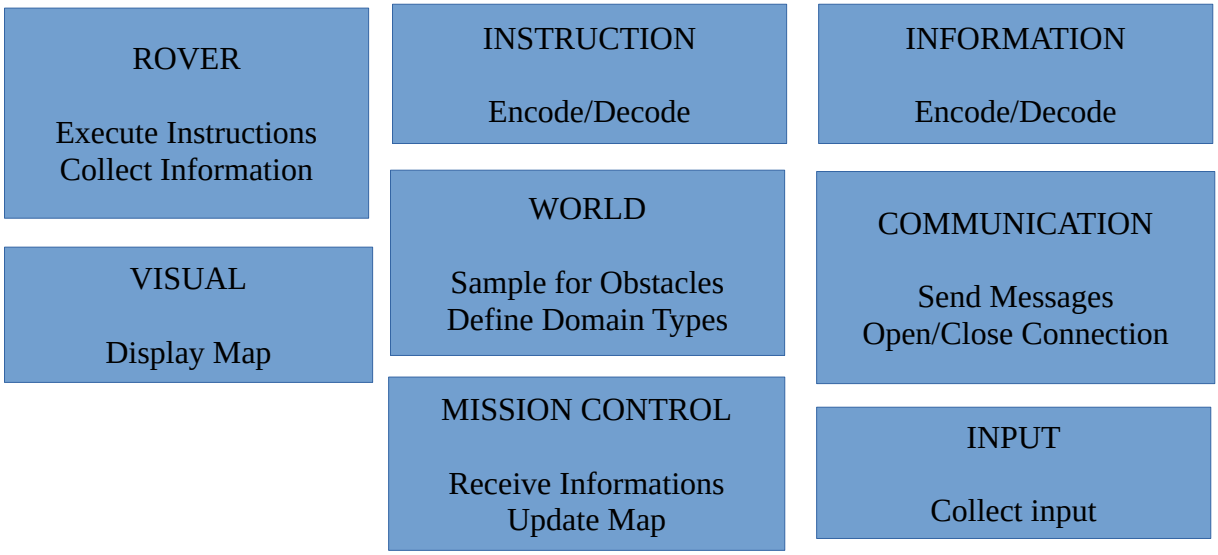
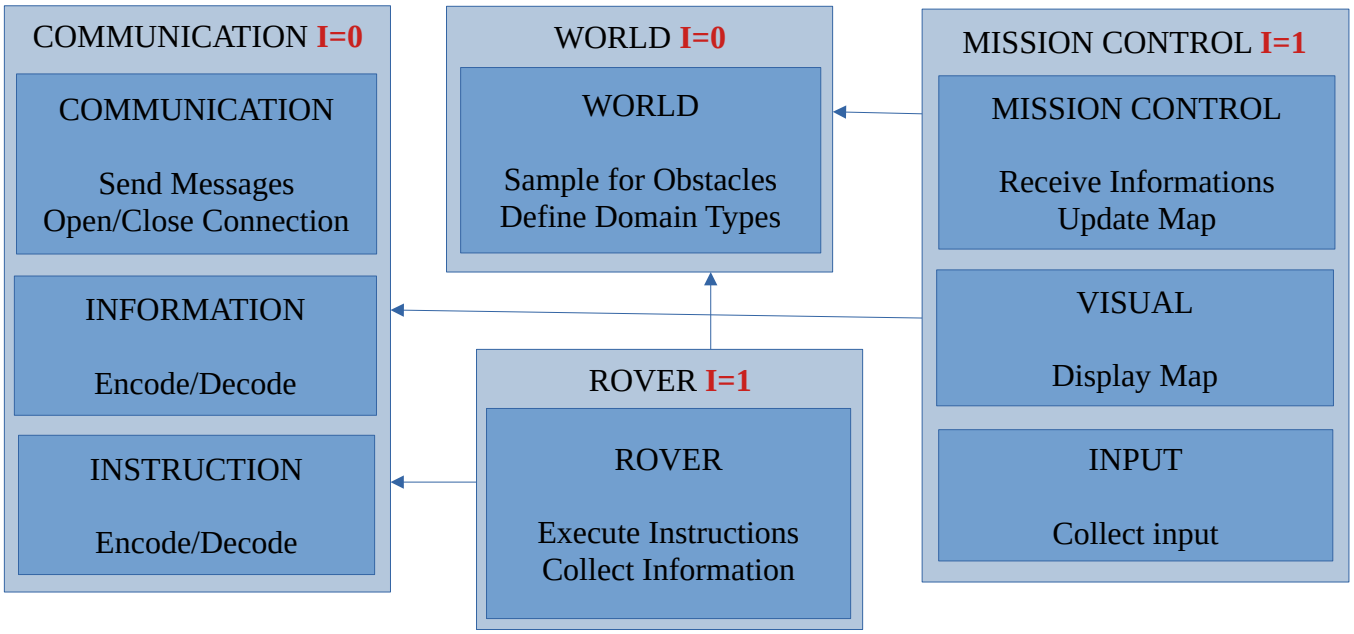
On commence par lister les “soucis” du Rover

- Définir les types du métier
- Établir la connexion entre Control et Rover
- Collecter les commandes de l’utilisateur
- Encoder/Décoder l’instruction pour le rover
- Envoyer des messages
- Recevoir des messages
- Exécuter l’instruction
- Interroger le monde pour des obstacles
- Encoder/Décoder l’information
- Mettre à jour la carte
- Représenter la carte à l’utilisateur

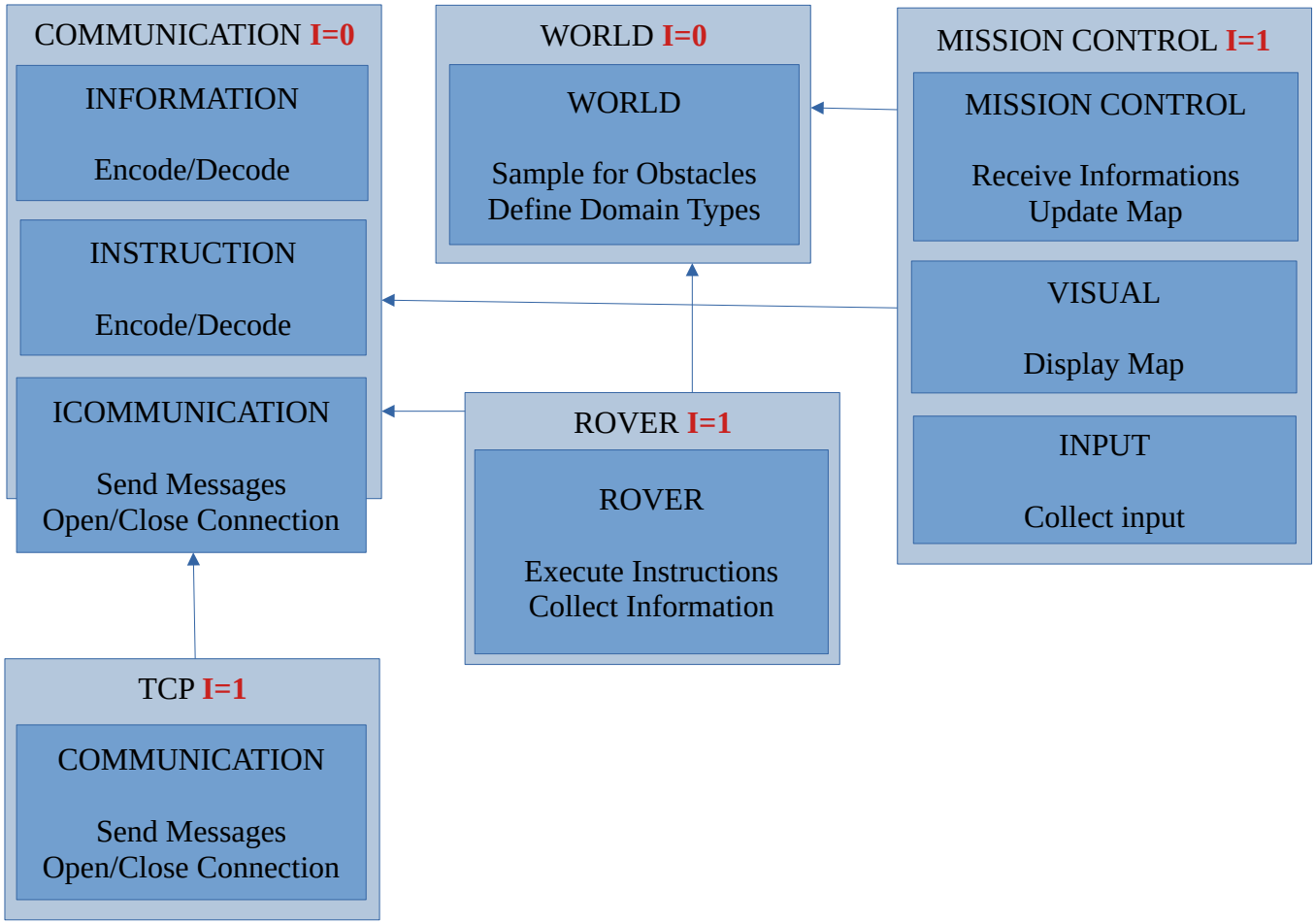
En combinant les soucis fortement liés on peut les répartir de la manière suivante.



Ces blocs peuvent être combinés en un nombre minimal de modules faiblement couplés de la manière suivante. Les flèches représentent les dépendances.



Ainsi MissionControl et Rover sont tous deux dépendants de Communication.
Mais Communication est un module volatil et stable, il est donc souhaitable d'isoler les spécificités de son implémentation du reste de l'application afin d'amortir les conséquences d'une changement de protocole.
Pour cela nous séparons communications, en :
- d'une part un module abstrait, intégrant les règles de communications de notre application et une interface que les différentes implémentations devront respecter.
- d'autre part, des modules individuels contenant l'implémentation pour chaque protocole, ici TCP.



Le style architectural résultant de ces choix est une architecture en Onion dans laquelle :

- le cœur de l'application repose sur les fondamentaux métier (planète, Vector, …)
- Sur lesquels viennent se greffer les applications spécifiques (rover, mission control)
- Autour de laquelle gravitent différente implémentations des communications dépendantes de l'infrastructure choisie, et découplées du reste par une interface côté Application afin de ne faire transiter que des messages neutres (ici du texte).

LISTE DES INTERFACES

WORLD expose :

- Une classe Vector2 entièrement publique. Incluant fonction mathématiques et constantes pour les directions
- Une classe Planète, exposant
  - GetElement(Vector2) : Renvoie l'élément situé aux coordonnées Vector2.
  - Des constructeurs
- Deux enums, pour le ElementType (la nature du contenu d'une case) et les directions cardinales.

COMMUNICATION expose :

- Deux classes Instruction et Information, régissant l'encodage des messages
- Une enum, listant les directions possibles dans l'instruction au rover.
- Une classe Connection contenant une entrée et une sortie texte.
- Une interface ICommunicator que les modules d'infra devront implémenter, exposant 3 mèthodes
  - HostCommunication : Ouvre la communication, et renvoie une Connection quand initialisé.
  - JoinCommunication : Rejoins la communication et renvoie une Connection.
  - CloseCommunication(Connection) : Ferme la communication liée à la Connection

TCP n'expose rien
ROVER n'expose rien
MISSION CONTROL n'expose rien

La synchronisation de TCP et de Planète sont assuré par des fichiers de configuration communs.
La communication entre ROVER et MISSION CONTROL, est assuré par deux objects Connection créés lors de l'ouverture de la Communication.

ANALYSE SWOT

Strengths :

- Ne conserve que l'interface Communication
- L'application est centré sur son module le moins volatil.
- Ses modules les plus volatils (TCP, …) sont isolés du reste par une interface.

Weaknesses :

- Développement additionnel d'un module de communication abstrait, pour permettre de remplacer les implémentations, possibilité qui ne sera plus employable en production.

Opportunities :

- Le module de communication est très simple à modifier/remplacer

Threats :

- Une modification des primitives Métiers auraient des répercussions partout.