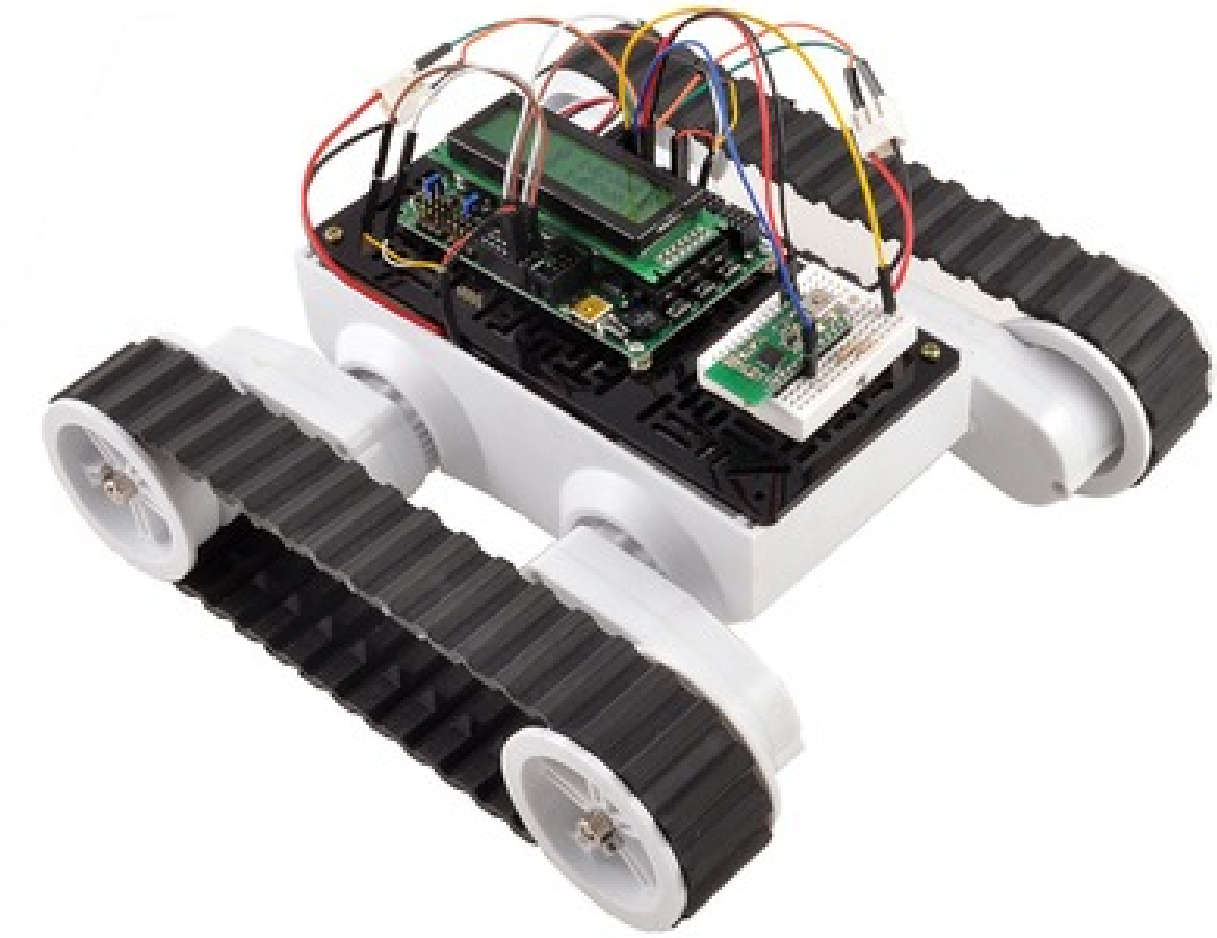
Projet SysML



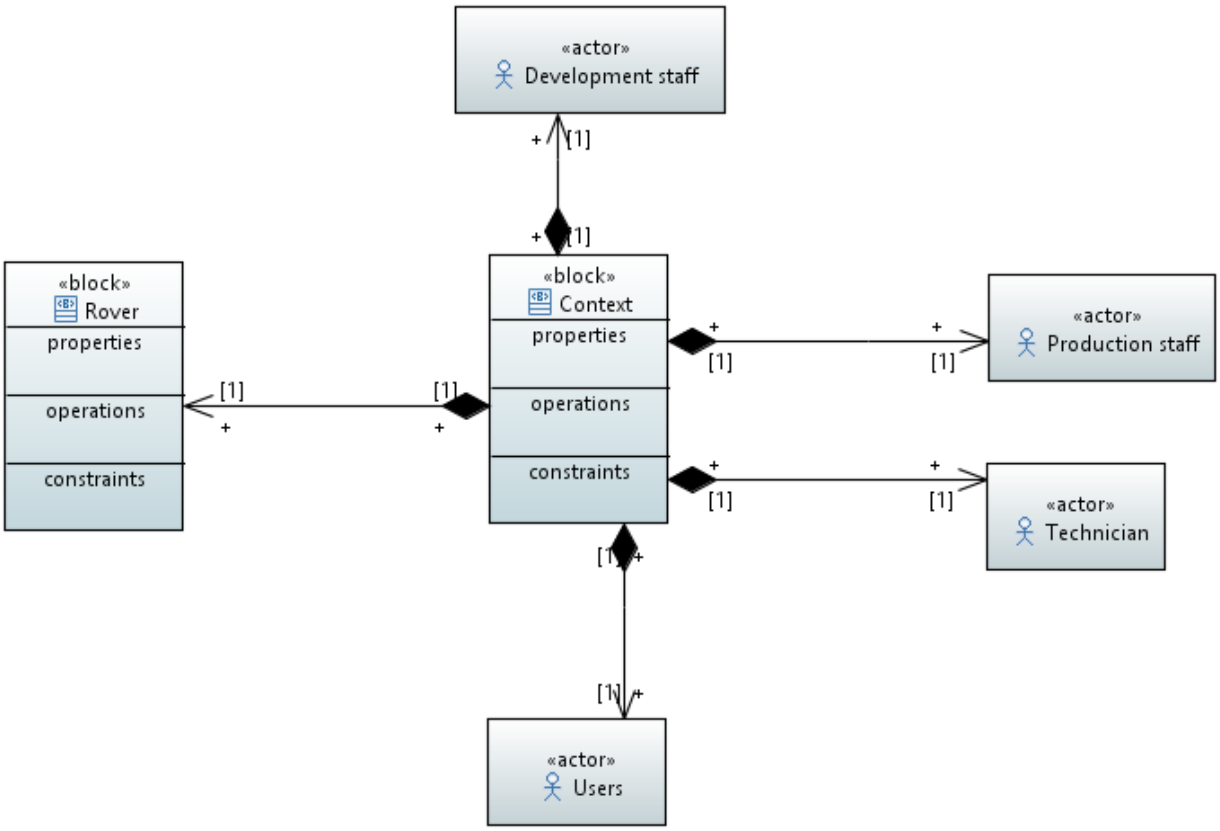
Akana MAO
Mariane NDIAYE
Antoine WILLEMS

Introduction

L'objectif de ce travail est de proposer une modélisation SysML du robot Rover en nous basant sur les exigences qui ont été exprimées par Eclipse.
L'outil utilisé pour faire la modélisation est Papyrus, qui est un plugin d'Eclipse.

Sommaire

1 - Diagramme de contexte



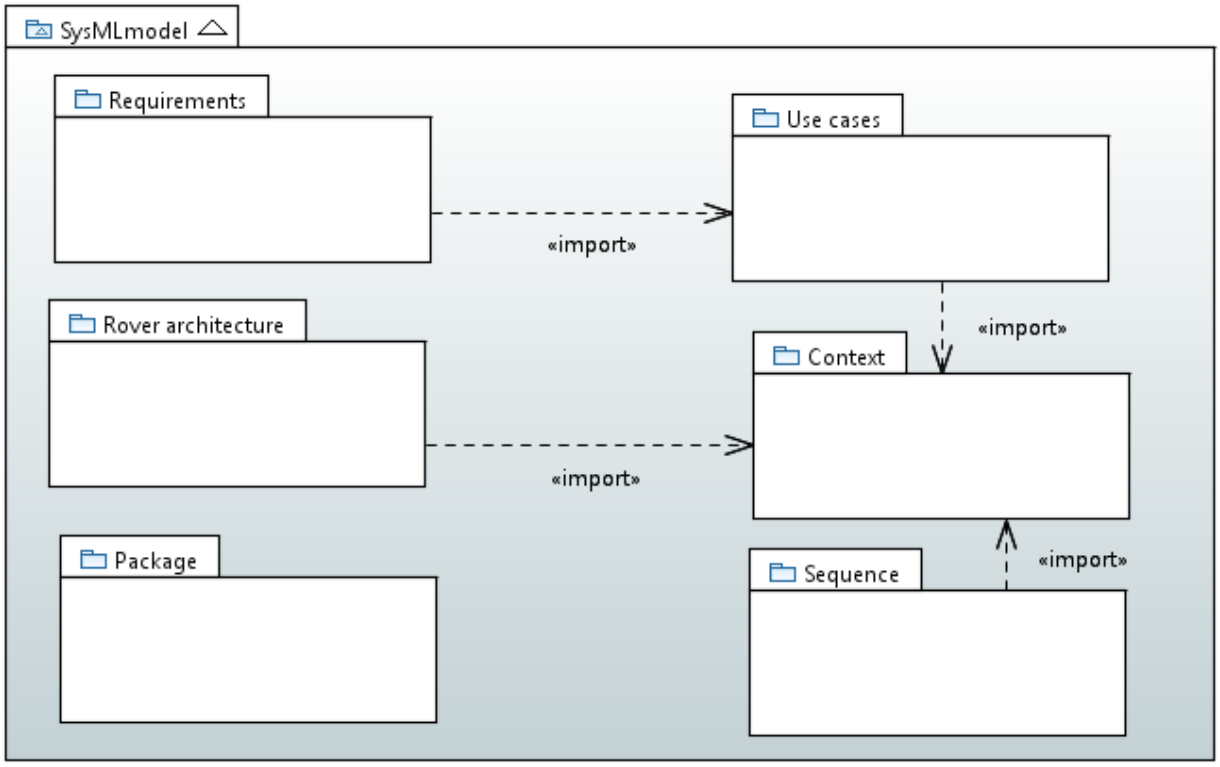
Users: Ce sont les personnes qui utilisent le Rover, sachant que celui-ci est principalement autonome, ils démarrent ou arrêtent simplement le Rover.

Development staff: C'est l'équipe qui a programmée le Rover.

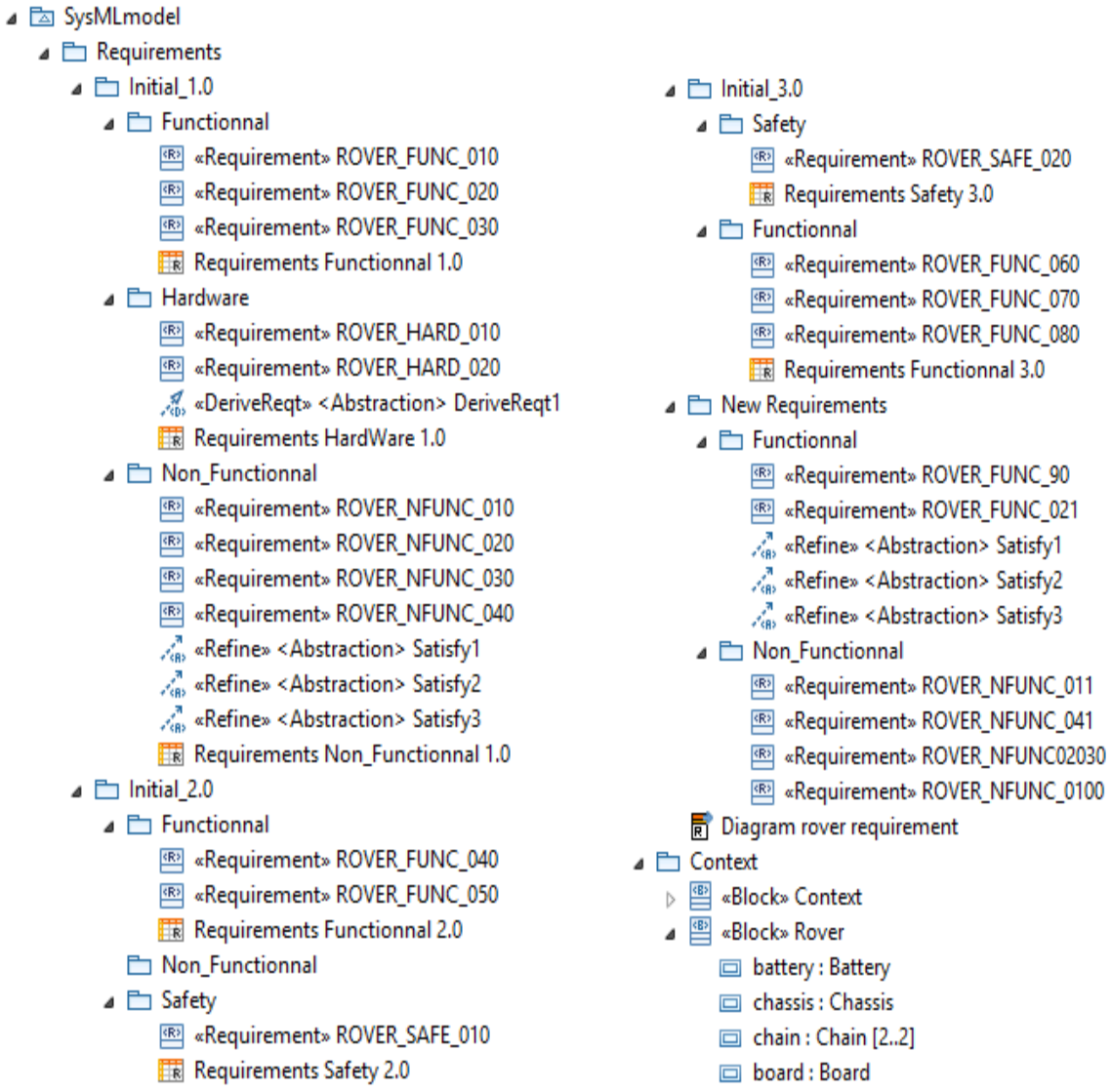
Technician: C'est la personne qui répare le Rover.

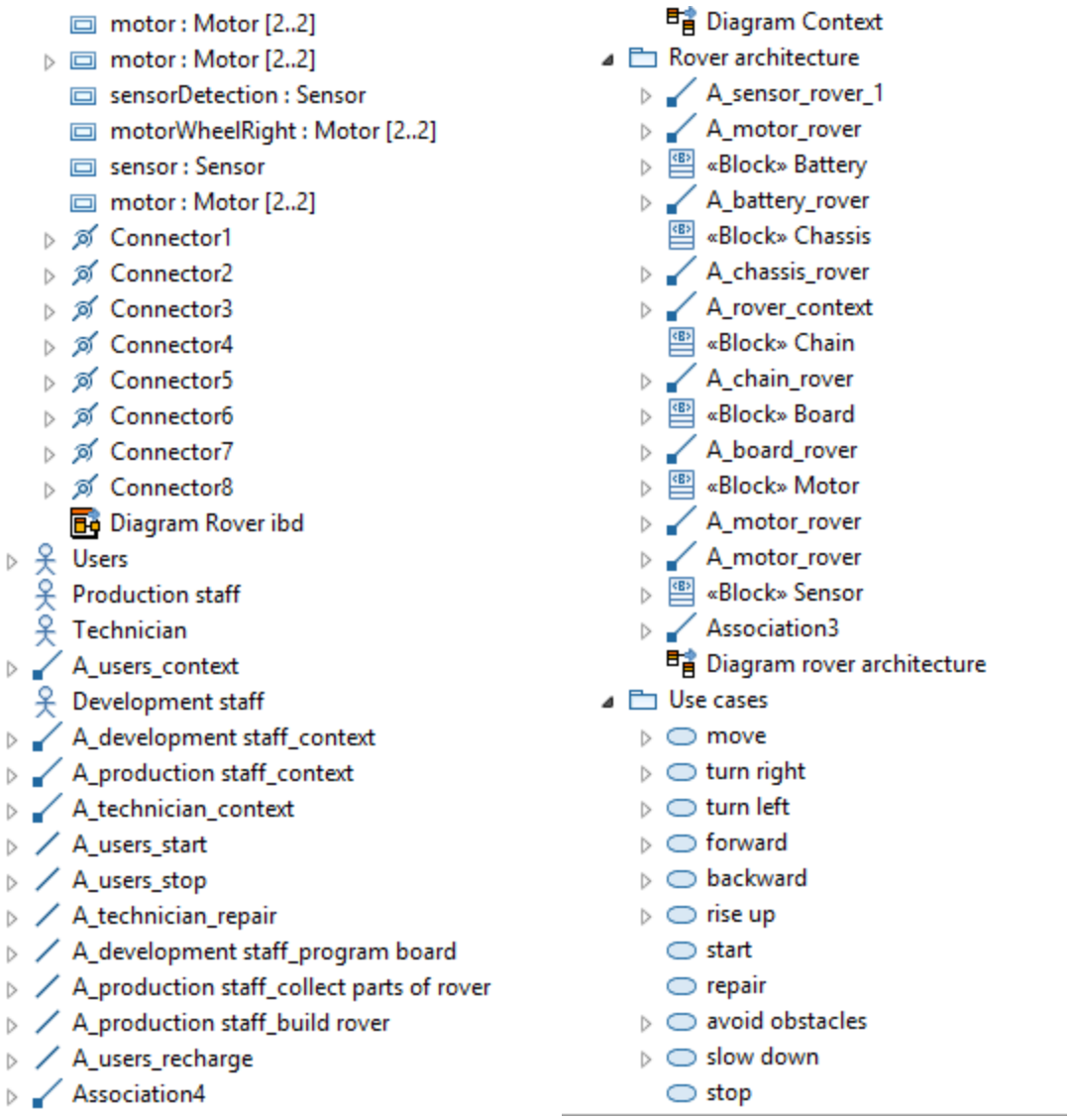
Production staff: C'est l'equipe qui fabrique le Rover.

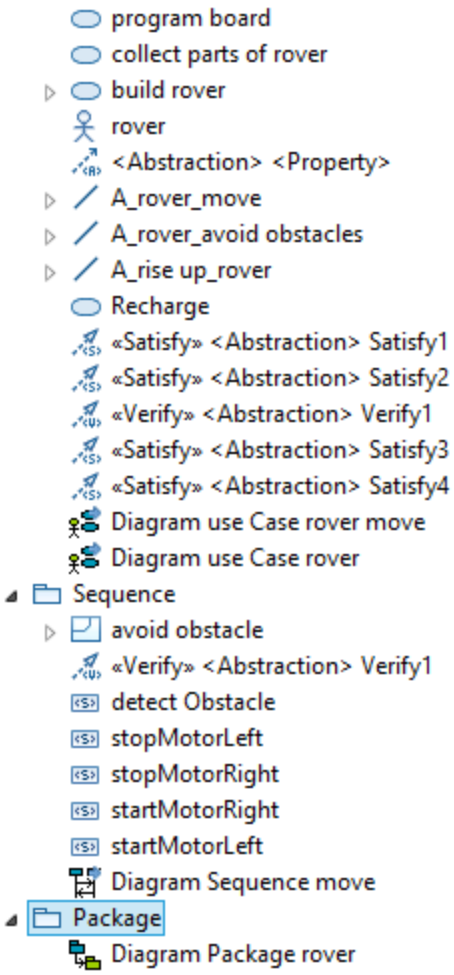
2 - Diagramme de Packetage



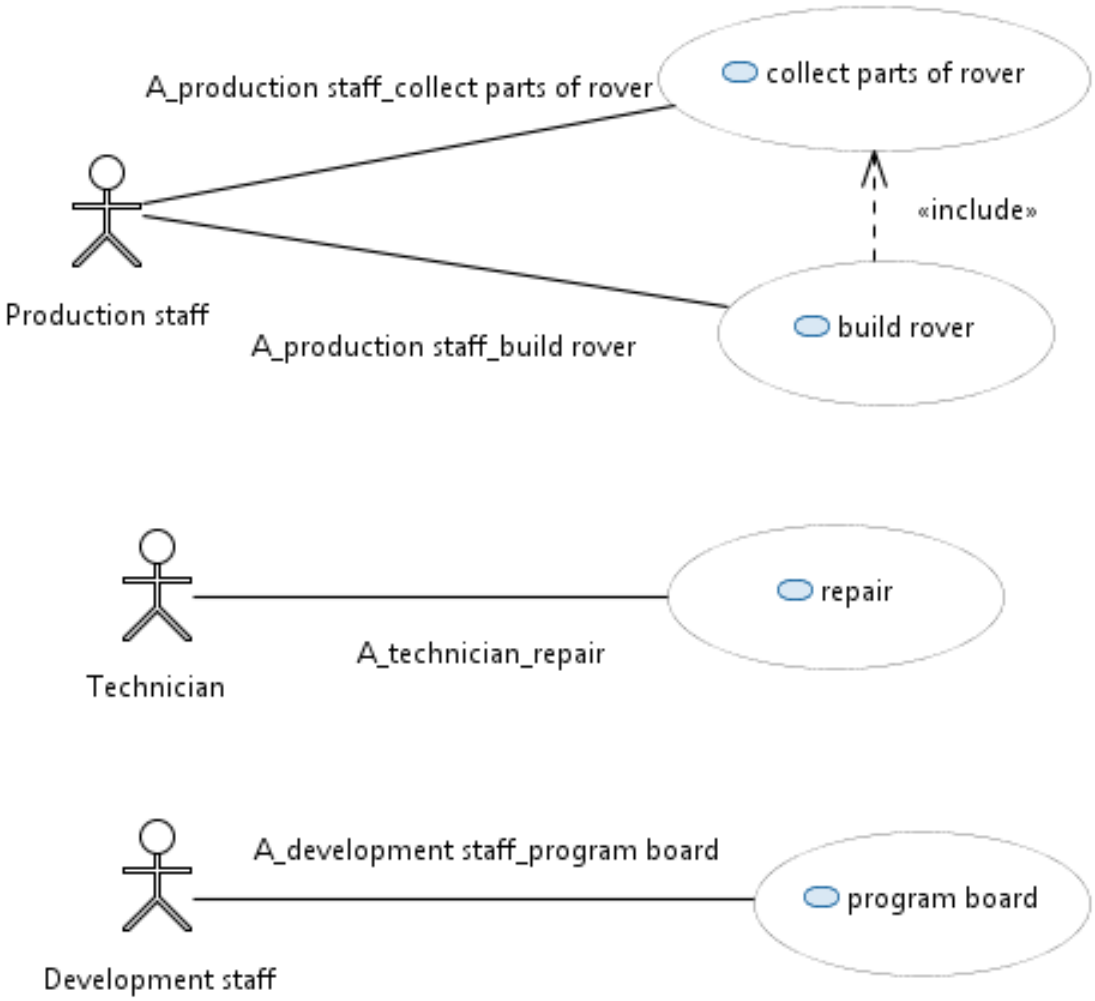
3 - Model Explorer



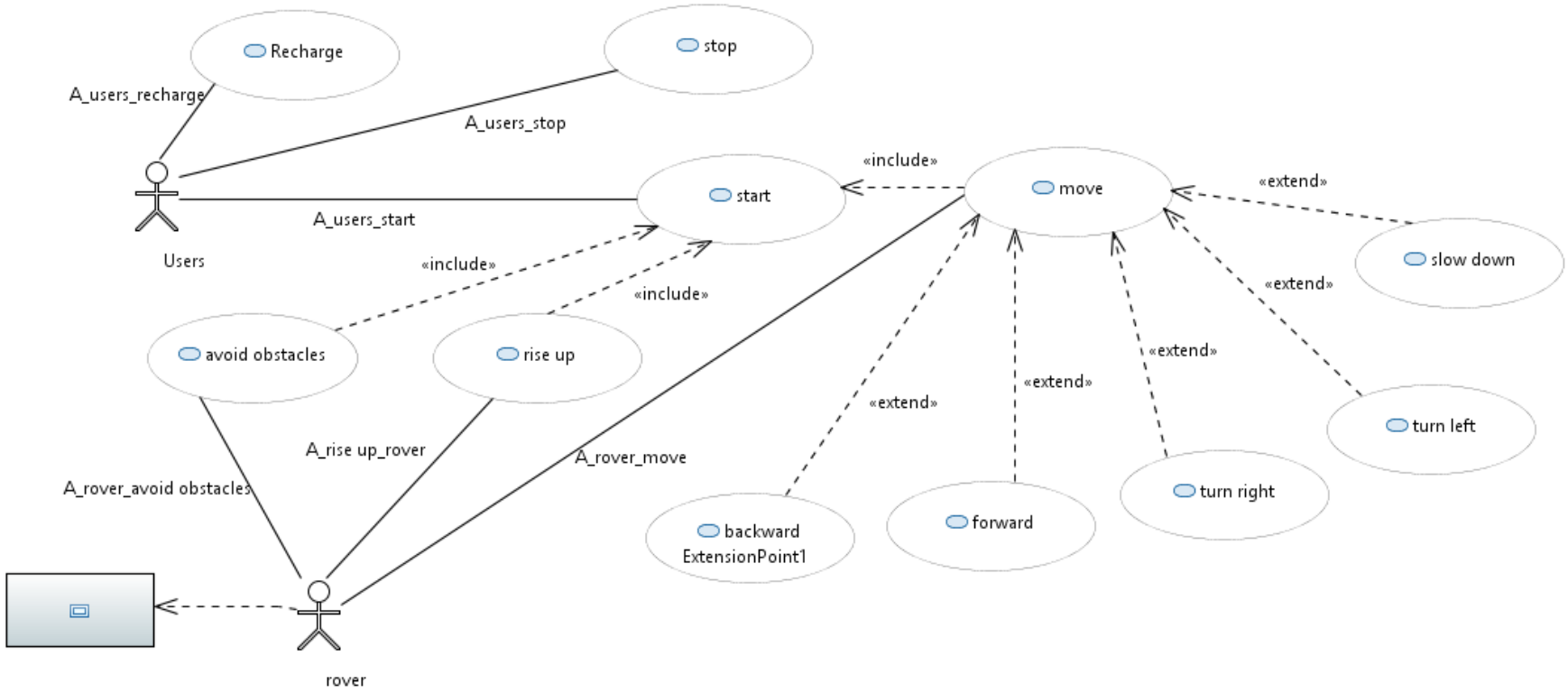

motor : Motor [2..2]
motor : Motor [2..2]
sensorDetection : Sensor
motorWheelRight : Motor [2..2]
sensor : Sensor
motor : Motor [2..2]
Connector1
Connector2
Connector3
Connector4
Connector5
Connector6
Connector7
Connector8
Diagram Rover ibd
Users
Production staff
Technician
A_users_context
Development staff
A_development staff_context
A_production staff_context
A_technician_context
A_users_start
A_users_stop
A_technician_repair
A_development staff_program board
A_production staff_collect parts of rover
A_production staff_build rover
A_users_recharge
Association4
Diagram Context
Rover architecture
A_sensor_rover_1
A_motor_rover
«Block» Battery
A_battery_rover
«Block» Chassis
A_chassis_rover
A_rover_context
«Block» Chain
A_chain_rover
«Block» Board
A_board_rover
«Block» Motor
A_motor_rover
A_motor_rover
«Block» Sensor
Association3
Diagram rover architecture
Use cases
move
turn right
turn left
forward
backward
rise up
start
repair
avoid obstacles
slow down
stop


program board
collect parts of rover
build rover
rover
<Abstraction> <Property>
A_rover_move
A_rover_avoid obstacles
A_rise up_rover
Recharge
«Satisfy» <Abstraction> Satisfy1
«Satisfy» <Abstraction> Satisfy2
«Verify» <Abstraction> Verify1
«Satisfy» <Abstraction> Satisfy3
«Satisfy» <Abstraction> Satisfy4
Diagram use Case rover move
Diagram use Case rover
Sequence
avoid obstacle
«Verify» <Abstraction> Verify1
detect Obstacle
stopMotorLeft
stopMotorRight
startMotorRight
startMotorLeft
Diagram Sequence move
Package
Diagram Package rover

4 - Diagramme des Use Cases



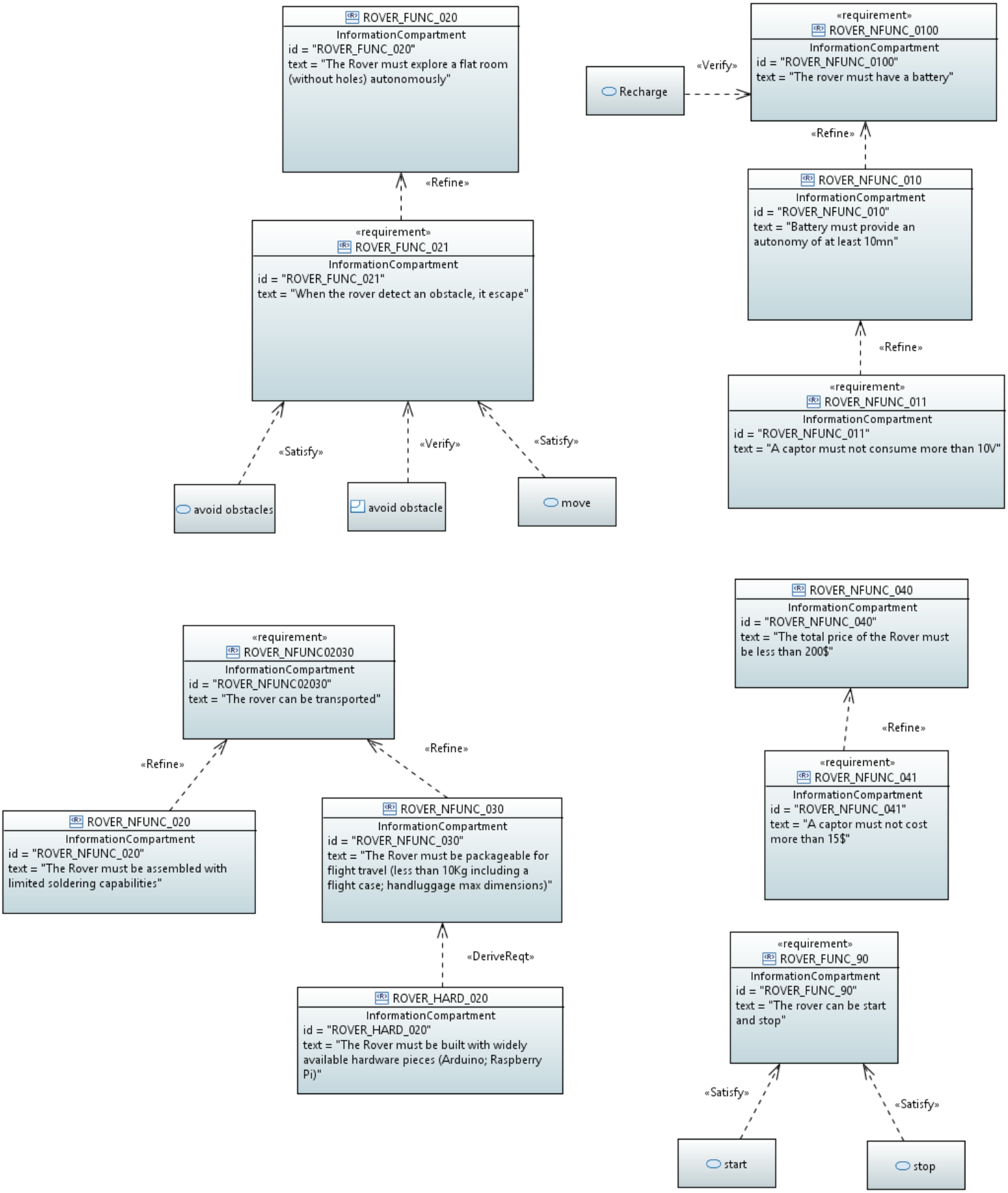
Ce diagramme de cas d'utilisation représente l'utilisation des différents acteurs sur le Rover.


Recharge
stop
A_users_recharge
A_users_stop
start
«include»
move
«extend»
Users
A_users_start
slow down
«include»
«include»
«extend»
avoid obstacles
rise up
«extend»
«extend»
turn left
A_rise up_rover
A_rover_move
«extend»
A_rover_avoid obstacles
turn right
backward
ExtensionPoint1
forward
rover

Ce diagramme de cas d'utilisation représente l’utilisation du Rover. L’utilisateur peut ainsi, démarrer, arrêter, recharger le Rover. Quant au Rover il peut se déplacer, ce qui inclus, avancer, reculer, tourner à gauche, tourner à droite. Mais il peut aussi éviter un obstacle, se soulever et s’abaisser.

L’acteur Rover est une abstraction du block Rover contenu dans le diagramme de contexte.

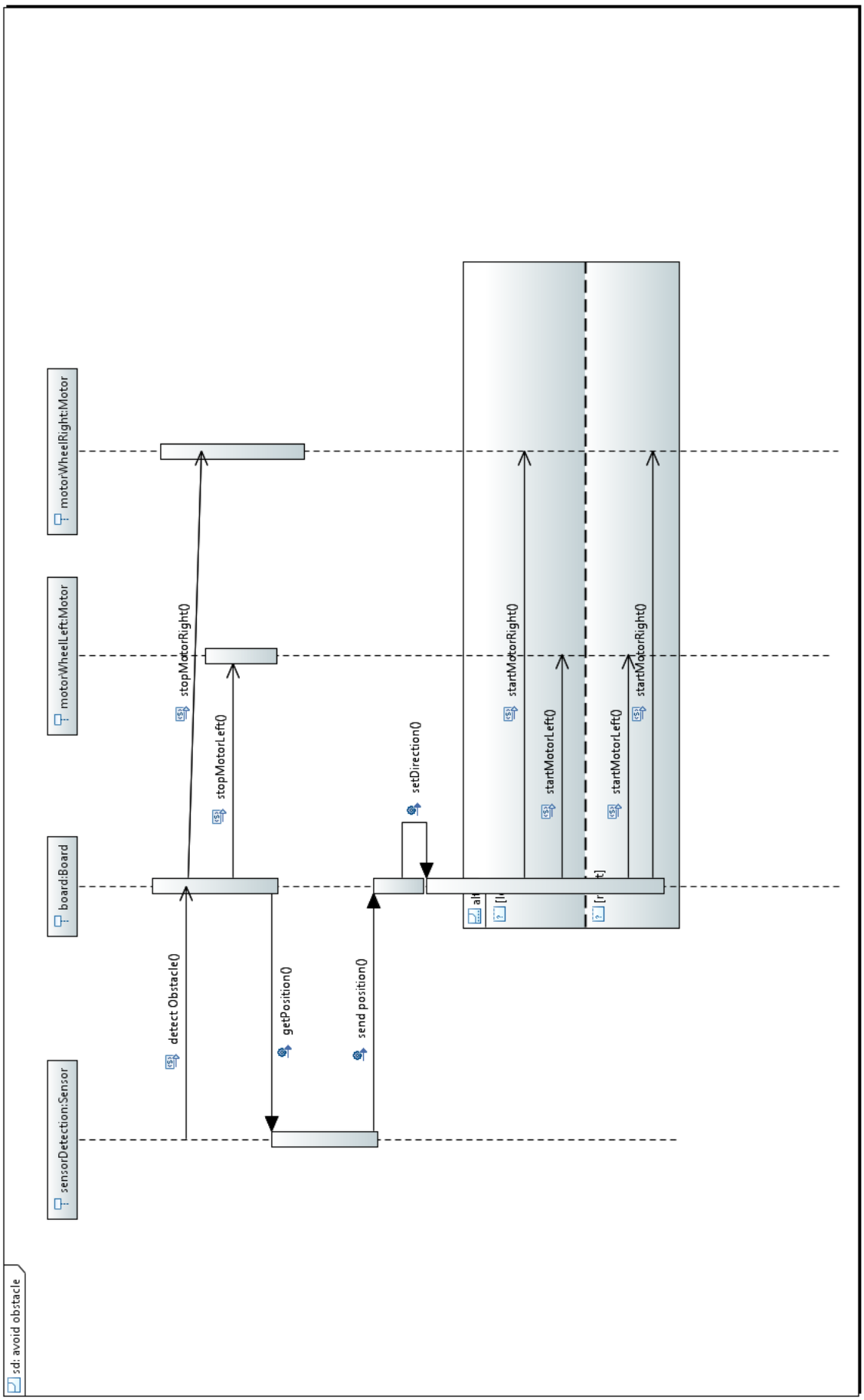
5 - Diagramme des Requirements



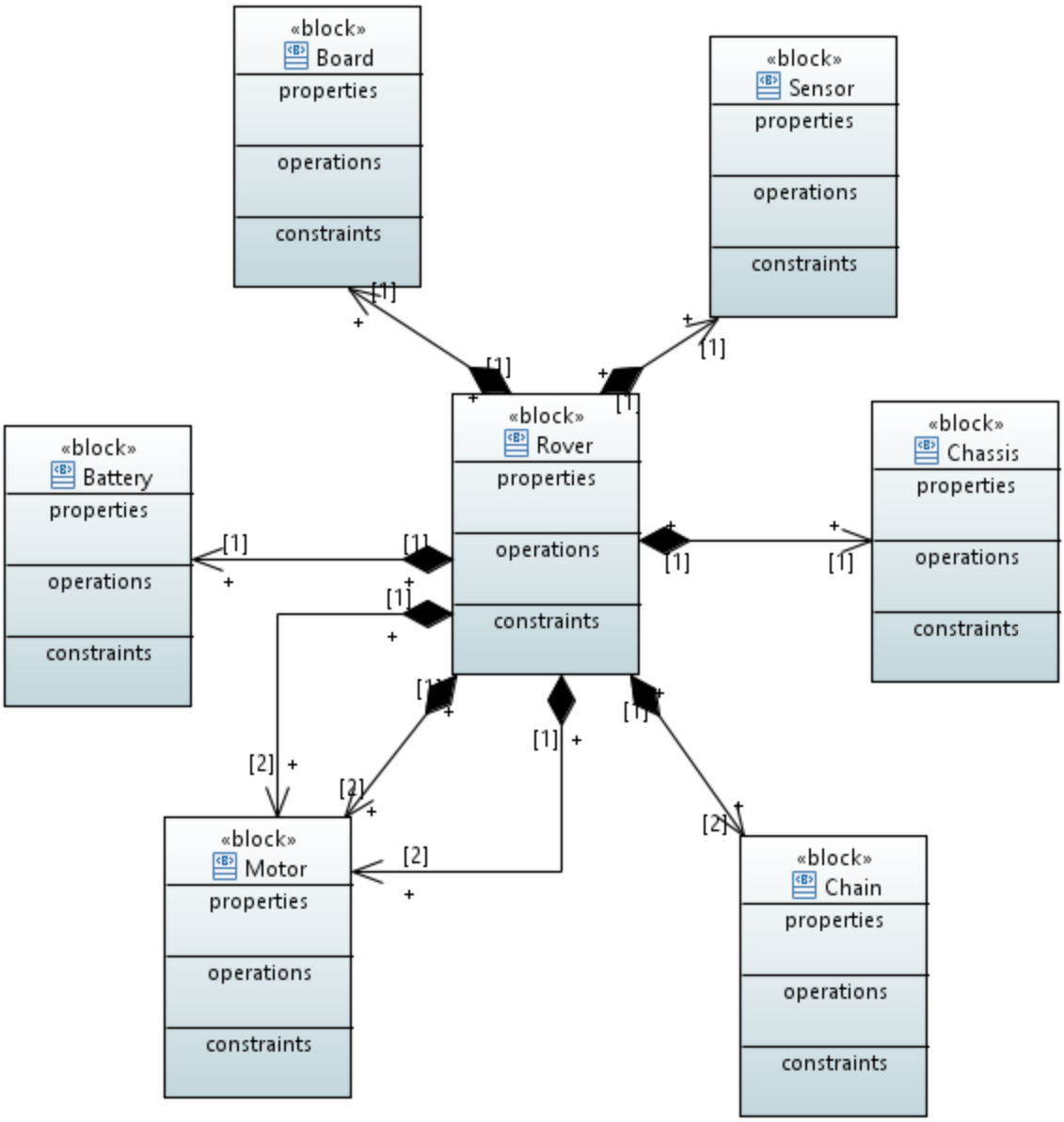
6 - Diagramme de Séquence

Éviter obstacle:

Lorsque le capteur détecte un obstacle, il envoie un signal à la puce(controleur) qui à son tour envoie un signal aux moteurs des chaînes pour leur demander de s'arrêter , immobilisant ainsi le robot. Une fois que les moteurs sont stoppés, la puce demande au capteur de lui fournir la position de l'obstacle à éviter. Avec les coordonnées de l'obstacle, la puce calcule si elle doit faire tourner le robot à gauche ou à droite. Dans le cas où le robot doit tourner à gauche, la puce demande aux moteurs de la chaîne de droite de redémarrer. Puis, après avoir évité l'obstacle , elle demande aux moteurs de la chaîne de gauche de se remettre en route. Avec les moteurs des deux cotés allumés , le robot peut continuer sa route. Dans ce cas où le robot doit tourner à droite c'est d'abord les moteurs pour la chaîne de gauche qui sont allumés suivit des moteurs pour la chaîne de droite.


sd: avoid obstacle
sensorDetection:Sensor
board:Board
motorWheelLeft:Motor
motorWheelRight:Motor
detect Obstacle()
getPosition()
send position()
stopMotorLeft()
stopMotorRight()
setDirection()
startMotorLeft()
startMotorRight()
startMotorLeft()
startMotorRight()

7 - Diagramme de Définition de Blocks



Battery: La batterie permet d'alimenter le Rover.
Chain: Elles permmetent au robot de pouvoir avancer, reculer, et tourner.
Motor: Pour les moteurs nous avons décidés de mettre 3 compositions puisqu'il y en a 2 pour les moteurs de la chaine gauche, 2 pour la chaine droite, et 2 pour soulever et abaisser le Rover.
Chassis: C'est la pièce qui permet de maintenir les autres pièces.
Board: C'est le circuit électrique dans lequel on programme et relie les pièces.

8 - Diagramme Interne du Block Rover

Si le Sensor détecte un obstacle il envoie le signal “detect obstacle” la Board.

Ensuite un signal de type “stopMotorLeft” sera envoyé aux deux moteurs gauches du robot, ce qui arrêtera la chaîne gauche du robot.

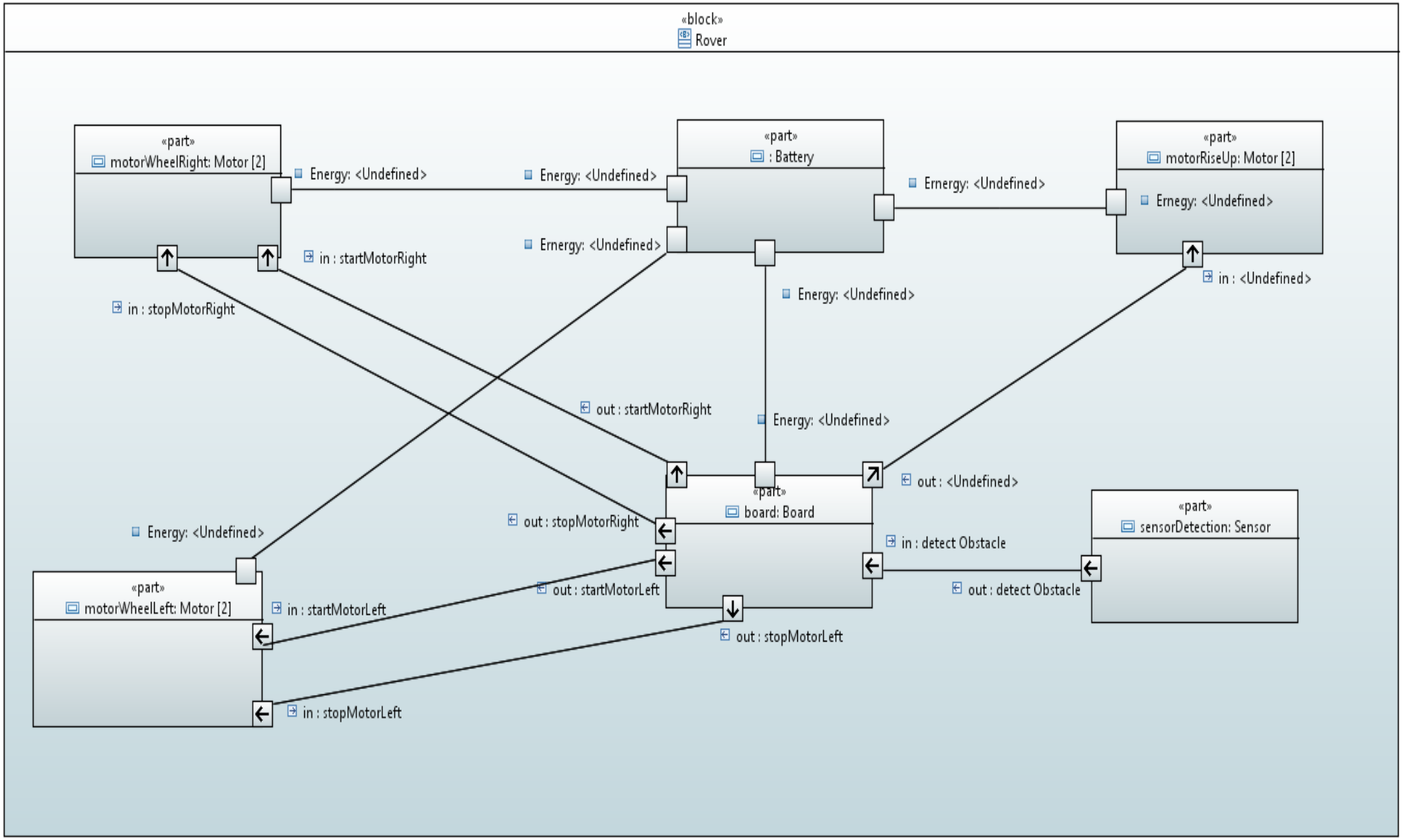
Un autre signal de type “stopMotorRight” sera envoyé aux deux moteurs droits du robot, ce qui arrêtera la chaîne droite du robot.

Pour redémarrer la chaîne gauche, le contrôleur(Board) enverra un signal de type “startMotorLeft” aux deux moteurs gauches.

Pour redémarrer la chaîne droite, le contrôleur(Board) enverra un signal de type “startMotorRight” aux deux moteurs droits.

Ensuite il y a les signaux Energy qui permet à la batterie d’alimenter les autres composants du robot.

Enfin il y a le dernier signal entre le contrôleur et les deux moteurs qui permettent de sur élever le robot. Ce signal n’a pas de type puisque nous ne l’avons pas modélisé dans un diagramme de séquence.


«block»
Rover
«part»
motorWheelRight: Motor [2]
Energy: <Undefined>
Energy: <Undefined>
«part»
: Battery
Energy: <Undefined>
«part»
motorRiseUp: Motor [2]
Ernegy: <Undefined>
in : startMotorRight
Ernergy: <Undefined>
in : <Undefined>
in : stopMotorRight
Energy: <Undefined>
out : startMotorRight
Energy: <Undefined>
out : <Undefined>
«part»
board: Board
Energy: <Undefined>
out : stopMotorRight
in : detect Obstacle
«part»
sensorDetection: Sensor
«part»
motorWheelLeft: Motor [2]
in : startMotorLeft
out : startMotorLeft
out : detect Obstacle
out : stopMotorLeft
in : stopMotorLeft