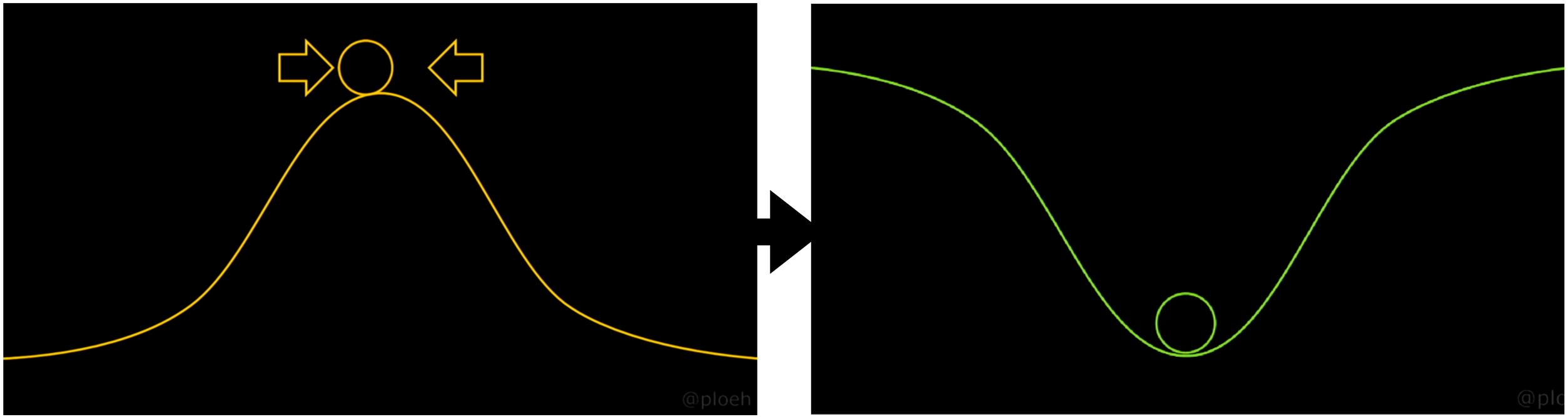
Rust-Botic Made My **DDD**ay

Luis Parada & Pierrick Mirabel


Time Destination Flight Gate Status
12:00 HONG-KONG HK4701 A56 CANCELLED
12:03 LONDON HT964 D15 CANCELLED
12:03 NEW YORK HK4701 B56 CANCELLED
12:12 Cutii HK487 C12 LA ROSE MAY
12:25 BUENOS AIRES BA2578 B6 CANCELLED
12:26 DUSSELDORF DS4307 E4 CANCELLED
12:40 OSLO OS258 B10
DUBAI DB1234 C31
OLOGNA BL9875 A4


PACBOT

Contexte



Mark Seeman - Functionnal architecture : pits of success

Robot Operating System

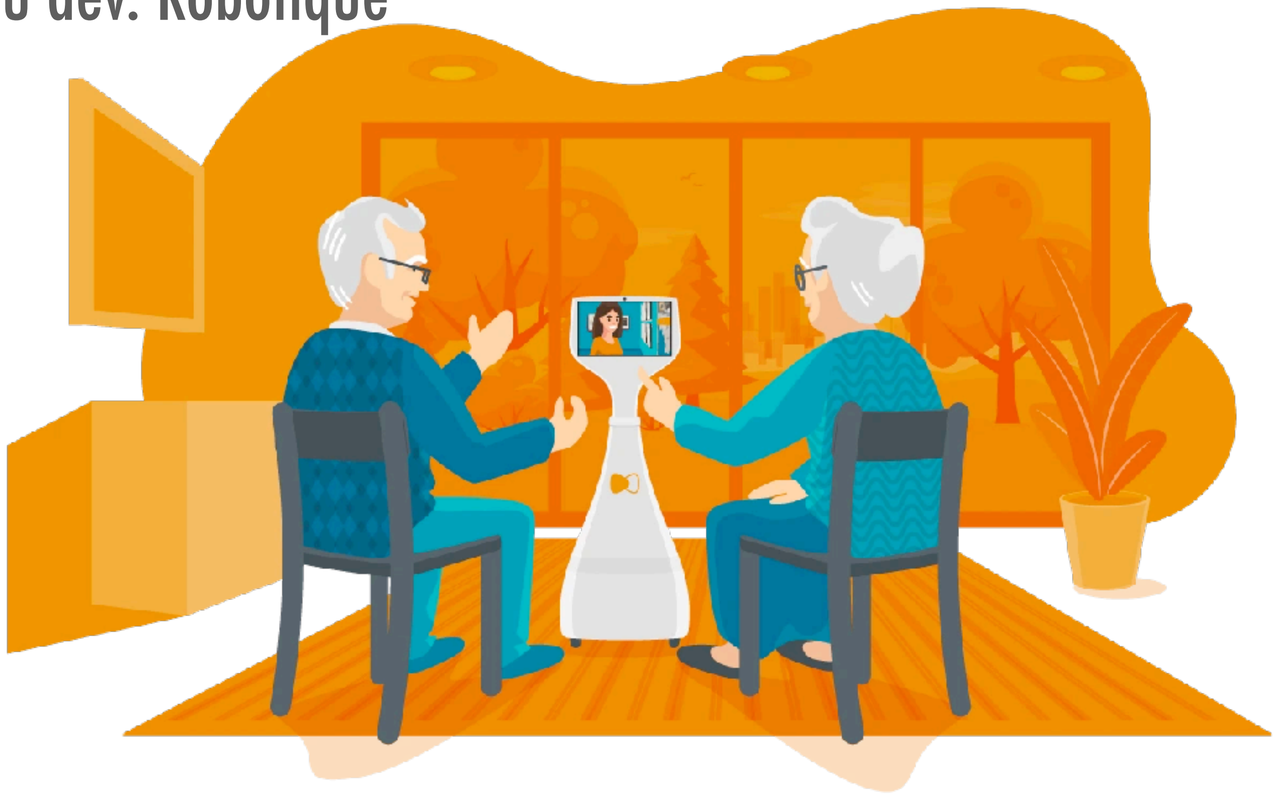
Problématique liées au dev. Robotique

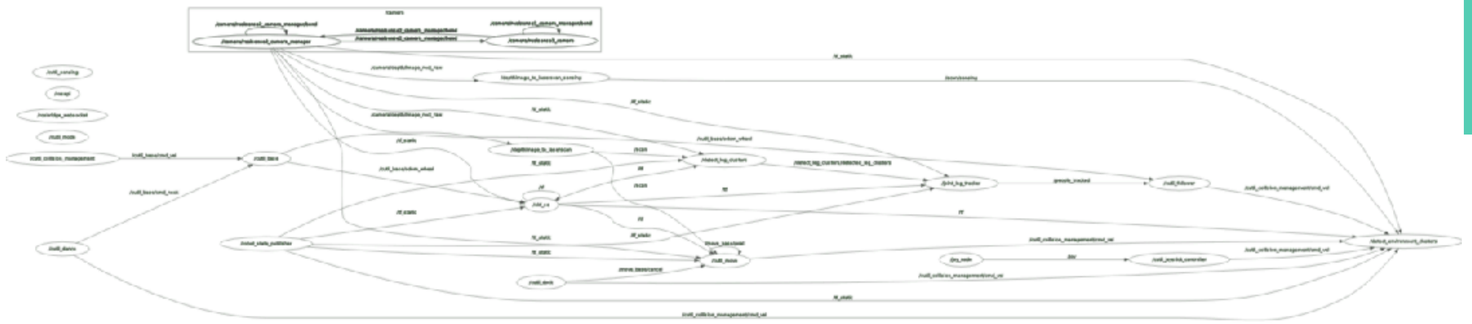
MOTORISATION

COU RÉTRACTABLE

NOEUD PAPILLON LUMINEUX

BANDEAU LUMINEUX



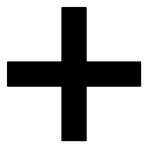


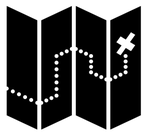
CAMÉRA 2D

CAPTEURS DE PROXIMITÉ

CAMÉRA 3D

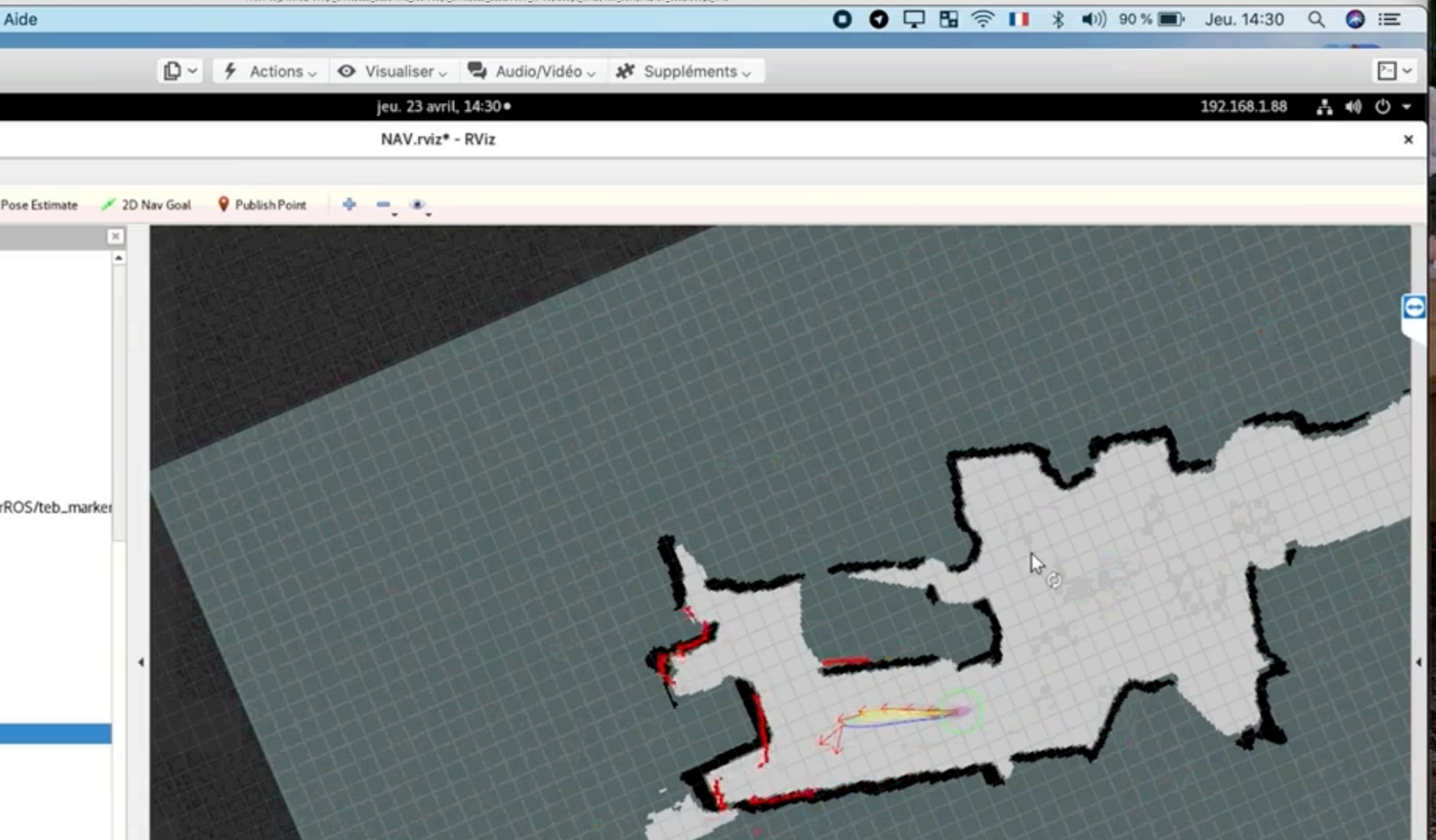
ROS = accélérateur

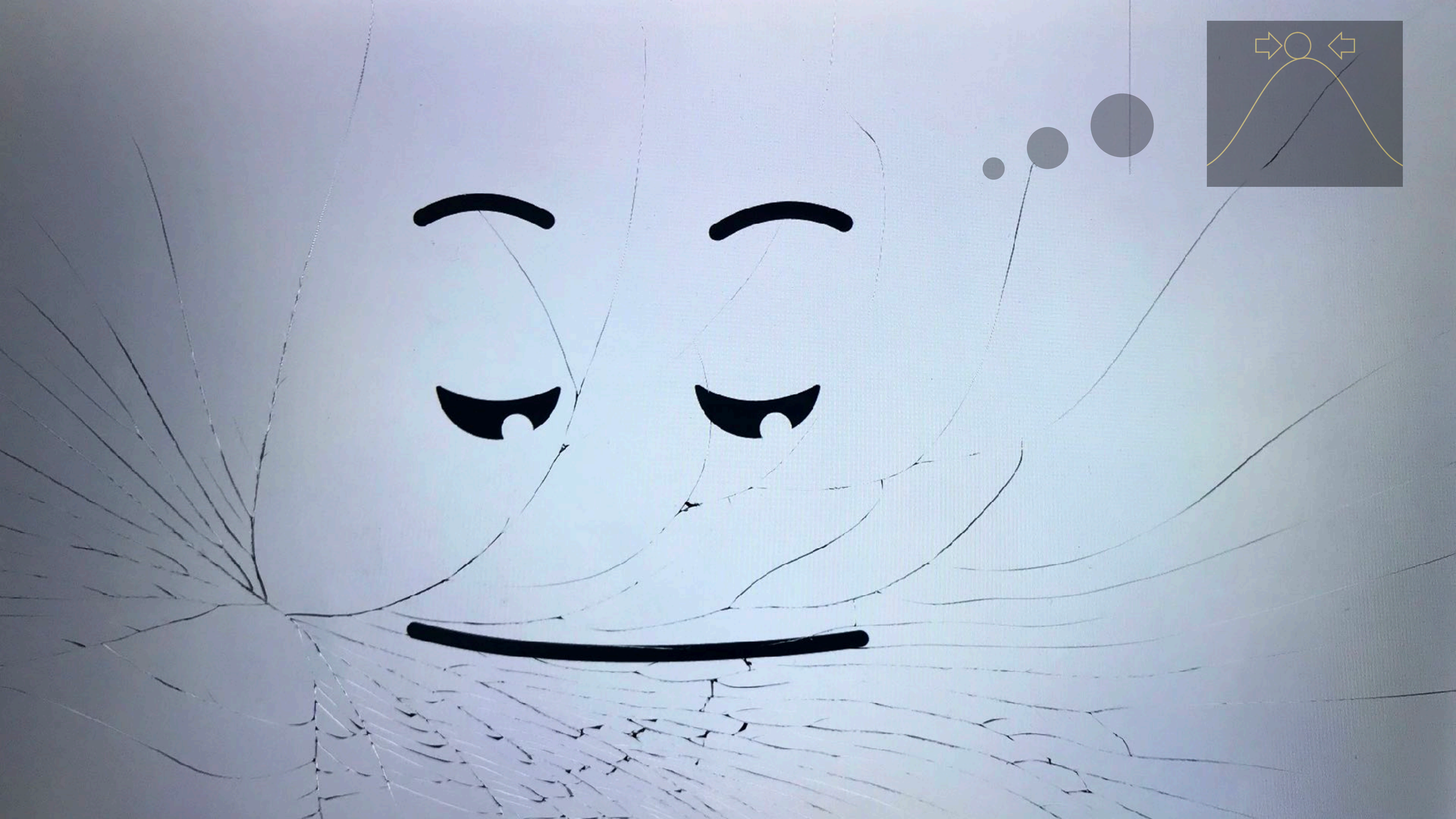





12
14
16
18
20
22
24
26
28
30
32
34
36
38

camera_link
camera_rgb_frame
camera_depth_frame
camera_rgb_optical_frame
camera_depth_optical_frame
object_position
vision
object_position0


Aide
90 %
Jeu. 14:30
Actions
Visualiser
Audio/Vidéo
Suppléments
jeu. 23 avril, 14:30
192.168.1.88
NAV.rviz* - RViz
Pose Estimate
2D Nav Goal
Publish Point

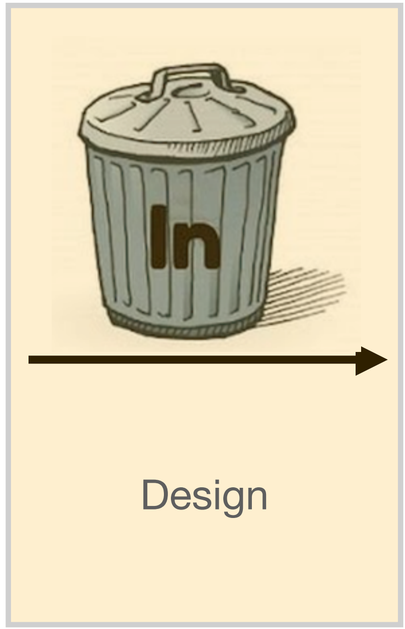


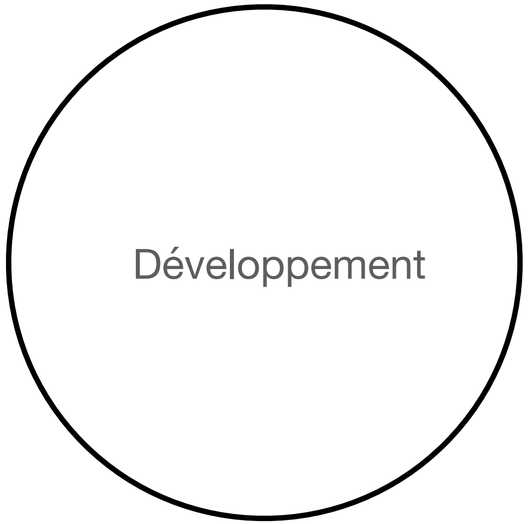
Changement d'approche: DDD

Approche DDD : Processus de développement

Dis papa,

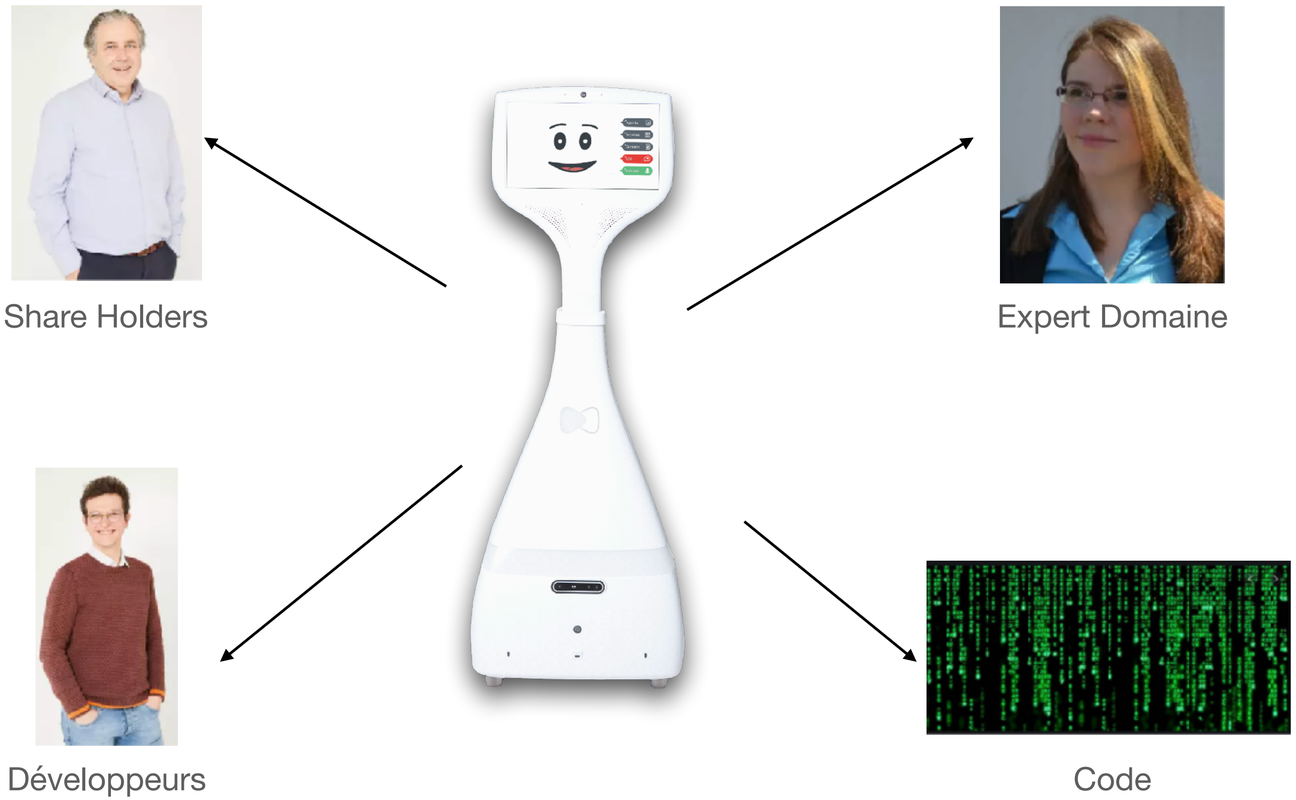
Comment on fait DDD ?





Approche DDD : Domaine

Modèle partagé du Domaine



Approche DDD : Développer un Lexicon « commun »

String ?= 

Approche DDD : L’importance du design

```
Type RobotConfiguration = {
    Collision_avoidance_distance_front : float
    Collision_avoidance_distance_back : float
    Speed: float
}

Type BatteryState = {
    // 1 for low, 2 for medium, 3 for high
    Battery_state_of_charge: integer
    Is_robot_charging: bool
}
```

Cela a-t-il du sens d’avoir une distance ou vitesse = 10^37 ?

Comment la désactivation est elle représentée

Que signifie une vitesse négative ?

Représentation d’états non gérés

« Vrai » si en charge ou si branché ?

Quid si branché et batterie pleine?

Approche DDD : Composition

Robot Mode

StandBy **OU** RemoteControl **OU** Follow **OU** **.....**

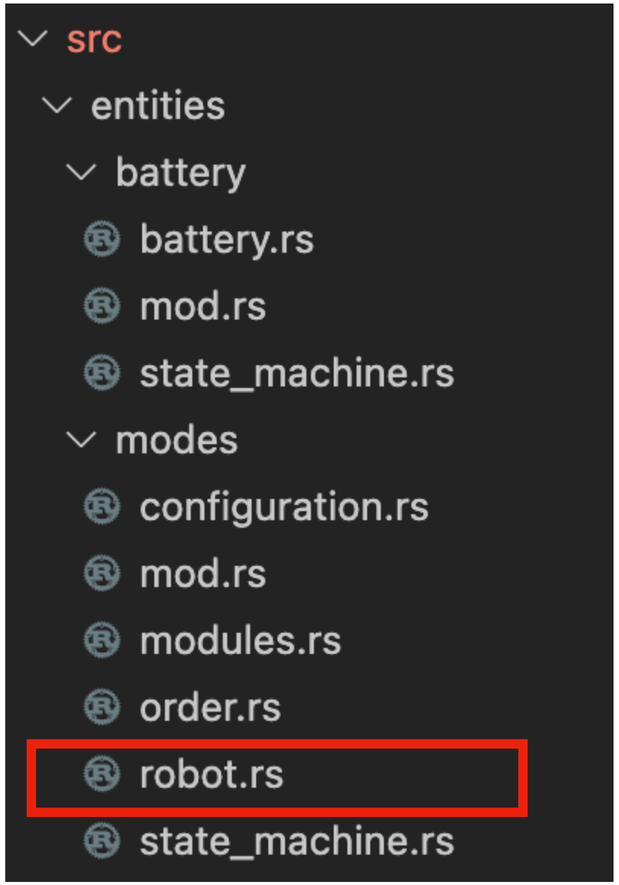
Modules Mode

SpeedMode **ET** NeckMode **ET** CollisionDetectionMode **ET** CollisionAvoidanceMode

Modules Configuration

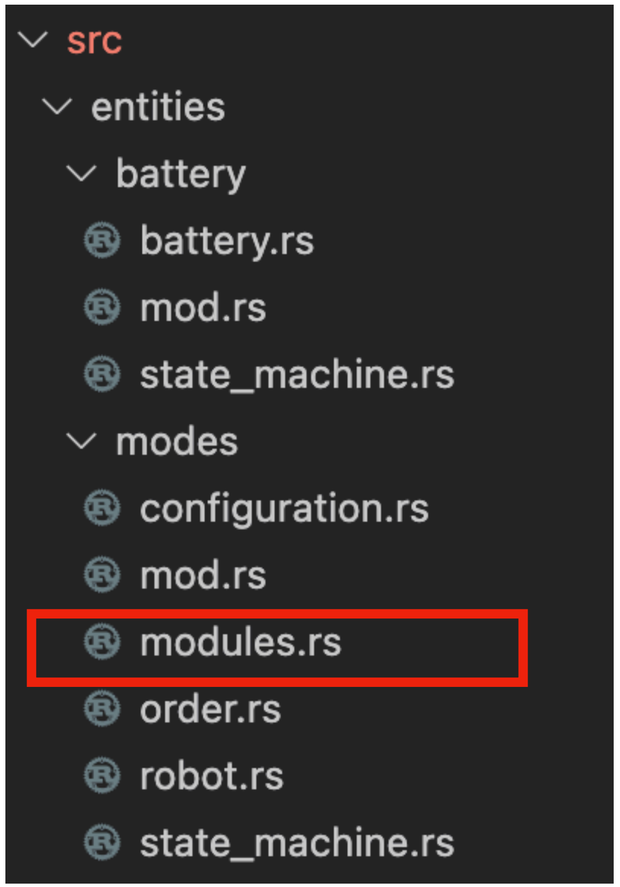
CollisionAvoidanceConfiguration **ET** CollisionDetectionConfiguration **ET** SpeedConfiguration **ET** NeckConfiguration **ET** **....**

Approche DDD : Communiquer le design dans le code



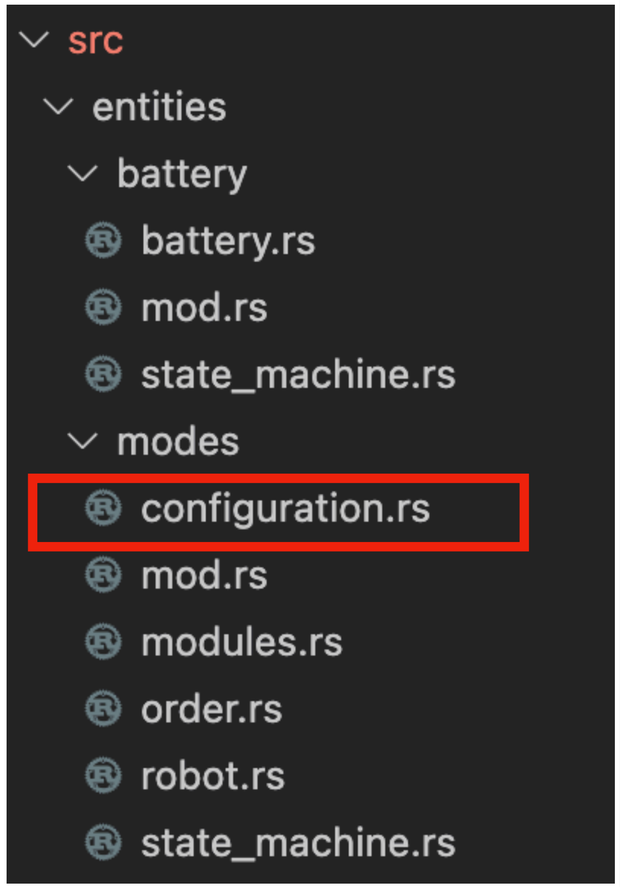

```rust
enum RobotMode {
    StandBy,
    RemoteControl,
    Follow(FollowMode),
}
enum FollowMode {
    Search,
    Move,
}
```

Approche DDD : Communiquer le design dans le code



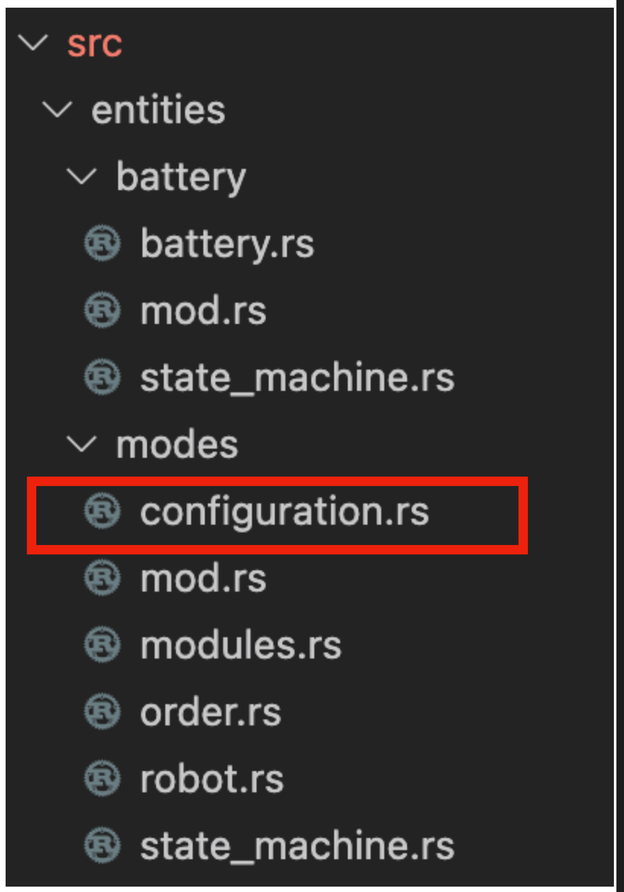

```rust
struct ModulesMode {
    collision_avoidance_mode: CollisionAvoidanceMode,
    speed_mode: SpeedMode,
}

enum CollisionAvoidanceMode {
    Off,
    Agressive,
    Precision,
}

enum SpeedMode {
    Agressive,
    Precision,
}
```

Approche DDD : Communiquer le design dans le code



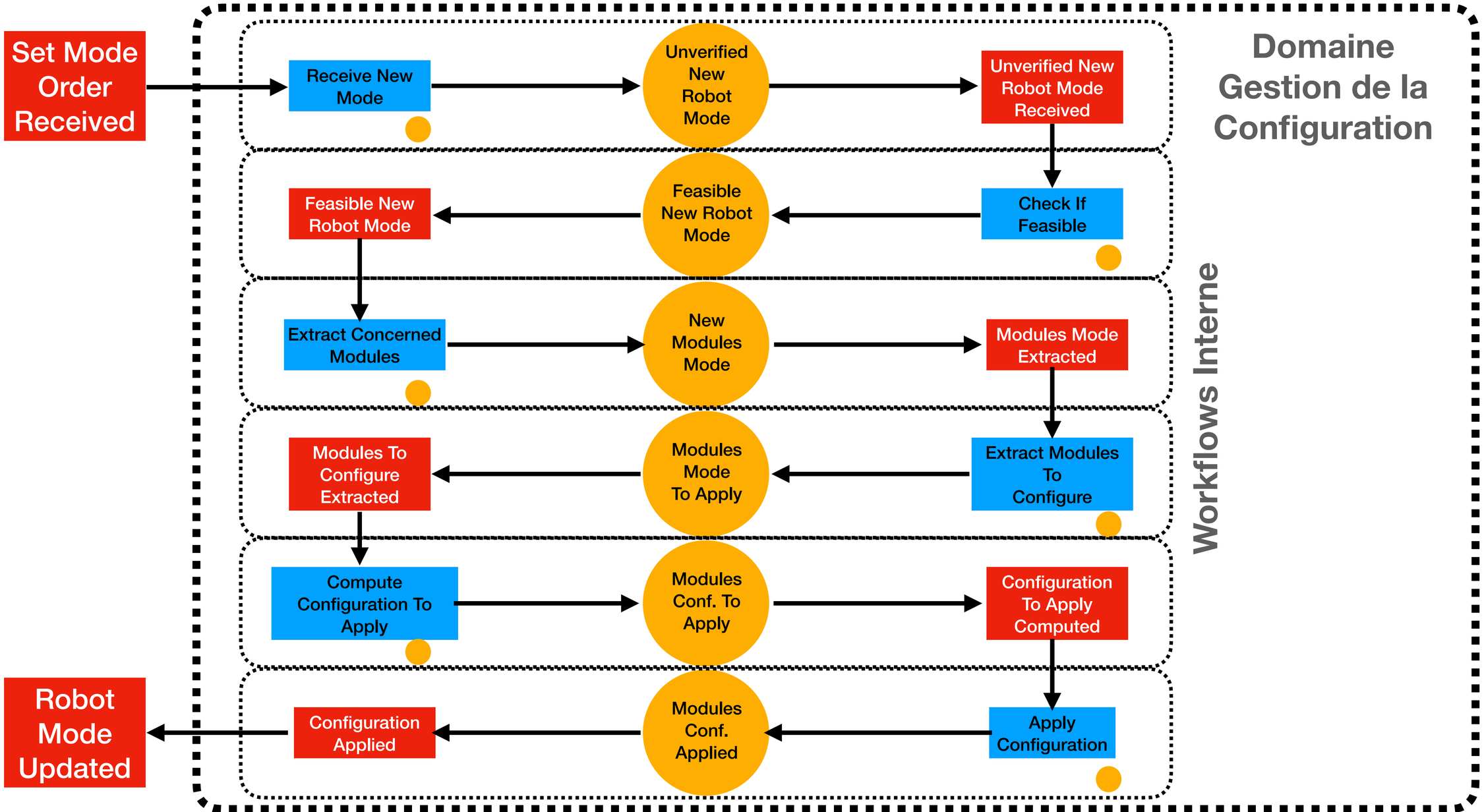

```rust
struct CollisionAvoidanceConfiguration {
    min_front_distance_threshold: CollisionAvoidanceValue,
    min_rear_distance_threshold: CollisionAvoidanceValue,
    min_angular_distance_threshold: CollisionAvoidanceValue,
}
struct SpeedConfiguration {
    max_frontward_vel: SpeedValue,
    max_absolute_backward_vel: SpeedValue,
    max_absolute_angular_vel: SpeedValue,
    max_speed_delta: SpeedValue,
}

struct ModulesConfiguration {
    collision_avoidance_configuration: Option<CollisionAvoidanceConfiguration>,
    speed_configuration: Option<SpeedConfiguration>,
}
```

Approche DDD : Communiquer le design dans le code

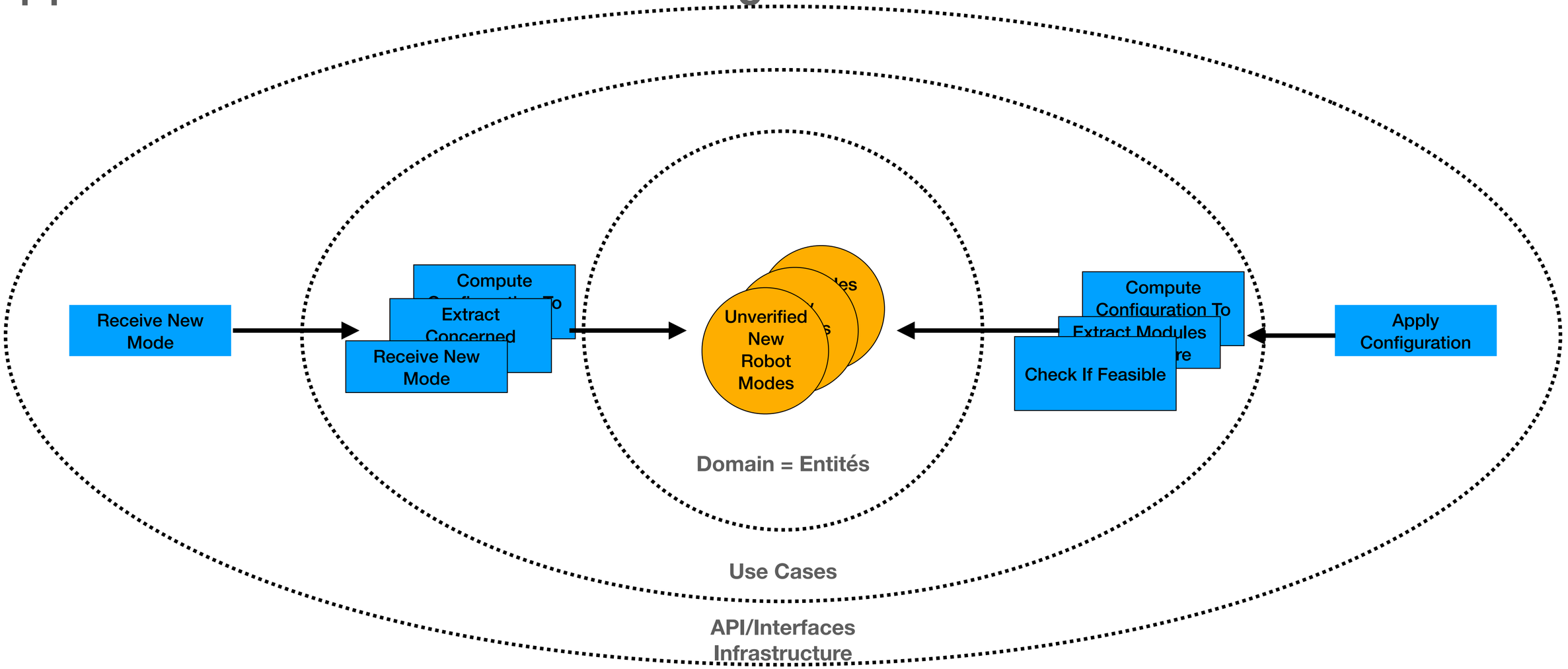



```rust
impl ConfigureSpeed for SpeedConfiguration {
    fn agressive() -> Self {
            SpeedConfiguration {
                max_frontward_vel: SpeedValue::new(1.0),
                max_absolute_backward_vel: SpeedValue::new(0.5),
                max_absolute_angular_vel: SpeedValue::new(1.0),
                max_speed_delta: SpeedValue::new(0.8),
            }
        }
       fn precision() -> Self {
            SpeedConfiguration {
                max_frontward_vel: SpeedValue::new(0.10),
                max_absolute_backward_vel: SpeedValue::new(0.05),
                max_absolute_angular_vel: SpeedValue::new(0.5),
                max_speed_delta: SpeedValue::new(0.4),
            }
        }
    }
```

Approche DDD : Communiquer le design dans le code



Approche DDD : Le modèle en Oignon

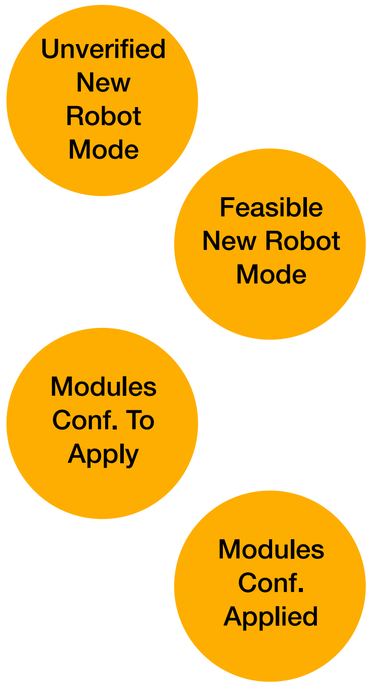


Workflow

Approche DDD: Workflow as functions

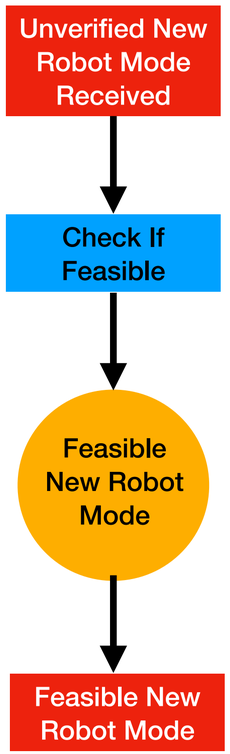
```rust
pub fn set_new_mode(new_mode: RobotMode) -> SetModeStateMachineResult
{
  SetModeStateMachine::receive_new_mode(new_mode, current_mode)?
        .check_if_feasible()?
        .extract_concerned_modules()
        .extract_modules_requiring_configuration()
        .compute_configuration_to_apply()
        .apply_configuration()?
        .update_robot_mode()
}
```

Approche DDD: Workflow as a FSM : states



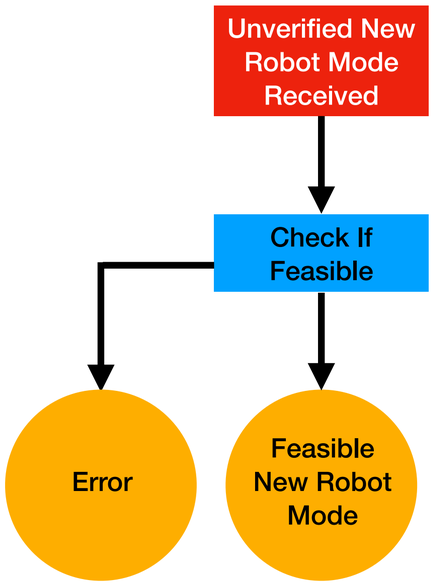

```rust
/// State Machine structure
struct RobotSetModeStateMachine<S> {
    mode_to_set: RobotMode,
    current_robot_mode: CurrentRobotMode,
    state: S,
}

/// State Machine entry point state
struct UnverifiedNewRobotModeState {
    new_mode_request: UnverifiedNewRobotMode,
}

struct FeasableNewRobotModeState {
    new_mode_feasible: FeasableNewRobotMode,
}

struct ModulesConfigurationToApplyState {
    modules_configuration_to_apply: ModulesConfigurationToApply,
}

struct ModulesConfigurationAppliedState {
    modules_configuration_applied: ModulesConfigurationApplied,
}
```

Approche DDD: Workflow as a FSM : transitions



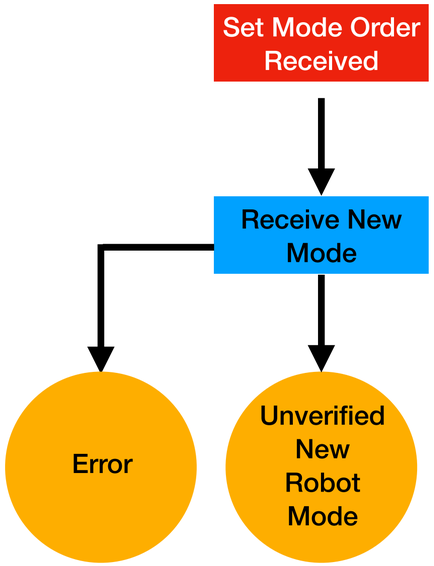

```rust
impl RobotSetModeStateMachine<UnverifiedNewRobotModeState> {
    pub fn check_if_feasible(self) -> Result<>{

        let new_mode_feasible =
                        check_if_feasible(self.state.new_mode_request, self.current_robot_mode)?;

       Ok(RobotSetModeStateMachine {
            mode_to_set: self.mode_to_set,
            current_robot_mode: self.current_robot_mode,
            state: FeasableNewRobotModeState {
                new_mode_feasible
            },
        })
    }
}
```

Approche DDD: Workflow errors



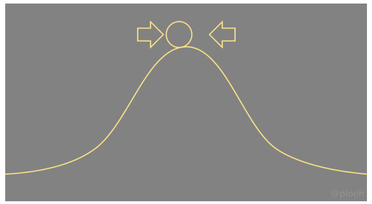

```rust
type FeasableNewRobotModeResult = Result<FeasableNewRobotMode, FeasableNewRobotModeError>;

struct RobotSetModeStateMachineError {
    pub kind: RobotSetModeStateMachineErrorKind,
    pub message: ErrorMessage,
}

impl From<FeasableNewRobotModeError> for RobotSetModeStateMachineError {
    fn from(from_error: FeasableNewRobotModeError) -> Self {

        RobotSetModeStateMachineError {
            kind: RobotSetModeStateMachineErrorKind::FeasableNewRobotModeError,
            message: from_error.message,
        }
    }
}


pub enum RobotSetModeStateMachineErrorKind {
    UnverifiedNewRobotModeError,
    FeasableNewRobotModeError,
    ConfigurationApplicationError,
}
```

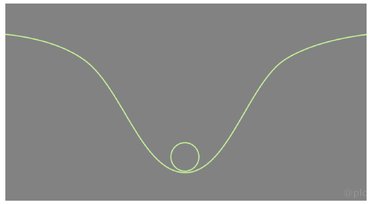
Approche DDD: DTO

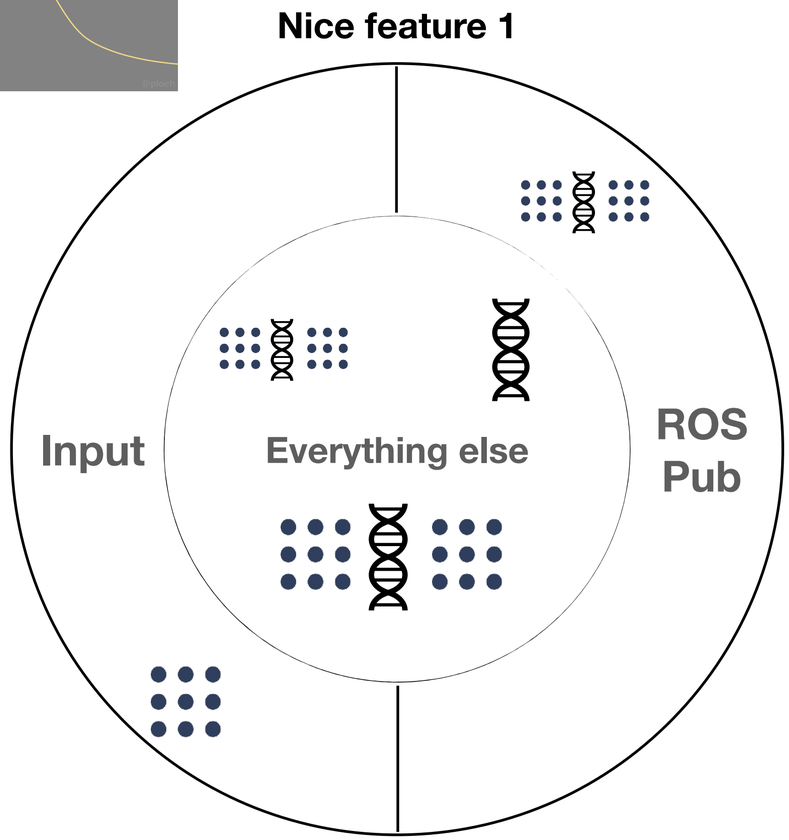


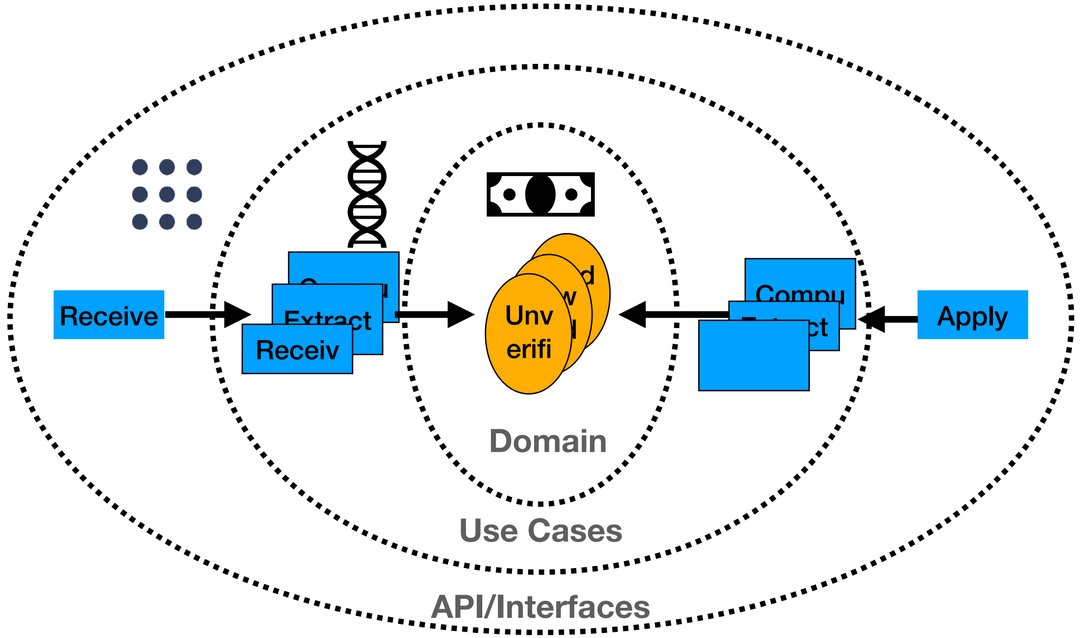

```rust
impl RobotSetModeStateMachine<UnverifiedNewRobotModeState> {
    pub fn receive_new_mode(message: String, current_mode: &mut CurrentRobotMode) -> Result<> {
        let new_mode_request = UnverifiedNewRobotModeState::new(message)?;
        Ok(RobotSetModeStateMachine {
            mode_to_set: new_mode_request.new_mode_request.0,
            current_robot_mode: *current_mode,
            state: new_mode_request,
        })
    }
}
impl UnverifiedNewRobotModeState {
    fn new(message: String) -> UnverifiedNewRobotModeStateResult {
        let new_mode_try = UnverifiedNewRobotMode::try_from(message)?;
        Ok(UnverifiedNewRobotModeState {
            new_mode_request: new_mode_try,
        })
    }
}
impl TryFrom<String> for UnverifiedNewRobotMode {
    type Error = UnverifiedNewRobotModeError;
    fn try_from(ros_msg: String) -> UnverifiedNewRobotModeResult {
        match &ros_msg[..] {
            "STANDBY" => Ok(UnverifiedNewRobotMode(RobotMode::StandBy(SlopeOrientation::Unknown))),
            "DOCK" => Ok(UnverifiedNewRobotMode(RobotMode::Dock(DockMode::DockFind))),
            "FOLLOW_MOVE" => Ok(UnverifiedNewRobotMode(RobotMode::Follow(FollowMode::Move))),

            other => Err(UnverifiedNewRobotModeError {
                kind: UnverifiedNewRobotModeErrorKind::UnknownMode,
                message: format!("{} : Unknown mode", other),
            }),
        }
    }
}
```

Conclusion









+ Evolutions simplifiées

+ Importance du Lexicon