



ROBOTIC RESTAURANT

DESIGN DOC

BEIXI HAO | FEBRUARY 28, 2020



CONTENTS

1. Introduction

- Design Goal
- Description
- Requirements
- Constraint
- Simplification

2. Design Specification

- Use Cases Analysis
- Entity
- UML Class Diagram
- Microservices Diagram
- Flow Chart (Sequence Diagram from Ordering to Delivery)
- API outline for the Ordering Interface

3. Further Consideration

- Reliability
- Scalability
- Maintainability



MAJOR COMPONENTS

- Order
- Food Item
- Robot
- Task
- Control Software

INTRODUCTION

DESIGN GOAL

The purpose of this design document is to build the software for a fully automated restaurant that accepts online orders (referenced from *Software Design Challenge* document).

DESCRIPTION

Customers can submit orders through the restaurant's website or mobile app. After receiving an order, the automated restaurant will prepare the food. The restaurant uses a series of specialized robots to prepare a variety of food including burgers, salads, sandwiches, side dishes, and desserts.



REQUIREMENTS

- Each order will have a series of food items that must be prepared.
- Each food item requires a series of steps that must be completed in a certain order, and most steps must be completed within a given duration of time.
- Each specialized robot only handles a specific task. For example, one robot may cook burger patties, another prepares vegetables, a third robot assembles and packages the completed burger.
- Control software must coordinate and dispatch commands to the robots.
- The restaurant currently only has one robot for each task (referenced from *Software Design Challenge* document).

CONSTRAINT

Multiple orders may arrive in a similar timeframe and that some tasks may be blocked while others are not, due to robot availability (referenced from *Software Design Challenge* document).



SIMPLIFICATION

Since only two use cases are considered, which is outlined to the top left (ordering and food prep), we can make the assumption that customers submit orders online only and receive their order through a drive-up window (referenced from *Interview Follow Up* email from Kanyon Edvall).



DESIGN

USE CASES

- **Place Order**
- **Modify Order**
- **Cancel Order**
- **Check Order Status**
- **Expedite Order**
- **Process Order**

USE CASES

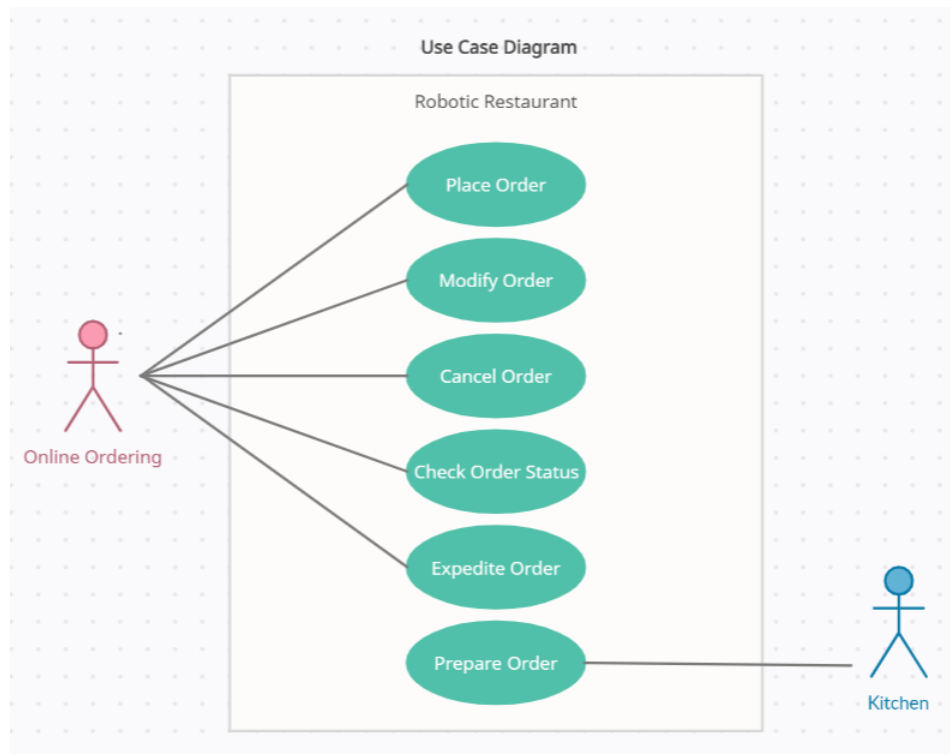
Here are the top use cases of the Robotic Restaurant.

- **Place order:** Add a new order in the system for a customer
- **Modify order:** Modify an order if the original order isn't started yet
- **Cancel order:** Cancel an order if it isn't started yet
- **Check order status:** Display the order status for the customer
- **Expedite order:** Customer can pay a certain fee to prioritize their order so that they may get their order faster if the order is not finished yet
- **Process order:** Retrieve meal items from an order and prepare the meals on the order according to a list of steps by assigning tasks to associated robots. Notification for pick-up is sent when the order is finished.



ACTORS

- **Online Ordering (Customer Application Interface)**
- **Kitchen (Robots Controller)**



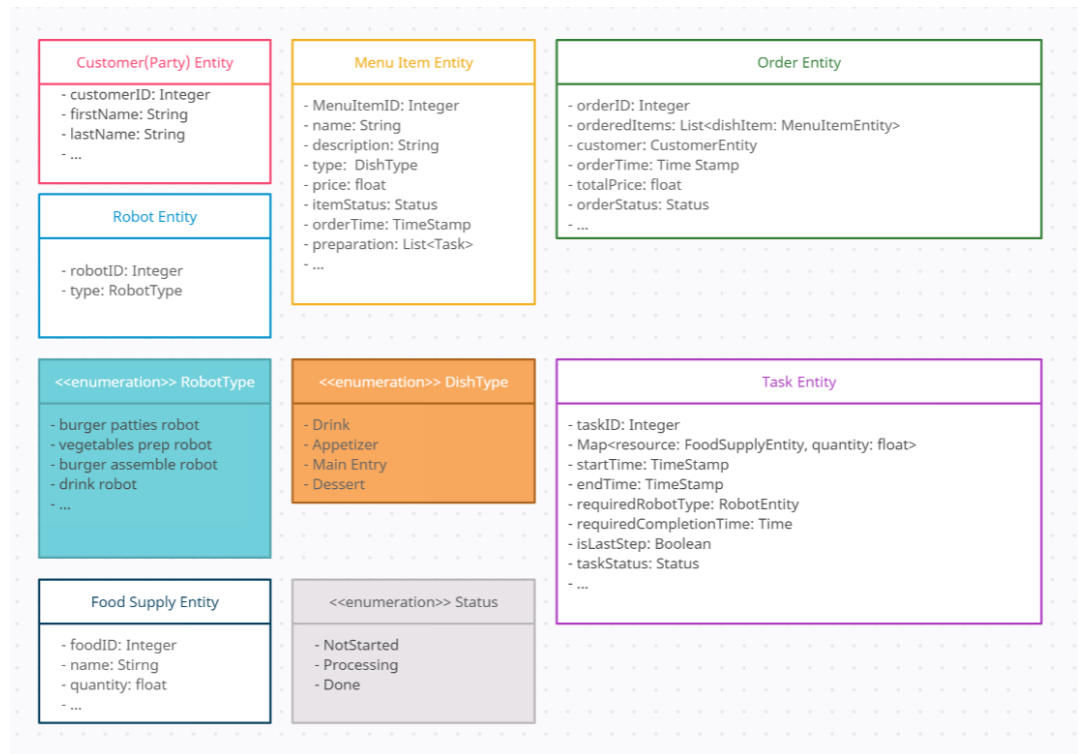
ENTITY

Here is the basic entity that will be used in the software's database. An entity is an object that exists, but it is different from the class itself. This is listed for the data consideration and can be used to establish the database.

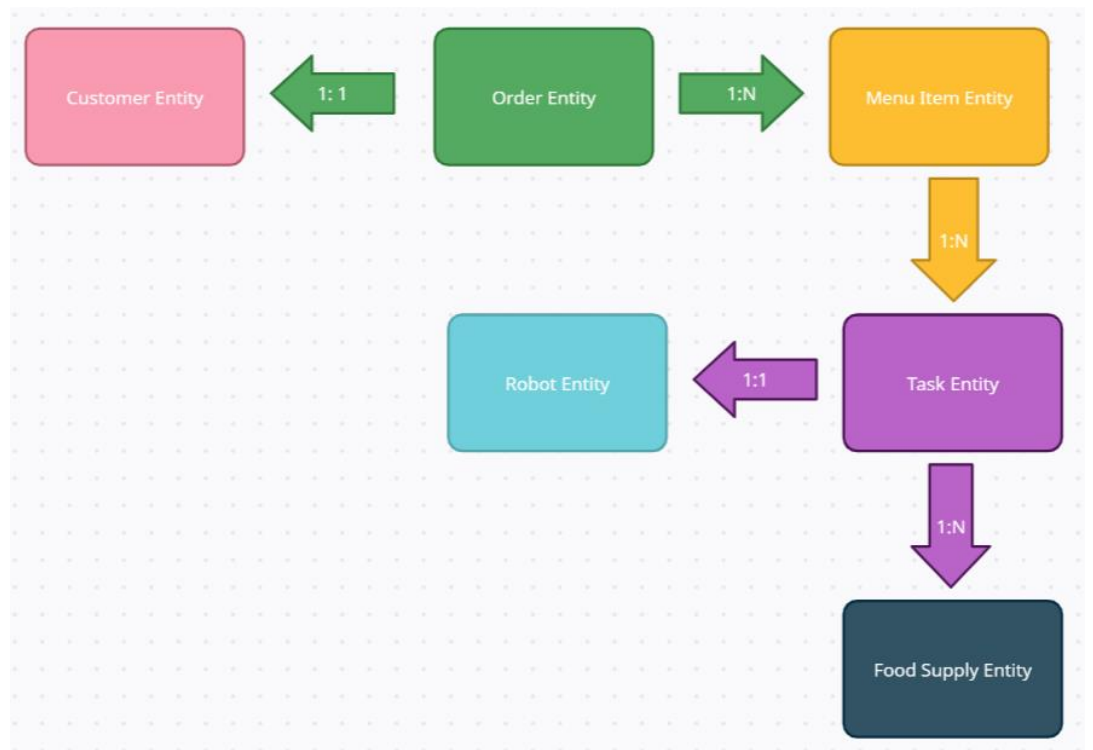


ENTITY

- Customer
- Robot
- Order
- Menu Item
- Task
- Food Supply

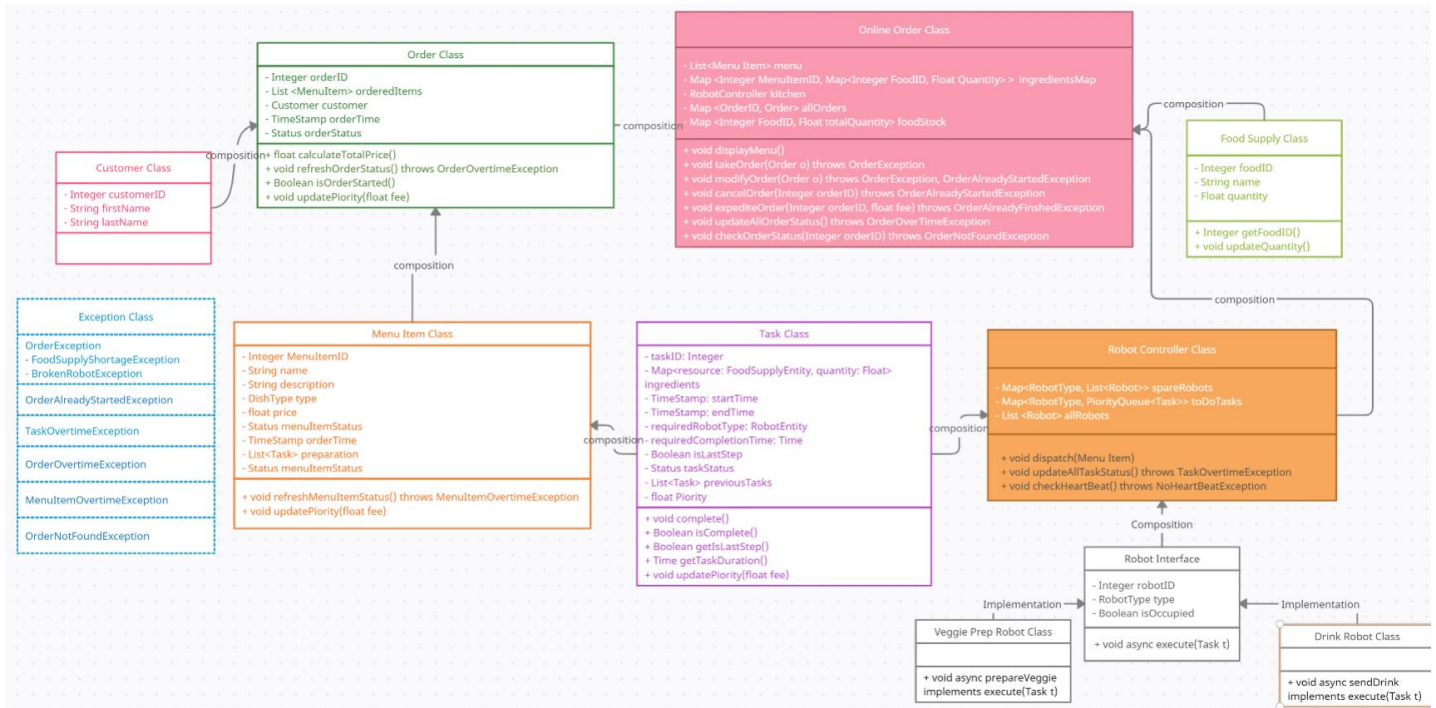


The relationship between the Entity is listed here and this relationship is also consistent in the class diagram in the next section.



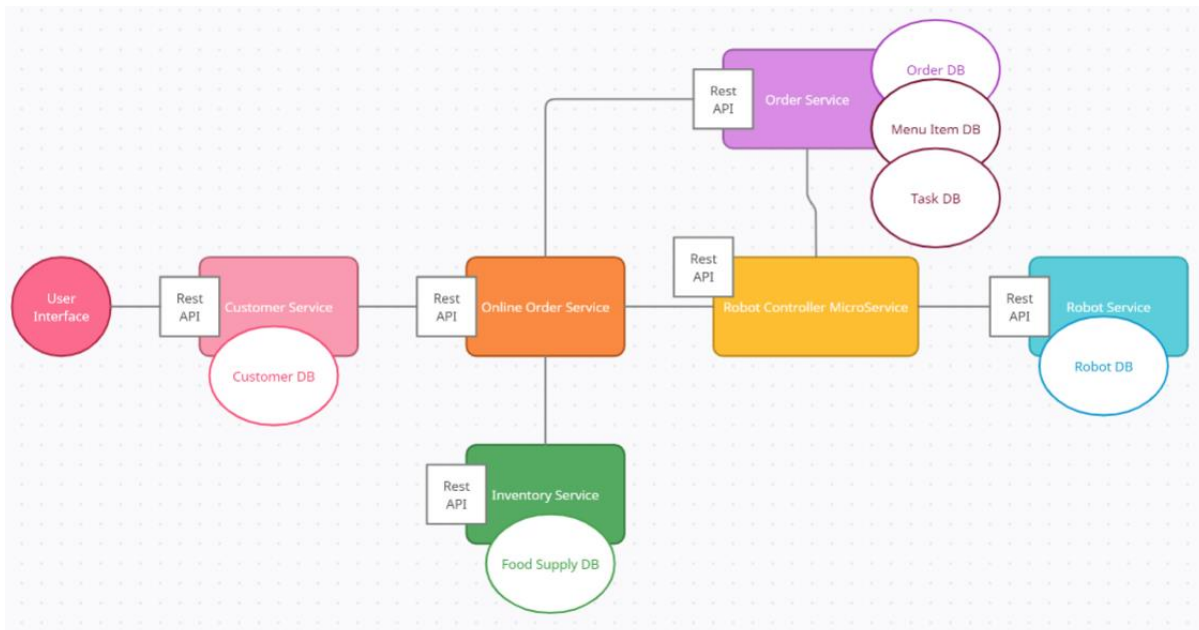
The relationship between task entity and robot entity is one-to-one because currently there is only one robot available for each type of task. This relationship may become one-to-many, many-to-one, or many-to-many as the robot are able to perform multiple tasks or if more robots are added.

CLASS DIAGRAM

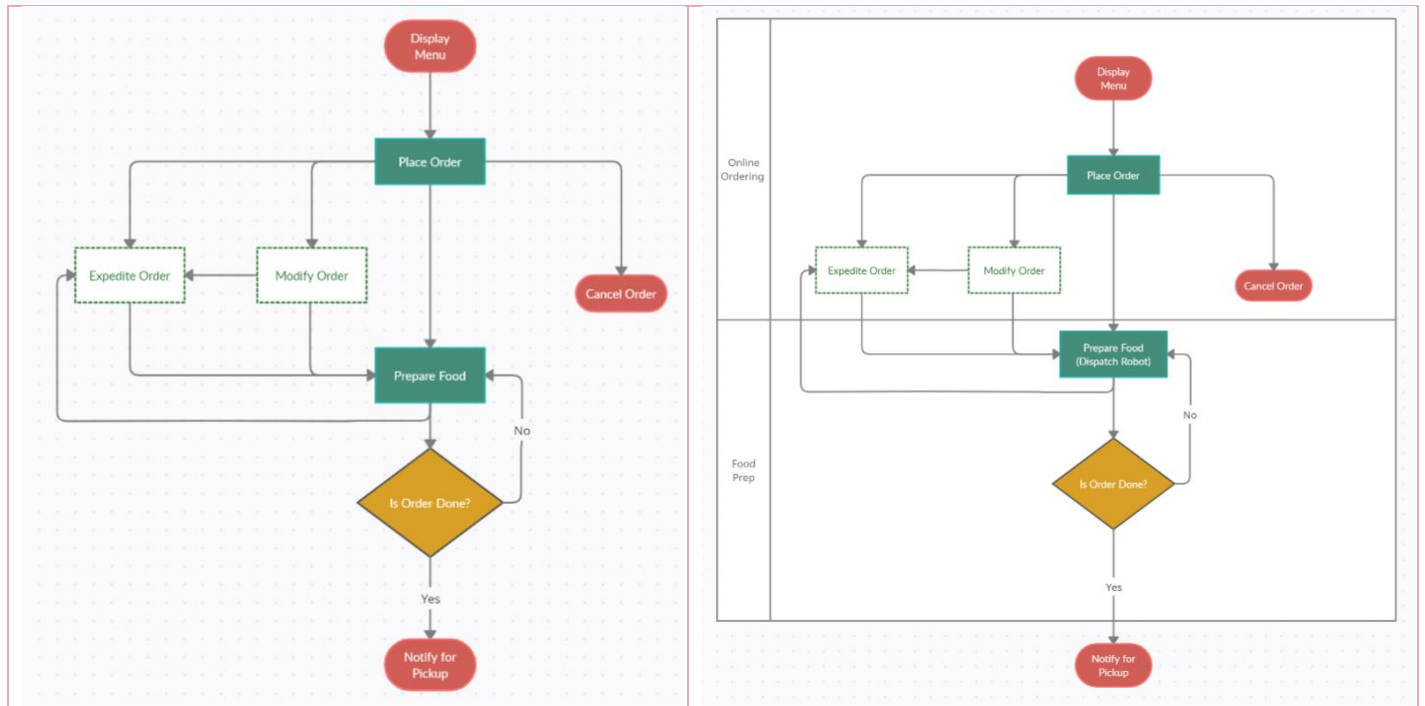


The enumeration type is already listed in the entity. For readability, the enumeration and the connection of Exception classes to the other classes are omitted in the class diagram above.

MICROSERVICE DIAGRAM



FLOW CHART



API OUTLINE

For the Ordering Interface, we can derive it from the Online Order Class by using Facet Design Pattern, which simply means to obtain a smaller interface that provides less authority by providing only a subset of the methods. Therefore, the functionality a customer has access to on the ordering page would be the following:

- ✚ Void displayMenu()
- ✚ Void takeOrder(Order o) throws OrderException
- ✚ Void modifyOrder(Order o) throws OrderException, OrderAlreadyStartedException
- ✚ Void cancelOrder(Integer orderID) throws OrderAlreadyStartedException
- ✚ Void checkOrderStatus(Integer orderID) throws OrderNotFoundException
- ✚ Void expediteOrder(Integer orderID, float fee) throws OrderAlreadyFinishedException

We will be using Rest API and JSON format to translate objects.

Void displayMenu()

- GET /v1/online-ordering/menu
- HTTP response status = 200 OK
- Body(JSON) MenuItem

Void takeOrder(Order o) throws OrderException

- POST /v1/online-ordering/place
- HTTP response status = 201 Created for success
- Request Body(Json) orderEntity
- HTTP response status = 404 Not Found if orderException is thrown

Void modifyOrder(Order o) throws OrderException, OrderAlreadyStartedException

- PUT /v1/online-ordering/<orderID>/modify
- HTTP response status = 200 OK for success
- Request Body(Json) orderEntity
- HTTP response status = 405 Method Not Allowed if Exception is thrown

Void cancelOrder(Integer orderID) throws OrderAlreadyStartedException


- DELETE /v1/online-ordering/<orderID>/cancel
- HTTP response status = 204 No Content for success
- HTTP response status = 405 Method Not Allowed if Exception is thrown

Void checkOrderStatus(Integer orderID) throws OrderNotFoundException

- GET /v1/online-ordering/<orderID>
- HTTP response status = 200 OK for success
- HTTP response status = 404 Not Found if Exception is thrown
- Return Response Body(Json) orderStatus

Void expediteOrder(Integer orderID, float fee) throws OrderAlreadyFinishedException

- POST /v1/online-ordering/<orderID>
- HTTP response status = 200 OK for success
- HTTP response status = 405 Method Not Allowed if Exception is thrown



WHAT'S NEXT

Reliability

The System should continue to work correctly even in the face of adversity, especially for a store run by robots. The system should be able to anticipate faults and cope with them.

For hardware faults, we can add redundancy to the individual hardware components to reduce the failure rate of the system, such as having dual power supplies and hot-swappable CPUs.

As for software failures, we can cope with by carefully evaluating assumptions and interactions in the system, extensive testing, process isolation, allowing processes to crash and restart, and monitoring system behavior in production.



USE CASES FOR THE FUTURE

- User Account
- Reserve a Pickup
- Payment Options
- ...

Scalability

As the system grows in data volume, traffic volume, or complexity, there should be an appropriate approach for dealing with the growth. Since robots are relatively expensive, we can evaluate the total workload of a robot (number of jobs * average time to complete a job) to decide what type of robot to add.

Maintainability

Over time, there may be different people working on the restaurant application on both engineering and operations to maintain current functionality and adding new use cases. We can adopt design patterns to improve the maintainability of the system since the costliest process for a software is not the initial development, but the long-term maintenance.