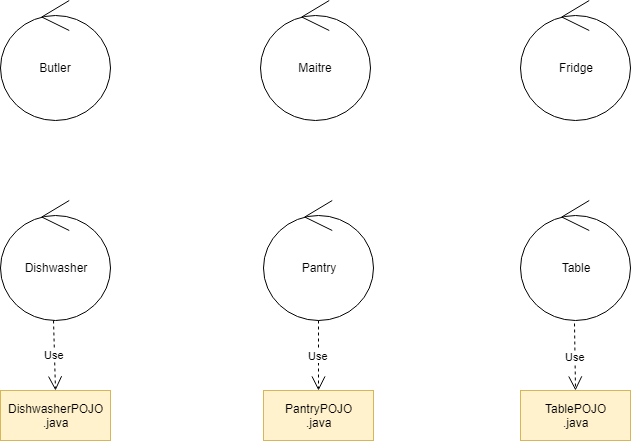
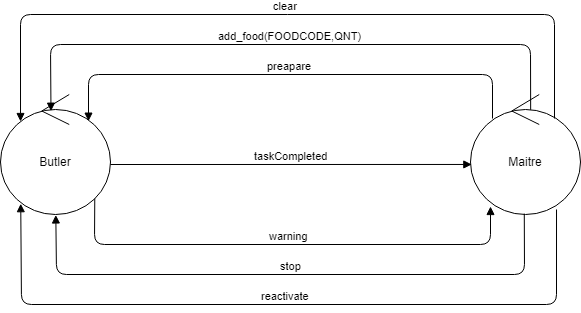
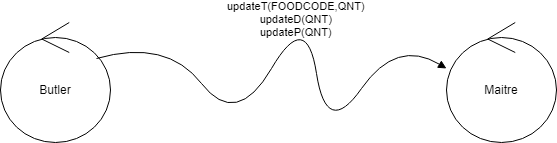
ANALISI INIZIALE

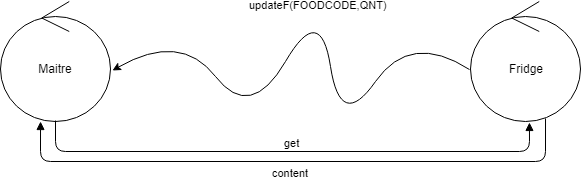
Struttura:

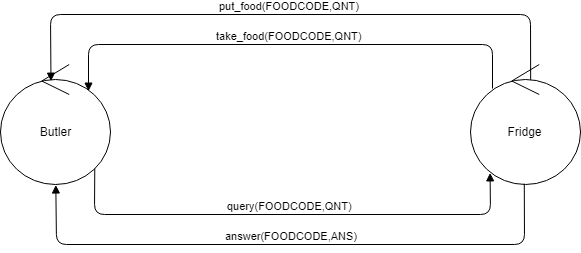
Interazione:

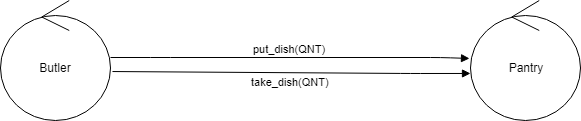
Butler-Maitre



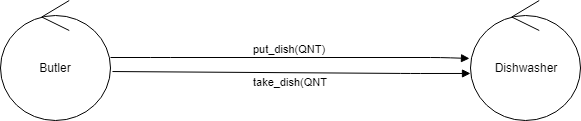


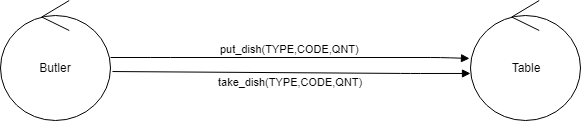
Maitre-Fridge

Butler-Fridge

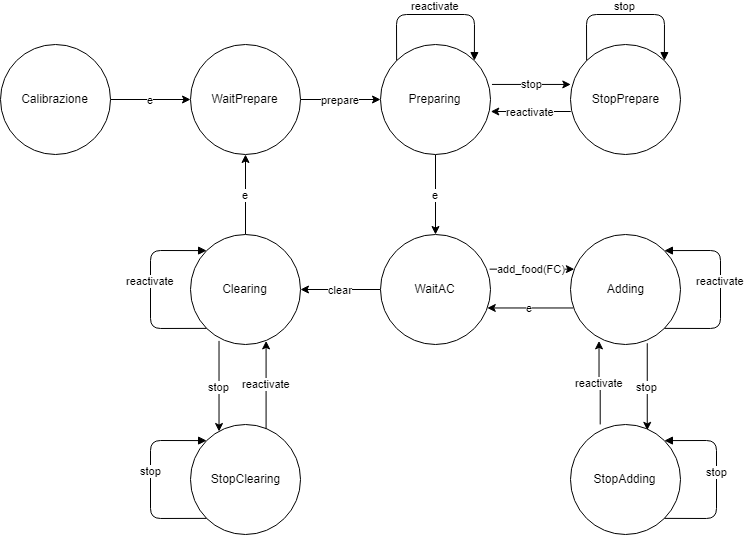
Butler-Pantry

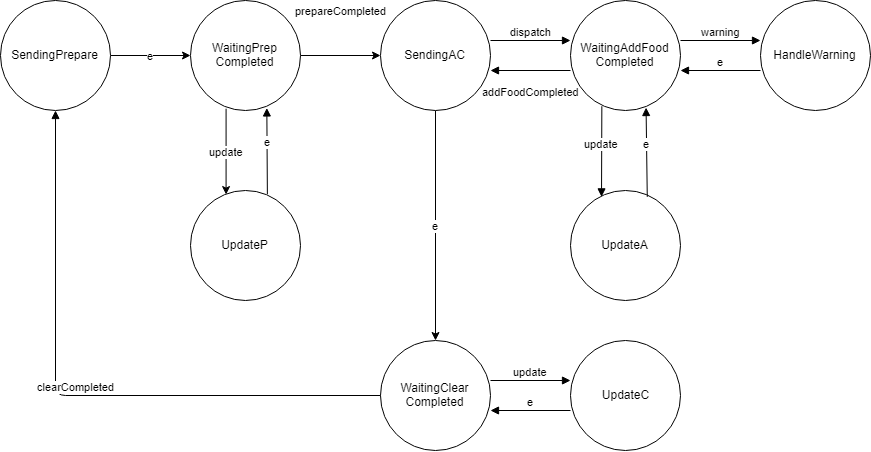
Butler-Dishwasher

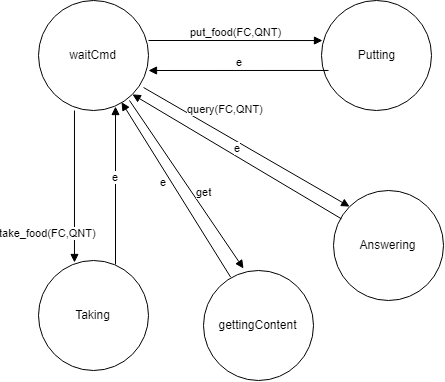


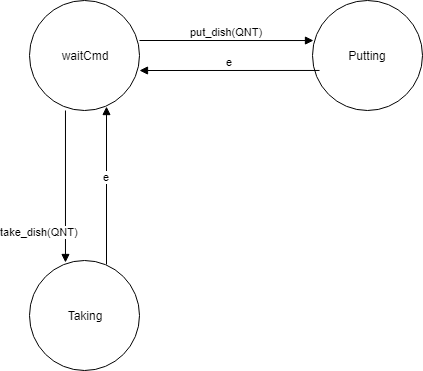
Butler-Table

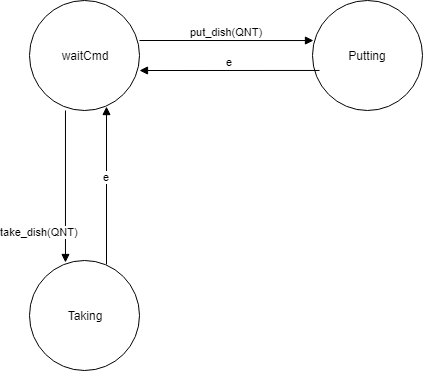
Comportamento:

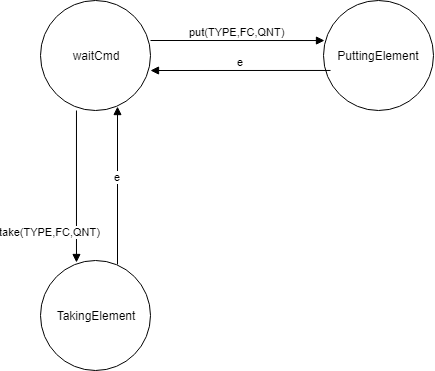
Butler

Maitre

Fridge

Pantry

Dishwasher

Table

TEST PLANS

Tutti i componenti sono modellati come automi a stati finiti, come tali i test dovrebbero essere in grado di stabili se effettivamente è avvenuta la transizione di stato desiderata.

Al fine di fare questo potrebbe essere una buona idea:

* Poter identificare univocamente uno stato
* Poter accedere allo stato corrente di un componente

Il primo punto può considerarsi già rispettato se consideriamo come nome univoco quello dato ai vari stati nei diagrammi precedenti.

Alla luce degli aspetti emersi fino a questo punto, vorremmo:

* Verificare la correttezza delle transizioni
  + se il robot è nello stato *waitPrepare* e riceve ‘prepare’ ci aspettiamo che successivamente si trovi nello stato di *preparing*.
* Avere accesso alla attuale posizione del robot
  + Negli stati di waiting il robot deve trovarsi nella posizione RH
* Avere accesso allo stato di movimento del robot
  + Negli stati di stop ci aspettiamo che il robot sia fermo

Sebbene gli esempi precedenti siano stati fatti per il robot analoghe considerazioni possono essere fatte per maitre, pantry, table, dishwasher e fridge

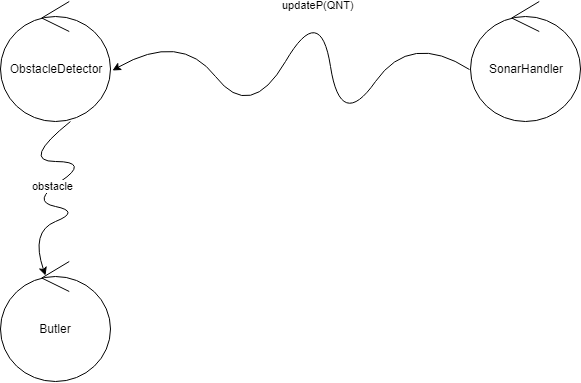
* Avere accesso al contenuto dei vari componenti
  + Dopo aver tolto 10 piatti dalla pantry voglio controllare che effettivamente siano stati tolti

Per quanto riguarda l’identificazione univoca degli stati è possibile utilizzare i nomi assegnati nei diagrammi precedenti. Per l’accesso allo stato dei componenti si può pensare alla creazione di un *MODELLO* per ognuno di questi che abbia lo scopo di monitorare ciò che avviene in un certo istante nel sistema, il modello cambierà in base alle varie azioni che verranno compiute. Un buon modo di modellare queste informazioni è quello di usare dei file Prolog che oltre alla flessibilità e la potenza espressiva che si ha tramite l’unificazione permettono di modificare facilmente le varie voci presenti all’interno.

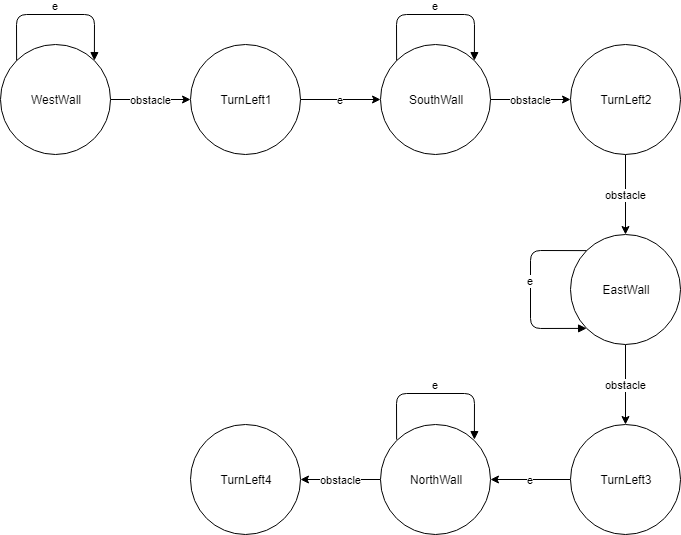
Di seguito le varie informazioni di cui si pensa possa essere utile tenere traccia per ogni componente modellate già sottoforma di facts:

* MAITRE
  + state(maitre,X)
  + task(maitre,X)
* PANTRY
  + state(pantry,X)
  + dish(pantry,QNT)
* FRIDGE
  + state(fridge,X)
  + food(fridge,FC,QNT)
* TABLE
  + state(table,X)
  + dish(table,X)
  + food(table,FC,QNT)
* DISHWASHER
  + state(dishwasher,X)
  + dish(dishwasher,QNT)
* BUTLER
  + state(butler,X)
  + position(butler,[X,Y])
  + action(butler,X)
  + task(butler,X)
  + onboardDish(butler,QNT)
  + onboardFood(butler,QNT)

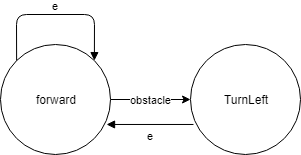
ZOOMING 1

Struttura e Interazione:

Comportamento:

Calibrazione

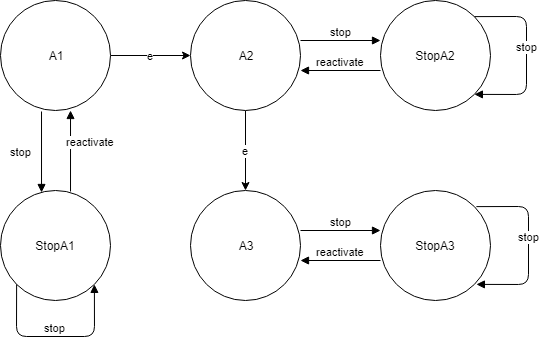
Con l’ausilio dell’infrastruttura qak che permette di tenere traccia di quante volte si è girato, è possibile semplificare lo schema sotto in:

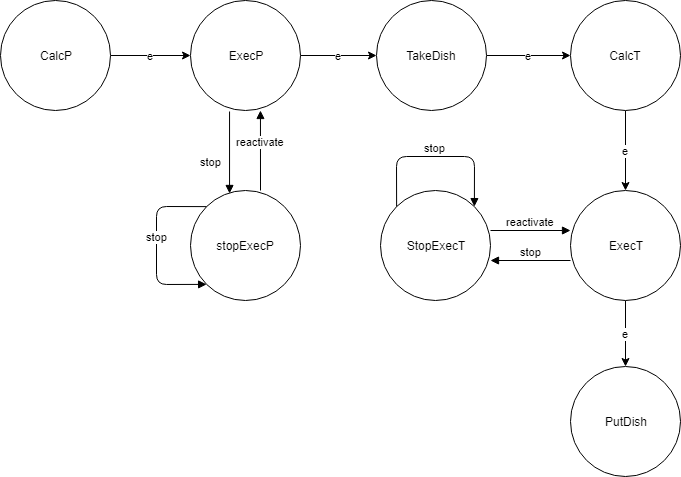


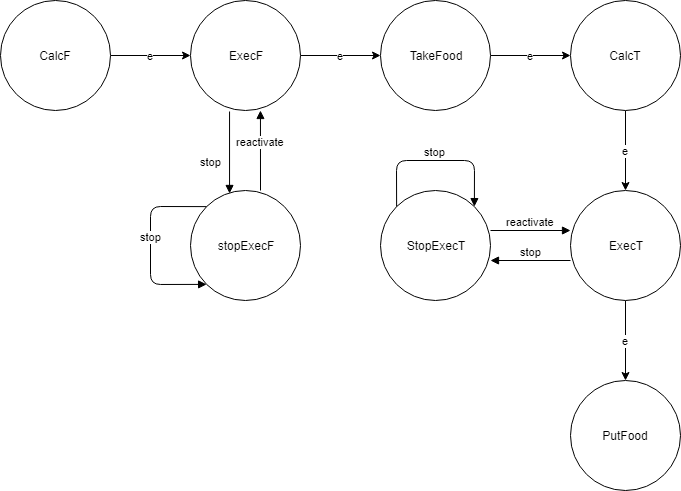
Preparing

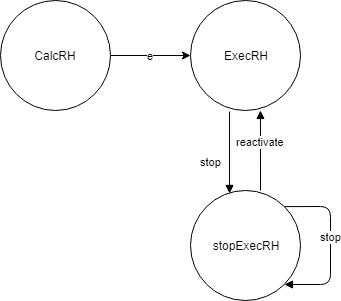
Divisa in:

* Prendere i piatti dalla Pantry e portarli al tavolo – A1
* Prendere il cibo dal Fridge e portarlo sul tavolo – A2
* Tornare in RH – A3



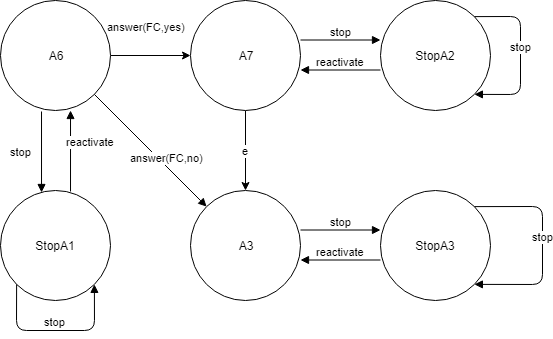
A1

A2

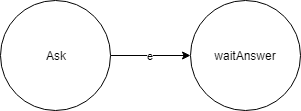
A3

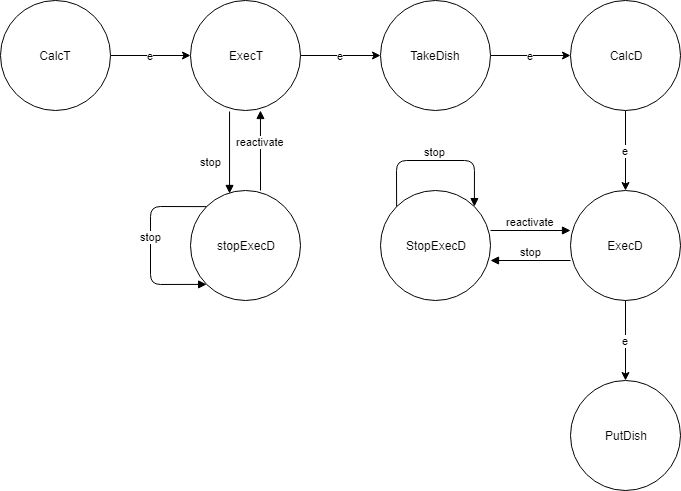
ADD FOOD

Divisa in:

* Fai richiesta disponibilità cibo al fridge – A6
* Prendi cibo – A7
* Tornare in RH – A3

A6

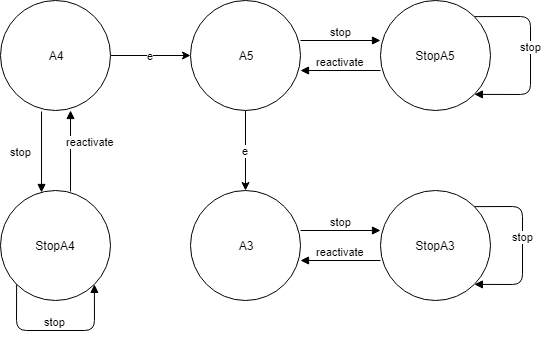


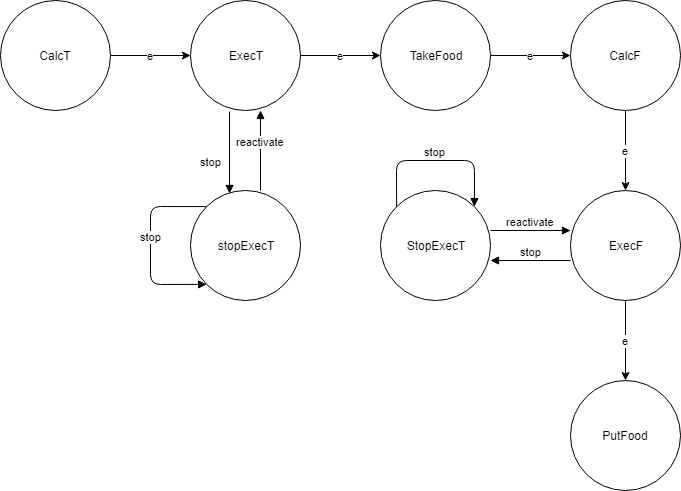
A7

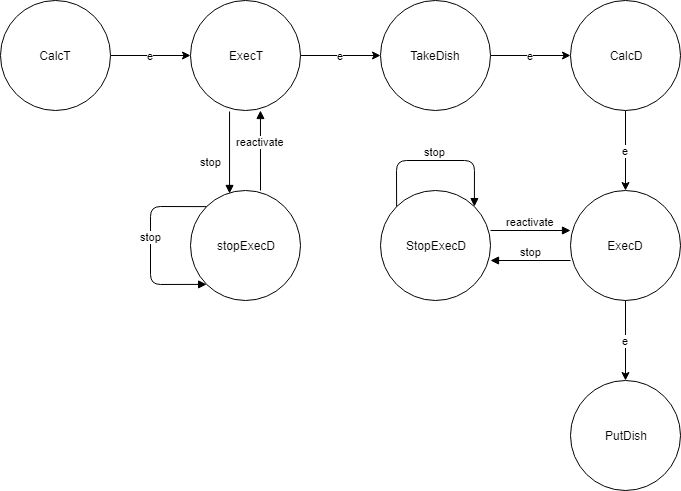
CLEAR

Divisa in:

* Portare cibo dal tavolo al fridge – A4
* Portare piatti dal tavolo alla dishwasher – A5
* Tornare in RH



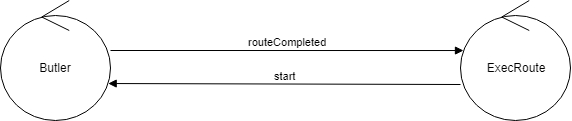
A4



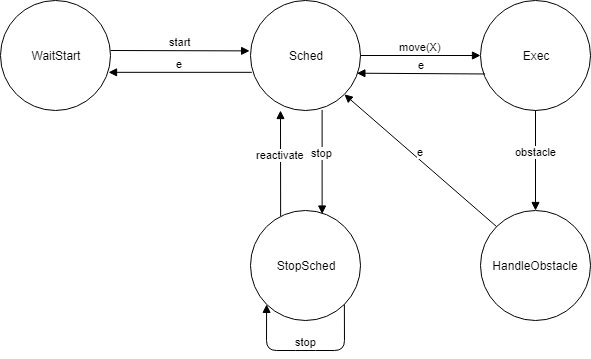
A5

ZOOMING 2

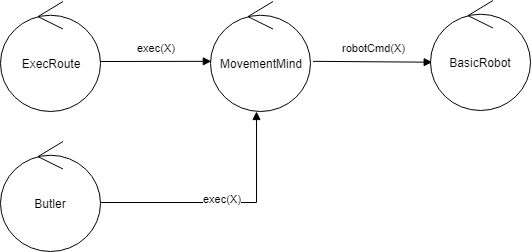
Interazione ExecRoute Butler



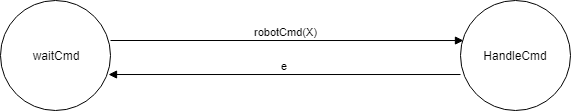
Comportamento ExecRoute

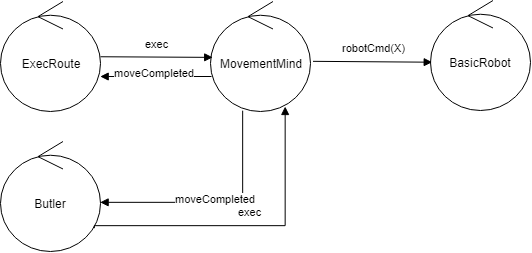


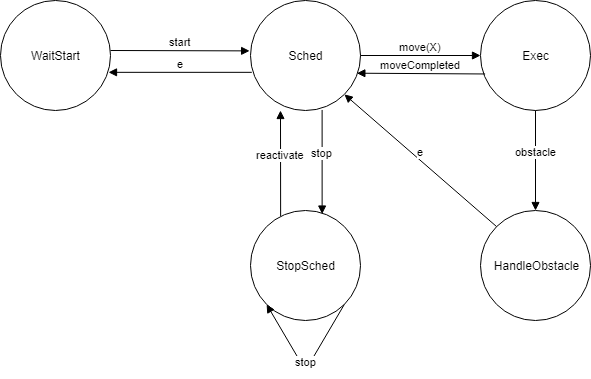
Interazione BasicRobot



Comportamento BaiscRobot



In realtà questa interazione è parziale perché bisogna che la parte di business logic resti in attesa fintanto che non arrivi un messaggio di completamento del movimento. L’interazione cambia in questo modo:

E quindi anche alcuni dei componenti gia definiti cambiano il loro comportamento perché devono attendere dei messaggi di tipo *movementCompleted*.

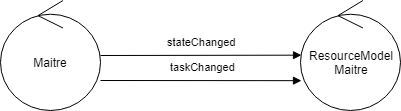
INTRODUZIONE DEL MODELLO

Struttura:



Interazione:

Maitre



Fridge



Pantry



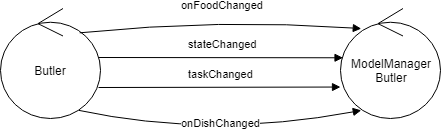
Dishwasher



Table



Butler



Per il butler la situazione è diversa, in particolare potrebbe essere adottato un architettura model-driven in cui nel sistema non cambia nulla se prima non cambia il modello

A livello di business logic i cambiamenti che si hanno riguardano

