Material

Let's consider a robot that drives from one office to another one to pick up an item and return to its initial location.

Before the robot can start driving, it undergoes an initialization step where it is calibrating its lidar unit and sensors before it is ready. It can then drive from one room to another room, wait to pick up an item and return to its original destination. Figure 3 shows a state chart indicating the states of this item pickup scenario.

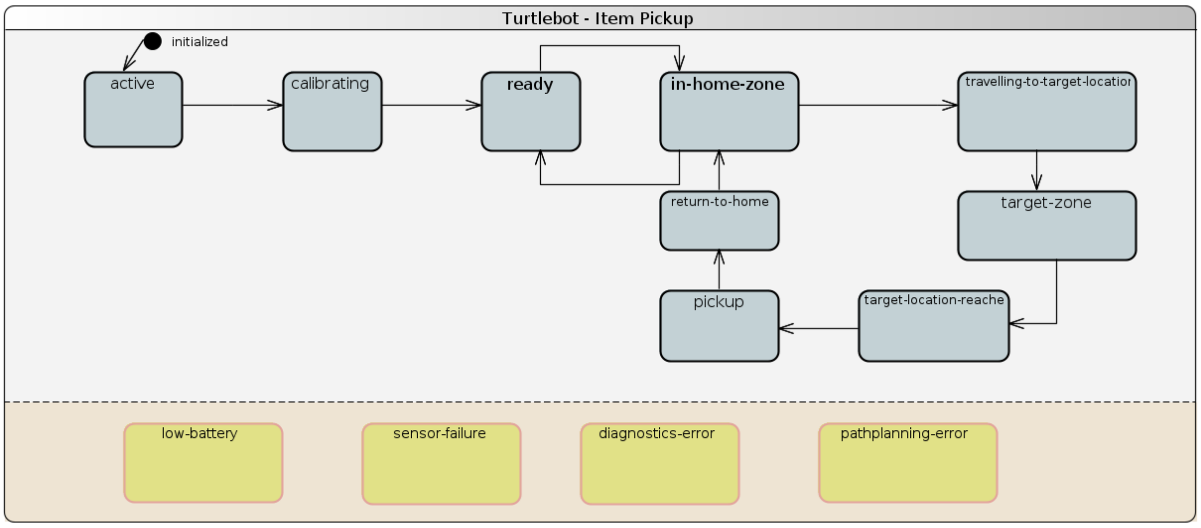

Figure 3: State Transition Diagram for the Turtlebot Item Pickup Scenario

To ensure that the robot operates correctly and does not collide with an obstacle, or move outside the designated area, it is necessary to monitor its behavior. One challenge when implementing this scenario is that different states require different information to be monitored. For instance, during the initialization phase, certain calibration checks are important to monitor, whereas they can be switched off when the robot is driving.

We developed a language that can help users to define the monitoring properties that should be monitored, as well as the period of monitoring and whether the monitored information should be sent to a central ground control station.

I: Preplanned Rules

In order to customize monitors in different states, "Preplanned Rules" can be specified in the DSL. Each Rule has a unique name, a specific trigger (the state where it becomes active), and a set of monitors it customizes. A "Monitor" describes a specific property of the system that is monitored, e.g., the Property "Bot.Diagnostics". For each Monitor, a period can be specified, as well as the scope, which indicates whether the property is only processed locally or it should be sent to the central server for further analysis.

Additionally, for a preplanned rule in the DSL (1) the *application context* can be specified, i.e., a rule can be applied to any bot (or to a bot with a specific ID, if needed), and (2) whether the rule is triggered when the state is entered or exited.

Examples of Preplanned Rules for the TurtleBot are shown in Listing 1.

Listing 1: Example of Preplanned Rules for the Turtlebot

```
Rule PREPLANNED "Rule␣1" applies forall // a preplanned rule with the name "Rule 1" used for all
     turtlebots
 Trigger Context["Bot.active"] ENTRY // it is triggered as soon as the state "active" is activated
     Monitor "Bot.Diagnostics":
      CHANGE_PERIOD: every 3 seconds  // once every three seconds,
      CHANGE_SCOPE: scope local      //the Diagnostics data should be locally monitored
     Monitor "Bot.Odometry":
      CHANGE_PERIOD: every 2 seconds  // once every two seconds,
      CHANGE_SCOPE: scope central    //the TurtleBot's odometry data should be
;                                     //reported centrally to the control station

Rule PREPLANNED "While␣moving␣to␣target"  applies forall
 Trigger Context["Bot.traveling-to-target-location"] ENTRY // when a bot enters the state "traveling-to-
      target-location"...
     Monitor "Bot.BatteryStatus":
      CHANGE_PERIOD: every 1 seconds
      CHANGE_SCOPE: scope central //... the Battery Status should be centrally monitored once every
           second
;     //...
```

II: Ubiquitous Rules

Besides the "normal course" of a use case, that is reflected by the different states in the state transition diagram, situations can occur that require special treatment and a deviation from the preplanned monitoring rules.

For example, when a Bot shows low battery, or a sensor failure, monitoring and data collected need to be adapted accordingly.

For these cases, our DSL allows us to specify *Ubiquitous Rules* which can be triggered by states outside of the state transition diagram. Specifying these rules works the same way as for Preplanned rules. However, they are not part of predefined states and transitions in the state chart but can occur at any time during the mission when a certain ubiquitous or error state is triggered.

An example is shown in Listing 2. The comments explain the meaning of this rule.

Listing 2: Ubiquitous Rule – Battery Warning

```
Rule UBIQUITOUS "Battery_State_Warning" applies forall //The rule holds for all turtlebots
 salience 1 //The rule has priority/salience 1.
            //If multiple ubiquitous rules can be applied, the one with the highest salience is selected
  Trigger Event['Bot.battery-state-warning'] ENTRY //When the event 'battery-state-warning' is triggered
      Monitor "Bot.Odometry":
       CHANGE_SCOPE: scope local
       CHANGE_PERIOD: every 5 seconds //The Odometry Data should be locally monitored every 5 seconds.
      Monitor "Bot.BatteryStatus":
       CHANGE_SCOPE: scope central
       CHANGE_PERIOD: every 2 seconds //The Battery Status should be centrally monitored every 2 seconds.
;
```

III: Default Values

To reduce the configuration effort of the monitors, (i.e., each monitor should be present in every rule), the DSL allows us to set *Default Values* for monitors.

This means that if a monitor is not configured in a certain rule, its default value is used. A monitor default value can (1) either be *off*, i.e., no data is collected (period 0), or (2) *keep*, which means the monitor does not change and keeps the same value (both period and scope) as it had before the transition to the new state occurred.

In the default values shown in Listing 3, "Bot.Odometry" has the value "keep", which entails that for the example rules in Listing 1, the property "Bot.Odometry" would be continued to be monitored with the same scope and period in the state "traveling-to-target-location". Note that if no default value is defined, the monitoring property needs to be explicitly defined in all preplanned rules.

Listing 3: Examples of Default Values

```
Default "Bot.BatteryStatus" off;
Default "Bot.Odometry" keep;
```

IV: Assumptions

To support basic validation of rules, the DSL allows users to define *Assumptions* about the monitored properties that should hold. These assumptions require domain knowledge or knowledge about the system, and define minimum guaranteed values for Monitors. This means that the system can guarantee that values are updated in a specific interval, and that new values are available for the monitors to be sent to the monitoring framework.

For example, Listing 4 specifies that the minimum interval with which we can expect updates is 1 second for *"Bot.BatteryStatus"* messages. A rule specifying a Monitor with a period of, for example, 0.5 seconds for *"Bot.BatteryStatus"* would be invalid, as data is not provided frequently enough by the system.

Listing 4: Assumption specification for Monitoring properties

```
Assumption "Bot.BatteryStatus"  MIN_PERIOD:  1 seconds;
Assumption "Bot.Odometry" MIN_PERIOD: 0.5 seconds;
```

Cheat Sheet

Listing 5 shows an example using all the language elements we have introduced.

Listing 5: Example of a DSL file for the TurtleBot System

```
namespace 'Bot';

/* Assumptions */
Assumption "Bot.BatteryStatus"  MIN_PERIOD:  1 seconds;
Assumption "Bot.Odometry" MIN_PERIOD: 0.5 seconds;

/* Default Values */
Default "Bot.BatteryStatus" off;
Default "Bot.Odometry" keep;

/* Preplanned Rules */
Rule PREPLANNED "Rule␣1" applies forall // a preplanned rule with the name "Rule 1" used for all
     turtlebots
 Trigger Context["Bot.active"] ENTRY // it is triggered as soon as the state "active" is activated
     Monitor "Bot.Diagnostics":
      CHANGE_PERIOD: every 3 seconds  // once every three seconds,
      CHANGE_SCOPE: scope local      //the Diagnostics data should be locally monitored
     Monitor "Bot.Odometry":
      CHANGE_PERIOD: every 2 seconds  // once every two seconds,
      CHANGE_SCOPE: scope central    //the TurtleBots odometry data should be
;                                     //reported centrally to the control station

Rule PREPLANNED "While␣moving␣to␣target"  applies forall
 Trigger Context["Bot.travelling-to-target-location"] ENTRY // when a bot enters the state "travelling-to
      -target-location"...
     Monitor "Bot.BatteryStatus":
      CHANGE_PERIOD: every 1 seconds
      CHANGE_SCOPE: scope central //The Battery Status should be centrally monitored once every second
;

/* UbiquitousRules
 */
Rule UBIQUITOUS "Battery_State_Warning" applies forall //The rule holds for all turtlebots
 salience 1  //The rule has priority/salience 1.
            //If multiple ubiquitous rules can be applied, the one with the highest salience is selected
 Trigger Event["Bot.low-battery"] and Event["Bot.sensor-failure"]  //When the event low-battery as well
      as the event sensor-failure occur.
  Monitor "Bot.Odometry":
   CHANGE_SCOPE: scope local
   CHANGE_PERIOD: every 5 seconds //The Odometry Data should be locally monitored once every 5 seconds.
  Monitor "Bot.BatteryStatus":
   CHANGE_SCOPE: scope central
   CHANGE_PERIOD: every 2 seconds //The Battery Status should be centrally monitored once every 2 seconds
        .
;
```

1.1 Create Monitoring Rules (Phase 2 - Task 1)

This task is concerned with specifying monitoring rules for a scenario in our domain-specific language. You can go through the following steps and work with the language editor in parallel.

Monitoring Property	**Description**	**Min. Assumed Period**
Bot.Odometry	Motion sensors and position estimate data	once every 0.5s
Bot.BatteryStatus	Battery Information (Voltage, Level, ...)	once per 1s
Bot.VersionInfo	Version Information of the ROS Core and packages	
Bot.SensorState	Sensor information about sensor state and accuracy	
Bot.LaserScan	Laser Scan data for mapping	
Bot.Velocity	Movement speed and Motor Data	
Bot.Diagnostics	Sensor and Actuator Diagnostics Data	

– **Step 1: Assumption**
Based on the guaranteed monitoring period by the system, create Assumptions for the minimum period of the two properties Bot.Odometry and Bot.BatteryStatus. If nothing is stated in the table, no assumption needs to be specified.

– **Step 2: Default Values**
Monitoring for all the above mentioned properties should be **turned off by default**. Define the default values in the Domain-Specific Language.

– **Step 3: Preplanned Rules**
Define pre-planned rules for the following states. All rules should apply for all bots and upon entry into a certain state.

1. Whenever a bot enters the state **calibrating**, its *Odometry data should be monitored every 0.5 seconds*, its *VersionInfo checks every 1 second*, and its *Sensor State every 2 seconds.* All of these properties should be centrally monitored.
2. Whenever a bot is **traveling to the target location**, the *Odometry data should be centrally monitored* every 10 seconds and *Laser Scan data locally every second.* The bot's *Version Info should not be monitored* anymore (0 seconds).

– **Step 4: Ubiquitous Rules**

Define the following ubiquitous rules:

1. When the **low battery** event is triggered, the *Odometry data of the bot should be locally monitored every 3 seconds*, and the *Battery Status should be monitored centrally every 2 seconds.* The rule has a salience level of 3.
2. When both a **low battery** and a **sensor failure** event occur, with both active, the *Odomentry data of the bot should be locally monitored every 10 seconds*, the *Diagnostics data should be monitored centrally every 5 seconds*, and the *Battery Status should be monitored centrally every 2 seconds.* The rule has a salience level of 5.

1.2 Glitch detector (Phase 2 – Task 2)

Now, we will use a glitch detector task, in which we show you an example specification in our DSL that may include errors. Your task is to go through the rules, describe what you see, and identify potential faults. The comments indicate what the specified instance of the DSL is supposed to describe. Faults are not syntax-related (caused by missing symbols or a strange order of keywords), but are semantic issues (resulting in undesirable monitoring behavior or conflicts between specified rules).

– Rule Description

- The preplanned rule "pre-mission-data" should be used for all bots in the fleet.
  It is triggered when the state "Bot.calibrating" is entered.
  The odometry should be monitored centrally once every 0.5 seconds.
  The Bot.SensorState should be locally monitored every 3 seconds.

- The ubiquitous rule "battery-and-planning-error" should be used for all Bots.
  Its salience value is 5.
  It is triggered when the events "low-battery" and "pathplanning-error" are triggered.
  The Bot.Odometry should be monitored locally once every 3 seconds.
  The Bot.SensorState should be monitored centrally once every 2 seconds.
  The battery should be monitored locally once every 10 seconds.