

# Plan détaillé : Anchoring

---

- Plan détaillé : Anchoring
  - Résumé (Français)
  - Résumé (Anglais)
  - Introduction
  - I - Présentation du Projet
    - 1) Cadre du projet en entreprise
      - a) Présentation de l'entreprise
      - b) Présentation de l'environnement du projet
    - 2 - Objectif du projet
      - a) Projet de recherche
      - b) Domaine du projet
    - 3) Présentation du projet : Ancrage Sémantique (Anchoring)
      - a) Présentation rapide
      - b) Terminogy
      - c) Presentation Anchoring avec remplissage d'une ontologie
      - d) But du projet
  - II - Recherche du moteur ontologique
    - 1) Choix du OWL
    - 2) Qu'est ce qu'on veut faire avec notre moteur ontologique
    - 3) Protege
    - 4) OWLAPI / Jena
    - 5) Ontologenius
    - 6) Owlready2
    - 7) KnowRob
    - 8) Autre moteur d'ontologie
    - 7) Qu'est ce qu'on peut améliorer ?
  - III - Algorithme
    - 1) Presentation Algorithme
      - a) Algorithme Simple Septembre
      - b) Algorithme Linéaire
    - 2) Algorithme : Ontology Manager
      - a) Besoins
      - b) Presenatation de l'algorithme
      - c) Réduction de Ontology Manager dans le cadre du projet
    - 3) Algorithme réduit : Algorithme implémenté
      - a) Explications de l'algorithme
      - b) SYSML
      - c) Implementation ROS Squidly
  - IV - Implementation
    - 1) Design
      - a) Role du design
      - b) Choix de l'Upper Ontologie
      - c) Présentation de SOMA

- d) Modification de SOMA
  - e) Ajout du Mid et du Domain
    - i) Mid ajout de la compatibilité avec les autres Node
    - ii) Domain
  - f) Comment le Design/domain est fait ?
- 2) Setup
  - a) Role du setup
  - b) Comment le populator est fait ?
- 3) Run Time
  - a) Role du run time
  - b) Intégrateur
    - i) Role de l'intégrateur
    - ii) Comment l'intégrateur est fait ?
  - c) Rulemaker
    - i) Role du rulemaker
    - ii) Comment le rulemaker est fait ?
  - d) Reasonner
    - i) Role du reasonner
    - ii) Comment le reasonner est fait ?
  - e) Reader
    - i) Role du reader
    - ii) Comment le reader est fait ?
- V - Apprentissages, critiques, autres chose à explorer
  - 1) Ontologies
  - 2) SWRL
  - 3) Moteur Ontologique
  - 4) Algorithme
  - 5) Design
  - 6) Setup
  - 7) Intergrator
  - 8) RuleMaker
  - 9) Reasonner
  - Moi-même
- Conclusion

## Résumé (Français)

Ancrage symbolique (Anchoring) processus qui transforme des données de Jumeaux numériques (fonctionnel ou géométrique) en une sémantique intelligente à l'aide d'une ontologie. Cette ontologie peut être lue pour donner des prédicats sous la forme *Sujet Action Complément*. De plus, grâce à un reasonner ontologique on peut vérifier la cohérence de l'ontologie, faire des inférences (créer des relations) et appliquer des règles. On peut ainsi dire que l'Anchoring raisonne et donc que c'est un processus intelligent. Grâce à l'Anchoring, l'ontologie est un savoir sémantique sur lequel d'autres applications peuvent interagir. Dans mon cas d'application, la robotique cognitive, le Task Planner et le jumeau numérique (Digital Twin) interagissent avec l'ontologie. Les Digital Twins permettent de remplir l'ontologie grâce à l'Anchoring (intégrateur), et l'Anchoring peut demander aux Digital Twins des informations supplémentaires si nécessaire. Le Task Planner reçoit les prédicats que fournit l'Anchoring pour créer une liste d'action pour résoudre un objectif.

L'Anchoring est temps réel et donc peut s'apercevoir de problème dans les actions. De plus, l'Anchoring est basé sur une ontologie qui est créé avec une upper ontologie pour permettre à ce que ce savoir sémantique soit partageable.

## Résumé (Anglais)

Symbolic anchoring (Anchoring) process that transforms data from Digital Twins (functional or geometric) into intelligent semantics using an ontology. This ontology can be read to give predicates in the form *Subject Action Complement*. Furthermore, an ontology reasoner can be used to check the consistency of the ontology, make inferences (create relations) and apply rules. Anchoring can therefore be considered an intelligent process. Thanks to Anchoring, the ontology is a semantic knowledge on which other applications can interact. In my case study, cognitive robotics, the Task Planner and the Digital Twin interact with the ontology. The Digital Twins fill in the ontology thanks to the Anchoring (integrator), and the Anchoring can request additional information from the Digital Twins if necessary. The Task Planner receives the predicates provided by the Anchoring to create an action list to solve a goal. Anchoring is real-time, so it can spot problems in actions. Furthermore, Anchoring is based on an ontology that is created with an upper ontology to enable this semantic knowledge to be shared.

## Introduction

Bla bla

mots-clés :

- cognitive Robotics
- Robot intelligent
- Symbolique
- Sémantique
- Ontologie
- Raisonnement
- Probleme complexe
- Ancrage
- Ancrage Symbolique

## I - Présentation du Projet

### 1) Cadre du projet en entreprise

#### a) Présentation de l'entreprise

- CEA : centre de recherche français
- EPIC : transfert technologique
- Projet européen
- Projet interne

#### b) Présentation de l'environnement du projet

- Projet européen
- Projet interne

- LSEA
- SQUIDLY
- Intelligence Robot Team
- moonshot

## 2 - Objectif du projet

### a) Projet de recherche

- Découvrir les technologies dans le domaine des ontologies et des règles ontologiques (SWRL)
- Découvrir comment faire de la robotique avec des ontologies
- Faire des conclusions sur les technologies et les pistes rechercher les points positifs et les points négatifs
- Première itération
- Faire des erreurs pour apprendre
- Faire un premier Anchoring

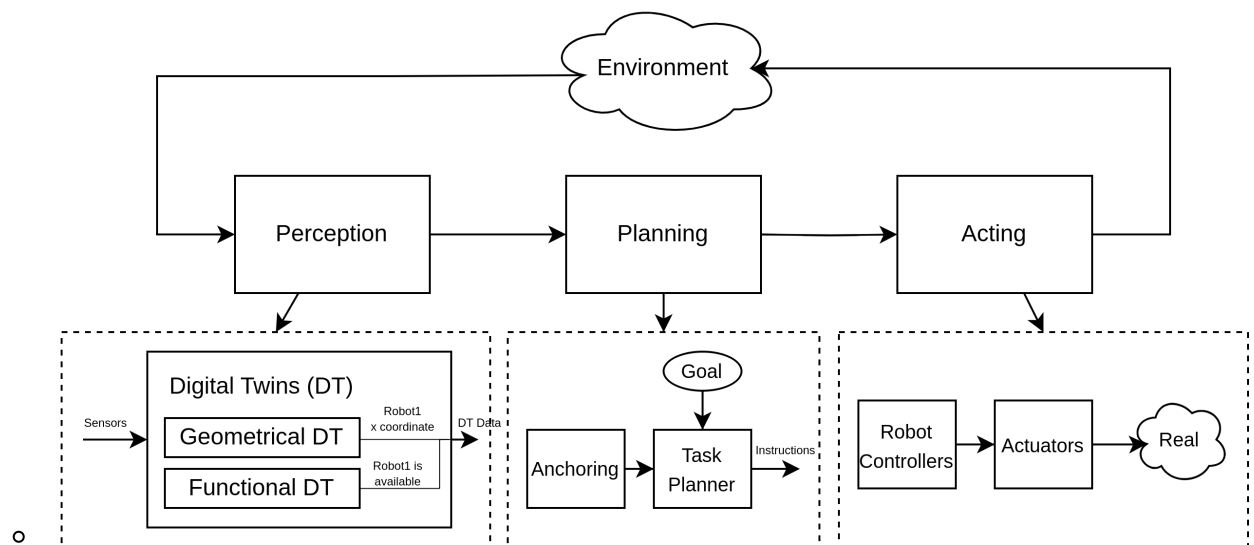
### b) Domaine du projet

- SQUIDLY : architecture généraliste pour la robotique
- Robotique cognitive
  - Qu'est ce que c'est ?
  - Qu'est ce que cela implique ?
- Lier des données géométriques et fonctionnelles (jumeaux numériques) à une base de données symbolique et donc SÉMANTIQUE pour faire des raisonnements fonctionnels et donc intelligence

## 3) Présentation du projet : Ancrage Sémantique (Anchoring)

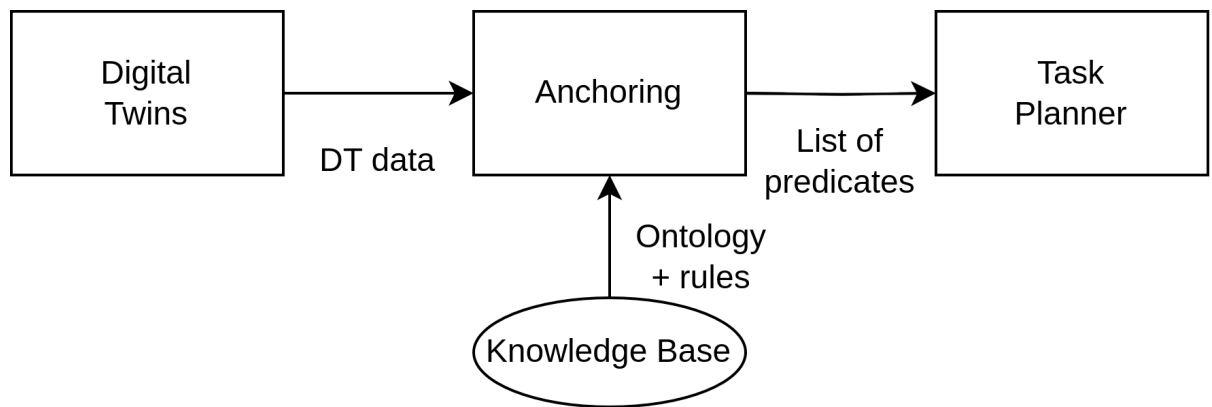
### a) Présentation rapide

- Schéma : Perception -> Planning -> Acting



- Perception : Capteurs -> **Jumeaux numériques** (Géométrie/Fonctionnel)
- Planification : **Anchoring** -> **Task Planner**
- Action : Controller -> Actuators

- Expliquer ce schéma :
  - Rôle de l'Anchoring
  - Place de l'Anchoring
- Schéma : DT -> Anchoring -> Task Planner



- 
- Anchoring -> Knowledge Base : Ontology + rules

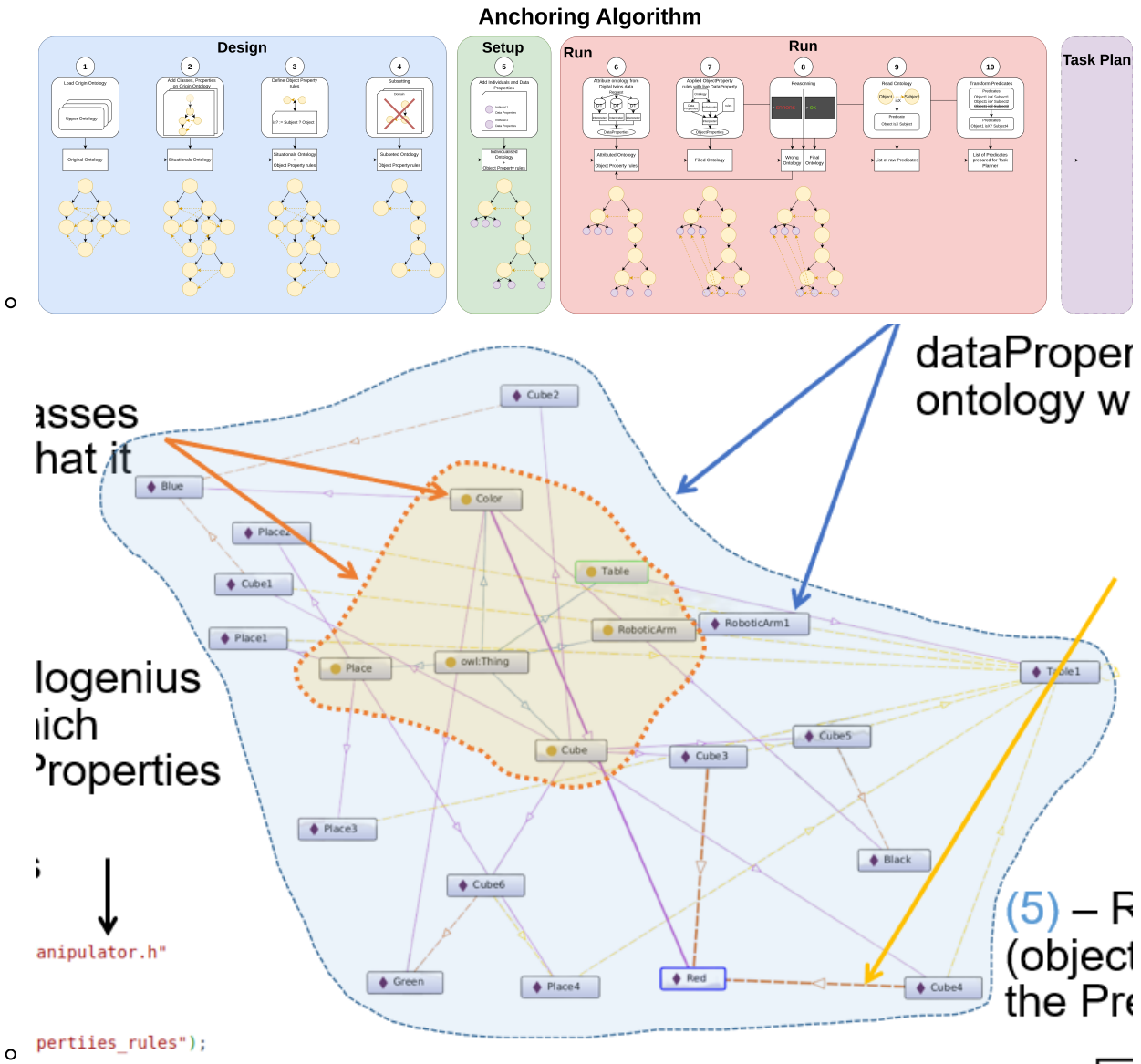
- Expliquer ce schéma

## b) Terminology

- Digital Twin - jumeaux numérique
  - Qu'est ce que c'est ?
  - DT géométrique + exemples
  - DT fonctionnels + exemples
- Task Planner - Planneur de taches
  - Qu'est ce que c'est ?
  - Exemples
- Ontologies (Développer cette partie)
  - Qu'est ce que c'est ?
  - Taxonomies associés voir taxonomie septembre
  - Exemples
  - Reasonner
  - Inferences
  - Illustrations

## c) Presentation Anchoring avec remplissage d'une ontologie

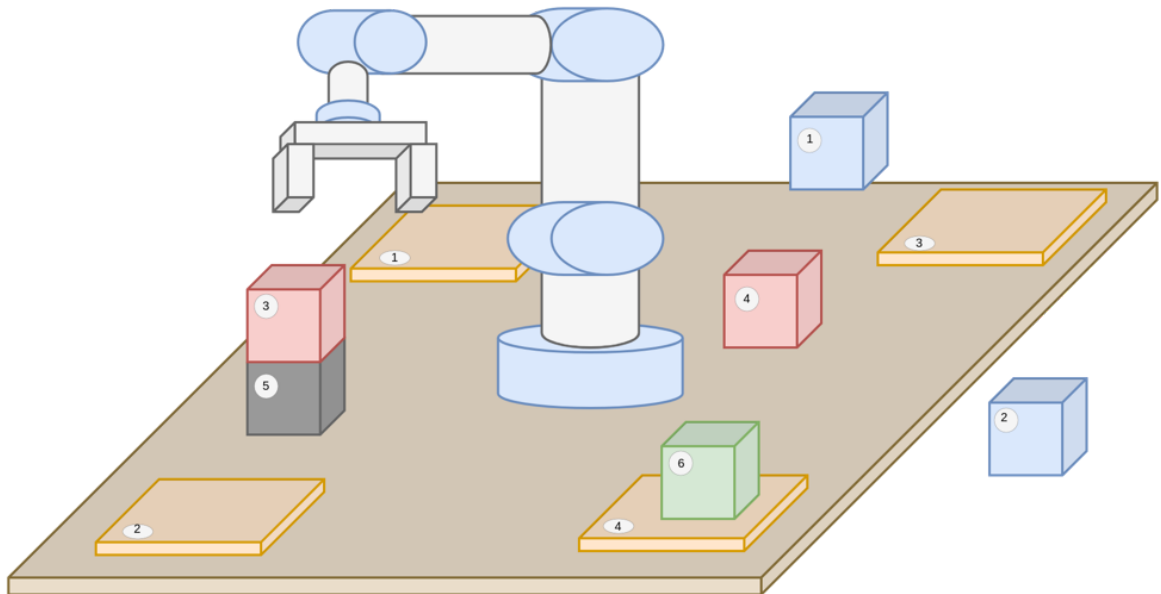
- Schema avec ontologie
  - Design
  - Setup
  - Run
- Schema ontologie : class - individus (10/10/23)



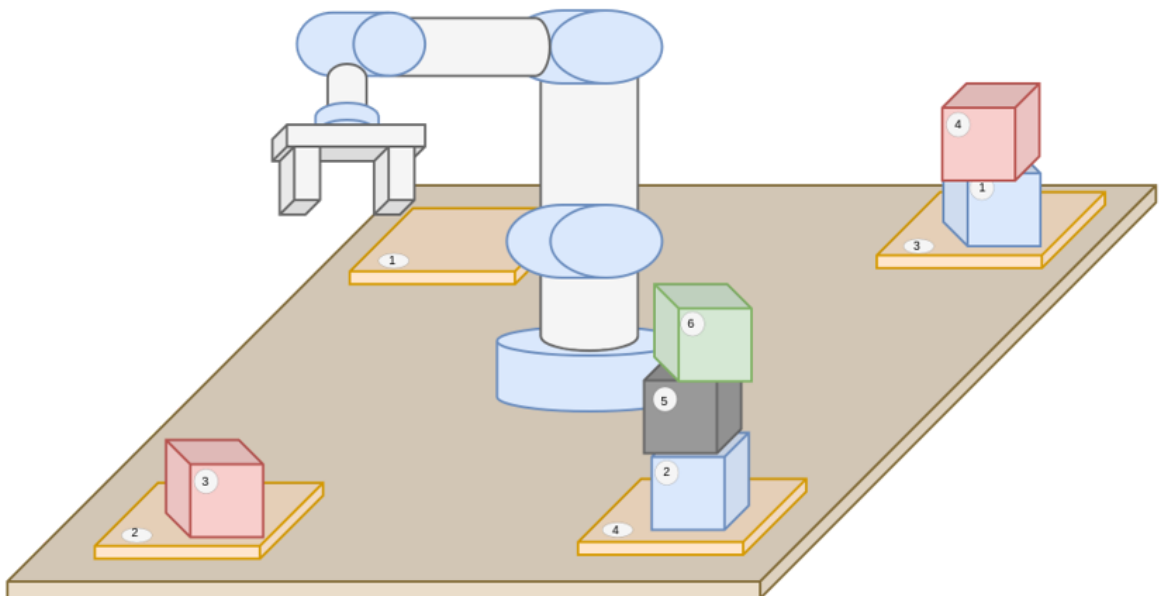
- Grace au raisonner et au regles -> IA

d) But du projet

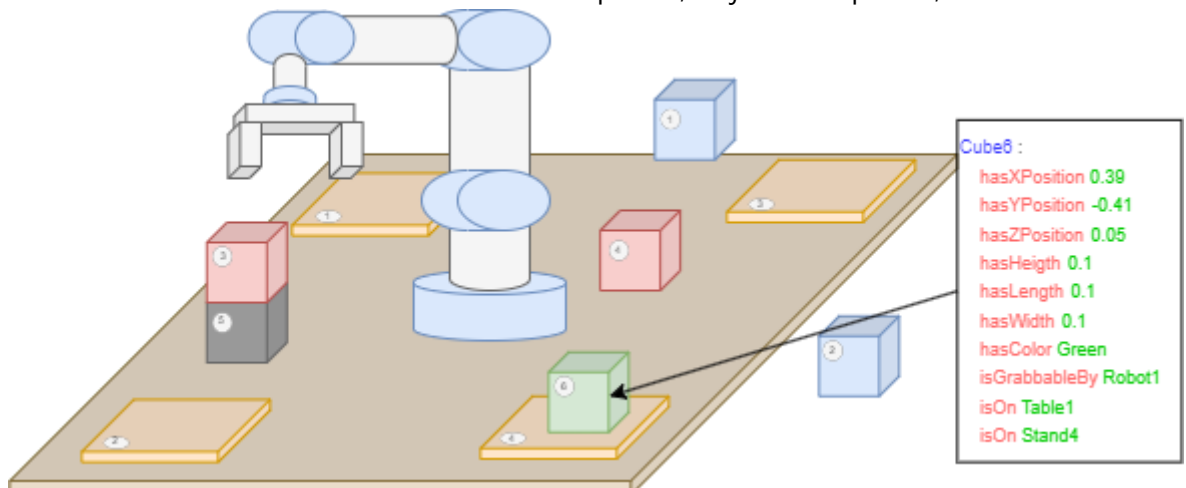
- Schema démonstration RUN 3IA
- Début



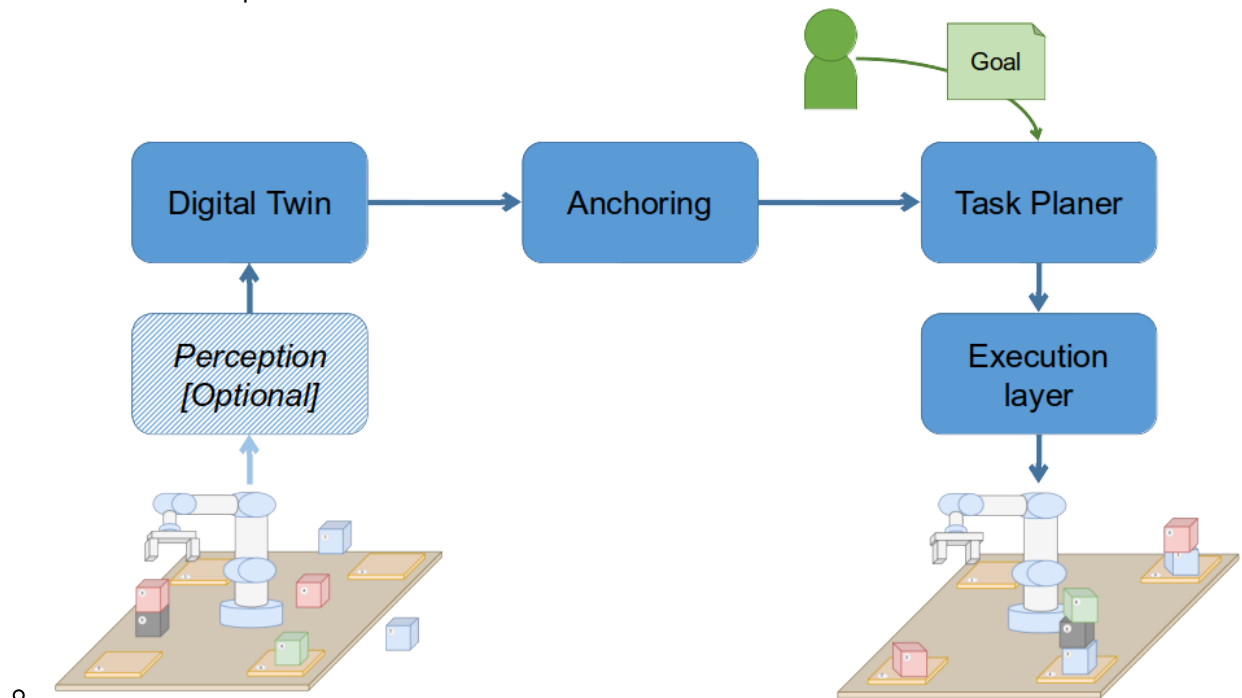
- Objectif



- Explications
- Comment elle a été créée ?
- Liste des différentes objets individuelles -> Classes
- Liste des différentes choses à faire attention : Data Properties, Object Properties, rules



- Explications
- Schema FeedBack Loop



- Explications
- Demonstration avec le Digital Twin XDE : Vidéos
  - Demander la vidéo à Raphaël

## II - Recherche du moteur ontologique

### 1) Choix du OWL

- Autre choix que le OWL :
  - OntoUML
  - First Order Logic
  - ....
- Facile
- Protege
- Upper Ontologie
- Communauté

### 2) Qu'est ce qu'on veut faire avec notre moteur ontologique

- Manipuler l'ontologie (S'appuyer sur les schémas du haut)
  - lire les classes / individus / Data Properties / Object Properties
  - Créer des classes / individus / Data Properties / Object Properties
- Reasonner
- Appliquer les regles SWRL
- Pouvoir etre utiliser avec ROS
- Rapidité pour robot

### 3) Protege

-  Protege avec Soma




- Atouts
  - visualisation
  - creation
  - on peut tout faire
  - Plugin
  - SWRL
  - ...
- Defaults
  - Programme -> pas ROS
- Conclusion
  - Programme majeur dans le developpement d'une ontologie


#### 4) OWLAPI / Jena

- Atouts
  - Api sur lequel repose Protege
  - La plus utilise pour manipulé du OWL
- Defaults
  - JAVA
- Conclusion
  - NON

#### 5) Ontolegenius

-  Logo Ontolegenius
- Atouts
  - Developper par un proche de la team
  - penser pour la robotique
  - Rapide (Benchmark)
  - Python / C++
  - ROS2
  - Developpement Actif
  - Developpement sous forme de plugin => contribution "facile"
- Defaults
  - Pas de SWRL
  - Manque de raisonner avec SWRL
  - Documentation pas super clair
  - ...
- Conclusion
  - Une solution a garder l'oeil
  - Pas garder par manque de SWRL

#### 6) Owlready2



-  Logo Owlready2
- Atouts
  - Python
  - Actif dans le developpement

- Reasonner pellet/HermiT
- Rapide (Benchmark)
- On peut tout faire
- Documentation + livre
- Defaults
  - Pas assez rapide a cause du reasonner java pellet
  - Documentation + livre incomplete das mon use case
- Conclusion
  - Moteur Ontologique garder garce au python, sa simplicité, et reasonner pellet

## 7) KnowRob

- Atouts
  - Université Bremen
  - relire Article
- Defaults
  - relire Article
- Conclusion
  - NON

## 8) Autre moteur d'ontologie

- Cowl
  -  Logo Cowl
  - Atouts
    - lightweight
    - C++
    - -developpemet actif
  - Defaults
    - Pas de reasonner du tout
  - Conclusion
    - NON
- ORO
  -  Logo ORO
  - Atouts
    - CNRS Sarthou
    - C++
  - Defaults
    - Pas de reasonner du tout
    - Plus dev depuis 2012
  - Conclusion
    - NON
  - Armor RDS
    - Atout
      - Penser Ros Docker
    - Defaults
      - Plus Dev

- Juste un Test
- Pas de raisonner
- NON

7) Qu'est ce qu'on peut améliorer ?

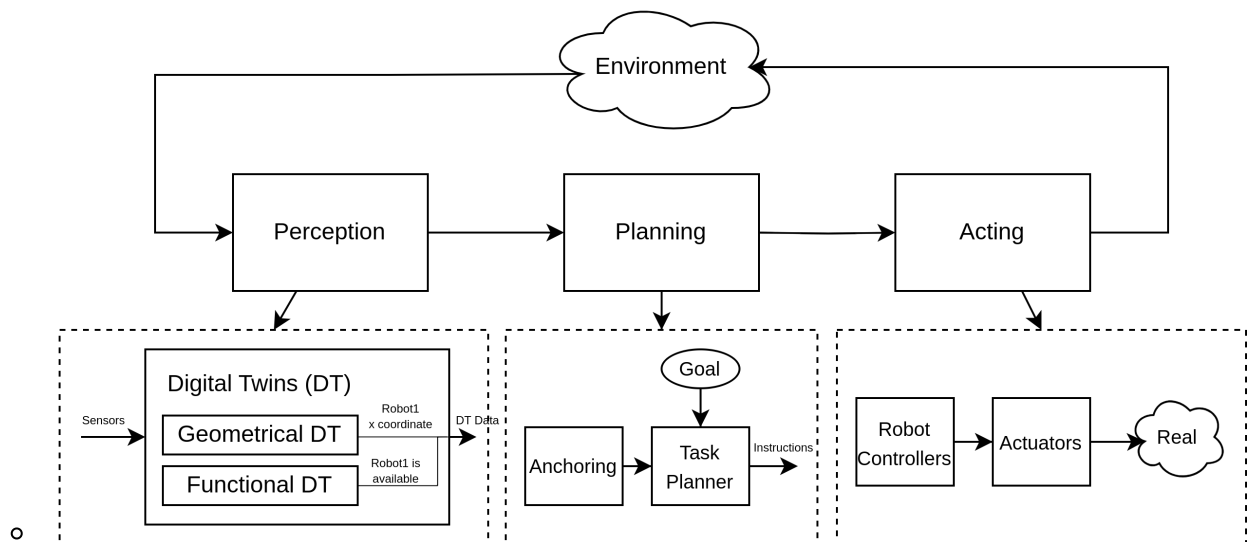
À mettre à la fin plutôt

### III - Algorithmme

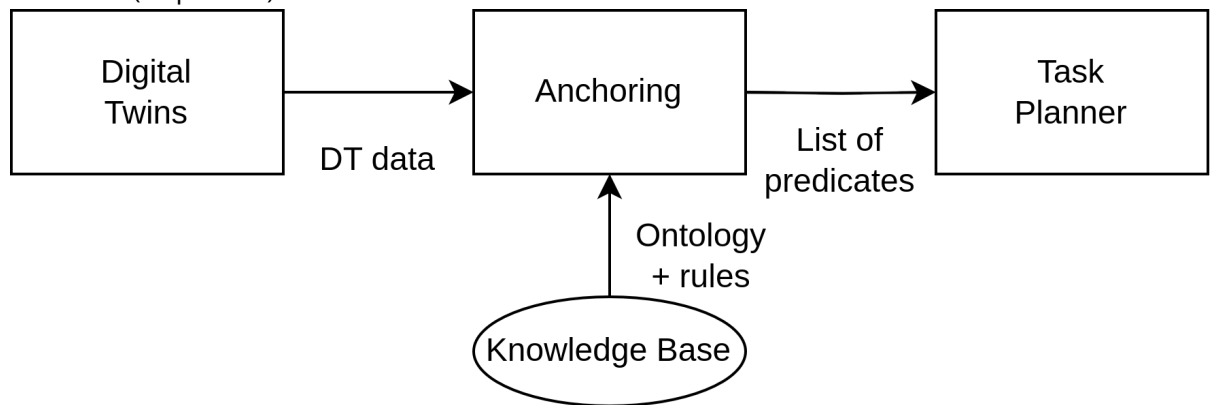
#### 1) Presentation Algorithmme


##### a) Algorithmme Simple Septembre

- Algorithmme Global



- Algoritme Local (Duplex DT)

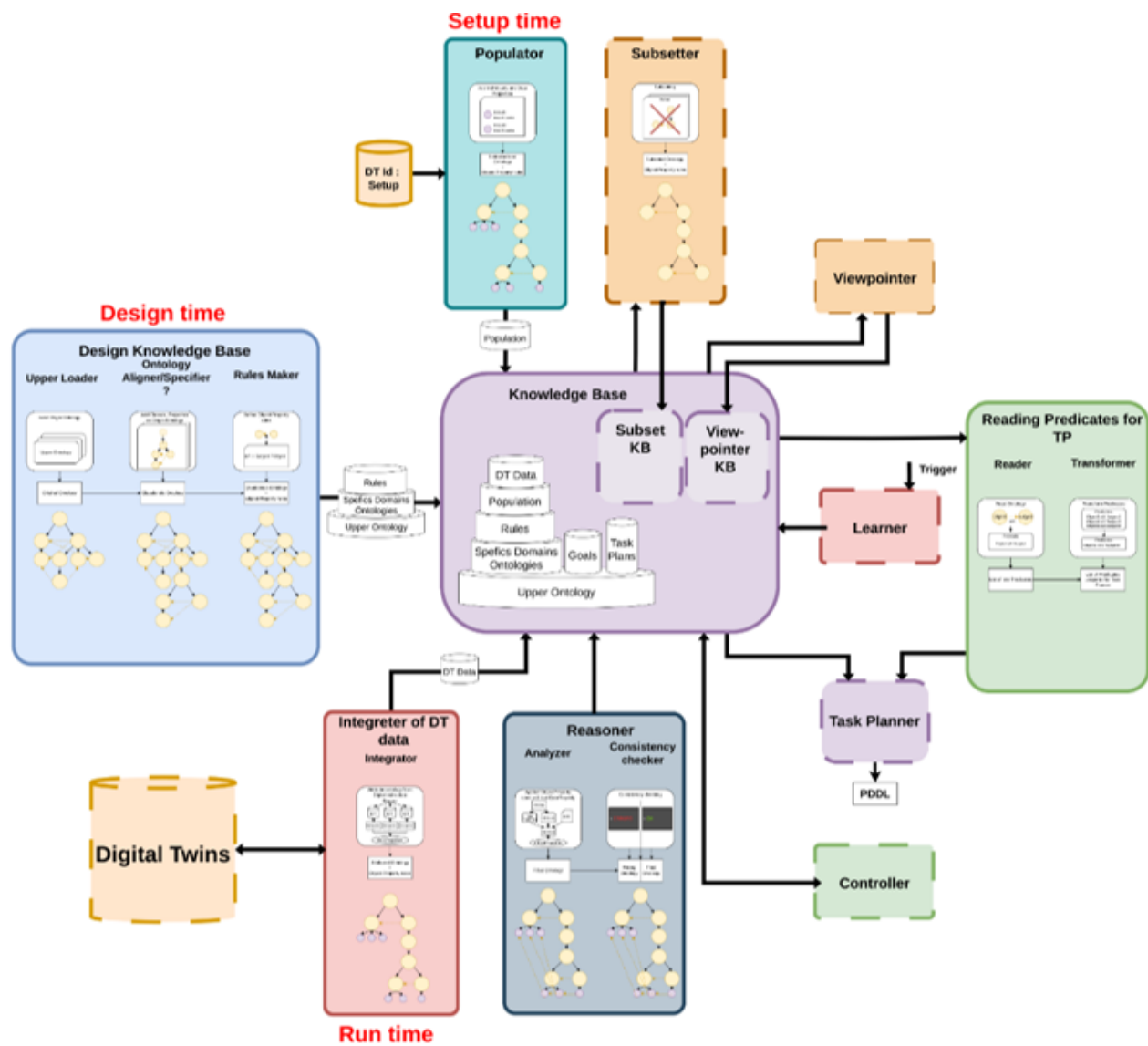


- Algorithme Micro
  -  Algorithme micro (à modifier)
- Comment on ils été pensé ?
- Quels sont les problemes ?

##### b) Algorithme Linéaire

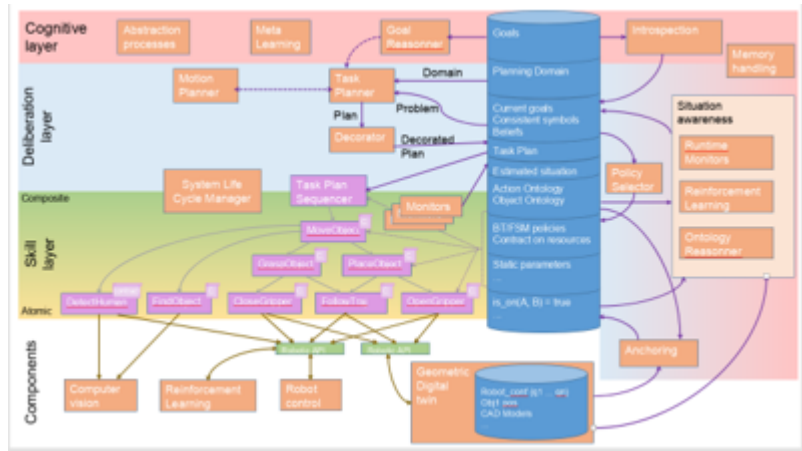
- Extension des premier algorithme avec la notion d'ontologie
- Comment a il été pensé ?
- Quels sont les problemes ?

2) Algorithme : Ontology Manager

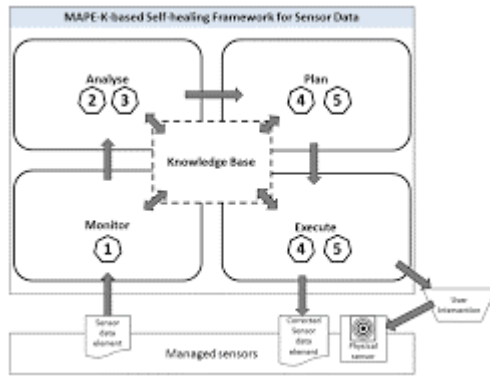


a) Besoins

- Squidly Compatible



- MAPE-K compatible



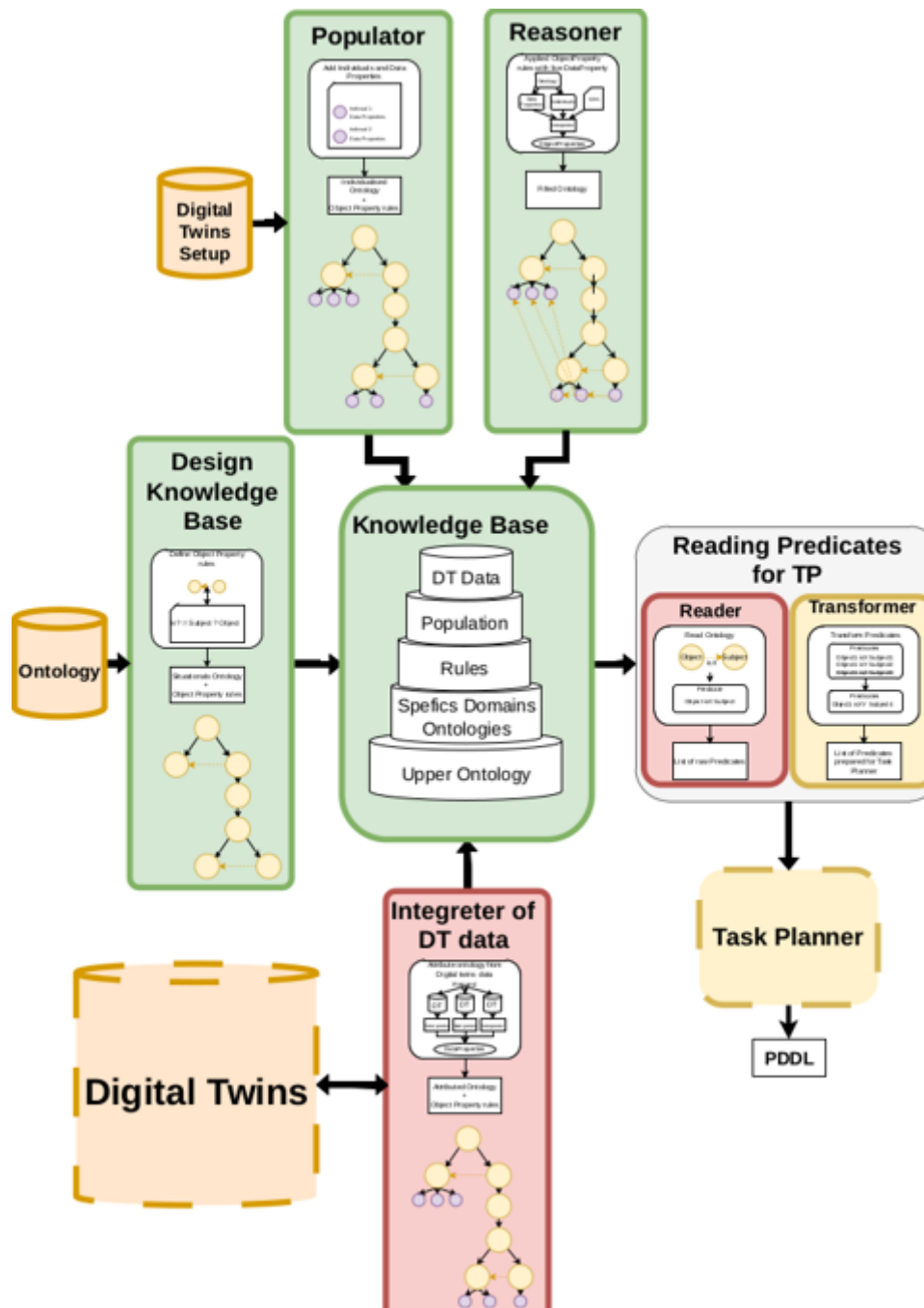
- 
- Decentralisation
- Penser Node Ros + Behavior Tree
- Modulable
  - learner
  - viewpoint
  - substracter
- Les point d'améliorations

## b) Présentation de l'algorithme

- Schéma voir au dessus
- Superposition des bloc = dépendance plugin
- Explications (Bien détailler)

## c) Réduction de Ontology Manager dans le cadre du projet

- Pourquoi?
- Quels Blocs Garder
- Schema :



o

### 3) Algorithme réduit : Algorithme implémenté

#### a) Explications de l'algorithme

- Design
- Setup
- Run

#### b) SYSML

- Annexe
- Comment a il été créer ?
- [Le mettre a jour ?]

#### c) Implementation ROS Squidly

- Diagramme de sequence
- Explication du knowlege domain
- Explication des path
- Explication du overwrite etc...

## IV - Implementation

L'implementation a été fait dans un premier temos avec des script python independent de ros/Squidly puis ensuite implemente dedans.

### 1) Design

#### a) Role du design

- Creer une ontologie de base qui a :
  - une base ontologique
  - qui convient à la situations
  - qui convient au domaine
  - qui convient aux node utiliséq
- Montrer le bloc Design
  - [Ontology manager Design](#)
- Expliquer les étapes

#### b) Choix de l'Upper Ontologie

- C'est quoi une upper ontologie ?
- Pourquoi ?
- Listes des Upper Ontologies :
  - IEEE1872
    - Description : +/-
  - SUMO
    - Description : +/-
  - ORO (ground)
    - Description : +/-
  - UFO
    - Description : +/-
  - SkiRos2
    - Description : +/-
  - DUL
    - Description : +/-
  - DOLCE
    - Description : +/-
  - ROBOTML
    - Description : +/-
  - ROSETTA
    - Description : +/-
  - SOMA
    - Description : +/-

- Je choisi SOMA car ...

### c) Présentation de SOMA

- Université de Bremen
- Laboratoire de IA
- Cognitive Robotic
- Social
- Goal
- ...

### d) Modification de SOMA

- Refait SOMA en enlevant des relation car marche pas avec pellet penser pour HerMiT
- Substract avec seulement ce dont j'ai besoin
- Toujours compatible Soma

### e) Ajout du Mid et du Domain

#### i) Mid ajout de la compatibilité avec les autres Node

- Compatibilité avec le DT
- Compatibilité avec le Task Planner

#### ii) Domain

- Ajout pour le domain robotics

Transition : Ajout des individus -> Setup + Run

### f) Comment le Design/domain est fait ?

- Design fait à la main avec protege
- Load par l'ontology manager lors du ros launch

## 2) Setup

### a) Role du setup

- Populer l'ontologie avec :
  - les individus initial
  - les configuration des nodes associé :
    - Digital Twin
    - Task Planner
  - Alias

### b) Comment le populator est fait ?

- Parsing de fichier json préalablement fait



- Ajout avec le moteur ontologique
- Ontology manager lors du ros launch ou peut etre activer a n'importe quel moment
- Explication de processus dans le code avec owlready2

### 3) Run Time

#### a) Role du run time

- Etre run time pour que l'ontology représente en live le réel

#### b) Intégrateur

##### i) Role de l'intégrateur

- C'est un peu juste ça l'"anchoring"
- Remplir l'ontology avec des data propriété que le DT lui donne
- Puis les faire valider par l'ontology manager pour qui infere pour créer des relations

##### ii) Comment l'intégrateur est fait ?

- parsing d'un message ros
- Ajout dans l'ontology avec le moteur ontologique
- Parler du processus dans le code avec les Alias/ID
- Parler du processus dans le code pour écraser les Data properties
- Parler du processus dans le code pour que ce soit automatique peut importer la propriété à remplir si il y a bien un alias
- Puis les faire valider par l'ontology manager pour qui infere pour créer des relations

#### c) Rulemaker

##### i) Role du rulemaker

- Ajout de règles python pour palier au problème du SWRL

##### ii) Comment le rulemaker est fait ?

- recherche et ajout dans l'ontology avec owlready 2
- Parler des règles
- Parler du processus de suppression de propriété fausse

#### d) Reasonner

##### i) Role du reasonner

- assure la cohérence de l'ontology
- infere
- Applique les SWRL rules
- pellet / Hermit

## ii) Comment le raisonner est fait ?

- Ligne python avec owlready2

## e) Reader

### i) Role du reader

- Transformer les fait sémantique de l'ontologie en predicat pour un Task Planner
- peut transforer pour plusieurs Task Planner PDDL / HRT / ...

### ii) Comment le reader est fait ?

- Parsing avec owlready 2 : [[ObjectProperty, Subject, Object], ...]

## V - Apprentissages, critiques, autres chose à explorer

### 1) Ontologies

- Apprentissages :
  - Domain ou l'IRT a appris et mis en pratique
  - Application de Semantique dans la robotique
  - relation symbolique
- Critiques :
  - Il faut etre expert en ontologies
    - j'ai moi meme pas ete rigoureux pour des expert en ontologie avec soma
  - interaction avec compliqué
  - domain tres vaste et donc peu clair
  - Schema : 2 ontologie differentes meme signification
- Autres chose à explorer :
  - Algo pour chaque regle (revient a ce qu'on fait avec rule maker) + SQL/DataLake

### 2) SWRL

- Apprentissages :
  - Pas de négations
  - pas de division dans aucun des raisonner
  - dependant du raisonner
  - peu utilisé par la communaute
  - déconseille de l'utilise par la communauté
  - long pour ecrire des regle
  - pas de var tmp
  - pas possible d'ajouter et/ou d'écraser des Data properties
- Critiques :
- Autres chose à explorer :
  - Biblioteque de regle python en axioms

### 3) Moteur Ontologique

- Apprentissages :
  - Owlready super bien mais utilise pellet qui est lent
- Critiques :
- Autres chose à explorer :
  - Maintenant plus de SWRL + maj de ontologienius pour ROS 2 => essayé Ontologienius

#### 4) Algorithme

- Apprentissages :
- Critiques :
- Autres chose à explorer :

#### 5) Design

- Apprentissages :
- Critiques :
  - Pas automatique
  - Pas import
  - En un seul bloc
- Autres chose à explorer :
  - Le faire automatiquement en dennat une UPPER les differentes nodes ros pour definir les domain ontologies a prendre et le tout avec des import

#### 6) Setup

- Apprentissages :
- Critiques :
- Autres chose à explorer :
  - Creer les ficheir de setup de node automatiquement
  - Creer le fichier des setup de la situation automatiquement

#### 7) Intergrator

- Apprentissages :
- Critiques :
- Autres chose à explorer :

#### 8) RuleMaker

- Apprentissages :
- Critiques :
- Autres chose à explorer :

#### 9) Reasonner

- Apprentissages :
  - Pellet c'est bien
  - pellet c'est puissant
  - Pellet necessaire pour SWRL

- Critiques :
- Autres chose à explorer :
  - On pourrait essayer Hermit ou essayer un resonner tiers plus rapide => Ontologenius

## Moi-même

- Apprentissages :
  - la recherche
  - le langage formel
  - la semantique
  - entreprise
  - cognitive robotique
- Critiques :
  - mauvaise estimation du temps
- Autres chose à explorer :

## Conclusion