Software Engineering
Final Case Study 2018-2019

Luca Begnardi
luca.begnardi@studio.unibo.it

1 Introduction

This report describes the development of the software system proposed by the case study, paying particular attention to the adopted production process. The aim of this project is to realize a distributed software system for the management of a room dedicated to support a Standing Buffet Service. A Differential Drive Robot (DDR) acting as a butler (therefore the name Room Butler Robot or RBR, as it will be called from now on) must be able to perform several tasks, including preparing the room for the buffet and clearing it after it is finished, inside an environment equipped with a set of (smart and non-smart) resources. It must also be controllable by a Maitre de salle (or simply Maitre), who will supervise all the operations, via a smart-phone.

2 Vision

Our vision is to develop the distributed system so that it can be possible to easily adjust it to reasonable changes in the requirements or to change its underlying technology with the same ease. Our system should also allow us to create, in the fastest and easiest possible manner, a working prototype that follows the Product Owner specifications, aiming at the same time to build it so that most of its parts will be reusable. In order to achieve this, the followed motto will be: "There is no code without project, no project without problem analysis and no problem without requirements". We will then approach the problem top-down, analyzing the requirements to identify models describing the logical architecture of the problem itself. These models must be formal: they cannot be ambiguous since they have to be understandable by a machine. Then, we also want them to be as general and reusable as possible. Such models should enable us to automatically produce most of our code directly from them via software factories, sharply decreasing the actual coding time and the cost of adaptations in case of changes in the requirements. They would also allow us to rapidly and automatically generate working prototypes, helping us to communicate effectively with the client in order to better understand the requirements. This leads to the need to develop our system incrementally. To do so we will use SCRUM,

organizing the work in sprints, for each of which the logical architecture defined in the previous one will be refined to satisfy more functional requirements.

3 Goals

Our goal is to realize a distributed software system consisting of three entities:

- a **Robot (RBR)**, able to operate inside a predetermined room and to execute some planned tasks when requested by a human operator;
- a **Fridge**, able to communicate with other machines and humans without explicitly knowing those entities;
- a **smart-phone application** used by the human Maitre to send commands to the RBR and the Fridge and to receive responses.

A model-driven top-down approach will be adopted, using meta-models and software factories to automatically generate as much code as possible, while applying the SCRUM framework as described in the Vision.

4 Requirements

A room dedicated to support a Standing Buffet Service is equipped with a set of (smart and non-smart) resources including a fridge, a dishwasher, a pantry, and a DDR robot able to work as a Room Butler (called from now on RBR or Room Butter Robot). Design and build the software to put on board of the **Fridge** and of the **RBR**. In particular, the **RBR** must be able to accept the following commands sent by the smart-phone of the **Maitre**:

- *prepare*: the **RBR** must execute in autonomous way the *Prepare the room* task;
- *add food*: the **RBR** must execute in autonomous way the *Add food* task;
- *clear*: the **RBR** must execute in autonomous way the *Clear the room* task.

These tasks are normally executed in sequence, and the main scenario can be summarized as follows:

1. At start, the room is empty (i.e. no people is in it, besides the Maitre) while the **pantry** and the **Fridge** are filled with a proper set of items. The **RBR** is in its home (RH) location and the **dishwasher** is empty.
2. The **Maitre** sends to the **RBR** the *prepare* command and waits for the completion of the related task, consisting in putting on the **table** dishes taken from the **pantry**, and food taken from the **Fridge**. The set of items to put on the **table** in this phase is fixed and properly described somewhere. At the end, the **RBR** is in its RH location again.

3. The Maitre opens the room to people. During the service, the **Maitre** can send to the **RBR** the *add food* command, by specifying a food-code. The **RBR** executes the task, consisting in bringing the food with the given code from the **Fridge** to the **table**, only if the specified food is available in the **Fridge**, otherwise it sends a warning to the **Maitre**. After the task completion, the **RBR** returns is in its RH location.

4. At the end of the party, the **Maitre** sends to the **RBR** the **clear** command and waits for the completion of the task, consisting in bringing non-consumed food again in the **Fridge** and the dishes in the **dishwasher**. The **RBR** returns is in its RH location again.

However, the **Maitre** is able, at any time, to use his/her smart-phone to:

- *consult* the state of the room, e.g. to known what are the objects related to each resource; for example, the object currently posed on the **table**, in the **dishwasher**, etc;

- *stop* or *reactivate* an activated task.

Finally, the **RBR** must be able to

- *avoid* the impact with mobile obstacles (e.g. The Maitre or other humans/animals present in the room).

The software to put on the **Fridge** should make the device able to:

- *expose* its current content on the **Maitre** smart-phone;

- *answer* to questions about its content (e.g. if it contains food with a given code).

5 Requirement analysis

5.1 Domain model

First of all, we need to understand clearly the meaning of the client's requests, starting from the entities involved in the system:

- the **RBR** is a robot able to receive and execute tasks sent by the the Maitre via a smart-phone. It can move inside a predetermined room and interact with the other smart and non-smart entities in several ways, knowing where they are located since the beginning. It is also able to avoid the impact with mobile obstacles while operating. Discussing with the client, it emerged that to do so the robot will be equipped with a sonar;

- the **Fridge** is a static smart device for food storage. It explicitly knows what its content is and it is able, on request, to communicate it with humans and machines in two different ways. After the discussion with the

client, it is clear that this communications must occur via CoAP, so that the device can operate without knowing explicitly the other entities of the system;

- the Maitre is a human operator who uses a smart-phones to interact with the entities and manage the operations inside the room. The client clarified that it is not necessary that the software running on the smart-phone is a native application;
- the table, the pantry and the dishwasher are static non-smart entities. They can store objects without having explicit knowledge of it.

Then we discussed with the client about what the RBR and the Fridge can and cannot do:

- **RBR**: The robot can receive commands only from the Maitre. Most of those involve moving automatically from a point of the room to another.
  - *Prepare the room*: this task consists in putting on the table dishes taken from the pantry, and food taken from the fridge. The set of items to put on the table in this phase is fixed and properly described somewhere;
  - *clear the room*: this task consists in bringing non-consumed food again in the fridge and the dishes in the dishwasher;
  - *add food on the table*: this task consists in bringing some specific food from the Fridge to the table. This will be only performed if the requested food is available in the fridge, so the robot will have to forward the request to the Fridge itself (it will need to use CoAP as well) and wait for the response. In case the fridge does not contain the specified food, the robot will send a warning to the Maitre instead of performing the task.

  The robot can also receive simpler less articulated commands, including:
  - *stop*: the RBR must pause the task it is performing in that moment. The client has made clear that if this command is received while the robot is not performing any task, it must be ignored;
  - *reactivate*: if the RBR was stopped while performing a task, the job is resumed, otherwise the command is ignored.
- **Fridge**: The only commands the Fridge can receive are related to possible questions the RBR or the Maitre can ask it about its content:
  - *expose*: the Fridge must reply with its whole content. It is sent only by the Maitre;
  - *answer*: the Fridge must reply with a binary answer if it contains a certain food or not. It can be sent by both the Maitre and the RBR.

Finally, the **Maitre** must be able to *consult* the state of the room at any given moment.

5.2 TestPlans

Technically we could already define several tests to prove the expected behavior of the domain entities discussed above. These tests could be easily formalized using the JUnit framework, which allows an almost one to one mapping between requirements and test cases. However, as anticipated, the work will be divided in Sprints following the instructions of the SCRUM software development framework. For each sprint we will deal with an increased amount of functional requirements, so we will report multiple problem analysis and subsequent workplans and projects. Multiple formal TestPlans will then be presented in the later sections. The requirement analysis phase will not be repeated since it has already been presented.

6 Sprint 1

6.1 Requirements

Design and build the software to put on board of the **Fridge** and of the **RBR**. In particular, the **RBR** must be able to accept the following commands sent by the smart-phone of the **Maitre**:

- *prepare*: the **RBR** must execute in autonomous way the *Prepare the room* task;
- *add food*: the **RBR** must execute in autonomous way the *Add food* task;
- *clear*: the **RBR** must execute in autonomous way the *Clear the room* task.

These tasks are normally executed in sequence, and the main scenario can be summarized as follows:

1. At start, the room is empty (i.e. no people is in it, besides the Maitre) while the **pantry** and the **Fridge** are filled with a proper set of items. The **RBR** is in its home (RH) location and the **dishwasher** is empty.
2. The **Maitre** sends to the **RBR** the *prepare* command and waits for the completion of the related task, consisting in putting on the **table** dishes taken from the **pantry**, and food taken from the **Fridge**. The set of items to put on the **table** in this phase is fixed and properly described somewhere. At the end, the **RBR** is in its RH location again.
3. The Maitre opens the room to people. During the service, the **Maitre** can send to the **RBR** the *add food* command, by specifying a food-code. The **RBR** executes the task, consisting in bringing the food with the given code from the **Fridge** to the **table**, only if the specified food is available in the **Fridge**, otherwise it sends a warning to the **Maitre**. After the task completion, the **RBR** returns is in its RH location. For now, the **Fridge** does not need to use CoAP to communicate.

4. At the end of the party, the **Maitre** sends to the **RBR** the **clear** command and waits for the completion of the task, consisting in bringing non-consumed food again in the **Fridge** and the dishes in the **dishwasher**. The **RBR** returns is in its RH location again.

6.2 Problem analysis

6.2.1 Abstraction gap

As stated in the vision and goal sections, our work will follow a model-driven top-down approach, using meta-models and software factories to automatically generate as much code as possible. This would be rather hard and time consuming to realize from scratch. Luckily we already have some tools we can make use of:

- **QActor (Qak) meta-model**: using this DSL we will be able to formally express the logical architecture of the system we have to build and it will automatically generate the code to make such a system actually work. It also provides a way to make the involved entities communicate with each other based on MQTT, a light and widely used messaging protocol;
- *basicrobot*: as stated in the first few lines of the requirements, we have to control a robot. The basicrobot script manages the basic actions a DDR can do (i.e. moving and handling the data of the sonar it is equipped with). Along with this script, we will make use of the (separated) Kotlin code that handles several possible implementations of the robot (both physical and virtual) and that allows it to identify obstacles through the sonar data;
- *robotmind*: a qak script that enables the actuator (the basicrobot) only after the model of the system has changed (following the MVC pattern). Since we are adopting a model-driven approach, our software will rely heavily on Prolog knowledge basis describing the model of the system;
- *planexecutor*: as stated in the requirements, the robot must be able to automatically move inside the room. This implies we need some kind of planning tool, which again would be quite hard to build from scratch. The planexecutor script, along with some other Kotlin code will allow us to focus more on the business logic part of the software system we are to develop;
- *onestepahead*: in order to work properly the planner tool needs the robot to move one "step" at a time (see section 6.5.2 for more details). The onestepahead script does exactly this and it is also able to tell if the robot encountered an obstacle during the step or not.

6.2.2 Logical architecture

The logical architecture of a system defines its structure, interactions and behavior:

- **Structure**: despite the system seems to be composed of six entities, since three of those (the **table**, the **pantry** and the **dishwasher**) are not smart, we will only consider the **RBR**, the **Fridge** and the **Maitre**. Then, at this level, we do not see the robot as an atomic entity anymore: in fact we can look deeper into how it is composed. As said before to make it work we will rely on some already developed virtual components (*basicrobot*, *onestepahead*, *robotmind* and *planexecutor*) that will be integrated in the robot creating a layered system and all of their communications will occur inside of the robot itself. We will then introduce another component inside the robot (but at the same layer of the *roombutlerrobot*) called *resourcemodel*. Its job is to handle the model of the resources (except for that of the *fridge*), modifying them and making part of those visible to the other actors when needed.

- **Interaction**: to be as technology independent as possible and since we are designing a distributed system, we can rely on two general different communication models provided by the Qak framework:

  - **messages**, which are point-to-point (i.e. emitted by a component with a specific destination), asynchronous and buffered, so that they will not be lost if they are not received immediately. The source entity doesn't wait for a response;
  - **events**, which are emitted by a component without a specific destination so every interested entity can receive them. If this doesn't occur, the events are lost. Clearly, as for messages, the source of the event does not wait for a response.

  Now we have to consider what interactions will occur between the system's entities:

  - **RBR**: it must be able to communicate with both the **Maitre** and the **Fridge**. As for how (non-basic) components communicate inside the robot:
    - *roombutlerrobot*: communicates with *resourcemodel*, *planexecutor* and **Maitre**;
    - *planexecutor*: communicates with *roombutlerrobot* and *resourcemodel*;
    - *resourcemodel*: communicates with *roombutlerrobot* and *planexecutor*.
  - **Fridge**: it will only communicate with the **RBR**.
  - **Maitre**: it will only communicate with the **RBR**.

- **Behavior**: using the QActor meta-model we will express the behavior of the entities as Finite State Machines (FSM).

roombutlerrobot

```
QActor roombutlerrobot context ctxRBR {
  ["
  var NextGoal = \"\"
  var GoalX = \"\"
  var GoalY = \"\"
  var CurObject = \"\"
  var actionDone = false
  "]

  State s0 initial {
    solve( consult("butlerRobotKb.pl") )
    solve( consult("sysRules.pl") )
  }
  Goto waitCmd

  State waitCmd { println("&&& RBR waitCmd ... ") }
  Transition t0
  whenMsg prepare -> prepareTheRoomInit
  whenMsg addFood -> addFoodOnTableInit
  whenMsg clear  -> clearTheRoomInit

  //PREPARE THE ROOM
  State prepareTheRoomInit {
    ["
    actionDone = false
    NextGoal = \"pantry\"
    GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
    GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
    "]
    forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)

    //at least one item from the pantry must be in the preparation set
    ["solve( \"preparation([H|T])\" )"]
    ifSolved {
      ["
      solve( \"replaceRule(preparation([H|T]), preparation(T))\" )
      CurObject = getCurSol(\"H\").toString()
      "]
    }
  }
  Goto waitPrepare

  State waitPrepare { println("WAIT PREPARE") }
  Transition t0
  whenMsg goalOk -> prepareTheRoomContinue
  whenEvent roomModelChanged -> prepareTheRoomContinue

  State prepareTheRoomContinue {
    printCurrentMessage
    delay 1000

    onMsg( goalOk : goalOk(pantry) ) {
      forward resourcemodel -m modelUpdate : modelUpdate(pantry, remove($CurObject))
    }
    onMsg( goalOk : goalOk(fridge) ) {
      forward fridge -m modelUpdate : modelUpdate(fridge, remove($CurObject))
    }
    onMsg( goalOk : goalOk(table) ) {
      forward resourcemodel -m modelUpdate : modelUpdate(table, add($CurObject))
    }
    onMsg( goalOk : goalOk(home) ) {
      ["actionDone = true"]
```

```
63          forward emulatedmaitre -m prepareDone : prepareDone
64        }
65
66        onMsg( roomModelChanged : modelChanged(pantry, remove(_)) ) {
67          ["
68          NextGoal = \"table\"
69          GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
70          GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
71          "]
72          forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
73        }
74        onMsg( roomModelChanged : modelChanged(fridge, remove(_)) ) {
75          ["
76          NextGoal = \"table\"
77          GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
78          GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
79          "]
80          forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
81        }
82        onMsg( roomModelChanged : modelChanged(table, add(_)) ) {
83          ["solve( \"preparation([H|T])\" )"]
84          println(currentSolution)
85          ifSolved {
86            ["
87            solve( \"replaceRule(preparation([H|T]), preparation(T))\" )
88            CurObject = getCurSol(\"H\").toString()
89            if(CurObject.equals(\"dish\"))
90            NextGoal = \"pantry\"
91            else
92            NextGoal = \"fridge\"
93            GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
94            GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
95            "]
96          }
97          else { //preparation list empty
98            ["
99            CurObject = \"\"
100           NextGoal = \"home\"
101           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
102           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
103           "]
104         }
105         println("$NextGoal")
106         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
107       }
108     }
109     Goto waitCmd if "actionDone" else waitPrepare
110
111     //ADD FOOD ON THE TABLE
112     State addFoodOnTableInit {
113       ["actionDone = false"]
114       onMsg(addFood : addFood(F)) {
115         //check if the fridge contains F
116         forward fridge -m request : request(roombutlerrobot, $payloadArg(0))
117         ["CurObject = \"${payloadArg(0)}\" "]
118       }
119     }
120     Transition t0
121     whenMsg answer -> checkFridgeAnswer
122
123     State checkFridgeAnswer {
124       printCurrentMessage
125       onMsg(answer : answer(yes)) {
126         ["
127         NextGoal = \"fridge\"
128         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
129         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
130         "]
```

```
131         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
132       }
133       onMsg(answer : answer(no)) {
134         ["CurObject = \"\"
135         NextGoal = \"\"
136         GoalX = \"\"
137         GoalY = \"\"
138         actionDone = true
139         "]
140         forward emulatedmaitre -m warning : warning
141       }
142     }
143     Goto waitCmd if "actionDone" else waitAddFood
144
145     State waitAddFood { println("WAIT ADD FOOD") }
146     Transition t0
147     whenMsg goalOk -> addFoodOnTheTableContinue
148     whenEvent roomModelChanged -> addFoodOnTheTableContinue
149
150     State addFoodOnTheTableContinue {
151       delay 1000
152
153       onMsg( goalOk : goalOk(fridge) ) {
154         forward fridge -m modelUpdate : modelUpdate(fridge, remove($CurObject))
155       }
156       onMsg( goalOk : goalOk(table) ) {
157         forward resourcemodel -m modelUpdate : modelUpdate(table, add($CurObject))
158       }
159       onMsg( goalOk : goalOk(home) ) {
160         ["actionDone = true"]
161         forward emulatedmaitre -m addDone : addDone
162       }
163
164       onMsg( roomModelChanged : modelChanged(fridge, remove(_)) ) {
165         ["
166         NextGoal = \"table\"
167         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
168         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
169         "]
170         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
171       }
172       onMsg( roomModelChanged : modelChanged(table, add(_)) ) {
173         ["
174         CurObject = \"\"
175         NextGoal = \"home\"
176         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
177         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
178         "]
179         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
180       }
181     }
182     Goto waitCmd if "actionDone" else waitAddFood
183
184     //CLEAR THE ROOM
185     State clearTheRoomInit {
186       ["actionDone = false"]
187       forward resourcemodel -m modelConsult : modelConsult(table)
188     }
189     Transition t0
190     whenEvent modelState -> checkTableState
191
192     State checkTableState {
193       printCurrentMessage
194       onMsg(modelState : modelState(S) ) {
195         ["solve( \"table(_)\" )"]
196         ifSolved {
197           ["solve( \"replaceRule(table(_), table(${payloadArg(0)}))\" )"]
198         } else {
```

```
199             ["solve( \"addRule(table(${payloadArg(0)}))\" )"]
200           }
201           println(currentSolution)
202         }
203         ["
204         NextGoal = \"table\"
205         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
206         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
207         "]
208         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
209       }
210       Goto waitClear
211
212       State waitClear { }
213       Transition t0
214       whenMsg goalOk -> clearTheRoomContinue
215       whenEvent roomModelChanged -> clearTheRoomContinue
216
217       State clearTheRoomContinue {
218         printCurrentMessage
219         delay 1000
220         onMsg( goalOk : goalOk(fridge) ) {
221           forward fridge -m modelUpdate : modelUpdate(fridge, add($CurObject))
222         }
223         onMsg( goalOk : goalOk(dishwasher) ) {
224           forward resourcemodel -m modelUpdate : modelUpdate(dishwasher, add($CurObject))
225         }
226         onMsg( goalOk : goalOk(table) ) {
227           ["solve( \"table([H|T])\" )"]
228           println(currentSolution)
229           ifSolved {
230             ["
231             solve( \"replaceRule(table([H|T]), table(T))\" )
232             CurObject = getCurSol(\"H\").toString()
233             if(CurObject.equals(\"dish\"))
234             NextGoal = \"dishwasher\"
235             else
236             NextGoal = \"fridge\"
237             GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
238             GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
239             "]
240             forward resourcemodel -m modelUpdate : modelUpdate(table, remove($CurObject))
241           }
242           else { //no more objects on the table
243             ["
244             CurObject = \"\"
245             NextGoal = \"home\"
246             GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
247             GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
248             "]
249             forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
250           }
251           //println("$NextGoal")
252         }
253         onMsg( goalOk : goalOk(home) ) {
254           ["actionDone = true"]
255           forward emulatedmaitre -m clearDone : clearDone
256         }
257
258         onMsg( roomModelChanged : modelChanged(dishwasher, add(_)) ) {
259           ["
260           NextGoal = \"table\"
261           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
262           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
263           "]
264           forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
265         }
266         onMsg( roomModelChanged : modelChanged(fridge, add(_)) ) {
```

```
        ["
        NextGoal = \"table\"
        GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
        GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
        "]
        forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
      }
      onMsg( roomModelChanged : modelChanged(table, remove(_)) ) {
        forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
      }
    }
    Goto waitCmd if "actionDone" else waitClear
  }
```

fridge

```
QActor fridge context ctxFridge{
  State s0 initial {
    solve( consult("sysRules.pl") )
    solve( consult("fridgeModel.pl") )
  }
  Goto waitCmd

  State waitCmd { }
  Transition t0
  whenMsg request -> handleRequest
  whenMsg modelUpdate -> updateModel

  State handleRequest {
    onMsg(request : request(S, F)) {
      run itunibo.fridge.fridgeModelSupport.updateFridgeModel(myself, payloadArg(1))
    }
  }
  Goto waitCmd

  State updateModel {
    onMsg(modelUpdate : modelUpdate(fridge, V)) {
      run itunibo.fridge.fridgeModelSupport.updateFridgeModel(myself, payloadArg(1))
    }
  }
  Goto waitCmd
}
```

resourcemodel

```
QActor resourcemodel context ctxRobotMind{

  State s0 initial {
    solve( consult("sysRules.pl") )
    solve( consult("resourceModel.pl") )
    solve( showResourceModel )
  }
  Goto waitMsg

  State waitMsg{ }
  Transition t0
  whenMsg modelChange -> changeModel
  whenMsg modelUpdate -> updateModel
  whenMsg modelConsult -> consultModel

  State updateModel{
    printCurrentMessage
    onMsg( modelUpdate : modelUpdate(robot, V ) ) {
      run itunibo.resModel.resourceModelSupport.updateRobotModel( myself, payloadArg(1) )
    }
    onMsg( modelUpdate : modelUpdate(sonarRobot,V ) ) {
```

```
22          run itunibo.resModel.resourceModelSupport.updateSonarRobotModel( myself, payloadArg(1)
                ↪ )
23        }
24        onMsg( modelUpdate : modelUpdate(pantry, V) ) {
25          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
26        }
27        onMsg( modelUpdate : modelUpdate(table, V) ) {
28          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
29        }
30        onMsg( modelUpdate : modelUpdate(dishwasher, V) ) {
31          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
32        }
33        onMsg( modelUpdate : modelUpdate(roomMap,V ) ) {
34          run itunibo.resModel.resourceModelSupport.updateRoomMapModel( myself, payloadArg(1) )
35        }
36      }
37      Goto waitMsg
38
39      State changeModel {
40        //ROBOT MOVE
41        onMsg( modelChange : modelChange( robot,V ) ) { // V= w | ...
42          run itunibo.resModel.resourceModelSupport.updateRobotModel( myself, payloadArg(1) )
43          emit local_robotModelChanged : modelChanged( robot, $payloadArg(1)) //for the
                ↪ robotmind
44        }
45      }
46      Goto waitMsg
47
48      State consultModel {
49        onMsg( modelConsult : modelConsult(_) ) {
50          run itunibo.resModel.resourceModelSupport.consultRoomResourceModel(myself, payloadArg
                ↪ (0))
51        }
52      }
53      Goto waitMsg
54    }
```

6.3 Product backlog

- Define the system infrastructure:
  - basic entities (30 minutes);
  - messages which will be exchanged (30 minutes);
  - room model, fridge model and other Prolog knowledge basis (1 hour).
- Implement tasks for the **RBR**:
  - prepare the room: requires to work on the planner tool, on the *room-butlerrobot*, *fridge* and *resourcemodel* qak scripts and on the related support codes (5 hours);
  - add food on the table: as before, except for the planner tool, which should not need other variations (3 hours);
  - clear the room: as before (3 hours).
- Create an emulated frontend for prototyping purposes (30 minutes).
- Create formal TestPlans with JUnit. (2 hours)

6.4 TestPlans

TestRBR

```
class TestRBR {
  var rbr : ActorBasic? = null
  var mqttClient : MqttClient? = null
  var broker : String = "tcp://localhost"
  var rbrTopic : String = "unibo/qak/roombutlerrobot"
  var clientId : String = "sprint_1"
  var qos : Int = 2;

  @Before
  fun systemSetUp() {

    try {
      println("%%%%%%%%%%%%%% Sprint 1 TestRBR starting Mqtt Client")
      mqttClient = MqttClient(broker, clientId)
      var connOpts = MqttConnectOptions()
      connOpts.setCleanSession(true)
      mqttClient!!.connect(connOpts)
    }
    catch (e : MqttException) {
      println("MQTT EXCEPTION IN SETUP")
    }
    GlobalScope.launch {
      it.unibo.ctxRBR.main()
    }
    delay(5000) //give the time to start
    rbr = sysUtil.getActor("roombutlerrobot")
  }

  @After
  fun terminate() {
    println("%%%%%%% Sprint 1 TestRBR terminate ")
    try {
      mqttClient!!.disconnect()
      mqttClient!!.close()
    } catch (e : MqttException) {
      println("MQTT EXCEPTION IN TERMINATE")
    }
  }

  @Test
  fun sprint1Test() {
    println("%%%%%%% Sprint 1 Functional TestRBR starts ")
    prepare()
    addFoodFail("fruit")
    addFoodOk("beef")
    clear()
  }

  fun prepare() {
    println("%%%%%%% Sprint 1 TestRBR prepareRoom Task")
    sendCmd("prepare", "")
    solveCheckGoal(rbr!!, 0, 0)
  }

  fun addFoodFail(foodCode : String) {
    println("%%%%%%% Sprint 1 TestRBR addFood Task")
    sendCmd("addFood", foodCode)
    solveCheckGoal(rbr!!, 0, 0)
  }

  fun addFoodOk(foodCode : String) {
    println("%%%%%%% Sprint 1 TestRBR addFood Task")
    sendCmd("addFood", foodCode)
    solveCheckGoal(rbr!!, 0, 0)
```

```
  }

  fun clear() {
    println("%%%%%%% Sprint 1 TestRBR clearRoom Task")
    sendCmd("clear", "")
    solveCheckGoal(rbr!!, 0, 0)
  }

  //-----------------------------------------------------------------------------

  fun sendCmd(cmd : String, content : String) {
    println("--- RBR performing performing task $cmd")
    var msg : String
    if (content != "") {
      msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd($content),1)"
    } else {
      msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd,1)"
    }
    try {
      var mqttMsg = MqttMessage(msg.toByteArray())
      mqttClient!!.publish(rbrTopic, mqttMsg)
    }
    catch (e : MqttException) {
      println("MQTT EXCEPTION IN DOTASK")
    }
    if(content == "fruit") {
      delay(5000)
    }
    else {
      delay(45000) //wait 45 seconds to check the results
    }
  }

  fun solveCheckGoal( actor : ActorBasic, x : Int, y : Int ){
    var result = itunibo.planner.moveUtils.getPosX(actor) == x && itunibo.planner.moveUtils.getPosY(actor
        ↪ ) == y
    println(" %%%%%%% actor={$actor.name} goal = RBR in ($x,$y) result = $result")
    assertTrue(result)
  }

  fun delay( time : Long ){
    Thread.sleep( time )
  }
}
```

TestResources

```
class TestResources {
  var resource : ActorBasic? = null
  var mqttClient : MqttClient? = null
  var broker : String = "tcp://localhost"
  var rbrTopic : String = "unibo/qak/roombutlerrobot"
  var clientId : String = "sprint_1"
  var qos : Int = 2;

  @Before
  fun systemSetUp() {

    try {
      println("%%%%%%%%%%%%%% Sprint 1 TestResources starting Mqtt Client")
      mqttClient = MqttClient(broker, clientId)
      var connOpts = MqttConnectOptions()
      connOpts.setCleanSession(true)
      mqttClient!!.connect(connOpts)
    }
    catch (e : MqttException) {
```

```
20        println("MQTT EXCEPTION IN SETUP")
21      }
22      GlobalScope.launch {
23        it.unibo.ctxRobotMind.main()
24      }
25      delay(5000) //give the time to start
26      resource = sysUtil.getActor("resourcemodel")
27    }
28
29    @After
30    fun terminate() {
31      println("%%%%%%% Sprint 1 TestResources terminate ")
32      try {
33        mqttClient!!.disconnect()
34        mqttClient!!.close()
35      } catch (e : MqttException) {
36        println("MQTT EXCEPTION IN TERMINATE")
37      }
38    }
39
40    @Test
41    fun sprint1Test() {
42      println("%%%%%%% Sprint 1 Functional TestResources starts ")
43      prepare()
44      clear()
45    }
46
47    fun prepare() {
48      println("%%%%%%% Sprint 1 TestResources prepareRoom Task")
49      sendCmd("prepare", "")
50      solveCheckGoal(resource!!, "model( actuator, robot, state( stopped ) )")
51      solveCheckGoal(resource!!, "model( resource, pantry, state([ dish, dish, dish, dish, dish,
            ↪ dish, dish, dish, dish, dish, dish, dish, dish, dish ]) )")
52      solveCheckGoal(resource!!, "model( resource, table, state([ fruit, dish ]) )")
53      solveCheckGoal(resource!!, "model( resource, dishwasher, state([ ]) )")
54    }
55
56    fun clear() {
57      println("%%%%%%% Sprint 1 TestResources clearRoom Task")
58      sendCmd("clear", "")
59      solveCheckGoal(resource!!, "model( actuator, robot, state( stopped ) )")
60      solveCheckGoal(resource!!, "model( resource, pantry, state([ dish, dish, dish, dish, dish,
            ↪ dish, dish, dish, dish, dish, dish, dish, dish, dish ]) )")
61      solveCheckGoal(resource!!, "model( resource, table, state([ ]) )")
62      solveCheckGoal(resource!!, "model( resource, dishwasher, state([ dish ]) )")
63    }
64
65    //-------------------------------------------------------------------------
66
67    fun sendCmd(cmd : String, content : String) {
68      println("--- RBR performing performing task $cmd")
69      var msg : String
70      if (content != "") {
71        msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd($content),1)"
72      } else {
73        msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd,1)"
74      }
75      try {
76        var mqttMsg = MqttMessage(msg.toByteArray())
77        mqttClient!!.publish(rbrTopic, mqttMsg)
78      }
79      catch (e : MqttException) {
80        println("MQTT EXCEPTION IN DOTASK")
81      }
82      if(content == "fruit") {
83        delay(5000)
84      }
85      else {
```

```
        delay(45000) //wait 45 seconds to check the results
    }
  }

  fun solveCheckGoal( actor : ActorBasic, goal : String ){
    actor.solve( goal )
    var result = actor.resVar
    println(" %%%%%%% actor={$actor.name} goal= $goal result = $result")
    assertTrue("", result == "success" )
  }

  fun delay( time : Long ){
    Thread.sleep( time )
  }
}
```

TestFridge

```
class TestFridge {
  var fridge : ActorBasic? = null
  var mqttClient : MqttClient? = null
  var broker : String = "tcp://localhost"
  var rbrTopic : String = "unibo/qak/roombutlerrobot"
  var clientId : String = "sprint_1"
  var qos : Int = 2;

  @Before
  fun systemSetUp() {

    try {
      println("%%%%%%%%%%%%%% Sprint 1 TestFridge starting Mqtt Client")
      mqttClient = MqttClient(broker, clientId)
      var connOpts = MqttConnectOptions()
      connOpts.setCleanSession(true)
      mqttClient!!.connect(connOpts)
    }
    catch (e : MqttException) {
      println("MQTT EXCEPTION IN SETUP")
    }
    GlobalScope.launch {
      it.unibo.ctxFridge.main()
    }
    delay(5000) //give the time to start
    fridge = sysUtil.getActor("fridge")
  }

  @After
  fun terminate() {
    println("%%%%%%% Sprint 1 TestFridge terminate ")
    try {
      mqttClient!!.disconnect()
      mqttClient!!.close()
    } catch (e : MqttException) {
      println("MQTT EXCEPTION IN TERMINATE")
    }
  }

  @Test
  fun sprint1Test() {
    println("%%%%%%% Sprint 1 Functional TestFridge starts ")
    prepare()
    addFoodFail("fruit")
    addFoodOk("beef")
    clear()
  }

```

```
49    fun prepare() {
50      println("%%%%%%% Sprint 1 TestFridge prepareRoom Task")
51      sendCmd("prepare", "")
52      solveCheckGoal(fridge!!, "model( resource, fridge, state([ beef ]) )")
53    }
54
55    fun addFoodFail(foodCode : String) {
56      println("%%%%%%% Sprint 1 TestFridge addFood Task")
57      sendCmd("addFood", foodCode)
58      solveCheckGoal(fridge!!, "model( resource, fridge, state([ beef ]) )")
59    }
60
61    fun addFoodOk(foodCode : String) {
62      println("%%%%%%% Sprint 1 TestFridge addFood Task")
63      sendCmd("addFood", foodCode)
64      solveCheckGoal(fridge!!, "model( resource, fridge, state([ ]) )")
65    }
66
67    fun clear() {
68      println("%%%%%%% Sprint 1 TestFridge clearRoom Task")
69      sendCmd("clear", "")
70      solveCheckGoal(fridge!!, "model( resource, fridge, state([ fruit, beef ]) )")
71    }
72
73    //---------------------------------------------------------------------------
74
75    fun sendCmd(cmd : String, content : String) {
76      println("--- RBR performing performing task $cmd")
77      var msg : String
78      if (content != "") {
79        msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd($content),1)"
80      } else {
81        msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd,1)"
82      }
83      try {
84        var mqttMsg = MqttMessage(msg.toByteArray())
85        mqttClient!!.publish(rbrTopic, mqttMsg)
86      }
87      catch (e : MqttException) {
88        println("MQTT EXCEPTION IN DOTASK")
89      }
90      if(content == "fruit") {
91        delay(5000)
92      }
93      else {
94        delay(45000) //wait 45 seconds to check the results
95      }
96    }
97
98    fun solveCheckGoal( actor : ActorBasic, goal : String ){
99      actor.solve( goal )
100     var result = actor.resVar
101     println(" %%%%%%% actor={$actor.name} goal= $goal result = $result")
102     assertTrue("", result == "success" )
103   }
104
105   fun delay( time : Long ){
106     Thread.sleep( time )
107   }
108 }
```

6.5 Project

6.5.1 System infrastructure

Basic entities

We already showed the whole behavior of the entities in section 6.2.2.

Exchanged messages

As said before we will rely on Dispatches and Events, the two kinds of communications provided by the Qak meta-model.

roombutlerrobot

```
Event modelState : modelState(VALUE)
Event roomModelChanged : modelChanged(RES, VALUE)

Dispatch modelUpdate : modelUpdate(TARGET, VALUE) //for resourcemodel
Dispatch modelConsult : modelConsult(TARGET)  //for resourcemodel

Dispatch request : request(S, F) //S = sender, F = food-code
Dispatch answer : answer(A)  //A = yes | no
Dispatch warning : warning

Dispatch prepare : prepare
Dispatch addFood : addFood(X) //X = food-code
Dispatch clear : clear

Dispatch stepOk : stepOk
Dispatch stepFail : stepFail(R, T) //R = ok | obstacle, T = time

Dispatch goalUpdate : goalUpdate(G, X, Y) //G = pantry | fridge | dishwasher | home, X = coord x
    ↪  of G, Y = coord y of G
Dispatch goalOk : goalOk(X)   //X = pantry | fridge | dishwasher | home
```

fridge

```
Dispatch modelUpdate : modelUpdate(TARGET, VALUE) //TARGET = always fridge, VALUE = add(food)
    ↪  | remove(food)
Dispatch request : request(S, F) //S = maitre | roombutlerrobot, F = food-code
```

Prolog knowledge basis

The room is expressed as a 2D space divided in tiles as large as the robot itself. The room will be loaded at run-time from a proper binary file. It will have a textual representation, such as this one:

```
|r, 1, 1, 1, 1, 1, X,
|1, 1, 1, 1, 1, 1, X,
|1, 1, 1, X, X, 1, X,
|1, 1, 1, X, X, 1, X,
|1, 1, 1, 1, 1, 1, X,
|X, X, X, X, X, X, X,
```

The "r" represents the robot, which in the initial conditions is in its home location.
The "x" represent fixed obstacles, including walls and the table in the center of the room.
The "1" represent accessible tiles.

As already mentioned, the *resourcemodel* actor will detain the knowledge of all the resources except for the **Fridge** (as in the requirements is clearly said that it has to own its own knowledge). Since the **RBR** is the main entity operating in the room, it appears clear that it needs to possess not only its own knowledge, but also those of all the other non-smart resources. Therefore the decision to place the *resourcemodel* actor on the robot.

resourcemodel.pl

```
model( environment, roommap, state(unknown) ).  %% the actual map will be obtained at the start
model( actuator, robot, state(stopped) ).  %% initial state
model( sensor, sonarRobot, state(unknown) ).  %% initial state
model( resource, table,  state([ ]) ).  %% initial state
model( resource, pantry, state([ dish, dish, dish, dish, dish, dish, dish, dish, dish, dish, dish, dish, dish, dish, dish,
    ↪ dish ]) ).  %% initial state
model( resource, dishwasher, state([ ]) ).  %% initial state

%% environment actions
action(roommap, update(V)) :- changeModel( environment, roommap, V ).

%% movement actions
action(robot, move(w)) :- changeModel( actuator, robot, movingForward ).
action(robot, move(s)) :- changeModel( actuator, robot, movingBackward ).
action(robot, move(a)) :- changeModel( actuator, robot, rotateLeft ).
action(robot, move(d)) :- changeModel( actuator, robot, rotateRight ).
action(robot, move(h)) :- changeModel( actuator, robot, stopped ).
action(robot, move(l)) :- changeModel( actuator, robot, rotateLeft90 ).
action(robot, move(r)) :- changeModel( actuator, robot, rotateRight90 ).

%%application actions
action(ROOMRES, remove(dish)) :- changeModel( resource, ROOMRES, removeDish ), !.
action(ROOMRES, add(dish)) :- changeModel( resource, ROOMRES, addDish ), !.
action(ROOMRES, remove(F)) :- changeModel( resource, ROOMRES, removeFood(F)).
action(ROOMRES, add(F)) :- changeModel( resource, ROOMRES, addFood(F)).

%%sensor actions
action(sonarRobot, V) :- changeModel( sensor, sonarRobot, V ).
```

```
28
29 changeModel( CATEG, NAME, addDish ) :-
30   replaceRule( model(CATEG, NAME, state(X)), model( CATEG, NAME, state([dish|X])) ), !.
31
32 changeModel( CATEG, NAME, removeDish ) :-
33   replaceRule( model(CATEG, NAME, state([dish|X])), model( CATEG, NAME, state(X)) ), !.
34
35 changeModel( CATEG, NAME, addFood(VALUE) ) :-
36   replaceRule( model(CATEG, NAME, state(X)), model( CATEG, NAME, state([VALUE|X])) ), !.
37
38 changeModel( CATEG, NAME, removeFood(VALUE) ) :-
39   model( CATEG, NAME, state(X)),
40   delete_one(VALUE, X, X1),
41   replaceRule( model(CATEG, NAME, state(X)), model( CATEG, NAME, state(X1)) ), !.
42
43 changeModel( CATEG, NAME, VALUE ) :-
44   replaceRule( model(CATEG,NAME,_), model(CATEG,NAME,state(VALUE)) ).
45   %% showResourceModel. %% at each change, show the model
46
47 showResourceModel :-
48   output("RESOURCE MODEL ---------- "),
49   showResources,
50   output("--------------------------").
51
52 showResources :-
53   model( CATEG, NAME, STATE ),
54   output( model( CATEG, NAME, STATE ) ),
55   fail.
56 showResources.
57
58 delete_one(_, [], []).
59 delete_one(X, [X|T], T).
60 delete_one(X, [H|T], [H|R]) :-
61   delete_one(X, T, R).
62
63 contains(X, [X|T]).
64 contains(X, [_|T]) :-
65   contains(X, T).
66
67 output( M ) :- stdout <- println( M ).
68
69 initResourceTheory :- output("resourceModel loaded").
70 :- initialization(initResourceTheory).
```

fridge.pl

```
1 model( resource, fridge, state([ fruit, beef ]) ). %% initial state
2
3 %%application actions
4 action(ROOMRES, remove(F)) :- changeModel( resource, ROOMRES, removeFood(F)).
5 action(ROOMRES, add(F)) :- changeModel( resource, ROOMRES, addFood(F)).
6
7 changeModel( CATEG, NAME, addFood(VALUE) ) :-
8   replaceRule( model(CATEG, NAME, state(X)), model( CATEG, NAME, state([VALUE|X])) ), !.
9
10 changeModel( CATEG, NAME, removeFood(VALUE) ) :-
11   model( CATEG, NAME, state(X)),
12   delete_one(VALUE, X, X1),
13   replaceRule( model(CATEG, NAME, state(X)), model( CATEG, NAME, state(X1)) ), !.
14
15 changeModel( CATEG, NAME, VALUE ) :-
16   replaceRule( model(CATEG,NAME,_), model(CATEG,NAME,state(VALUE)) ).
17
18 showFridgeModel :-
19   output("FRIDGE MODEL ---------- "),
20   showFridge,
```

```
21    output("----------------------------").
22
23  showFridge :-
24    model( CATEG, NAME, STATE ),
25    output( model( CATEG, NAME, STATE ) ),
26    fail.
27  showResources.
28
29  delete_one(_, [], []).
30  delete_one(X, [X|T], T).
31  delete_one(X, [H|T], [H|R]) :-
32    delete_one(X, T, R).
33
34  contains(X, [X|T]).
35  contains(X, [_|T]) :-
36    contains(X, T).
37
38  output( M ) :- stdout <- println( M ).
39
40  initResourceTheory :- output("fridgeModel loaded").
41  :- initialization(initResourceTheory).
```

Since the *roombutlerrobot* actor is entitled of the business logic of the system, it will detain the knowledge about fixed information, including the location of the resources in the room and of the preparation set needed for the *prepare the room* task.

butlerRobotKb.pl

```
1   preparation([dish, fruit]).
2
3   goal(pantry, 0, 4).
4   goal(fridge, 5, 0).
5   goal(dishwasher, 5, 4).
6   goal(table, 5, 3).
7   goal(home, 0, 0).
8
9   showPreparationSet :-
10    preparation(X),
11    output(preparation(X)),
12    fail.
13  showPreparationSet.
14
15  output( M ) :- stdout <- println( M ).
16
17  initPreparationTheory :- output("preparationSet loaded").
18  :- initialization(initPreparationTheory).
```

6.5.2 RBR tasks

To implement the tasks, in addition to the already presented QActors, we needed to modify the planner and add some support code.

planexecutor

```
1   QActor planexecutor context ctxRBR {
2     ["
3     var mapEmpty = true
4     val mapname = \"roommap\"
5     //var Tback = 100
```

```
6
7   var Curmove = \"\"
8   var curmoveIsForward = false
9
10  var CurGoal = \"\"
11
12  //REAL ROBOT
13  //var StepTime = 1000
14  //var PauseTime = 500L
15
16  //VIRTUAL ROBOT
17  var StepTime = 330
18  var PauseTime = 400 //increased because it wouldn't do two rotations in a row
19
20  var PauseTimeL = PauseTime.toLong()
21  "]
22
23  State s0 initial {
24    solve ( consult("moves.pl") )
25    run itunibo.planner.plannerUtil.initAI()
26    run itunibo.planner.moveUtils.loadRoomMap(myself, mapname)
27    run itunibo.planner.moveUtils.showCurrentRobotState()
28    ["val MapStr = itunibo.planner.plannerUtil.getMapOneLine()"]
29    forward resourcemodel -m modelUpdate : modelUpdate(roomMap, $MapStr)
30  }
31  Goto waitCmd
32
33  State waitCmd { }
34  Transition t0
35  whenMsg goalUpdate -> createPlan
36
37  State createPlan {
38    printCurrentMessage
39    onMsg( goalUpdate : goalUpdate(G, X, Y) ) {
40
41      ["CurGoal = payloadArg(0)"]
42      run itunibo.planner.plannerUtil.setGoal(payloadArg(1), payloadArg(2))
43    }
44    run itunibo.planner.moveUtils.doPlan(myself)
45  }
46  Goto executePlannedActions
47
48  State executePlannedActions {
49    //solve( showMoves )
50    solve( retract(move(M)) )
51    ifSolved {
52      ["
53      Curmove = getCurSol(\"M\").toString()
54      curmoveIsForward = (Curmove == \"w\")
55      "]
56    } else {
57      ["
58      Curmove = \"\"
59      curmoveIsForward = false
60      "]
61    }
62    println("executePlannedActions doing $Curmove")
63  }
64  Goto cheakAndDoAction if "(Curmove.length > 0)" else goalOk
65
66  State goalOk {
67    forward roombutlerrobot -m goalOk : goalOk($CurGoal)
68  }
69  Goto waitCmd
70
71  //Execute the move if it is a rotation or halt
72  State cheakAndDoAction { }
73  Goto doForwardMove if "curmoveIsForward" else doTheMove
```

```
74
75    State doTheMove {
76      //println("ROTATION")
77      run itunibo.planner.moveUtils.rotate(myself, Curmove, PauseTime)
78    }
79    Goto executePlannedActions
80
81    State doForwardMove {
82      //println("FORWARD")
83      delayVar PauseTimeL //Otherwise is too fast, even with remote interaction
84      run itunibo.planner.moveUtils.attemptTomoveAhead(myself, StepTime)
85    }
86    Transition t0
87    whenMsg stepOk -> handleStepOk
88    whenMsg stepFail -> handleStepFail
89
90    State handleStepOk {
91      run itunibo.planner.moveUtils.updateMapAfterAheadOk(myself)
92      ["val MapStr = itunibo.planner.plannerUtil.getMapOneLine()"]
93      forward resourcemodel -m modelUpdate : modelUpdate(roomMap, $MapStr)
94      run itunibo.planner.moveUtils.showCurrentRobotState()
95    }
96    Goto executePlannedActions
97
98    State handleStepFail {
99      println("NEVER HERE!")
100   }
101 }
```

roomButlerRobotSupport.kt

```
1  package itunibo.rbr
2
3  import it.unibo.kactor.ActorBasic
4
5  object roomButlerRobotSupport {
6    fun getGoalCoordX( actor : ActorBasic, goal : String) : String{
7      actor.solve( "goal($goal, X, _)" )
8      var x = actor.getCurSol("X")
9      return "$x"
10   }
11
12   fun getGoalCoordY( actor : ActorBasic, goal : String) : String{
13     actor.solve( "goal($goal, _, Y)" )
14     var y = actor.getCurSol("Y")
15     return "$y"
16   }
17 }
```

resourceModelSupport.kt

```
1  package itunibo.resModel
2
3  import it.unibo.kactor.ActorBasic
4  import kotlinx.coroutines.launch
5
6  object resourceModelSupport {
7    fun updateRobotModel( actor: ActorBasic, content: String ){
8      actor.solve( "action(robot, move($content) )" ) //change the robot state model
9      actor.solve( "model( A, robot, STATE )" )
10     val RobotState = actor.getCurSol("STATE")
11     actor.scope.launch{
12       actor.emit( "local_robotModelChanged" , "modelChanged( robot, $content)" ) //for the
             ↪ robotmind
```

```
      actor.emit( "modelContent" , "content( robot( $RobotState ) )" )
    }
  }

  fun updateSonarRobotModel( actor: ActorBasic, content: String ){
    actor.solve( "action( sonarRobot, $content )" ) //change the robot state model
    actor.solve( "model( A, sonarRobot, STATE )" )
    val SonarState = actor.getCurSol("STATE")
    actor.scope.launch{
      actor.emit( "modelContent" , "content( sonarRobot( $SonarState ) )" )
    }
  }

  fun updateRoomResourceModel( actor: ActorBasic, resource: String, content: String ){
    actor.solve( "action( $resource, $content )" ) //change the room state model
    actor.solve( "model( A, $resource, STATE )" )
    val RoomResState = actor.getCurSol("STATE")
    actor.scope.launch{
      //sent to notify to the RBR that the change in the model has been performed
      actor.emit( "local_robotModelChanged" , "modelChanged( robot, $content)" ) //for the
          ↪ robotmind
      //the action is performed by the robot immediately. In case of real robots a callback
          ↪ system has to be implemented
      actor.emit( "roomModelChanged" , "modelChanged( $resource, $content)" ) //for the rbr
      actor.emit( "modelContent" , "content($resource( $RoomResState ))" )
    }
  }

  fun updateRoomMapModel( actor: ActorBasic, content: String ) {
    actor.solve( "action( roommap, update('$content'))" )
    actor.scope.launch{
      actor.emit( "modelContent" , "content( roomMap( state( '$content' ) ) )" )
    }
  }

  fun getRobotModel( actor: ActorBasic){
    actor.solve( "model( A, robot, STATE )" )
    val RobotState = actor.getCurSol("STATE")
    //println("  resourceModelSupport updateModel RobotState=$RobotState")
    actor.scope.launch{
      actor.emit( "modelContent" , "content( robot( $RobotState ) )" )
    }
  }

  fun getSonarRobotModel( actor: ActorBasic ){
    actor.solve( "model( A, sonarRobot, STATE )" )
    val SonarState = actor.getCurSol("STATE")
    //println("  resourceModelSupport updateSonarRobotModel SonarState=$SonarState")
    actor.scope.launch{
      actor.emit( "modelContent" , "content( sonarRobot( $SonarState ) )" )
    }
  }

  fun consultRoomResourceModel( actor: ActorBasic, resource: String) {
    actor.solve( "model( A, $resource, state(STATE) )" )
    val RoomResState = actor.getCurSol("STATE")
    actor.scope.launch{
      actor.emit( "modelState" , "modelState( $RoomResState )")
    }
  }
}
```

fridgeModelSupport.kt

```
package itunibo.fridge
import it.unibo.kactor.ActorBasic
```

```
import kotlinx.coroutines.launch

object fridgeModelSupport {

  fun answerRequest(actor: ActorBasic, source: String, foodcode: String) {
    actor.solve( "model( resource, fridge, state(STATE) )" )
    actor.solve( "contains($foodcode, ${actor.getCurSol("STATE")})" )
    actor.scope.launch {
      if(actor.solveOk()) {
        actor.emit("answer", "answer(yes)")
      } else {
        actor.emit("answer", "answer(no)")
      }
    }
  }

  fun updateFridgeModel( actor: ActorBasic, content: String ){
    actor.solve( "action( fridge, $content )" ) //change the fridge state model
    actor.solve( "model( A, fridge, STATE )" )
    actor.scope.launch{
      //sent to notify to the RBR that the change in the model has been performed
      actor.emit( "roomModelChanged" , "modelChanged(fridge, $content)" )
    }
  }
}
```

6.5.3 Emulated frontend

This is just an actor running on the same computational node of the **RBR**, sending in sequence the basic commands to it. This feature is temporary and it only has prototyping purposes, since at the moment we have not developed a proper frontend to allow human interactions.
It also must be removed when trying to run the JUnit tests presented in the next section.

emulatedmaitre

```
QActor emulatedmaitre context ctxRBR {
  State s0 initial {
    println("EMULATED MAITRE STARTS")
  }
  Goto s1 //COMMENT IF USING THE TESTPLANS

  State s1 {
    forward roombutlerrobot -m prepare : prepare
  }
  Transition t0
  whenEvent prepareDone -> s2

  State s2 {
    printCurrentMessage
    delay 2000
    forward roombutlerrobot -m addFood : addFood(lasagna)
  }
  Transition t0
  whenMsg warning -> s3
  whenEvent addDone -> s4

  State s3 {
    printCurrentMessage
    delay 2000
    forward roombutlerrobot -m addFood : addFood(beef)
```

```
26    }
27    Transition t0
28    whenEvent addDone -> s4
29
30    State s4 {
31      printCurrentMessage
32      delay 2000
33      forward roombutlerrobot -m clear : clear
34    }
35  }
```

6.6 Testing

Since we have to present a prototype to the client in a reasonable time, we can not manage to deploy the system to physical machines. Instead we will exploit a virtual environment, implemented in the project "it.unibo.robots19", to which the *basicrobot* is already able to adapt. Since we have no way to represent the dishes and foods that the robot has to carry around the room, the action of removing and adding those items will not be actually performed and we will only have evidence of their occurrence in the models. However, attaining to our vision, the system has been developed so that it would be easy to add those feature with minimum changes in the *basicrobot*.
However, the project already contains some low-level software (to which the basic robot is able to adapt) that could control the basic actions and sensors of a DDR.

6.7 Deployment

The following instructions must be followed for the three main actors (*roombutlerrobot*, *resourcemodel*, *fridge*):

1. Edit the generated file "build.gradle" as follows:
   - uncomment: id 'application'
   - uncomment: mainClassName = 'it.unibo.TODOKt'
   - uncomment: the 'jar' section.
2. Set mainClassName = 'it.unibo.ctxRBR.MainCtxRBR' for the RBR, for example.
3. Run 'gradle build eclipse'.
4. Unzip (somewhere) the file "it.unibo.project_name/build/distributions/it.unibo.project_name-1.0.zip" where "project_name" depends on the project.
5. Copy into the bin directory all the configurations file *name_file.pl*, for example *sysRules.pl*.
6. Edit *basicRobotConfig.pl* to denote the robot to be used (it has to be put in the folder generated by the ctxRobotMind).
7. Execute "it.unibo.project_name.bat"

7 Sprint 2

7.1 Requirements

Design and build the software to put on board of the **Fridge** and of the **RBR**. In particular, the **RBR** must be able to accept the following commands sent by the smart-phone of the **Maitre**:

- *prepare*: the **RBR** must execute in autonomous way the *Prepare the room* task;
- *add food*: the **RBR** must execute in autonomous way the *Add food* task;
- *clear*: the **RBR** must execute in autonomous way the *Clear the room* task.

These tasks are normally executed in sequence, and the main scenario can be summarized as follows:

1. At start, the room is empty (i.e. no people is in it, besides the Maitre) while the **pantry** and the **Fridge** are filled with a proper set of items. The **RBR** is in its home (RH) location and the **dishwasher** is empty.
2. The **Maitre** sends to the **RBR** the *prepare* command and waits for the completion of the related task, consisting in putting on the **table** dishes taken from the **pantry**, and food taken from the **Fridge**. The set of items to put on the **table** in this phase is fixed and properly described somewhere. At the end, the **RBR** is in its RH location again.
3. The Maitre opens the room to people. During the service, the **Maitre** can send to the **RBR** the *add food* command, by specifying a food-code. The **RBR** executes the task, consisting in bringing the food with the given code from the **Fridge** to the **table**, only if the specified food is available in the **Fridge**, otherwise it sends a warning to the **Maitre**. After the task completion, the **RBR** returns is in its RH location. For now, the **Fridge** does not need to use CoAP to communicate.
4. At the end of the party, the **Maitre** sends to the **RBR** the **clear** command and waits for the completion of the task, consisting in bringing non-consumed food again in the **Fridge** and the dishes in the **dishwasher**. The **RBR** returns is in its RH location again.

However, the **Maitre** is able, at any time, to use his/her smart-phone to:

- *consult* the state of the room, e.g. to known what are the objects related to each resource; for example, the object currently posed on the **table**, in the **dishwasher**, etc;
- *stop* or *reactivate* an activated task.

The software to put on the **Fridge** should make the device able to:

- *expose* its current content on the **Maitre** smart-phone;
- *answer* to questions about its content (e.g. if it contains food with a given code).

7.2 Problem analysis

7.2.1 Logical architecture

The logical architecture of a system defines its structure, interactions and behavior.

- **Structure**: the structure of the system defined in the Sprint 1 has not changed.
- **Interaction**: since the new requirements are all related to the implementation of the frontend, all of the smart entities will now communicate between each other. In particular, some of the communications of the **RBR** with **Maitre** and **Fridge** will occur through the *resourcemodel* component:
  - **RBR**: it must be able to communicate with both the **Maitre** and the **Fridge**. In particular:
    - *roombutlerrobot*: communicates with *resourcemodel*, *planexecutor*, **Maitre** and **Fridge**.
    - *planexecutor*: communicates with *roombutlerrobot* and *resourcemodel*;
    - *resourcemodel*: communicates with *roombutlerrobot*, *planexecutor*, **Maitre** and **Fridge**.
  - **Fridge**: it will communicate with both the **RBR** and the **Maitre**.
  - **Maitre**: it will communicate with both the **RBR** and the **Fridge**.
- **Behavior**: using the QActor meta-model we will express the behavior of the entities as Finite State Machines (FSM).

roombutlerrobot

```
1  QActor roombutlerrobot context ctxRBR {
2    ["
3    var NextGoal = \"\"
4    var GoalX = \"\"
5    var GoalY = \"\"
6    var CurObject = \"\"
7    var actionDone = false
8    "]
9
10   State s0 initial {
11     solve( consult("butlerRobotKb.pl") )
12     solve( consult("sysRules.pl") )
13   }
14   Goto waitCmd
15
```

```
16      State waitCmd { println("&&& RBR waitCmd ... ") }
17      Transition t0
18      whenMsg prepChange -> changePrepSet
19      whenMsg prepare -> prepareTheRoomInit
20      whenMsg addFood -> addFoodOnTableInit
21      whenMsg clear  -> clearTheRoomInit
22
23      State changePrepSet {
24        onMsg(prepChange : prepChange(P)) {
25          run itunibo.rbr.roomButlerRobotSupport.changePrepSet(myself, payloadArg(0))
26        }
27      }
28      Goto waitCmd
29
30      //PREPARE THE ROOM
31      State prepareTheRoomInit {
32        ["
33        actionDone = false
34        NextGoal = \"pantry\"
35        GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
36        GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
37        "]
38        forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
39
40        //at least one item from the pantry must be in the preparation set
41        ["solve( \"preparation([H|T])\" )"]
42        ifSolved {
43          ["
44          solve( \"replaceRule(preparation([H|T]), preparation(T))\" )
45          CurObject = getCurSol(\"H\").toString()
46          "]
47        }
48      }
49      Goto waitPrepare
50
51      State waitPrepare { println("WAIT PREPARE") }
52      Transition t0
53      whenMsg goalOk -> prepareTheRoomContinue
54      whenEvent roomModelChanged -> prepareTheRoomContinue
55
56      State prepareTheRoomContinue {
57        printCurrentMessage
58        delay 1000
59
60        onMsg( goalOk : goalOk(pantry) ) {
61          forward resourcemodel -m modelUpdate : modelUpdate(pantry, remove($CurObject))
62        }
63        onMsg( goalOk : goalOk(fridge) ) {
64          forward fridge -m modelUpdate : modelUpdate(fridge, remove($CurObject))
65        }
66        onMsg( goalOk : goalOk(table) ) {
67          forward resourcemodel -m modelUpdate : modelUpdate(table, add($CurObject))
68        }
69        onMsg( goalOk : goalOk(home) ) {
70          ["actionDone = true"]
71        }
72
73        onMsg( roomModelChanged : modelChanged(pantry, remove(_)) ) {
74          ["
75          NextGoal = \"table\"
76          GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
77          GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
78          "]
79          forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
80        }
81        onMsg( roomModelChanged : modelChanged(fridge, remove(_)) ) {
82          ["
83          NextGoal = \"table\"
```

```
84          GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
85          GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
86          "]
87          forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
88        }
89        onMsg( roomModelChanged : modelChanged(table, add(_)) ) {
90          ["solve( \"preparation([H|T])\" )"]
91          println(currentSolution)
92          ifSolved {
93            ["
94            solve( \"replaceRule(preparation([H|T]), preparation(T))\" )
95            CurObject = getCurSol(\"H\").toString()
96            if(CurObject.equals(\"dish\"))
97            NextGoal = \"pantry\"
98            else
99            NextGoal = \"fridge\"
100           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
101           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
102           "]
103         }
104         else { //preparation list empty
105           ["
106           CurObject = \"\"
107           NextGoal = \"home\"
108           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
109           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
110           "]
111         }
112         println("$NextGoal")
113         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
114       }
115     }
116     Goto waitCmd if "actionDone" else waitPrepare
117
118     //ADD FOOD ON THE TABLE
119     State addFoodOnTableInit {
120       ["actionDone = false"]
121       onMsg(addFood : addFood(F)) {
122         //check if the fridge contains F
123         forward fridge -m request : request(roombutlerrobot, $payloadArg(0))
124         ["CurObject = \"${payloadArg(0)}\" "]
125       }
126     }
127     Transition t0
128     whenMsg answer -> checkFridgeAnswer
129
130     State checkFridgeAnswer {
131       printCurrentMessage
132       onMsg(answer : answer(yes)) {
133         ["
134         NextGoal = \"fridge\"
135         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
136         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
137         "]
138         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
139       }
140       onMsg(answer : answer(no)) {
141         ["CurObject = \"\"
142         NextGoal = \"\"
143         GoalX = \"\"
144         GoalY = \"\"
145         actionDone = true
146         "]
147         forward resourcemodel -m warning : warning
148       }
149     }
150     Goto waitCmd if "actionDone" else waitAddFood
151
```

```
152     State waitAddFood { println("WAIT ADD FOOD") }
153     Transition t0
154     whenMsg goalOk -> addFoodOnTheTableContinue
155     whenEvent roomModelChanged -> addFoodOnTheTableContinue
156
157     State addFoodOnTheTableContinue {
158       delay 1000
159
160       onMsg( goalOk : goalOk(fridge) ) {
161         forward fridge -m modelUpdate : modelUpdate(fridge, remove($CurObject))
162       }
163       onMsg( goalOk : goalOk(table) ) {
164         forward resourcemodel -m modelUpdate : modelUpdate(table, add($CurObject))
165       }
166       onMsg( goalOk : goalOk(home) ) {
167         ["actionDone = true"]
168       }
169
170       onMsg( roomModelChanged : modelChanged(fridge, remove(_)) ) {
171         ["
172         NextGoal = \"table\"
173         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
174         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
175         "]
176         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
177       }
178       onMsg( roomModelChanged : modelChanged(table, add(_)) ) {
179         ["
180         CurObject = \"\"
181         NextGoal = \"home\"
182         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
183         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
184         "]
185         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
186       }
187     }
188     Goto waitCmd if "actionDone" else waitAddFood
189
190     //CLEAR THE ROOM
191     State clearTheRoomInit {
192       ["actionDone = false"]
193       forward resourcemodel -m modelConsult : modelConsult(table)
194     }
195     Transition t0
196     whenEvent modelState -> checkTableState
197
198     State checkTableState {
199       printCurrentMessage
200       onMsg(modelState : modelState(S) ) {
201         ["solve( \"table(_)\" )"]
202         ifSolved {
203           ["solve( \"replaceRule(table(_), table(${payloadArg(0)}))\" )"]
204         } else {
205           ["solve( \"addRule(table(${payloadArg(0)}))\" )"]
206         }
207         println(currentSolution)
208       }
209       ["
210       NextGoal = \"table\"
211       GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
212       GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
213       "]
214       forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
215     }
216     Goto waitClear
217
218     State waitClear { }
219     Transition t0
```

```
220     whenMsg goalOk -> clearTheRoomContinue
221     whenEvent roomModelChanged -> clearTheRoomContinue
222
223     State clearTheRoomContinue {
224       printCurrentMessage
225       delay 1000
226       onMsg( goalOk : goalOk(fridge) ) {
227         forward fridge -m modelUpdate : modelUpdate(fridge, add($CurObject))
228       }
229       onMsg( goalOk : goalOk(dishwasher) ) {
230         forward resourcemodel -m modelUpdate : modelUpdate(dishwasher, add($CurObject))
231       }
232       onMsg( goalOk : goalOk(table) ) {
233         ["solve( \"table([H|T])\" )"]
234         println(currentSolution)
235         ifSolved {
236           ["
237           solve( \"replaceRule(table([H|T]), table(T))\" )
238           CurObject = getCurSol(\"H\").toString()
239           if(CurObject.equals(\"dish\"))
240           NextGoal = \"dishwasher\"
241           else
242           NextGoal = \"fridge\"
243           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
244           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
245           "]
246           forward resourcemodel -m modelUpdate : modelUpdate(table, remove($CurObject))
247         }
248         else { //no more objects on the table
249           ["
250           CurObject = \"\"
251           NextGoal = \"home\"
252           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
253           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
254           "]
255           forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
256         }
257         //println("$NextGoal")
258       }
259       onMsg( goalOk : goalOk(home) ) {
260         ["actionDone = true"]
261       }
262
263       onMsg( roomModelChanged : modelChanged(dishwasher, add(_)) ) {
264         ["
265         NextGoal = \"table\"
266         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
267         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
268         "]
269         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
270       }
271       onMsg( roomModelChanged : modelChanged(fridge, add(_)) ) {
272         ["
273         NextGoal = \"table\"
274         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
275         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
276         "]
277         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
278       }
279       onMsg( roomModelChanged : modelChanged(table, remove(_)) ) {
280         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
281       }
282     }
283     Goto waitCmd if "actionDone" else waitClear
284   }
```

fridge

```
QActor fridge context ctxFridge{
  State s0 initial {
    solve( consult("sysRules.pl") )
    solve( consult("fridgeModel.pl") )
  }
  Goto waitCmd

  State waitCmd { }
  Transition t0
  whenMsg request -> handleRequest
  whenMsg expose -> exposeModel
  whenMsg modelUpdate -> updateModel

  State handleRequest {
    onMsg(request : request(S, F)) {
      run itunibo.fridge.fridgeModelSupport.answerRequest(myself, payloadArg(0), payloadArg(1))
    }
  }
  Goto waitCmd

  State exposeModel {
    run itunibo.fridge.fridgeModelSupport.exposeFridgeModel(myself)
  }
  Goto waitCmd

  State updateModel {
    onMsg(modelUpdate : modelUpdate(fridge, V)) {
      //["var Action = payloadArg(0)+\"(\"+payloadArg(1)+\")\""]
      run itunibo.fridge.fridgeModelSupport.updateFridgeModel(myself, payloadArg(1))
    }
  }
  Goto waitCmd
}
```

resourcemodel

```
QActor resourcemodel context ctxRobotMind{

  State s0 initial {
    solve( consult("sysRules.pl") )
    solve( consult("resourceModel.pl") )
    solve( showResourceModel )
  }
  Goto waitMsg

  State waitMsg{ }
  Transition t0
  whenMsg modelChange -> changeModel
  whenMsg modelUpdate -> updateModel
  whenMsg modelConsult -> consultModel
  whenMsg modelExpose -> exposeModel
  whenMsg warning  -> emitWarning

  State updateModel{
    printCurrentMessage
    onMsg( modelUpdate : modelUpdate(robot, V ) ) {
      run itunibo.resModel.resourceModelSupport.updateRobotModel( myself, payloadArg(1) )
    }
    onMsg( modelUpdate : modelUpdate(sonarRobot,V ) ) {
      run itunibo.resModel.resourceModelSupport.updateSonarRobotModel( myself, payloadArg(1)
          ↪ )
    }
    onMsg( modelUpdate : modelUpdate(pantry, V) ) {
```

```
27          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
28        }
29        onMsg( modelUpdate : modelUpdate(table, V) ) {
30          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
31        }
32        onMsg( modelUpdate : modelUpdate(dishwasher, V) ) {
33          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
34        }
35        onMsg( modelUpdate : modelUpdate(roomMap,V ) ) {
36          run itunibo.resModel.resourceModelSupport.updateRoomMapModel( myself, payloadArg(1) )
37        }
38      }
39      Goto waitMsg
40
41      State changeModel{
42        //ROBOT MOVE
43        onMsg( modelChange : modelChange( robot,V ) ) { // V= w | ...
44          run itunibo.resModel.resourceModelSupport.updateRobotModel( myself, payloadArg(1) )
45          emit local_robotModelChanged : modelChanged( robot, $payloadArg(1)) //for the
                ↪ robotmind
46        }
47      }
48      Goto waitMsg
49
50      State consultModel {
51        onMsg( modelConsult : modelConsult(_) ) {
52          run itunibo.resModel.resourceModelSupport.consultRoomResourceModel(myself, payloadArg
                ↪ (0))
53        }
54      }
55      Goto waitMsg
56
57      State exposeModel {
58        onMsg( modelExpose : modelExpose ) {
59          run itunibo.resModel.resourceModelSupport.getRobotModel( myself )
60          run itunibo.resModel.resourceModelSupport.getRoomMapModel(myself)
61          run itunibo.resModel.resourceModelSupport.exposeRoomResourceModel( myself, "table" )
62          run itunibo.resModel.resourceModelSupport.exposeRoomResourceModel( myself, "pantry" )
63          run itunibo.resModel.resourceModelSupport.exposeRoomResourceModel( myself, "
                ↪ dishwasher" )
64        }
65      }
66      Goto waitMsg
67
68      State emitWarning {
69        onMsg(warning : warning) {
70          run itunibo.resModel.resourceModelSupport.sendWarning(myself)
71        }
72      }
73      Goto waitMsg
74    }
```

7.3 Product backlog

- Implement functionalities for the **RBR**:
  - expose (30 minutes).
- Implement functionalities for the **Fridge**:
  - ask (30 minutes);

– expose (30 minutes).

- Create a proper frontend. (3 hours).
- Create formal TestPlans with JUnit. (2 hours).

7.4 TestPlans

TestResources

```
class TestResources {
  var resource : ActorBasic? = null
  var mqttClient : MqttClient? = null
  var broker : String = "tcp://localhost"
  var resmodelTopic : String = "unibo/qak/resourcemodel"
  var eventTopic : String = "unibo/qak/events"
  var resStates = ArrayList<String>()
  var clientId : String = "sprint_2"
  var qos : Int = 2;

  val eventCallback = object : MqttCallback {
    override fun connectionLost(cause: Throwable) {
      //connectionStatus = false
      // Give your callback on failure here
    }
    override fun messageArrived(topic: String, message: MqttMessage) {
      try {
        val data = String(message.payload, charset("UTF-8"))
        // data is the desired received message
        // Give your callback on message received here
        resStates.add(data)
      } catch (e: Exception) {
        // Give your callback on error here
      }
    }
    override fun deliveryComplete(token: IMqttDeliveryToken) {
      // Acknowledgement on delivery complete
    }
  }

  @Before
  fun systemSetUp() {

    try {
      println("%%%%%%%%%%%%%% Sprint 2 TestResources starting Mqtt Client")
      mqttClient = MqttClient(broker, clientId)
      var connOpts = MqttConnectOptions()
      connOpts.setCleanSession(true)
      mqttClient!!.connect(connOpts)
      mqttClient!!.setCallback(eventCallback)
      mqttClient!!.subscribe(eventTopic)
    }
    catch (e : MqttException) {
      println("MQTT EXCEPTION IN SETUP")
    }
    GlobalScope.launch {
      it.unibo.ctxRobotMind.main()
    }
    delay(5000) //give the time to start
    resource = sysUtil.getActor("resourcemodel")
  }

  @After
  fun terminate() {
    println("%%%%%%% Sprint 2 TestResources terminate ")
```

```
56      try {
57        mqttClient!!.disconnect()
58        mqttClient!!.close()
59      } catch (e : MqttException) {
60        println("MQTT EXCEPTION IN TERMINATE")
61      }
62    }
63
64    @Test
65    fun sprint1Test() {
66      println("%%%%%%% Sprint 2 Functional TestResources starts ")
67      expose()
68    }
69
70    fun expose() {
71      println("%%%%%%% Sprint 2 TestResources expose Task")
72      sendCmd("modelExpose", "")
73      delay(1000)
74      solveCheckGoal(resource!!, resStates.get(0), "msg(modelContent,event,resourcemodel,none,content(
            ↪ robot(state(stopped))),16)")
75      solveCheckGoal(resource!!, resStates.get(1), "msg(modelContent,event,resourcemodel,none,content(
            ↪ roomMap(state(unknown))),17)")
76      solveCheckGoal(resource!!, resStates.get(2), "msg(modelContent,event,resourcemodel,none,content(
            ↪ table(state([]))),18)")
77      solveCheckGoal(resource!!, resStates.get(3), "msg(modelContent,event,resourcemodel,none,content(
            ↪ pantry(state([dish,dish,dish,dish,dish,dish,dish,dish,dish,dish,dish,dish,dish,
            ↪ dish,dish]))),19)")
78      solveCheckGoal(resource!!, resStates.get(4), "msg(modelContent,event,resourcemodel,none,content(
            ↪ dishwasher(state([]))),20)")
79    }
80
81    //-----------------------------------------------------------------------------
82
83    fun sendCmd(cmd : String, content : String) {
84      println("--- RBR performing performing task $cmd")
85      var msg : String
86      if (content != "") {
87        msg = "msg($cmd,dispatch,js,resourcemodel,$cmd($content),1)"
88      } else {
89        msg = "msg($cmd,dispatch,js,resourcemodel,$cmd,1)"
90      }
91      try {
92        var mqttMsg = MqttMessage(msg.toByteArray())
93        mqttClient!!.publish(resmodelTopic, mqttMsg)
94      }
95      catch (e : MqttException) {
96        println("MQTT EXCEPTION IN DOTASK")
97      }
98    }
99
100   fun solveCheckGoal( actor : ActorBasic, goal : String ){
101     actor.solve( goal )
102     var result = actor.resVar
103     println(" %%%%%%% actor={$actor.name} goal= $goal result = $result")
104     assertTrue("", result == "success" )
105   }
106
107   fun solveCheckGoal( actor : ActorBasic, ans : String, goal : String ){
108     println(" %%%%%%% actor={$actor.name} goal= $goal ans= $ans")
109     assertTrue("", ans == goal )
110   }
111
112
113   fun delay( time : Long ){
114     Thread.sleep( time )
115   }
116 }
```

TestFridge

```
class TestFridge {
  var fridge : ActorBasic? = null
  var mqttClient : MqttClient? = null
  var broker : String = "tcp://localhost"
  var fridgeTopic : String = "unibo/qak/fridge"
  var eventTopic : String = "unibo/qak/events"
  var fridgeAns : String = ""
  var clientId : String = "sprint_2"
  var qos : Int = 2;

  val eventCallback = object : MqttCallback {
    override fun connectionLost(cause: Throwable) {
      //connectionStatus = false
      // Give your callback on failure here
    }
    override fun messageArrived(topic: String, message: MqttMessage) {
      try {
        val data = String(message.payload, charset("UTF-8"))
        // data is the desired received message
        // Give your callback on message received here
        fridgeAns = data
      } catch (e: Exception) {
        // Give your callback on error here
      }
    }
    override fun deliveryComplete(token: IMqttDeliveryToken) {
      // Acknowledgement on delivery complete
    }
  }

  @Before
  fun systemSetUp() {

    try {
      println("%%%%%%%%%%%%%% Sprint 2 TestFridge starting Mqtt Client")
      mqttClient = MqttClient(broker, clientId)
      var connOpts = MqttConnectOptions()
      connOpts.setCleanSession(true)
      mqttClient!!.connect(connOpts)
      mqttClient!!.setCallback(eventCallback)
      mqttClient!!.subscribe(eventTopic)
    }
    catch (e : MqttException) {
      println("MQTT EXCEPTION IN SETUP")
    }
    GlobalScope.launch {
      it.unibo.ctxFridge.main()
    }
    delay(5000) //give the time to start
    fridge = sysUtil.getActor("fridge")
  }

  @After
  fun terminate() {
    println("%%%%%%% Sprint 2 TestFridge terminate ")
    try {
      mqttClient!!.disconnect()
      mqttClient!!.close()
    } catch (e : MqttException) {
      println("MQTT EXCEPTION IN TERMINATE")
    }
  }

  @Test
  fun sprint2Test() {
```

```
66      println("%%%%%%% Sprint 2 Functional TestFridge starts ")
67      expose()
68      consultOk("fruit")
69      consultFail("pasta")
70    }
71
72    fun expose() {
73      println("%%%%%%% Sprint 2 TestFridge expose Task")
74      sendCmd("expose", "")
75      delay(1000)
76      solveCheckGoal(fridge!!, fridgeAns, "msg(fridgeContent,event,fridge,none,content(fridge(state([
            ↪ fruit,beef]))),6)")
77    }
78
79    fun consultOk(foodCode : String) {
80      println("%%%%%%% Sprint 2 TestFridge consultOk Task")
81      sendCmd("request", "fridge, $foodCode")
82      delay(1000)
83      solveCheckGoal(fridge!!, fridgeAns, "msg(answer,event,fridge,none,answer(content(yes)),7)")
84    }
85
86    fun consultFail(foodCode : String) {
87      println("%%%%%%% Sprint 2 TestFridge consultOFail Task")
88      sendCmd("request", "fridge, $foodCode")
89      delay(1000)
90      solveCheckGoal(fridge!!, fridgeAns, "msg(answer,event,fridge,none,answer(content(no)),8)")
91    }
92
93    //-----------------------------------------------------------------------------
94
95    fun sendCmd(cmd : String, content : String) {
96      println("--- Fridge performing performing task $cmd")
97      var msg : String
98      if (content != "") {
99        msg = "msg($cmd,dispatch,js,fridge,$cmd($content),1)"
100     } else {
101       msg = "msg($cmd,dispatch,js,fridge,$cmd,1)"
102     }
103     try {
104       var mqttMsg = MqttMessage(msg.toByteArray())
105       mqttClient!!.publish(fridgeTopic, mqttMsg)
106     }
107     catch (e : MqttException) {
108       println("MQTT EXCEPTION IN DOTASK")
109     }
110   }
111
112   fun solveCheckGoal( actor : ActorBasic, goal : String ){
113     actor.solve( goal )
114     var result = actor.resVar
115     println(" %%%%%%% actor={$actor.name} goal= $goal result = $result")
116     assertTrue("", result == "success" )
117   }
118
119   fun solveCheckGoal( actor : ActorBasic, ans : String, goal : String ){
120     println(" %%%%%%% actor={$actor.name} goal= $goal ans= $ans")
121     assertTrue("", ans == goal )
122   }
123
124   fun delay( time : Long ){
125     Thread.sleep( time )
126   }
127 }
```

7.5 Project

7.5.1 RBR functionalities

Expose This functionality allows the **Maitre** to see, through the smart-phone, the state of the resources in the room. Since that particular piece of information is handled by the *resourcemodel*, the expose functionality will also be delegated to it. The command will be sent by the **Maitre** device on startup, then every time a change occur in the model, a communication will be sent to keep everything up to date.

roomButlerRobotSupport.kt

```kotlin
package itunibo.rbr

import it.unibo.kactor.ActorBasic

object roomButlerRobotSupport {
  fun getGoalCoordX( actor : ActorBasic, goal : String ) : String{
    actor.solve( "goal($goal, X, _)" )
    var x = actor.getCurSol("X")
    return "$x"
  }

  fun getGoalCoordY( actor : ActorBasic, goal : String ) : String{
    actor.solve( "goal($goal, _, Y)" )
    var y = actor.getCurSol("Y")
    return "$y"
  }

  fun changePrepSet( actor : ActorBasic, content : String ) {
    actor.solve( "replaceRule(preparation(_), preparation([$content]))" )
  }
}
```

resourceModelSupport.kt

```kotlin
package itunibo.resModel

import it.unibo.kactor.ActorBasic
import kotlinx.coroutines.launch

object resourceModelSupport {
  fun updateRobotModel( actor: ActorBasic, content: String ){
    actor.solve( "action(robot, move($content) )" ) //change the robot state model
    actor.solve( "model( A, robot, STATE )" )
    val RobotState = actor.getCurSol("STATE")
    actor.scope.launch{
      actor.emit( "robotModelChanged" , "modelChanged( robot, $content)" ) //for the robotmind
      actor.emit( "modelContent" , "content( robot( $RobotState ) )" )
    }
  }

  fun updateSonarRobotModel( actor: ActorBasic, content: String ){
    actor.solve( "action( sonarRobot, $content )" ) //change the robot state model
    actor.solve( "model( A, sonarRobot, STATE )" )
    val SonarState = actor.getCurSol("STATE")
    actor.scope.launch{
      actor.emit( "modelContent" , "content( sonarRobot( $SonarState ) )" )
    }
  }

```

```
26    fun updateRoomResourceModel( actor: ActorBasic, resource: String, content: String ){
27      actor.solve( "action( $resource, $content )" ) //change the room state model
28      actor.solve( "model( A, $resource, STATE )" )
29      val RoomResState = actor.getCurSol("STATE")
30      actor.scope.launch{
31        //sent to notify to the RBR that the change in the model has been performed
32        actor.emit( "local_robotModelChanged" , "modelChanged( robot, $content)" ) //for the
              ↪ robotmind
33        //the action is performed by the robot immediately. In case of real robots a callback
              ↪ system has to be implemented
34        actor.emit( "roomModelChanged" , "modelChanged( $resource, $content)" ) //for the rbr
35        actor.emit( "modelContent" , "content($resource( $RoomResState ))" )
36      }
37    }
38
39    fun updateRoomMapModel( actor: ActorBasic, content: String ) {
40      actor.solve( "action( roommap, update('$content'))" )
41      actor.scope.launch{
42        actor.emit( "modelContent" , "content( roomMap( state( '$content' ) ) )" )
43      }
44    }
45
46    fun getRobotModel( actor: ActorBasic){
47      actor.solve( "model( A, robot, STATE )" )
48      val RobotState = actor.getCurSol("STATE")
49      actor.scope.launch{
50        actor.emit( "modelContent" , "content( robot( $RobotState ) )" )
51      }
52    }
53
54    fun getSonarRobotModel( actor: ActorBasic ){
55      actor.solve( "model( A, sonarRobot, STATE )" )
56      val SonarState = actor.getCurSol("STATE")
57      actor.scope.launch{
58        actor.emit( "modelContent" , "content( sonarRobot( $SonarState ) )" )
59      }
60    }
61
62    fun consultRoomResourceModel( actor: ActorBasic, resource: String) {
63      actor.solve( "model( A, $resource, state(STATE) )" )
64      val RoomResState = actor.getCurSol("STATE")
65      actor.scope.launch{
66        actor.emit( "modelState" , "modelState( $RoomResState )")
67      }
68    }
69
70    fun exposeRoomResourceModel( actor: ActorBasic, resource: String ){
71      actor.solve( "model( A, $resource, STATE )" )
72      val RoomResState = actor.getCurSol("STATE")
73      actor.scope.launch{
74        actor.emit( "modelContent" , "content($resource( $RoomResState ) )" )
75      }
76    }
77
78    fun getRoomMapModel( actor: ActorBasic ) {
79      actor.solve( "model(A, roommap, STATE)" )
80      val RoomMapState = actor.getCurSol("STATE")
81      actor.scope.launch{
82        actor.emit( "modelContent" , "content( roomMap( $RoomMapState ) )" )
83      }
84    }
85
86    fun sendWarning(actor: ActorBasic) {
87      actor.scope.launch{
88        actor.emit( "warning" , "content( warning )" )
89      }
90    }
91  }
```

7.5.2 Fridge functionalities

Ask This functionality was actually already implemented in the previous Sprint. Since the **RBR** needed to do the same thing in the *add food on the table* task, we designed the interaction so that it contains the source of the message. Now we just have to check it and react differently to a different sender.

fridgeModelSupport.kt

```
package itunibo.fridge

import it.unibo.kactor.ActorBasic
import kotlinx.coroutines.launch

object fridgeModelSupport{

  fun answerRequest(actor: ActorBasic, source: String, foodcode: String) {
    actor.solve( "model( resource, fridge, state(STATE) )" )
    actor.solve( "contains($foodcode, ${actor.getCurSol("STATE")})" )
    actor.scope.launch {
      if(actor.solveOk()) {
        if(source == "roombutlerrobot") {
          actor.emit("answer", "answer(yes)")
        } else {
          actor.emit("answer", "answer(content(yes))")
        }
      } else {
        if(source == "roombutlerrobot") {
          actor.emit("answer", "answer(no)")
        } else {
          actor.emit("answer", "answer(content(no))")
        }
      }
    }
  }

  fun exposeFridgeModel( actor: ActorBasic ){
    actor.solve( "model( A, fridge, STATE )" )
    val FridgeState = actor.getCurSol("STATE")
    actor.scope.launch{
      actor.emit( "fridgeContent" , "content( fridge( $FridgeState ) )" )
    }
  }

  fun updateFridgeModel( actor: ActorBasic, content: String ){
    actor.solve( "action( fridge, $content )" ) //change the fridge state model
    actor.solve( "model( A, fridge, STATE )" )
    //val FridgeState = actor.getCurSol("STATE")
    actor.scope.launch{
      //sent to notify to the RBR that the change in the model has been performed
      actor.emit( "roomModelChanged" , "modelChanged(fridge, $content)" ) //for the RBR
    }
  }
}
```

7.5.3 Frontend

We already have available a developed frontend, based on Express (a web framework for Node.js), which provides communications via MQTT. We will then use an adapted version of it, as developing a native application for smart-phone would require at least ten time as estimated.

The frontend also have a section where it is possible to modify the resources of the room. This feature is needed for testing purposes and, in particular, to test in a convincing way the *clear the room task*: if a participant of the party removes a food from the table, the model of the robot is not updated and, since it is not able to understand by itself what lies on the table, it will try to get everything it put onto it.

applCode.js

```
/*
frontend/uniboSupports/applCode
*/
const express  = require('express');
const path  = require('path');
//const favicon = require('serve-favicon');
const logger  = require('morgan'); //see 10.1 of nodeExpressWeb.pdf;
//const cookieParser= require('cookie-parser');
const bodyParser  = require('body-parser');
const fs  = require('fs');
const index = require('./appServer/routes/index');
var io ;  //Upgrade for socketIo;

//for delegate
const mqttUtils = require('./uniboSupports/mqttUtils');

var app = express();


// view engine setup;
app.set('views', path.join(__dirname, 'appServer', 'views'));
app.set('view engine', 'ejs');

//create a write stream (in append mode) ;
var accessLogStream = fs.createWriteStream(path.join(__dirname, 'morganLog.log'), {flags: 'a'})
app.use(logger("short", {stream: accessLogStream}));

//Creates a default route. Overloads app.use('/', index);
//app.get("/", function(req,res){ res.send("Welcome to frontend Server"); } );

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));   //shows commands, e.g. GET /pi 304 23.123 ms - -;
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
//app.use(cookieParser());

app.use(express.static(path.join(__dirname, 'public')));
app.use(express.static(path.join(__dirname, 'jsCode'))); //(***)


app.get('/', function(req, res) {
  res.render("index");
  console.log("starting")
  //when the server starts acquire the state of the resources in the room.
  //they will be kept updated in real-time
  setTimeout(delegateForResource, 200, "modelExpose", req, res);
});

/*
* ====================== COMMANDS ================
*/
//TESTING
app.post("/changePrepSet", function (req, res, next) {
  content = req.body.prep_set;
  delegateForAppl("prepChange", req, res, content);
```

```
57    next();
58  });
59  app.post("/addFridge", function (req, res, next) {
60    content = "fridge, add(" + req.body.foodcode_resfridge + ")";
61    delegateForFridge("modelUpdate", req, res, content);
62    next();
63  });
64  app.post("/removeFridge", function (req, res, next) {
65    content = "fridge, remove(" + req.body.foodcode_resfridge + ")";
66    delegateForFridge("modelUpdate", req, res, content);
67    next();
68  });
69  app.post("/addTable", function (req, res, next) {
70    content = "table, add(" + req.body.itemcode_table + ")";
71    delegateForResource("modelUpdate", req, res, content);
72    next();
73  });
74  app.post("/removeTable", function (req, res, next) {
75    content = "table, remove(" + req.body.itemcode_table + ")";
76    delegateForResource("modelUpdate", req, res, content);
77    next();
78  });
79  app.post("/addPantry", function (req, res, next) {
80    content = "pantry, add(" + req.body.itemcode_pantry + ")";
81    delegateForResource("modelUpdate", req, res, content);
82    next();
83  });
84  app.post("/removePantry", function (req, res, next) {
85    content = "pantry, remove(" + req.body.itemcode_pantry + ")";
86    delegateForResource("modelUpdate", req, res, content);
87    next();
88  });
89  app.post("/addDishwasher", function (req, res, next) {
90    content = "dishwasher, add(" + req.body.itemcode_dishwasher + ")";
91    delegateForResource("modelUpdate", req, res, content);
92    next();
93  });
94  app.post("/removeDishwasher", function (req, res, next) {
95    content = "dishwasher, remove(" + req.body.itemcode_dishwasher + ")";
96    delegateForResource("modelUpdate", req, res, content);
97    next();
98  });
99
100 //APPLICATION
101 app.post("/prepare", function(req, res,next) {
102   delegateForAppl("prepare", req, res);
103   next();
104 });
105 app.post("/clear", function(req, res,next) {
106   delegateForAppl("clear", req, res);
107   next();
108 });
109 app.post("/addFood", function (req, res, next) {
110   content = req.body.foodcode_app
111   delegateForAppl("addFood", req, res, content);
112   next();
113 });
114 app.post("/expose", function (req, res, next) {
115   delegateForFridge("expose", req, res);
116   next();
117 });
118 app.post("/ask", function (req, res, next) {
119   content = req.body.foodcode_fridge
120   delegateForFridge("request", req, res, content);
121   next();
122 });
123
124 //================== UTILITIES ========================
```

```
125
126 var result = "";
127
128 app.setIoSocket = function( iosock ){
129   io = iosock;
130   mqttUtils.setIoSocket(iosock);
131   coap.setIoSocket(iosock);
132   console.log("app SETIOSOCKET io=" + io);
133 }
134
135 function delegateForAppl(cmd, req, res, content) {
136   console.log("app delegateForAppl cmd=" + cmd);
137   result = "Web server delegateForAppl: " + cmd;
138
139   if(arguments.length === 4) {
140     publishMsgToRobotapplication(cmd, content);
141   }
142   else {
143     publishMsgToRobotapplication(cmd);
144   }
145 }
146
147 function delegateForResource(cmd, req, res) {
148   console.log("app delegateForResource cmd=" + cmd);
149   result = "Web server delegateForResource: " + cmd;
150
151   publishMsgToResourceModel(cmd);
152 }
153
154 function delegateForFridge(cmd, req, res, content) {
155   console.log("app delegateForFridge cmd=" + cmd);
156   result = "Web server delegateForFridge: " + cmd;
157
158   if (arguments.length === 4) {
159     publishMsgToFridgeapplication(cmd, content);
160   }
161   else {
162     publishMsgToFridgeapplication(cmd);
163   }
164 }
165
166 /*
167 * ============ TO THE BUSINESS LOGIC =======
168 */
169
170 var publishMsgToRobotapplication = function (cmd, content) {
171   var msgstr;
172   if (arguments.length === 2) {
173     msgstr = "msg(" + cmd + ",dispatch,js,roombutlerrobot," + cmd + "(" + content + "),1)";
174   } else {
175     msgstr = "msg(" + cmd + ",dispatch,js,roombutlerrobot," + cmd + ",1)";
176   }
177   console.log("publishMsgToRobotapplication/" + arguments.length +" forward> " + msgstr);
178   mqttUtils.publish(msgstr, "unibo/qak/roombutlerrobot");
179 }
180
181 var publishMsgToResourceModel = function (cmd, content) {
182   var msgstr;
183   msgstr = "msg(" + cmd + ",dispatch,js,resourcemodel," + cmd + ",1)";
184   console.log("publishMsgToResourceModel/ forward> " + msgstr);
185   mqttUtils.publish(msgstr, "unibo/qak/resourcemodel");
186 }
187
188 var publishMsgToFridgeapplication = function (cmd, content) {
189   var msgstr;
190   if (arguments.length === 2) {
191     if (cmd == "modelUpdate") {
192       msgstr = "msg(" + cmd + ",dispatch,js,fridge," + cmd + "(" + content + "),1)";
```

```
    } else {
      msgstr = "msg(" + cmd + ",dispatch,js,fridge," + cmd + "(maitre," + content + "),1)";
    }
  } else {
    msgstr = "msg(" + cmd + ",dispatch,js,fridge," + cmd + ",1)";
  }
  console.log("publishMsgToFridgeapplication/" + arguments.length + " forward> " + msgstr);
  mqttUtils.publish(msgstr, "unibo/qak/fridge");
}

/*
* ====================== REPRESENTATION ================
*/
app.use( function(req,res){
  console.info("SENDING THE ANSWER " + result + " json:" + req.accepts('josn') );
  try{
    console.log("answer> "+ result );
    /*
    if (req.accepts('json')) {
      return res.send(result);  //give answer to curl / postman
    } else {
      return res.render('index' );
    };
    */
    //res.send(result);
    //return res.render('index' ); //NO: we loose the message sent via socket.io
  }catch(e){console.info("SORRY ..." + e);}
}
);

//app.use(converter());

/*
* ============ ERROR HANDLING =======
*/

// catch 404 and forward to error handler;
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler;
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page;
  res.status(err.status || 500);
  res.render('error');
});

/*
* ========= EXPORTS =======
*/

module.exports = app;
```

mqttUtils.js

```
/*
* ====================================
* frontend/uniboSupports/mqttUtils.js
* ====================================
```

```
*/
const mqtt = require ('mqtt'); //npm install --save mqtt
const topic = "unibo/qak/events";

var mqttAddr = 'mqtt://localhost'

var client = mqtt.connect(mqttAddr);
var io ;  //Upgrade for socketIo;
var robotModel = "none";
var roomMapModel = "none";

console.log("mqtt client= " + client );

exports.setIoSocket = function ( iosock ) {
  io = iosock;
  console.log("mqtt SETIOSOCKET io=" + io);
}


client.on('connect', function () {
  client.subscribe( topic );
  console.log('client has connected successfully with ' + mqttAddr);
});

//The message usually arrives as buffer, so it has to be had to converted into string data type;
client.on('message', function (topic, message){
  //console.log("mqtt io="+ io );
  //msg(modelContent,event,resourcemodel,none,content(robot(state(5))),74)
  console.log("mqtt RECEIVES:" + message.toString()); //if toString is not given, the message comes as buffer
  var msgStr = message.toString();
  if(msgStr.indexOf("content")<0) return;  //it is some other message sent via MQTT
  var spRobot = msgStr.indexOf("robot");
  var spRoomMap = msgStr.indexOf("roomMap");
  var spTable = msgStr.indexOf("table");
  var spPantry = msgStr.indexOf("pantry");
  var spDishwasher = msgStr.indexOf("dishwasher");
  var spWarning = msgStr.indexOf("warning");
  var spFridgeContent = msgStr.indexOf("fridgeContent");
  var spAnswer = msgStr.indexOf("answer");
  var sp1 = msgStr.indexOf("state");
  var msgStr = msgStr.substr(sp1);
  var sp2 = msgStr.indexOf("))");
  var msg = "";
  if (spWarning >= 0) {
    content = "The fridge does not contain the selected food.";
    msg = msg + "Warning: ";
    warning = msg + content;
  } else if (spAnswer >= 0) {
    msgStr = message.toString()
    sp1 = msgStr.indexOf("content");
    msgStr = msgStr.substr(sp1);
    var sp2 = msgStr.indexOf("))");
    var content = message.toString().substr(sp1, sp2 + 1);
    msg = msg + "Answer: "
    if (content == "content(yes)") {
      content = "The fridge contains the requested food.";
    } else {
      content = "The fridge does not contain the requested food.";
    }
    answer = msg + content;
  } else if (spFridgeContent >= 0) {
    var content = message.toString().substr(sp1, sp2 + 1);
    msg = msg + "Fridge exposing its content: ";
    fridgeModel = msg + content;
  } else {
    var content = message.toString().substr(sp1, sp2 + 1);
```

```
71      if (spRobot > 0) { msg = msg + "robotState:"; robotModel = msg + content; };
72      if (spRoomMap > 0) { msg = msg + "roomMap:"; roomMapModel = msg + content; };
73      if (spTable > 0) { msg = msg + "table:"; tableModel = msg + content; };
74      if (spPantry > 0) { msg = msg + "pantry:"; pantryModel = msg + content; };
75      if (spDishwasher > 0) { msg = msg + "dishwasher:"; dishwasher = msg + content; };
76    };
77    msg = msg + content;
78    console.log("mqtt send on io.sockets| " + msg + " content=" + content);
79    io.sockets.send(msg);
80  });
81
82  exports.publish = function( msg, topic ){
83    //console.log('mqtt publish ' + client);
84    client.publish(topic, msg);
85  }
```

7.6 Deployment

Instructions to use the frontend:

- From the terminal, get into the "it.unibo.frontend.tfbo19iss/nodeCode/frontend" folder.
- Execute npm install to download the requred dependencies.
- Execute startFrontEnd.bat to launch the node server.
- Open a browser on http://localhost:8080.

8 Sprint 3

8.1 Requirements

Design and build the software to put on board of the **Fridge** and of the **RBR**. In particular, the **RBR** must be able to accept the following commands sent by the smart-phone of the **Maitre**:

- *prepare*: the **RBR** must execute in autonomous way the *Prepare the room* task;
- *add food*: the **RBR** must execute in autonomous way the *Add food* task;
- *clear*: the **RBR** must execute in autonomous way the *Clear the room* task.

These tasks are normally executed in sequence, and the main scenario can be summarized as follows:

1. At start, the room is empty (i.e. no people is in it, besides the Maitre) while the **pantry** and the **Fridge** are filled with a proper set of items. The **RBR** is in its home (RH) location and the **dishwasher** is empty.

2. The **Maitre** sends to the **RBR** the *prepare* command and waits for the completion of the related task, consisting in putting on the **table** dishes taken from the **pantry**, and food taken from the **Fridge**. The set of items to put on the **table** in this phase is fixed and properly described somewhere. At the end, the **RBR** is in its RH location again.

3. The Maitre opens the room to people. During the service, the **Maitre** can send to the **RBR** the *add food* command, by specifying a food-code. The **RBR** executes the task, consisting in bringing the food with the given code from the **Fridge** to the **table**, only if the specified food is available in the **Fridge**, otherwise it sends a warning to the **Maitre**. After the task completion, the **RBR** returns is in its RH location.

4. At the end of the party, the **Maitre** sends to the **RBR** the **clear** command and waits for the completion of the task, consisting in bringing non-consumed food again in the **Fridge** and the dishes in the **dishwasher**. The **RBR** returns is in its RH location again.

However, the **Maitre** is able, at any time, to use his/her smart-phone to:

- *consult* the state of the room, e.g. to known what are the objects related to each resource; for example, the object currently posed on the **table**, in the **dishwasher**, etc;

- *stop* or *reactivate* an activated task.

Finally, the **RBR** must be able to

- *avoid* the impact with mobile obstacles (e.g. The Maitre or other humans/animals present in the room).

The software to put on the **Fridge** should make the device able to:

- *expose* its current content on the **Maitre** smart-phone;

- *answer* to questions about its content (e.g. if it contains food with a given code).

8.2 Problem analysis

8.2.1 Logical architecture

The logical architecture of a system defines its structure, interactions and behavior.

- **Structure**: the structure of the system defined in the Sprint 1 has not changed.

- **Interaction**: ne of the new requirements is that all of the interactions involving the **Fridge** must occur via CoAP. This protocol allows the entities to communicate without any knowledge of the implementation detail of the others.

- **RBR**: it must be able to communicate with both the **Maitre** and the **Fridge**. In particular:
  - *roombutlerrobot*: communicates via MQTT with *resourcemodel*, *planexecutor* and **Maitre**. It only receive messages from the last one;
  - *planexecutor*: communicates via MQTT with *roombutlerrobot* and *resourcemodel*;
  - *resourcemodel*: communicates via MQTT with *roombutlerrobot*, *planexecutor*, **Maitre** and via CoAP with **Fridge**. This actor now also takes care of sending all of the **RBR**'s messages to other entities.
- **Fridge**: it will communicate with both the **RBR** and the **Maitre** via CoAP.
- **Maitre**: it will communicate with the **RBR** via MQTT and the **Fridge** via CoAP.

- **Behavior**: using the QActor meta-model we will express the behavior of the entities as Finite State Machines (FSM).

roombutlerrobot

```
QActor roombutlerrobot context ctxRBR {
  ["
  var NextGoal = \"\"
  var GoalX = \"\"
  var GoalY = \"\"
  var CurObject = \"\"
  var actionDone = false
  var CurTask = \"\"
  "]

  State s0 initial {
    solve( consult("butlerRobotKb.pl") )
    solve( consult("sysRules.pl") )
  }
  Goto waitCmd

  State waitCmd {
    ["CurTask = \"\""]
    println("&&& RBR waitCmd ... ")
  }
  Transition t0
  whenMsg prepChange -> changePrepSet
  whenMsg prepare -> prepareTheRoomInit
  whenMsg addFood -> addFoodOnTableInit
  whenMsg clear  -> clearTheRoomInit

  State changePrepSet {
    onMsg(prepChange : prepChange(P)) {
      run itunibo.rbr.roomButlerRobotSupport.changePrepSet(myself, payloadArg(0))
    }
  }
  Goto waitCmd

  //PREPARE THE ROOM
  State prepareTheRoomInit {
    ["
```

```
37        CurTask = \"prepare\"
38        actionDone = false
39        NextGoal = \"pantry\"
40        GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
41        GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
42        "]
43        forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
44
45        //at least one item from the pantry must be in the preparation set
46        ["solve( \"preparation([H|T])\" )"]
47        ifSolved {
48          ["
49          solve( \"replaceRule(preparation([H|T]), preparation(T))\" )
50          CurObject = getCurSol(\"H\").toString()
51          "]
52        }
53      }
54      Goto waitPrepare
55
56      State waitPrepare { println("WAIT PREPARE") }
57      Transition t0
58      whenMsg goalOk -> prepareTheRoomContinue
59      whenEvent stop -> stopTask
60      whenEvent roomModelChanged -> prepareTheRoomContinue
61
62      State prepareTheRoomContinue {
63        printCurrentMessage
64        delay 1000
65
66        onMsg( goalOk : goalOk(pantry) ) {
67          forward resourcemodel -m modelUpdate : modelUpdate(pantry, remove($CurObject))
68        }
69        onMsg( goalOk : goalOk(fridge) ) {
70          forward resourcemodel -m modelUpdate : modelUpdate(fridge, remove($CurObject))
71        }
72        onMsg( goalOk : goalOk(table) ) {
73          forward resourcemodel -m modelUpdate : modelUpdate(table, add($CurObject))
74        }
75        onMsg( goalOk : goalOk(home) ) {
76          ["actionDone = true"]
77        }
78
79        onMsg( roomModelChanged : modelChanged(pantry, remove(_)) ) {
80          ["
81          NextGoal = \"table\"
82          GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
83          GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
84          "]
85          forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
86        }
87        onMsg( roomModelChanged : modelChanged(fridge, remove(_)) ) {
88          ["
89          NextGoal = \"table\"
90          GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
91          GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
92          "]
93          forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
94        }
95        onMsg( roomModelChanged : modelChanged(table, add(_)) ) {
96          ["solve( \"preparation([H|T])\" )"]
97          println(currentSolution)
98          ifSolved {
99            ["
100           solve( \"replaceRule(preparation([H|T]), preparation(T))\" )
101           CurObject = getCurSol(\"H\").toString()
102           if(CurObject.equals(\"dish\"))
103           NextGoal = \"pantry\"
104           else
```

```
105           NextGoal = \"fridge\"
106           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
107           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
108           "]
109         }
110         else { //preparation list empty
111           ["
112           CurObject = \"\"
113           NextGoal = \"home\"
114           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
115           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
116           "]
117         }
118         println("$NextGoal")
119         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
120       }
121     }
122     Goto waitCmd if "actionDone" else waitPrepare
123
124     //ADD FOOD ON THE TABLE
125     State addFoodOnTableInit {
126       ["
127       CurTask = \"addFood\"
128       actionDone = false
129       "]
130       onMsg(addFood : addFood(F)) {
131         //check if the fridge contains F
132         forward resourcemodel -m request : request($payloadArg(0))
133         ["CurObject = \"${payloadArg(0)}\" "]
134       }
135     }
136     Transition t0
137     whenMsg answer -> checkFridgeAnswer
138
139     State checkFridgeAnswer {
140       printCurrentMessage
141       onMsg(answer : answer(yes)) {
142         ["
143         NextGoal = \"fridge\"
144         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
145         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
146         "]
147         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
148       }
149       onMsg(answer : answer(no)) {
150         ["CurObject = \"\"
151         NextGoal = \"\"
152         GoalX = \"\"
153         GoalY = \"\"
154         actionDone = true
155         "]
156         forward resourcemodel -m warning : warning
157       }
158     }
159     Goto waitCmd if "actionDone" else waitAddFood
160
161     State waitAddFood { println("WAIT ADD FOOD") }
162     Transition t0
163     whenMsg goalOk -> addFoodOnTheTableContinue
164     whenEvent stop -> stopTask
165     whenEvent roomModelChanged -> addFoodOnTheTableContinue
166
167     State addFoodOnTheTableContinue {
168       delay 1000
169
170       onMsg( goalOk : goalOk(fridge) ) {
171         forward resourcemodel -m modelUpdate : modelUpdate(fridge, remove($CurObject))
172       }
```

```
173       onMsg( goalOk : goalOk(table) ) {
174         forward resourcemodel -m modelUpdate : modelUpdate(table, add($CurObject))
175       }
176       onMsg( goalOk : goalOk(home) ) {
177         ["actionDone = true"]
178       }
179
180       onMsg( roomModelChanged : modelChanged(fridge, remove(_)) ) {
181         ["
182         NextGoal = \"table\"
183         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
184         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
185         "]
186         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
187       }
188       onMsg( roomModelChanged : modelChanged(table, add(_)) ) {
189         ["
190         CurObject = \"\"
191         NextGoal = \"home\"
192         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
193         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
194         "]
195         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
196       }
197     }
198     Goto waitCmd if "actionDone" else waitAddFood
199
200     //CLEAR THE ROOM
201     State clearTheRoomInit {
202       ["
203       CurTask = \"clear\"
204       actionDone = false
205       "]
206       forward resourcemodel -m modelConsult : modelConsult(table)
207     }
208     Transition t0
209     whenEvent modelState -> checkTableState
210
211     State checkTableState {
212       printCurrentMessage
213       onMsg(modelState : modelState(S) ) {
214         ["solve( \"table(_)\" )"]
215         ifSolved {
216           ["solve( \"replaceRule(table(_), table(${payloadArg(0)}))\" )"]
217         } else {
218           ["solve( \"addRule(table(${payloadArg(0)}))\" )"]
219         }
220         println(currentSolution)
221       }
222       ["
223       NextGoal = \"table\"
224       GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
225       GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
226       "]
227       forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
228     }
229     Goto waitClear
230
231     State waitClear { }
232     Transition t0
233     whenMsg goalOk -> clearTheRoomContinue
234     whenEvent stop -> stopTask
235     whenEvent roomModelChanged -> clearTheRoomContinue
236
237     State clearTheRoomContinue {
238       printCurrentMessage
239       delay 1000
240       onMsg( goalOk : goalOk(fridge) ) {
```

```
241         forward resourcemodel -m modelUpdate : modelUpdate(fridge, add($CurObject))
242       }
243       onMsg( goalOk : goalOk(dishwasher) ) {
244         forward resourcemodel -m modelUpdate : modelUpdate(dishwasher, add($CurObject))
245       }
246       onMsg( goalOk : goalOk(table) ) {
247         ["solve( \"table([H|T])\" )"]
248         println(currentSolution)
249         ifSolved {
250           ["
251           solve( \"replaceRule(table([H|T]), table(T))\" )
252           CurObject = getCurSol(\"H\").toString()
253           if(CurObject.equals(\"dish\"))
254           NextGoal = \"dishwasher\"
255           else
256           NextGoal = \"fridge\"
257           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
258           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
259           "]
260           forward resourcemodel -m modelUpdate : modelUpdate(table, remove($CurObject))
261         }
262         else { //no more objects on the table
263           ["
264           CurObject = \"\"
265           NextGoal = \"home\"
266           GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
267           GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
268           "]
269           forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
270         }
271         //println("$NextGoal")
272       }
273       onMsg( goalOk : goalOk(home) ) {
274         ["actionDone = true"]
275       }
276
277       onMsg( roomModelChanged : modelChanged(dishwasher, add(_)) ) {
278         ["
279         NextGoal = \"table\"
280         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
281         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
282         "]
283         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
284       }
285       onMsg( roomModelChanged : modelChanged(fridge, add(_)) ) {
286         ["
287         NextGoal = \"table\"
288         GoalX = itunibo.rbr.roomButlerRobotSupport.getGoalCoordX(myself, NextGoal)
289         GoalY = itunibo.rbr.roomButlerRobotSupport.getGoalCoordY(myself, NextGoal)
290         "]
291         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
292       }
293       onMsg( roomModelChanged : modelChanged(table, remove(_)) ) {
294         forward planexecutor -m goalUpdate : goalUpdate($NextGoal, $GoalX, $GoalY)
295       }
296     }
297     Goto waitCmd if "actionDone" else waitClear
298
299     State stopTask {
300       println("RBR STOPPED")
301       emit local_stop : stop
302     }
303     Goto waitReactivation
304
305     State waitReactivation { }
306     Transition t0
307     whenMsg resume -> resumeTask
308
```

```
    State resumeTask {
      emit local_resume : resume($CurTask)
    }
    Goto waitTaskResumed

    State waitTaskResumed { println("WAIT TASK RESUMED") }
    Transition t0
    whenEvent local_prepareResumed -> waitPrepare
    whenEvent local_addFoodResumed -> waitAddFood
    whenEvent local_clearResumed -> waitClear
  }
```

fridge

```
QActor fridge context ctxFridge{
  State s0 initial {
    solve( consult("sysRules.pl") )
    solve( consult("fridgeModel.pl") )
    run itunibo.coap.fridgeResourceCoap.create( myself, "fridge" )
  }
  Goto waitCmd

  State waitCmd { }
  Transition t0
  whenMsg request -> handleRequest
  whenMsg expose -> exposeModel
  whenMsg modelUpdate -> updateModel

  State handleRequest {
    onMsg(request : request(S, F)) {
      run itunibo.fridge.fridgeModelSupport.answerRequest(myself, payloadArg(0), payloadArg(1))
    }
  }
  Goto waitCmd

  State exposeModel {
    run itunibo.fridge.fridgeModelSupport.exposeFridgeModel(myself)
  }
  Goto waitCmd

  State updateModel {
    onMsg(modelUpdate : modelUpdate(fridge, V)) {
      run itunibo.fridge.fridgeModelSupport.updateFridgeModel(myself, payloadArg(1))
    }
  }
  Goto waitCmd
}
```

resourcemodel

```
QActor resourcemodel context ctxRobotMind{

  State s0 initial {
    solve( consult("sysRules.pl") )
    solve( consult("resourceModel.pl") )
    solve( showResourceModel )
    run itunibo.coap.client.coapClientResModel.createClient(myself, "localhost", 5683, "fridge")
  }
  Goto waitMsg

  State waitMsg{ }
  Transition t0
  whenMsg modelChange -> changeModel
```

```
14      whenMsg modelUpdate -> updateModel
15      whenMsg modelConsult -> consultModel
16      whenMsg modelExpose -> exposeModel
17      whenMsg request  -> sendRequest
18      whenMsg warning  -> emitWarning
19
20      State updateModel{
21        printCurrentMessage
22        onMsg( modelUpdate : modelUpdate(robot, V ) ) {
23          run itunibo.resModel.resourceModelSupport.updateRobotModel( myself, payloadArg(1) )
24        }
25        onMsg( modelUpdate : modelUpdate(sonarRobot,V ) ) {
26          run itunibo.resModel.resourceModelSupport.updateSonarRobotModel( myself, payloadArg(1)
                ↪ )
27        }
28        onMsg( modelUpdate : modelUpdate(pantry, V) ) {
29          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
30        }
31        onMsg( modelUpdate : modelUpdate(table, V) ) {
32          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
33        }
34        onMsg( modelUpdate : modelUpdate(dishwasher, V) ) {
35          run itunibo.resModel.resourceModelSupport.updateRoomResourceModel( myself, payloadArg
                ↪ (0), payloadArg(1) )
36        }
37        onMsg( modelUpdate : modelUpdate(fridge, V) ) {
38          run itunibo.coap.client.coapClientResModel.put(payloadArg(1))
39        }
40        onMsg( modelUpdate : modelUpdate(roomMap,V ) ) {
41          run itunibo.resModel.resourceModelSupport.updateRoomMapModel( myself, payloadArg(1) )
42        }
43      }
44      Goto waitMsg
45
46      //ONLY FOR THE ROBOT PART OF THE MODEL
47      State changeModel{
48        //ROBOT MOVE
49        onMsg( modelChange : modelChange( robot,V ) ) { // V= w | ...
50          run itunibo.resModel.resourceModelSupport.updateRobotModel( myself, payloadArg(1) )
51          emit local_robotModelChanged : modelChanged( robot, $payloadArg(1)) //for the
                ↪ robotmind
52        }
53      }
54      Goto waitMsg
55
56      State consultModel {
57        onMsg( modelConsult : modelConsult(_) ) {
58          run itunibo.resModel.resourceModelSupport.consultRoomResourceModel(myself, payloadArg
                ↪ (0))
59        }
60      }
61      Goto waitMsg
62
63      State exposeModel {
64        onMsg( modelExpose : modelExpose ) {
65          run itunibo.resModel.resourceModelSupport.getRobotModel( myself )
66          run itunibo.resModel.resourceModelSupport.getRoomMapModel(myself)
67          run itunibo.resModel.resourceModelSupport.exposeRoomResourceModel( myself, "table" )
68          run itunibo.resModel.resourceModelSupport.exposeRoomResourceModel( myself, "pantry" )
69          run itunibo.resModel.resourceModelSupport.exposeRoomResourceModel( myself, "
                ↪ dishwasher" )
70        }
71      }
72      Goto waitMsg
73
74      State emitWarning {
```

```
        onMsg(warning : warning) {
          run itunibo.resModel.resourceModelSupport.sendWarning(myself)
        }
      }
      Goto waitMsg

      State sendRequest {
        onMsg(request : request(F)) {
          run itunibo.coap.client.coapClientResModel.synchGet(payloadArg(0))
        }
      }
      Goto waitMsg
    }
```

8.3 Product backlog

- Implement tasks and functionalities for the **RBR**:
  - stop (1 hour);
  - reactivate (1 hour);
  - avoid obstacles (2 hours).
- Implement **Fridge** communications with CoAP:
  - create CoAP resource for **Fridge** (1 hour);
  - create CoAP client for **RBR** (1 hour);
  - modify frontend to use CoAP (2 hour).
- Create formal TestPlans with JUnit (2 hours).

8.4 TestPlans

TestRBR

```
class TestRBR {
  var rbr : ActorBasic? = null
  var planexec : ActorBasic? = null
  var mqttClient : MqttClient? = null
  var broker : String = "tcp://localhost"
  var rbrTopic : String = "unibo/qak/roombutlerrobot"
  var eventTopic : String = "unibo/qak/events"
  var clientId : String = "sprint_3"
  var qos : Int = 2;

  @Before
  fun systemSetUp() {

    try {
      println("%%%%%%%%%%%%%% Sprint 3 TestRBR starting Mqtt Client")
      mqttClient = MqttClient(broker, clientId)
      var connOpts = MqttConnectOptions()
      connOpts.setCleanSession(true)
      mqttClient!!.connect(connOpts)
    } catch (e : MqttException) {
      println("MQTT EXCEPTION IN SETUP")
    }
    GlobalScope.launch {
      it.unibo.ctxRBR.main()
```

```
25      }
26      delay(5000) //give the time to start
27      rbr = sysUtil.getActor("roombutlerrobot")
28      //planexec = sysUtil.getActor("plaexecutor")
29    }
30
31    @After
32    fun terminate() {
33      println("%%%%%%% Sprint 3 TestRBR terminate ")
34      try {
35        mqttClient!!.disconnect()
36        mqttClient!!.close()
37      } catch (e : MqttException) {
38        println("MQTT EXCEPTION IN TERMINATE")
39      }
40    }
41
42    @Test
43    fun sprint3Test() {
44      println("%%%%%%% Sprint 3 Functional TestRBR starts ")
45      stop()
46      reactivate()
47      avoidObstacle()
48    }
49
50    fun stop() {
51      println("%%%%%%% Sprint 3 TestRBR stop command")
52      sendCmd("prepare", "")
53      delay(5000)
54      emitEvent("stop", "")
55      delay(5000)
56      solveCheckGoal(rbr!!, 0, 0, false)
57    }
58
59    fun reactivate() {
60      println("%%%%%%% Sprint 3 TestRBR reactivate command")
61      emitEvent("resume", "")
62      delay(40000)
63      solveCheckGoal(rbr!!, 0, 0, true)
64    }
65
66    fun avoidObstacle() {
67      println("%%%%%%% Sprint 3 TestRBR reactivate command")
68      sendCmd("addFood", "beef")
69      delay(1500)
70      emitEvent("obstacle", "5")
71      delay(45000)
72      solveCheckGoal(rbr!!, 0, 0, true)
73    }
74
75    //-------------------------------------------------------------------------
76
77    fun sendCmd(cmd : String, content : String) {
78      println("--- RBR performing performing task $cmd")
79      var msg : String
80      if (content != "") {
81        msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd($content),1)"
82      } else {
83        msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd,1)"
84      }
85      try {
86        var mqttMsg = MqttMessage(msg.toByteArray())
87        mqttClient!!.publish(rbrTopic, mqttMsg)
88      } catch (e : MqttException) {
89        println("MQTT EXCEPTION IN DOTASK")
90      }
91    }
92
```

```
93    fun emitEvent(cmd : String, content : String) {
94      println("--- RBR performing performing cmd $cmd")
95      var msg : String
96      if (content != "") {
97        msg = "msg($cmd,event,js,none,$cmd($content),1)"
98      } else {
99        msg = "msg($cmd,event,js,none,$cmd,1)"
100     }
101     try {
102       var mqttMsg = MqttMessage(msg.toByteArray())
103       mqttClient!!.publish(eventTopic, mqttMsg)
104     } catch (e : MqttException) {
105       println("MQTT EXCEPTION IN DOTASK")
106     }
107   }
108
109   fun solveCheckGoal( actor : ActorBasic, x : Int, y : Int, goal : Boolean ) {
110     if(goal) {
111       var result = itunibo.planner.moveUtils.getPosX(actor) == x && itunibo.planner.moveUtils.getPosY(
              ↪ actor) == y
112       println(" %%%%%%% actor={$actor.name} goal = RBR in ($x,$y) result = $result")
113       assertTrue(result)
114
115     } else {
116       var result = itunibo.planner.moveUtils.getPosX(actor) != x && itunibo.planner.moveUtils.getPosY(
              ↪ actor) != y
117       println(" %%%%%%% actor={$actor.name} goal = RBR not in ($x,$y) result = $result")
118       assertTrue(result)
119     }
120   }
121
122   fun delay( time : Long ){
123     Thread.sleep( time )
124   }
125 }
```

TestResources

```
1  class TestResources {
2    var resource : ActorBasic? = null
3    var mqttClient : MqttClient? = null
4    var broker : String = "tcp://localhost"
5    var rbrTopic : String = "unibo/qak/roombutlerrobot"
6    var eventTopic : String = "unibo/qak/events"
7    var clientId : String = "sprint_3"
8    var qos : Int = 2;
9
10   @Before
11   fun systemSetUp() {
12
13     try {
14       println("%%%%%%%%%%%%%% Sprint 3 TestResources starting Mqtt Client")
15       mqttClient = MqttClient(broker, clientId)
16       var connOpts = MqttConnectOptions()
17       connOpts.setCleanSession(true)
18       mqttClient!!.connect(connOpts)
19     }
20     catch (e : MqttException) {
21       println("MQTT EXCEPTION IN SETUP")
22     }
23     GlobalScope.launch {
24       it.unibo.ctxRobotMind.main()
25     }
26     delay(5000) //give the time to start
27     resource = sysUtil.getActor("resourcemodel")
28   }
```

```
29
30  @After
31  fun terminate() {
32    println("%%%%%%% Sprint 3 TestResources terminate ")
33    try {
34      mqttClient!!.disconnect()
35      mqttClient!!.close()
36    } catch (e : MqttException) {
37      println("MQTT EXCEPTION IN TERMINATE")
38    }
39  }
40
41  @Test
42  fun sprint1Test() {
43    println("%%%%%%% Sprint 3 Functional TestResources starts ")
44    stop()
45    reactivate()
46  }
47
48  fun stop() {
49    println("%%%%%%% Sprint 3 TestResources stop command")
50    sendCmd("prepare", "")
51    delay(5000)
52    emitEvent("stop")
53    delay(5000)
54    solveCheckGoal(resource!!, "model( actuator, robot, state( stopped ) )")
55    solveCheckGoal(resource!!, "model( resource, table, state([ ]) )")
56  }
57
58  fun reactivate() {
59    println("%%%%%%% Sprint 3 TestResources reactivate command")
60    emitEvent("resume")
61    delay(40000)
62    solveCheckGoal(resource!!, "model( actuator, robot, state( stopped ) )")
63    solveCheckGoal(resource!!, "model( resource, pantry, state([ dish, dish, dish, dish, dish,
          ↪ dish, dish, dish, dish, dish, dish, dish, dish, dish ]) )")
64    solveCheckGoal(resource!!, "model( resource, table, state([ fruit, dish ]) )")
65    solveCheckGoal(resource!!, "model( resource, dishwasher, state([ ]) )")
66  }
67
68  //------------------------------------------------------------------------
69
70  fun sendCmd(cmd : String, content : String) {
71    println("--- RBR performing performing task $cmd")
72    var msg : String
73    if (content != "") {
74      msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd($content),1)"
75    } else {
76      msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd,1)"
77    }
78    try {
79      var mqttMsg = MqttMessage(msg.toByteArray())
80      mqttClient!!.publish(rbrTopic, mqttMsg)
81    } catch (e : MqttException) {
82      println("MQTT EXCEPTION IN DOTASK")
83    }
84    // if(content == "fruit") {
85      //  delay(5000)
86      // } else {
87      //  delay(45000) //wait 45 seconds to check the results
88      // }
89  }
90
91  fun emitEvent(cmd : String) {
92    println("--- RBR performing performing cmd $cmd")
93    var msg : String
94    msg = "msg($cmd,event,js,none,$cmd,1)"
95
```

```
96      try {
97        var mqttMsg = MqttMessage(msg.toByteArray())
98        mqttClient!!.publish(eventTopic, mqttMsg)
99      } catch (e : MqttException) {
100       println("MQTT EXCEPTION IN DOTASK")
101     }
102     // if(cmd == "stop") {
103       //  delay(5000)
104       // } else {
105       //  delay(45000) //wait 45 seconds to check the results
106       // }
107   }
108
109   fun solveCheckGoal( actor : ActorBasic, goal : String ){
110     actor.solve( goal )
111     var result = actor.resVar
112     println(" %%%%%%% actor={$actor.name} goal= $goal result = $result")
113     assertTrue("", result == "success" )
114   }
115
116   fun delay( time : Long ){
117     Thread.sleep( time )
118   }
119 }
```

The introduction of CoAP provides us a new method to develop the Fridge test plan: being a CoAP resource the Fridge now has an internal state, which can be used instead of consulting the prolog knowledge base to check if the tasks have worked properly.

TestFridge

```
1  class TestFridge {
2    var fridge : ActorBasic? = null
3    var mqttClient : MqttClient? = null
4    var broker : String = "tcp://localhost"
5    var rbrTopic : String = "unibo/qak/roombutlerrobot"
6    var eventTopic : String = "unibo/qak/events"
7    var clientId : String = "sprint_3"
8    var qos : Int = 2;
9
10   @Before
11   fun systemSetUp() {
12
13     try {
14       println("%%%%%%%%%%%%%% Sprint 3 TestFridge starting Mqtt Client")
15       mqttClient = MqttClient(broker, clientId)
16       var connOpts = MqttConnectOptions()
17       connOpts.setCleanSession(true)
18       mqttClient!!.connect(connOpts)
19     }
20     catch (e : MqttException) {
21       println("MQTT EXCEPTION IN SETUP")
22     }
23     GlobalScope.launch {
24       it.unibo.ctxFridge.main()
25     }
26     delay(5000) //give the time to start
27     fridge = sysUtil.getActor("fridge")
28   }
29
30   @After
31   fun terminate() {
32     println("%%%%%%% Sprint 3 TestFridge terminate ")
```

```
33      try {
34        mqttClient!!.disconnect()
35        mqttClient!!.close()
36      } catch (e : MqttException) {
37        println("MQTT EXCEPTION IN TERMINATE")
38      }
39    }
40
41    @Test
42    fun sprint3Test() {
43      println("%%%%%%% Sprint 3 Functional TestFridge starts ")
44      prepare()
45      addFoodFail("fruit")
46      addFoodOk("beef")
47      clear()
48    }
49
50    fun prepare() {
51      println("%%%%%%% Sprint 3 TestFridge prepareRoom Task")
52      sendCmd("prepare", "")
53      solveCheckGoal(fridge!!, "fridge(state([beef]))")
54    }
55
56    fun addFoodFail(foodCode : String) {
57      println("%%%%%%% Sprint 3 TestFridge addFood Task")
58      sendCmd("addFood", foodCode)
59      solveCheckGoal(fridge!!, "fridge(state([beef]))")
60      solveCheckGoal(fridge!!, "no")
61    }
62
63    fun addFoodOk(foodCode : String) {
64      println("%%%%%%% Sprint 3 Test addFood Task")
65      sendCmd("addFood", foodCode)
66      solveCheckGoal(fridge!!, "yes")
67      solveCheckGoal(fridge!!, "fridge(state([]))")
68    }
69
70    fun clear() {
71      println("%%%%%%% Sprint 3 TestFridge clearRoom Task")
72      sendCmd("clear", "")
73      solveCheckGoal(fridge!!, "fridge(state([fruit,beef]))")
74    }
75
76    //-----------------------------------------------------------------------------
77
78    fun sendCmd(cmd : String, content : String) {
79      println("--- RBR performing performing task $cmd")
80      var msg : String
81      if (content != "") {
82        msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd($content),1)"
83      } else {
84        msg = "msg($cmd,dispatch,js,roombutlerrobot,$cmd,1)"
85      }
86      try {
87        var mqttMsg = MqttMessage(msg.toByteArray())
88        mqttClient!!.publish(rbrTopic, mqttMsg)
89      }
90      catch (e : MqttException) {
91        println("MQTT EXCEPTION IN DOTASK")
92      }
93      if(content == "fruit") {
94        delay(5000)
95      }
96      else {
97        delay(45000) //wait 45 seconds to check the results
98      }
99    }
100
```

```
101     fun solveCheckGoal( actor: ActorBasic, goal : String ){
102       var result : Boolean
103       if(goal.equals("yes") || goal.equals("no")) {
104         result = (goal == itunibo.coap.fridgeResourceCoap.getAnswer())
105       } else {
106         println(itunibo.coap.fridgeResourceCoap.getModel())
107         result = (goal == itunibo.coap.fridgeResourceCoap.getModel())
108       }
109       println(" %%%%%% actor={$actor.name} goal= $goal result = $result")
110       assertTrue(result)
111     }
112
113     fun delay( time : Long ){
114       Thread.sleep( time )
115     }
116   }
```

8.5 Project

8.5.1 RBR tasks and functionalities

Stop and Reactivate

When the *stop* command is received by the *roombutlerrobot* it is propagated to the *planexecutor*, which finishes to execute the action it is performing and then stops the robot. Both the actors move to a state called *waitReactivation* until the *reactivate* command is received by the *roombutlerrobot* and propagated to the *planexecutor*. As said in the requirement analysis, if the two commands are received while the robot is not performing any task they must be ignored. This implies the use of events, which are not buffered.

Avoid obstacles

When the robot detects an obstacle, it moves back for the same amount of time it moved forward before the detection to return inside the map grid. The tile the robot was attempting to move into is labeled as a "temporary obstacle" (temporary obstacles list is cleared every time a goal is reached). Then, all the actions of the planner are deleted and a new plan is computed. If the plan contains no actions because either the robot is trapped between temporary obstacles or one of those is right on the goal tile, the RBR stops for 2 seconds, it clears the temporary obstacles list and tries to compute a new plan.

planexecutor

```
1   QActor planexecutor context ctxRBR {
2     ["
3     var mapEmpty = true
4     val mapname = \"roommap\"
5     //var Tback = 100
6
7     var emptyPlan = false
8
9     var Curmove = \"\"
10    var curmoveIsForward = false
11    var Stopped = false
12
13    var CurGoal = \"\"
14
```

```
15  //REAL ROBOT
16  //var StepTime = 1000
17  //var PauseTime = 500L
18
19  //VIRTUAL ROBOT
20  var StepTime = 330
21  var PauseTime = 400 //increased because it wouldn't do two rotations in a row
22
23  var PauseTimeL = PauseTime.toLong()
24  "]
25
26  State s0 initial {
27    solve ( consult("moves.pl") )
28    run itunibo.planner.plannerUtil.initAI()
29    run itunibo.planner.moveUtils.loadRoomMap(myself, mapname)
30    run itunibo.planner.moveUtils.showCurrentRobotState()
31    ["val MapStr = itunibo.planner.plannerUtil.getMapOneLine()"]
32    forward resourcemodel -m modelUpdate : modelUpdate(roomMap, $MapStr)
33  }
34  Goto waitCmd
35
36  State waitCmd { }
37  Transition t0
38  whenMsg goalUpdate -> createPlan
39
40  State createPlan {
41    //printCurrentMessage
42    onMsg( goalUpdate : goalUpdate(G, X, Y) ) {
43      ["CurGoal = payloadArg(0)"]
44      run itunibo.planner.plannerUtil.setGoal(payloadArg(1), payloadArg(2))
45    }
46    run itunibo.planner.moveUtils.doPlan(myself)
47    solve( move(M) )
48    ifSolved {
49      ["emptyPlan = false"]
50    } else {
51      ["emptyPlan = true"]
52    }
53  }
54  Goto executePlannedActions if "! emptyPlan" else handleEmptyPlan
55
56  State executePlannedActions {
57    solve( retract(move(M)) )
58    ifSolved {
59      ["
60      Curmove = getCurSol(\"M\").toString()
61      curmoveIsForward = (Curmove == \"w\")
62      "]
63    } else {
64      ["
65      Curmove = \"\"
66      curmoveIsForward = false
67      "]
68    }
69    println("executePlannedActions doing $Curmove")
70  }
71  Goto cheakAndDoAction if "(Curmove.length > 0)" else goalOk
72
73  State goalOk {
74    run itunibo.planner.plannerUtil.clearTempObstacles()
75    ["val MapStr = itunibo.planner.plannerUtil.getMapOneLine()"]
76    forward resourcemodel -m modelUpdate : modelUpdate(roomMap, $MapStr)
77    forward roombutlerrobot -m goalOk : goalOk($CurGoal)
78  }
79  Goto waitCmd
80
81  //Execute the move if it is a rotation or halt
82  State cheakAndDoAction { }
```

```
83  Goto doForwardMove if "curmoveIsForward" else doTheMove
84
85  State doTheMove {
86    // println("ROTATION")
87    run itunibo.planner.moveUtils.rotate(myself, Curmove, PauseTime)
88  }
89  Goto executePlannedActions
90
91  State doForwardMove {
92    //println("FORWARD")
93    delayVar PauseTimeL //Otherwise is too fast, even with remote interaction
94    run itunibo.planner.moveUtils.attemptTomoveAhead(myself, StepTime)
95  }
96  Transition t0
97  whenMsg stepOk  -> handleStepOk
98  whenMsg stepFail -> handleStepFail
99  whenEvent local_stop -> handleStop
100
101 State handleStepOk {
102   run itunibo.planner.moveUtils.updateMapAfterAheadOk(myself)
103   ["val MapStr = itunibo.planner.plannerUtil.getMapOneLine()"]
104   forward resourcemodel -m modelUpdate : modelUpdate(roomMap, $MapStr)
105   run itunibo.planner.moveUtils.showCurrentRobotState()
106 }
107 Goto waitReactivation if "Stopped" else executePlannedActions
108
109 State handleStepFail {
110   ["
111   var Direction = itunibo.planner.plannerUtil.getDirection()
112   var ObsPosX = itunibo.planner.plannerUtil.getPosX()
113   var ObsPosY = itunibo.planner.plannerUtil.getPosY()
114   when( Direction ){
115     \"upDir\"  -> ObsPosY -= 1
116     \"rightDir\" -> ObsPosX += 1
117     \"downDir\" -> ObsPosY += 1
118     \"leftDir\" -> ObsPosX -= 1
119   }
120   "]
121   println("($ObsPosX, $ObsPosY)")
122   onMsg(stepFail : stepFail(O, D)) {
123     ["var BackStepTime = Integer.parseInt(payloadArg(1))"]
124     run itunibo.planner.moveUtils.backToCompensate(myself, BackStepTime)
125   }
126   run itunibo.planner.plannerUtil.addTempObstacle(ObsPosX, ObsPosY)
127   ["val MapStr = itunibo.planner.plannerUtil.getMapOneLine()"]
128   forward resourcemodel -m modelUpdate : modelUpdate(roomMap, $MapStr)
129 }
130 Goto waitReactivation if "Stopped" else createPlan
131
132 State handleEmptyPlan {
133   //Impossible to reach the goal because the rbr is "trapped" in between obstacles or the
        ↪ goal itself is obstructed
134   //Wait 2 seconds and check if the obstacles have moved
135   println("EMPTY PLAN: DELAY 2 SECONDS AND TRY AGAIN")
136   delay 2000
137   run itunibo.planner.plannerUtil.clearTempObstacles()
138   ["val MapStr = itunibo.planner.plannerUtil.getMapOneLine()"]
139   forward resourcemodel -m modelUpdate : modelUpdate(roomMap, $MapStr)
140 }
141 Goto waitReactivation if "Stopped" else createPlan
142
143 State handleStop {
144   ["Stopped = true"]
145   run itunibo.planner.plannerUtil.setStopped(true)
146 }
147 Goto waitReactivation
148
149 State waitReactivation {
```

```
150       println("PLANEXECUTOR: WAIT REACTIVATION")
151     }
152     Transition t0
153     whenMsg stepOk  -> handleStepOk
154     whenMsg stepFail -> handleStepFail
155     whenEvent local_resume -> handleResume
156
157     State handleResume {
158       ["Stopped = false"]
159       printCurrentMessage
160       run itunibo.planner.plannerUtil.setStopped(false)
161       onMsg (local_resume : resume(prepare)) {
162         emit local_prepareResumed : prepareResumed
163       }
164       onMsg (local_resume : resume(addFood)) {
165         emit local_addFoodResumed : addFoodResumed
166       }
167       onMsg (local_resume : resume(clear)) {
168         emit local_clearResumed : clearResumed
169       }
170     }
171     Goto executePlannedActions
172 }
```

plannerUtils.kt

```
1  package itunibo.planner
2
3  import java.util.ArrayList
4  import aima.core.agent.Action
5  import aima.core.search.framework.SearchAgent
6  import aima.core.search.framework.problem.GoalTest
7  import aima.core.search.framework.problem.Problem
8  import aima.core.search.framework.qsearch.GraphSearch
9  import aima.core.search.uninformed.BreadthFirstSearch
10 import java.io.PrintWriter
11 import java.io.FileWriter
12 import java.io.ObjectOutputStream
13 import java.io.FileOutputStream
14 import java.io.ObjectInputStream
15 import java.io.FileInputStream
16 import itunibo.planner.model.RobotState
17 import itunibo.planner.model.Functions
18 import itunibo.planner.model.RobotState.Direction
19 import itunibo.planner.model.RobotAction
20 import itunibo.planner.model.RoomMap
21 import itunibo.planner.model.Box
22
23 object plannerUtil {
24   private var initialState: RobotState? = null
25   private var actions: List<Action>? = null
26   private var tempObstacles = ArrayList<Pair<Int,Int>>()
27   /*
28   * ------------------------------------------------
29   * PLANNING
30   * ------------------------------------------------
31   */
32   private var search: BreadthFirstSearch? = null
33   var goalTest: GoalTest = Functions()  //init
34   private var timeStart: Long = 0
35   private var stopped = false
36
37   @Throws(Exception::class)
38   fun initAI() {
39     println("plannerUtil initAI")
40     initialState = RobotState(0, 0, RobotState.Direction.DOWN)
```

```
41    search = BreadthFirstSearch(GraphSearch())
42  }
43
44  fun resetRobotPos(x: Int, y:Int, oldx: Int, oldy: Int, direction: String ){
45    //println("plannerUtil resetRobotPos direction=$direction")
46    RoomMap.getRoomMap().put(oldx,oldy, Box(false, false, false))
47    RoomMap.getRoomMap().put(x,y, Box(false, false, true) )
48
49    var dir = RobotState.Direction.DOWN //init
50    when( direction ){
51      "down" -> dir = RobotState.Direction.DOWN
52      "up" -> dir = RobotState.Direction.UP
53      "left" -> dir = RobotState.Direction.LEFT
54      "right" -> dir = RobotState.Direction.RIGHT
55    }
56    initialState = RobotState(x,y, dir)
57    var canMove = RoomMap.getRoomMap().canMove( x,y, initialState!!.direction );
58    println("resetRobotPos $x,$y from: ${oldy},${oldy} direction=${getDirection()} canMove=
          ↪ $canMove")
59  }
60
61  var currentGoalApplicable = true;
62
63  fun getActions() : List<Action>{
64    return actions!!
65  }
66
67  @Throws(Exception::class)
68  fun doPlan(): List<Action>? {
69
70    if( ! currentGoalApplicable ){
71      println("plannerUtil doPlan cannot go into an obstacle")
72      return null
73    }
74
75    val searchAgent: SearchAgent
76    //println("plannerUtil doPlan newProblem (A) $goalTest" );
77    val problem = Problem(initialState, Functions(), Functions(), goalTest, Functions())
78
79
80    //println("plannerUtil doPlan newProblem (A) search " );
81    searchAgent = SearchAgent(problem, search!!)
82    actions = searchAgent.actions
83
84    println("plannerUtil doPlan actions=$actions")
85
86    if (actions == null || actions!!.isEmpty()) {
87      println("plannerUtil doPlan NO MOVES !!!!!!!!!!! $actions!!" )
88      if (!RoomMap.getRoomMap().isClean) RoomMap.getRoomMap().setObstacles()
89      //actions = ArrayList()
90      return null
91    } else if (actions!![0].isNoOp) {
92      println("plannerUtil doPlan NoOp")
93      return null
94    }
95
96    //println("plannerUtil doPlan actions=$actions")
97    return actions
98  }
99
100 fun executeMoves( ) {
101   if( actions == null ) return
102   val iter = actions!!.iterator()
103   while (iter.hasNext() && !stopped) {
104     plannerUtil.doMove(iter.next().toString())
105   }
106 }
107
```

```
/*
* ------------------------------------------------
* MAP UPDATE
* ------------------------------------------------
*/

fun getPosX() : Int{ return initialState!!.x }
fun getPosY() : Int{ return initialState!!.y }

fun doMove(move: String) {
  val dir = initialState!!.direction
  val dimMapx = RoomMap.getRoomMap().dimX
  val dimMapy = RoomMap.getRoomMap().dimY
  val x = initialState!!.x
  val y = initialState!!.y
  // println("plannerUtil: doMove move=$move dir=$dir x=$x y=$y dimMapX=$dimMapx dimMapY=
      ↪ $dimMapy")
  try {
    when (move) {
      "w" -> {
        RoomMap.getRoomMap().put(x, y, Box(false, false, false)) //clean the cell
        initialState = Functions().result(initialState!!, RobotAction(RobotAction.FORWARD)) as
            ↪ RobotState
        RoomMap.getRoomMap().put(initialState!!.x, initialState!!.y, Box(false, false, true))
      }
      "s" -> {
        initialState = Functions().result(initialState!!, RobotAction(RobotAction.BACKWARD)) as
            ↪ RobotState
        RoomMap.getRoomMap().put(initialState!!.x, initialState!!.y, Box(false, false, true))
      }
      "a" -> {
        initialState = Functions().result(initialState!!, RobotAction(RobotAction.TURNLEFT)) as
            ↪ RobotState
        RoomMap.getRoomMap().put(initialState!!.x, initialState!!.y, Box(false, false, true))
      }
      "l" -> {
        initialState = Functions().result(initialState!!, RobotAction(RobotAction.TURNLEFT)) as
            ↪ RobotState
        RoomMap.getRoomMap().put(initialState!!.x, initialState!!.y, Box(false, false, true))
      }
      "d" -> {
        initialState = Functions().result(initialState!!, RobotAction(RobotAction.TURNRIGHT)) as
            ↪ RobotState
        RoomMap.getRoomMap().put(initialState!!.x, initialState!!.y, Box(false, false, true))
      }
      "r" -> {
        initialState = Functions().result(initialState!!, RobotAction(RobotAction.TURNRIGHT)) as
            ↪ RobotState
        RoomMap.getRoomMap().put(initialState!!.x, initialState!!.y, Box(false, false, true))
      }
      "c" //forward and clean
      -> {
        RoomMap.getRoomMap().put(x, y, Box(false, false, false))
        initialState = Functions().result(initialState!!, RobotAction(RobotAction.FORWARD)) as
            ↪ RobotState
        RoomMap.getRoomMap().put(initialState!!.x, initialState!!.y, Box(false, false, true))
      }
      //Box(boolean isObstacle, boolean isDirty, boolean isRobot)
      "rightDir" -> RoomMap.getRoomMap().put(x + 1, y, Box(true, false, false))
      "leftDir" -> RoomMap.getRoomMap().put(x - 1, y, Box(true, false, false))
      "upDir" -> RoomMap.getRoomMap().put(x, y - 1, Box(true, false, false))
      "downDir" -> RoomMap.getRoomMap().put(x, y + 1, Box(true, false, false))

    }//switch
    //RoomMap.getRoomMap().setObstacles()
  } catch (e: Exception) {
    println("plannerUtil doMove: ERROR:" + e.message)
  }
```

```
168   }
169
170   fun showMap() {
171     println(RoomMap.getRoomMap().toString())
172   }
173
174   fun saveMap( fname : String) : Pair<Int,Int> {
175     println("saveMap in $fname")
176     val pw = PrintWriter( FileWriter(fname+".txt") )
177     pw.print( RoomMap.getRoomMap().toString() )
178     pw.close()
179
180     val os = ObjectOutputStream( FileOutputStream(fname+".bin") )
181     os.writeObject(RoomMap.getRoomMap())
182     os.flush()
183     os.close()
184     return getMapDims()
185   }
186
187   fun loadRoomMap( fname: String ) : Pair<Int,Int> {
188
189     try{
190       val inps = ObjectInputStream(FileInputStream("${fname}.bin"))
191       val map = inps.readObject() as RoomMap;
192       println("loadRoomMap = $fname DONE")
193       RoomMap.setRoomMap( map )
194     }catch(e:Exception){
195       println("loadRoomMap = $fname FAILURE")
196     }
197     return getMapDims()//Pair(dimMapx,dimMapy)
198   }
199
200   fun getMapDims() : Pair<Int,Int> {
201     if( RoomMap.getRoomMap() == null ){
202       return Pair(0,0)
203     }
204     val dimMapx = RoomMap.getRoomMap().getDimX()
205     val dimMapy = RoomMap.getRoomMap().getDimY()
206     //println("getMapDims dimMapx = $dimMapx, dimMapy=$dimMapy")
207     return Pair(dimMapx,dimMapy)
208   }
209
210   fun getMap() : String{
211     return RoomMap.getRoomMap().toString()
212   }
213   fun getMapOneLine() : String{
214     return "'"+RoomMap.getRoomMap().toString().replace("\n","@").replace("|","").replace(",","") +"'"
215   }
216
217   /*
218   * ------------------------------------------------------
219   */
220   fun setGoalInit() {
221     goalTest = Functions()
222   }
223
224   fun setGoal( x: String, y: String) {
225     setGoal( Integer.parseInt(x), Integer.parseInt(y))
226   }
227
228   //Box(boolean isObstacle, boolean isDirty, boolean isRobot)
229   fun setGoal( x: Int, y: Int) {
230     try {
231       println("setGoal $x,$y while robot in cell: ${getPosX()}, ${getPosY()} direction=${
          ↪ getDirection()}")
232       RoomMap.getRoomMap().put(x, y, Box(false, true, false))
233       //initialState = RobotState(getPosX(), getPosY(), initialState!!.direction )
234       goalTest = GoalTest { state : Any ->
```

```
235             val robotState = state as RobotState
236             (robotState.x == x && robotState.y == y)
237           }
238         } catch (e: Exception) {
239           e.printStackTrace()
240         }
241       }
242
243       fun setStopped(stopped: Boolean) {
244         this.stopped = stopped
245       }
246
247       fun getStopped() : Boolean {
248         return stopped
249       }
250
251       fun startTimer() {
252         timeStart = System.currentTimeMillis()
253       }
254
255       fun getDuration() : Int{
256         val duration = (System.currentTimeMillis() - timeStart).toInt()
257         println("DURATION = $duration")
258         return duration
259       }
260
261       fun getDirection() : String{
262         //val direction = initialState!!.direction.toString()
263         val direction = initialState!!.direction
264         when( direction ){
265           Direction.UP -> return "upDir"
266           Direction.RIGHT -> return "rightDir"
267           Direction.LEFT -> return "leftDir"
268           Direction.DOWN -> return "downDir"
269           else -> return "unknownDir"
270         }
271       }
272
273       /*
274       * Direction
275       */
276       fun rotateDirection() {
277         //println("before rotateDirection: " + initialState.getDirection() );
278         initialState = Functions().result(initialState!!, RobotAction(RobotAction.TURNLEFT)) as RobotState
279         initialState = Functions().result(initialState!!, RobotAction(RobotAction.TURNLEFT)) as RobotState
280         //println("after rotateDirection: " + initialState.getDirection() );
281         //update the kb
282         val x = initialState!!.x
283         val y = initialState!!.y
284         val newdir = initialState!!.direction.toString().toLowerCase() + "Dir"
285       }
286
287       fun setObstacles( ){
288         RoomMap.getRoomMap().setObstacles()
289       }
290
291       fun addTempObstacle(posX: Int, posY: Int) {
292         tempObstacles.add(Pair(posX, posY))
293         //set box as obstacle
294         RoomMap.getRoomMap().put(posX, posY, Box(true, false, false))
295       }
296
297       fun clearTempObstacles() {
298         println("CLEAR TEMP OBSTACLES")
299         for (obs in tempObstacles) {
300           RoomMap.getRoomMap().put(obs.first, obs.second, Box(false, false, false))
301         }
302         tempObstacles.clear()
```

```
303     }
304
305     fun setObstacleWall( dir: Direction, x:Int, y:Int){
306       when( dir ){
307         Direction.DOWN -> RoomMap.getRoomMap().put(x, y + 1, Box(true, false, false))
308         //Direction.UP -> RoomMap.getRoomMap().put(x, y - 1, Box(true, false, false))
309         //Direction.LEFT -> RoomMap.getRoomMap().put(x - 1, y, Box(true, false, false))
310         Direction.RIGHT -> RoomMap.getRoomMap().put(x + 1, y, Box(true, false, false))
311       }
312     }
313
314     fun wallFound(){
315       val dimMapx = RoomMap.getRoomMap().getDimX()
316       val dimMapy = RoomMap.getRoomMap().getDimY()
317       val dir = initialState!!.getDirection()
318       val x = initialState!!.getX()
319       val y = initialState!!.getY()
320       setObstacleWall( dir,x,y )
321       println("wallFound dir=$dir x=$x y=$y dimMapX=$dimMapx dimMapY=$dimMapy");
322       doMove( dir.toString() ) //set cell
323       if( dir == Direction.UP) setWallRight(dimMapx,dimMapy,x,y)
324       if( dir == Direction.RIGHT) setWallDown(dimMapx,dimMapy,x,y)
325     }
326
327     fun setWallDown(dimMapx: Int,dimMapy: Int,x: Int,y: Int ){
328       var k = 0
329       while( k < dimMapx ) {
330         RoomMap.getRoomMap().put(k, y+1, Box(true, false, false))
331         k++
332       }
333
334     }
335
336     fun setWallRight(dimMapx: Int,dimMapy: Int, x: Int,y: Int){
337       var k = 0
338       while( k < dimMapy ) {
339         RoomMap.getRoomMap().put(x+1, k, Box(true, false, false))
340         k++
341       }
342
343     }
344   }
```

moveUtils.kt

```
1   package itunibo.planner
2
3   import aima.core.agent.Action
4   import it.unibo.kactor.ActorBasic
5   import kotlinx.coroutines.delay
6   import itunibo.planner.model.RobotState.Direction
7
8   object moveUtils{
9     private var actions = ArrayList<Action>()
10    private var existPlan = false
11
12    private var mapDims : Pair<Int,Int> = Pair(0,0)
13    private var curPos : Pair<Int,Int> = Pair(0,0)
14    private var curGoal : Pair<Int,Int> = Pair(0,0)
15    private var direction = "downDir"
16    private val PauseTime = 250
17
18    private var MaxX = 0
19    private var MaxY = 0
20    private var CurX = 0
21    private var CurY = 0
```

```
22
23
24  private fun storeMovesInActor( actor : ActorBasic, actions : List<Action>? ) {
25    if( actions == null ) return
26    val iter = actions.iterator()
27    while (iter.hasNext()) {
28      val a = iter.next()
29      this.actions.add(a)
30      actor.solve("assert( move($a) )")
31    }
32  }
33
34  private fun clearMovesFromActor(actor: ActorBasic) {
35    for( a in actions) {
36      actor.solve("retract( move($a) )")
37    }
38    actions.clear()
39  }
40
41  fun loadRoomMap( actor : ActorBasic, fname : String ){
42    val dims = plannerUtil.loadRoomMap( fname )
43    memoMapDims( actor, dims )
44  }
45  fun saveMap( actor : ActorBasic, fname : String) {
46    val dims = plannerUtil.saveMap( fname )
47    memoMapDims( actor, dims )
48  }
49  fun memoMapDims( actor : ActorBasic, dims : Pair<Int,Int> ){
50    mapDims = dims
51    MaxX = dims.first
52    MaxY = dims.second
53    actor.solve("retract( mapdims(_,_) )") //remove old data
54    actor.solve("assert( mapdims( ${dims.first},${dims.second} ) )")
55  }
56
57  fun getMapDimX( ) : Int{ return mapDims.first }
58  fun getMapDimY( ) : Int{ return mapDims.second }
59  fun getPosX(actor : ActorBasic) : Int{ setPosition(actor); return curPos.first }
60  fun getPosY(actor : ActorBasic) : Int{ setPosition(actor);return curPos.second }
61  fun getDirection(actor : ActorBasic) : String{ setPosition(actor);return direction.toString() }
62  fun mapIsEmpty() : Boolean{return (getMapDimX( )==0 && getMapDimY( )==0 ) }
63
64
65  fun showCurrentRobotState(){
66    println("=================================================")
67    plannerUtil.showMap()
68    println("RobotPos=(${curPos.first},${curPos.second}) in map($MaxX,$MaxY) direction=
          ↪ $direction")
69    println("=================================================")
70  }
71  fun setObstacleOnCurrentDirection( actor : ActorBasic ){
72    doPlannedMove(actor, direction )
73  }
74
75  fun setDuration( actor : ActorBasic ){
76    val time = plannerUtil.getDuration()
77    actor.solve("retract( wduration(_) )") //remove old data
78    actor.solve("assert( wduration($time) )")
79  }
80
81  fun setDirection( actor : ActorBasic ) {
82    direction = plannerUtil.getDirection()
83    //println("moveUtils direction=$direction")
84    actor.solve("retract( direction(_) )") //remove old data
85    actor.solve("assert( direction($direction) )")
86  }
87
88  fun setGoal( actor : ActorBasic, x: String, y: String) {
```

```
    val xv = Integer.parseInt(x)
    val yv = Integer.parseInt(y)
    plannerUtil.setGoal( xv,yv )
    curGoal=Pair(xv,yv)
    actor.solve("retract( curGoal(_,_) )") //remove old data
    actor.solve("assert( curGoal($x,$y) )")
  }


  fun doPlan(actor : ActorBasic ){
    clearMovesFromActor(actor)
    val plan = plannerUtil.doPlan()
    existPlan = plan != null
    if( existPlan ) storeMovesInActor(actor,plan)
  }

  fun existPlan() : Boolean{ return existPlan }

  fun doPlannedMove(actor : ActorBasic, move: String){
    plannerUtil.doMove( move )
    setPosition(actor)
  }

  fun setPosition(actor : ActorBasic){
    direction = plannerUtil.getDirection()
    val posx = plannerUtil.getPosX()
    val posy = plannerUtil.getPosY()
    curPos = Pair( posx,posy )

    //println("setPosition curPos=($posx,$posy,$direction)")
    actor.solve("retract( curPos(_,_) )") //remove old data
    actor.solve("assert( curPos($posx,$posy) )")
    actor.solve("retract( curPos(_,_,_) )") //remove old data
    actor.solve("assert( curPos($posx,$posy,$direction) )")
  }

  suspend fun rotate(actor:ActorBasic,move:String,pauseTime:Int=PauseTime){
    when( move ){
      "a" -> rotateLeft(actor, pauseTime)
      "d" -> rotateRight(actor, pauseTime)
      else -> println("rotate $move unknown")
    }
  }
  suspend fun rotateRight(actor : ActorBasic, pauseTime : Int = PauseTime){
    actor.forward("modelChange", "modelChange(robot,d)", "resourcemodel")
    doPlannedMove(actor, "d" ) //update map
    delay( pauseTime.toLong() )
  }
  suspend fun rotateLeft(actor : ActorBasic, pauseTime : Int = PauseTime){
    actor.forward("modelChange", "modelChange(robot,a)", "resourcemodel")
    doPlannedMove(actor, "a" ) //update map
    delay( pauseTime.toLong() )
  }
  suspend fun moveAhead(actor:ActorBasic, stepTime:Int, pauseTime:Int = PauseTime, dest:String ="
      ↪ resourcemodel"){
    println("moveUtils moveAhead stepTime=$stepTime")
    actor.forward("modelChange", "modelChange(robot,w)", dest)
    delay( stepTime.toLong() )
    actor.forward("modelChange", "modelChange(robot,h)", dest)
    doPlannedMove(actor, "w" ) //update map
    delay( pauseTime.toLong() )
  }
  suspend fun attemptTomoveAhead(actor:ActorBasic,stepTime:Int, dest:String ="onestepahead"){
    //println("moveUtils attemptTomoveAhead stepTime=$stepTime")
    actor.forward("onestep", "onestep(${stepTime})", dest)
  }
  fun updateMapAfterAheadOk(actor : ActorBasic ){
    doPlannedMove(actor , "w")
```

```
156     }
157     suspend fun backToCompensate(actor : ActorBasic, stepTime : Int, pauseTime : Int = PauseTime){
158       println("moveUtils backToCompensate stepTime=$stepTime")
159       actor.forward("modelChange", "modelChange(robot,s)", "resourcemodel")
160       delay( stepTime.toLong() )
161       actor.forward("modelChange", "modelChange(robot,h)", "resourcemodel")
162       delay( pauseTime.toLong() )
163     }
164 }
```

8.5.2 CoAP

Fridge CoAP Resource

Being the **Fridge** the only entity designed as a CoAP resource, the CoAP server is created inside the **Fridge** node. The *consult* (or *ask* if the source is the **RBR**) and the expose functions are handled as GET methods, while the update of the state is, of course, handled as a PUT method.

fridgeResourceCoAP.kt

```
1  package itunibo.coap
2  import org.eclipse.californium.core.coap.CoAP.ResponseCode.BAD_REQUEST
3  import org.eclipse.californium.core.coap.CoAP.ResponseCode.CHANGED
4  import org.eclipse.californium.core.CoapResource
5  import org.eclipse.californium.core.coap.CoAP.ResponseCode
6  import org.eclipse.californium.core.coap.MediaTypeRegistry
7  import org.eclipse.californium.core.server.resources.CoapExchange
8  import it.unibo.kactor.ActorBasic
9  import it.unibo.kactor.MsgUtil
10 import org.eclipse.californium.core.CoapServer
11 import kotlinx.coroutines.launch
12 import kotlinx.coroutines.delay
13 import kotlinx.coroutines.GlobalScope
14 import org.eclipse.californium.core.coap.CoAP.Type
15 import itunibo.fridge.fridgeModelSupport
16
17 class fridgeResourceCoap (name : String ) : CoapResource(name) {
18
19   companion object {
20     lateinit var actor : ActorBasic
21     var curmodelval = "unknown"
22     var curanswer = "unknown"
23     lateinit var resourceCoap : fridgeResourceCoap
24
25     fun create( a: ActorBasic, name: String ){
26       actor = a
27       val server = CoapServer(5683);  //COAP SERVER
28       resourceCoap = fridgeResourceCoap( name )
29       server.add( resourceCoap );
30       println("-------------------------------------------------")
31       println("Coap Server started");
32       println("-------------------------------------------------")
33       server.start();
34       fridgeModelSupport.setCoapResource(resourceCoap) //Injects a reference
35     }
36
37     fun getModel() : String {
38       return curmodelval
39     }
40
41     fun getAnswer() : String {
42       return curanswer
43     }
```

```
44    }
45
46    init {
47      println("-----------------------------------------------------")
48      println("fridgeResourceCoap init")
49      println("-----------------------------------------------------")
50      setObservable(true)    // enable observing !!!!!!!!!!!!!!
51      setObserveType(Type.CON)  // configure the notification type to CONs
52      //getAttributes().setObservable(); // mark observable in the Link-Format
53    }
54
55    fun updateAnswer( answeritem : String ){
56      curanswer = answeritem
57      //println("%%%%%%%%%%%%%%%% updateState from $curState to $curmodelval" )
58      changed() // notify all CoAp observers
59      /*
60      * Notifies all CoAP clients that have established an observe relation with
61      * this resource that the state has changed by reprocessing their original
62      * request that has established the relation. The notification is done by
63      * the executor of this resource or on the executor of its parent or
64      * transitively ancestor. If no ancestor defines its own executor, the
65      * thread that has called this method performs the notification.
66      */
67    }
68
69    fun updateState( modelitem : String ){
70      curmodelval = modelitem
71      //println("%%%%%%%%%%%%%%%% updateState from $curState to $curmodelval" )
72      changed() // notify all CoAp observers
73      /*
74      * Notifies all CoAP clients that have established an observe relation with
75      * this resource that the state has changed by reprocessing their original
76      * request that has established the relation. The notification is done by
77      * the executor of this resource or on the executor of its parent or
78      * transitively ancestor. If no ancestor defines its own executor, the
79      * thread that has called this method performs the notification.
80      */
81    }
82
83    override fun handleGET(exchange: CoapExchange?) {
84      try {
85        val value = exchange!!.getRequestText()  //new String(payload, "UTF-8");
86        println(value)
87        GlobalScope.launch{
88          if(value == "") {
89            MsgUtil.sendMsg( "expose", "expose", actor )
90            delay(100)
91            exchange.respond(ResponseCode.CONTENT, curmodelval, MediaTypeRegistry.TEXT_PLAIN)
92          }
93          else {
94            MsgUtil.sendMsg( "request", "request(maitre, $value)", actor)
95            delay(100)
96            exchange.respond(ResponseCode.CONTENT, curanswer, MediaTypeRegistry.TEXT_PLAIN)
97          }
98        }
99      } catch (e: Exception) {
100       exchange!!.respond(BAD_REQUEST, "Invalid String")
101     }
102
103   }
104
105   override fun handlePOST(exchange: CoapExchange?) {
106     //println("%%%%%%%%%%%%%%%% handlePOST " )
107     handlePUT( exchange )
108   }
109
110   override fun handlePUT(exchange: CoapExchange?) {
111     try {
```

```
        val value = exchange!!.getRequestText()//new String(payload, "UTF-8");'
        //val curState = curmodelval
        GlobalScope.launch {
          MsgUtil.sendMsg( "modelUpdate", "modelUpdate(fridge, $value )", actor )
          delay(100) //give the time to change the model
          //updateState()
          exchange.respond(CHANGED, value)
        }
      } catch (e: Exception) {
        exchange!!.respond(BAD_REQUEST, "Invalid String")
      }
    }
  }
}
```

fridgeModelSupport.kt

```
package itunibo.fridge
import it.unibo.kactor.ActorBasic
import kotlinx.coroutines.launch
import itunibo.coap.fridgeResourceCoap

object fridgeModelSupport{
  lateinit var resourcecoap : fridgeResourceCoap

  fun setCoapResource( rescoap : fridgeResourceCoap ) {
    resourcecoap = rescoap
  }

  fun answerRequest(actor: ActorBasic, source: String, foodcode: String) {
    actor.solve( "model( resource, fridge, state(STATE) )" )
    actor.solve( "contains($foodcode, ${actor.getCurSol("STATE")})" )
    actor.scope.launch {
      if(actor.solveOk()) {
        if(source == "roombutlerrobot") {
          actor.emit("answer", "answer(yes)")
        } else {
          resourcecoap.updateAnswer( "yes" )
        }
      } else {
        if(source == "roombutlerrobot") {
          actor.emit("answer", "answer(no)")
        } else {
          resourcecoap.updateAnswer( "no" )
        }
      }
    }
  }

  fun exposeFridgeModel( actor: ActorBasic ){
    actor.solve( "model( A, fridge, STATE )" )
    val FridgeState = actor.getCurSol("STATE")
    actor.scope.launch {
      resourcecoap.updateState( "fridge($FridgeState)" )
    }
  }

  fun updateFridgeModel( actor: ActorBasic, content: String ){
    actor.solve( "action( fridge, $content )" ) //change the robot state model
    actor.solve( "model( A, fridge, STATE )" )
    val FridgeState = actor.getCurSol("STATE")
    actor.scope.launch {
      //sent to notify to the RBR that the change in the model has been performed
      resourcecoap.updateState( "fridge($FridgeState)" )
    }
  }
}
```

CoAP Client To remove from the *roombutlerrobot* actor every interaction with other entities, the CoAP client is delegated to the *resourcemodel* actor.

resModelClientCoap.kt

```
package itunibo.coap.client

import org.eclipse.californium.core.CoapClient
import org.eclipse.californium.core.CoapResponse
import org.eclipse.californium.core.Utils
import org.eclipse.californium.core.coap.MediaTypeRegistry
import org.eclipse.californium.core.CoapHandler
import org.eclipse.californium.core.coap.Request
import org.eclipse.californium.core.coap.CoAP.Code
import it.unibo.kactor.ActorBasic
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import it.unibo.kactor.MsgUtil
import kotlinx.coroutines.delay

object coapClientResModel {

  private lateinit var coapClient: CoapClient
  private lateinit var coapURI: String
  private lateinit var actor : ActorBasic

  fun createClient(a: ActorBasic, serverAddr: String, port: Int, resourceName: String?) {
    actor = a
    coapClient = CoapClient("coap://$serverAddr:" + port + "/" + resourceName)
    coapURI = "coap://$serverAddr:" + port + "/" + resourceName
    println("Client started")
  }

  fun synchGet(v: String) { //Synchronously send the GET message (blocking call)
    println("%%% synchGet ")
    val request = Request(Code.GET)
    request.setPayload(v)

    val coapResp = coapClient.advanced(request)
    println(coapResp.responseText)

    var answer = coapResp.responseText
    //The "CoapResponse" message contains the response.
    //println(Utils.prettyPrint(coapResp))
    GlobalScope.launch{
      actor.emit("answer", "answer($answer)")
    }
  }

  fun put(v: String) {
    val coapResp = coapClient.put(v, MediaTypeRegistry.TEXT_PLAIN)
    //The "CoapResponse" message contains the response.
    println("%%% ANSWER put $v:")
    println(coapResp.responseText)
    GlobalScope.launch{
      actor.emit("roomModelChanged", "modelChanged(fridge, ${coapResp.responseText})")
    }
  }

  fun asynchGet() {
    coapClient.get( AsynchListener );
  }

}
```

Frontend

To make the frontend CoAP enabled, we added the script *coapClientToFridge.js* and introduced some changes in *applCode.js*

coapClientToFridge.js

```
/*
frontend/uniboSupports/coapClientToFridge
*/
const coap = require("node-coap-client").CoapClient;
var coapAddr = "coap://localhost:5683"
var coapResourceAddr = coapAddr + "/fridge"
var io;  //Upgrade for socketIo;
/*
coap
.tryToConnect( coapAddr )
.then((result ) => { // true or error code or Error instance
  console.log("coap connection done"); // do something with the result
})
;
*/

exports.setIoSocket = function (iosock) {
  io = iosock;
  console.log("coap SETIOSOCKET io=" + io);
}

exports.setcoapAddr = function ( addr ){
  coapAddr = "coap://"+ addr + ":5683";
  coapResourceAddr = coapAddr + "/fridge";
  console.log("coap coapResourceAddr " + coapResourceAddr);
}

exports.coapGet = function ( param ){
  coap
  .request(
  coapResourceAddr,
  "get", //"get" | "post" | "put" | "delete"
  new Buffer(param) //payload Buffer
  //[options]] // RequestOptions
  )
  .then(response => {   /* handle response */
    var msgStr = response.payload
    console.log("coap get done> " + response.payload);
    if (msgStr.indexOf("fridge") < 0) {
      if (msgStr == "yes") {
        content = "Answer: The fridge contains the requested food.";
      } else {
        content = "Answer: The fridge does not contain the requested food.";
      }
    } else
    content = "Fridge exposing its content: " + msgStr;

    console.log("coap send on io.sockets| content=" + content);
    io.sockets.send(content);

  })
  .catch(err => { /* handle error */
    console.log("coap get error> " + err );}
  )
  ;

}//coapPut

exports.coapPut = function ( cmd ){
```

```
  coap
  .request(
  coapResourceAddr,
  "put" ,  // "get" | "post" | "put" | "delete"
  new Buffer(cmd ) // payload Buffer
  //[options]]     // RequestOptions
  )
  .then(response => {   // handle response

    console.log("coap put done> " + cmd);}
  )
  .catch(err => { // handle error
    console.log("coap put error> " + err + " for cmd=" + cmd);}
  )
  ;

}//coapPut

const myself = require('./coapClientToFridge');

//test()

/*
* ========= EXPORTS =======
*/

//module.exports = coap;
```

applCode.js

```
/*
frontend/applCode
*/
const express  = require('express');
const path  = require('path');
//const favicon = require('serve-favicon');
const logger  = require('morgan'); //see 10.1 of nodeExpressWeb.pdf;
//const cookieParser= require('cookie-parser');
const bodyParser  = require('body-parser');
const fs  = require('fs');
const index = require('./appServer/routes/index');
var io ;  //Upgrade for socketIo;

//for delegate
const mqttUtils = require('./uniboSupports/mqttUtils');
const coap = require('./uniboSupports/coapClientToFridge');

var app = express();

// view engine setup;
app.set('views', path.join(__dirname, 'appServer', 'views'));
app.set('view engine', 'ejs');

//create a write stream (in append mode) ;
var accessLogStream = fs.createWriteStream(path.join(__dirname, 'morganLog.log'), {flags: 'a'})
app.use(logger("short", {stream: accessLogStream}));

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));   //shows commands, e.g. GET /pi 304 23.123 ms - -;
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
//app.use(cookieParser());

app.use(express.static(path.join(__dirname, 'public')));
app.use(express.static(path.join(__dirname, 'jsCode'))); //(***)
```

```
37
38 app.get('/', function(req, res) {
39   res.render("index");
40   console.log("starting")
41   setTimeout(delegateForResource, 200, "modelExpose", req, res);
42 });
43
44 /*
45 * ====================== COMMANDS ================
46 */
47 //TESTING
48 app.post("/changePrepSet", function (req, res, next) {
49   content = req.body.prep_set;
50   delegateForAppl("prepChange", req, res, content);
51   next();
52 });
53 app.post("/addFridge", function (req, res, next) {
54   content = "add(" + req.body.foodcode_resfridge + ")";
55   delegateForFridge("addFridge", req, res, content);
56   next();
57 });
58 app.post("/removeFridge", function (req, res, next) {
59   content = "remove(" + req.body.foodcode_resfridge + ")";
60   delegateForFridge("removeFridge", req, res, content);
61   next();
62 });
63 app.post("/addTable", function (req, res, next) {
64   content = "table, add(" + req.body.itemcode_table + ")";
65   delegateForResource("modelUpdate", req, res, content);
66   next();
67 });
68 app.post("/removeTable", function (req, res, next) {
69   content = "table, remove(" + req.body.itemcode_table + ")";
70   delegateForResource("modelUpdate", req, res, content);
71   next();
72 });
73 app.post("/addPantry", function (req, res, next) {
74   content = "pantry, add(" + req.body.itemcode_pantry + ")";
75   delegateForResource("modelUpdate", req, res, content);
76   next();
77 });
78 app.post("/removePantry", function (req, res, next) {
79   content = "pantry, remove(" + req.body.itemcode_pantry + ")";
80   delegateForResource("modelUpdate", req, res, content);
81   next();
82 });
83 app.post("/addDishwasher", function (req, res, next) {
84   content = "dishwasher, add(" + req.body.itemcode_dishwasher + ")";
85   delegateForResource("modelUpdate", req, res, content);
86   next();
87 });
88 app.post("/removeDishwasher", function (req, res, next) {
89   content = "dishwasher, remove(" + req.body.itemcode_dishwasher + ")";
90   delegateForResource("modelUpdate", req, res, content);
91   next();
92 });
93
94 //APPLICATION
95 app.post("/stop", function(req, res,next) {
96   delegateForEvents("stop", req, res);
97   next();
98 });
99 app.post("/reactivate", function(req, res,next) {
100   delegateForEvents("resume", req, res);
101   next();
102 });
103 app.post("/explore", function(req, res,next) {
104   delegateForAppl("explore", req, res);
```

```
105   next();
106 });
107 app.post("/prepare", function(req, res,next) {
108   delegateForAppl("prepare", req, res);
109   next();
110 });
111 app.post("/clear", function(req, res,next) {
112   delegateForAppl("clear", req, res);
113   next();
114 });
115 app.post("/addFood", function (req, res, next) {
116   content = req.body.foodcode_app
117   delegateForAppl("addFood", req, res, content);
118   next();
119 });
120 app.post("/expose", function (req, res, next) {
121   delegateForFridge("expose", req, res, "");
122   next();
123 });
124 app.post("/ask", function (req, res, next) {
125   food = req.body.foodcode_fridge
126   //getFridgeModelCoap(food)
127   delegateForFridge("ask", req, res, food);
128   next();
129 });
130
131 //================== UTILITIES ========================
132
133 var result = "";
134
135 app.setIoSocket = function( iosock ){
136   io = iosock;
137   mqttUtils.setIoSocket(iosock);
138   coap.setIoSocket(iosock);
139   console.log("app SETIOSOCKET io=" + io);
140 }
141
142 function delegateForEvents(cmd, req, res) {
143   console.log("app delegateForEvents cmd=" + cmd);
144   result = "Web server delegateForEvents: " + cmd;
145
146   emitEvent(cmd);
147 }
148
149 function delegateForAppl(cmd, req, res, content) {
150   console.log("app delegateForAppl cmd=" + cmd);
151   result = "Web server delegateForAppl: " + cmd;
152
153   if (arguments.length === 4) {
154     publishMsgToRobotapplication(cmd, content);
155   }
156   else {
157     publishMsgToRobotapplication(cmd);
158   }
159 }
160
161 function delegateForResource(cmd, req, res, content) {
162   console.log("app delegateForResourceModel cmd=" + cmd);
163   result = "Web server delegateForResourceModel: " + cmd;
164
165   if (arguments.length === 4) {
166     publishMsgToResourceModel(cmd, content);
167   }
168   else {
169     publishMsgToResourceModel(cmd);
170   }
171 }
172
```

```
function delegateForFridge(cmd, req, res, content) {
  console.log("app delegateForFridge cmd=" + cmd);
  result = "Web server delegateForFridge: " + cmd;

  if (cmd == "expose" || cmd == "ask") {
    console.log(content);
    getFridgeModelCoap(content);
  }
  else {
    console.log(content)
    changeFridgeModelCoap(content);
  }
}

/*
* ============ TO THE BUSINESS LOGIC =======
*/

var publishMsgToRobotapplication = function (cmd, content) {
  var msgstr;
  if (arguments.length === 2) {
    msgstr = "msg(" + cmd + ",dispatch,js,roombutlerrobot," + cmd + "(" + content + "),1)";
  } else {
    msgstr = "msg(" + cmd + ",dispatch,js,roombutlerrobot," + cmd + ",1)";
  }
  console.log("publishMsgToRobotapplication/" + arguments.length +" forward> " + msgstr);
  mqttUtils.publish(msgstr, "unibo/qak/roombutlerrobot");
}

var publishMsgToResourceModel = function (cmd, content) {
  var msgstr;
  if (arguments.length === 2) {
    msgstr = "msg(" + cmd + ",dispatch,js,resourcemodel," + cmd + "(" + content + "),1)";
  } else {
    msgstr = "msg(" + cmd + ",dispatch,js,resourcemodel," + cmd + ",1)";
  }
  console.log("publishMsgToResourceModel/" + arguments.length + " forward> " + msgstr);
  mqttUtils.publish(msgstr, "unibo/qak/resourcemodel");
}

var emitEvent = function (cmd) {
  var msgstr;
  msgstr = "msg(" + cmd + ",event,js,none," + cmd + ",1)";

  console.log("emitEvent/ forward> " + msgstr);
  mqttUtils.publish(msgstr, "unibo/qak/events");
}

var getFridgeModelCoap = function (param) {
  console.log("coap GET> ");
  coap.coapGet(param); //see fridgeResourceCoap
}

var changeFridgeModelCoap = function (cmd) {
  console.log("coap PUT> " + cmd);
  coap.coapPut(cmd); //see fridgeResourceCoap
}

/*
* ====================== REPRESENTATION ================
*/
app.use( function(req,res){
  console.info("SENDING THE ANSWER " + result + " json:" + req.accepts('josn') );
  try{
    console.log("answer> "+ result );
    /*
    if (req.accepts('json')) {
      return res.send(result); //give answer to curl / postman
```

```
241     } else {
242       return res.render('index' );
243     };
244     */
245     //res.send(result);
246     //return res.render('index' ); //NO: we loose the message sent via socket.io
247   }catch(e){console.info("SORRY ..." + e);}
248 }
249 );
250
251 //app.use(converter());
252
253 /*
254 * ============ ERROR HANDLING =======
255 */
256
257 // catch 404 and forward to error handler;
258 app.use(function(req, res, next) {
259   var err = new Error('Not Found');
260   err.status = 404;
261   next(err);
262 });
263
264 // error handler;
265 app.use(function(err, req, res, next) {
266   // set locals, only providing error in development
267   res.locals.message = err.message;
268   res.locals.error = req.app.get('env') === 'development' ? err : {};
269
270   // render the error page;
271   res.status(err.status || 500);
272   res.render('error');
273 });
274
275 /*
276 * ========= EXPORTS =======
277 */
278
279 module.exports = app;
```

9 Maintenance

The maintenance of the presented software should be eased by the development process and practice we adopted. The model-driven approach we followed using the QActor meta-model ensures that low level (i.e. the code controlling the physical devices) components and configuration details can be modified without interfering with the system's logical architecture and vice-versa. Also, the software factory provided by Qak would make it quite easy to introduce new high level features and functionalities.