**Cook Agent**

Data Messages

List<Order> orders; hereIsAnOrder(Waiter waiter, String choice, int table)

List<MyMarket> markets; orders.add(new Order(waiter, choice, table))

List<FoodNeeded> foodToOrder; foodDone(Order order)

class Order { order.state = done

Waiter waiter; orderNotFulfilled(MarketAgent market, String food,

String choice; int amount)

int table; foodToOrder.add(new FoodNeeded(food, amount))

State state; if Ǝ mm in markets ϶ m.market = market

} m.outOf(food);

class MyMarket { orderFinished(MarketAgent market, String food,

MarketAgent market int amount)

MarketState state; foodToOrder.remove(food)

boolean outOfSteak; foods.get(food).inventory += amount;

boolean outOfChicken;

boolean outOfPizza;

boolean outOfSalad;

}

class Food {

String type;

int cookingTimer;

int inventory;

int low;

int capacity;

}

class FoodNeeded {

String type

int amount;

FoodNeededState state;

}

String name;

enum State {pending, outofstock, cooking, done}

enum FoodNeededState {needtoorder, ordered, cantbeordered}

enum MarketState {available, outofstock}

Timer timer;

map(String, Food) foods

Scheduler Actions

if Ǝ o in orders ϶ o.state = pending tryToCookIt(Order order) {

tryToCookIt(o) order.state = cooking; food.inventory--;

if Ǝ o in order ϶ o.state = done timer.start( run(foodDone(order)),

plateFood(o) cookingTime); checkIfFoodIsLow(); }

if Ǝ o in order ϶ o.state = outOfStock plateFood(Order order) {

notifyWaiterOutOfStock(o) order.waiter.orderReady(

if Ǝ item in foodToOrder ϶ item.state = order.choice, order.table)

needToOrder orders.remove(order) }

if Ǝ myMarket in markets ϶ notifyWaiterOutOfSTock(Order order) {

myMarket has item order.waiter.OutOf(order.choice, order.table)

orderLowfood(myMarket) orders.remove(order) }

orderLowFood(MyMarket myMarket) {

myMarket.market.OrderFood(this, foodToOrder) }