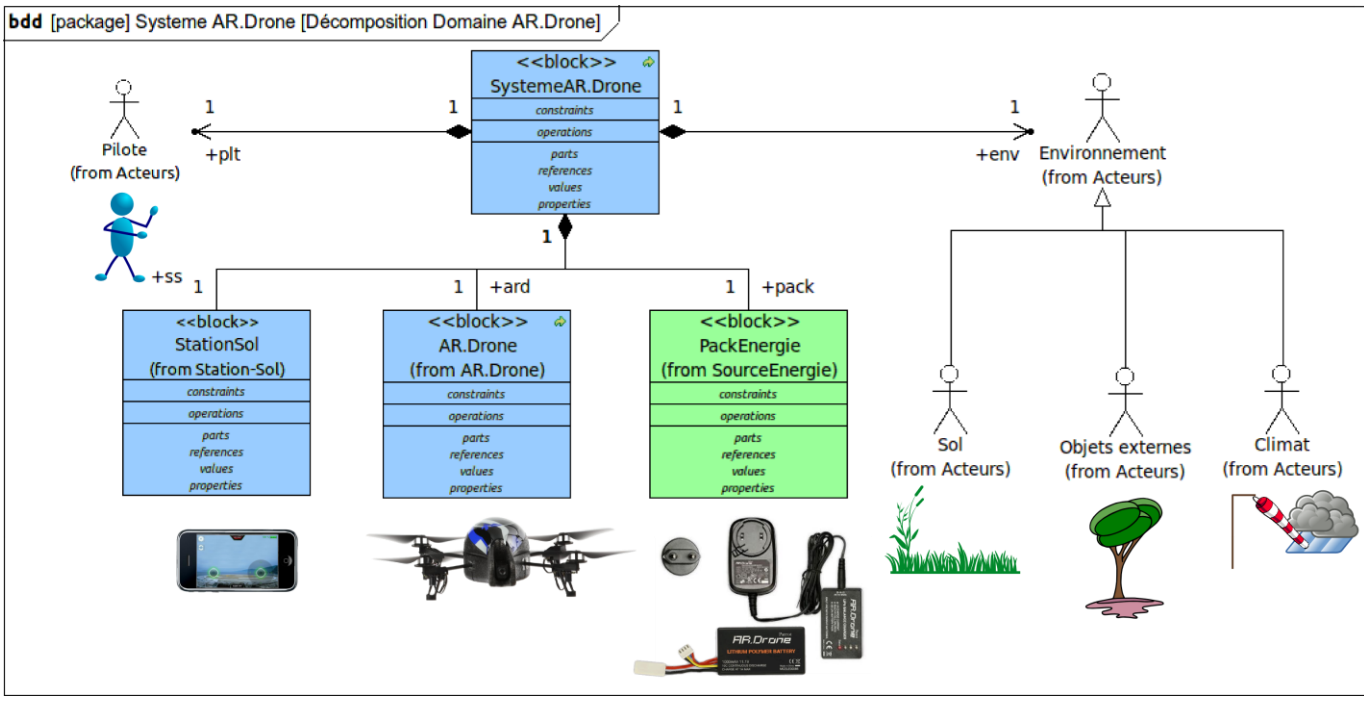


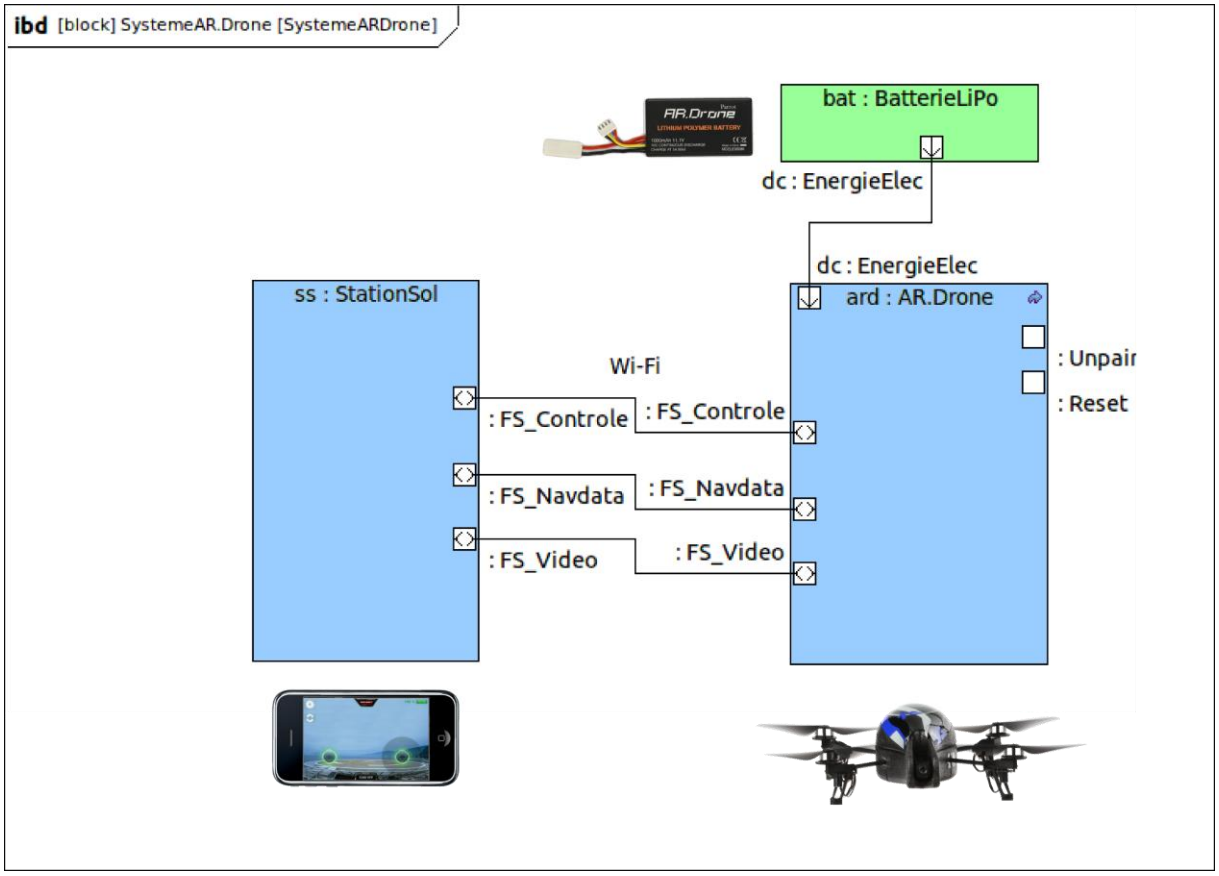
Fig. 1: Performances AR.Drone

4) Diagramme bloc

Les blocs permettent de réaliser une décomposition logique ou physique d'un système complexe. Une fois les blocs identifiées, la modélisation va utiliser deux types de diagrammes :

- **Block Definition Diagram ou BDD (Le diagramme de définition de blocs)** : il permet de décrire comment le système est conçu, illes relationsentre les blocs et / ou les acteurs du système doivent être énoncées.
- **Internal Bloc Diagram ou IBD (Le diagramme de bloc interne)** : Il montre les interactions et les échanges de flux (physiques ou informationnels) entre les blocs.





Dans le diagramme interne de bloc les flux échangés entre les instances de blocs sont notifiés, ses échanges se font par l'intermédiaire de ports.

- Ports de flux (flow port) permettent les échanges de fluide, d'énergie. Ce port est matérialisé par un rectangle dans lequel la flèche indique le sens des échanges (dc :EnergieElec).
- Port de flux de données matérialisé par un rectangle et le « ‹› «
- Port standard muni d'aucun symbole ils permettent de décrire des services rendus par un bloc.

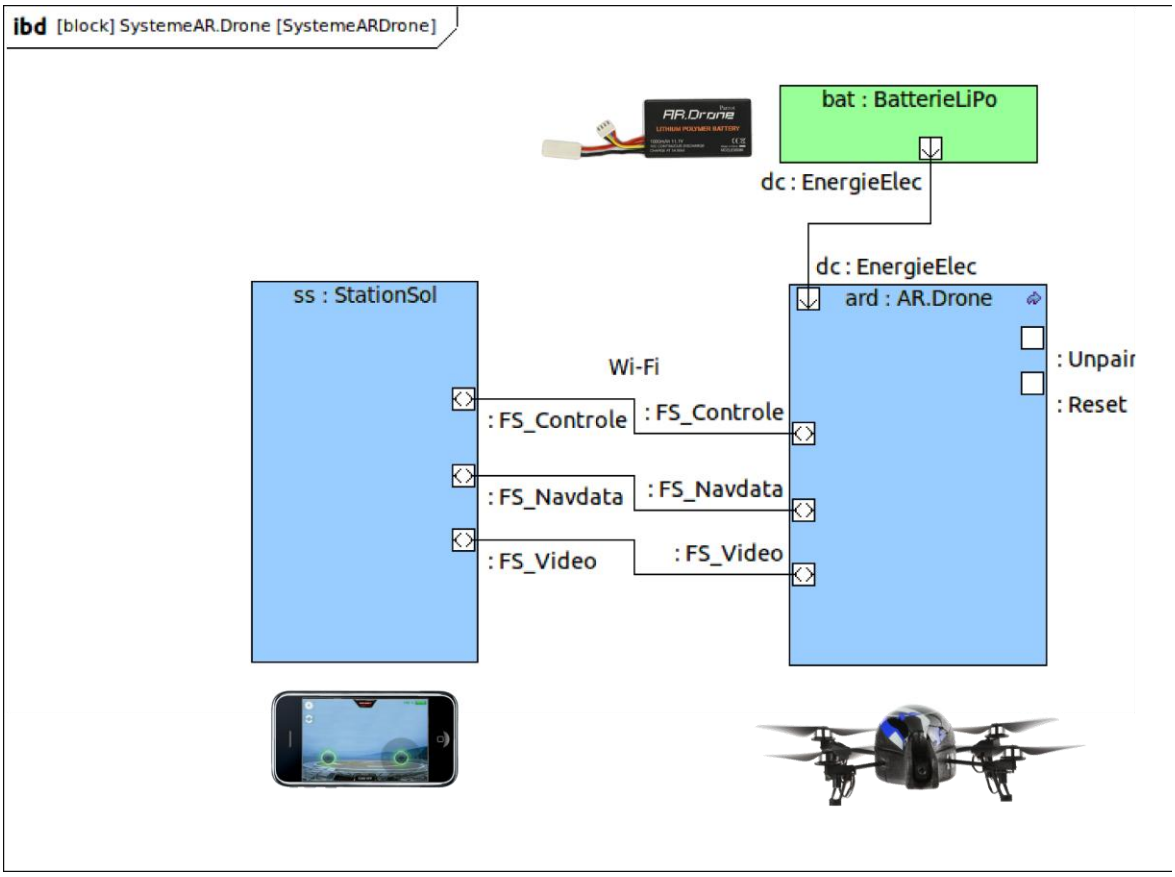


Illustration 1: IBD Système AR.Drone