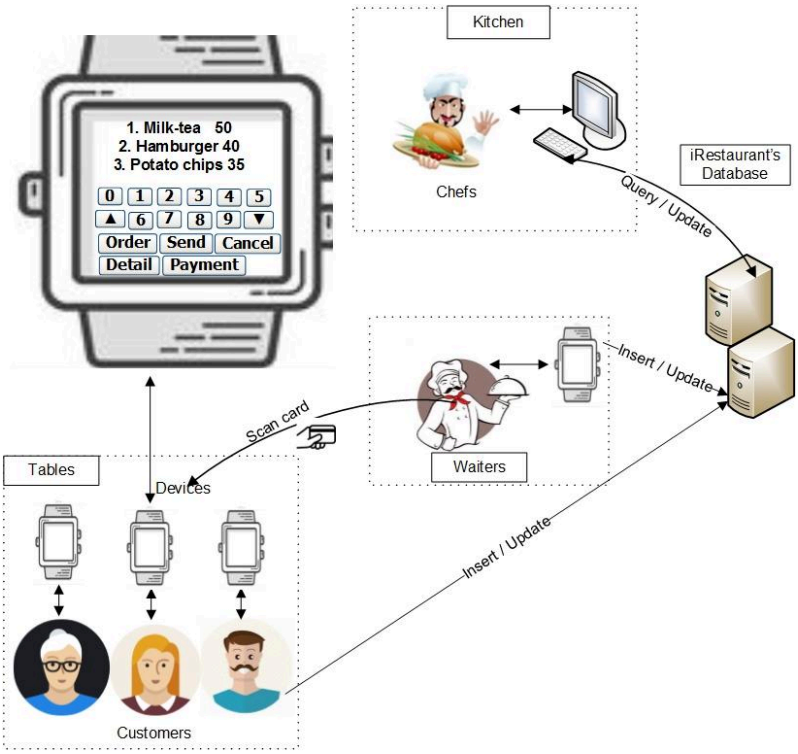
iRestaurant Case Revisited

The figure to the right depicts the overall architecture of the iRestaurant system, which consists of a server and two wearable devices. The server allows the restaurant chef to manage all meal orders placed by the clients. This module, being run on a computer in the restaurant kitchen, serves the two wearable devices described further below. The clients may place their order using a wearable that is of mobility, i.e., can be moved around or rigidly mounted on the table where they are seated. This food-ordering device manages a list of food and drink items. The client should be able to browse/view them and later add them to their meal order. Each food item is described with a name, a price, and a short textual description. The clients get a summary of their order, including the total amount of the meal and an estimated period of time they have to wait for their meal. Once the client confirms their order, this wearable communicates with the chef's module to inform the restaurant chef of the newly-placed order. The second wearable of iRestaurant is designed to serve restaurant waiters. A restaurant waiter is supposed to use his wearable to communicate with the system twice: when the client's order is ready for delivery and when he has delivered it. The clients may pay for their order using their credit card or in cash upon meal delivery. In addition to the LCD used for displaying menus, the two wearables come with controls such as buttons and lighting for showing the status of the client's order (e.g., confirmed, cooking, delivering, delivered).



Here are the analysis classes proposed for iRestaurant: Menu, Menu Item, Order, Customer, Waiter.

Observable states of the waiter’s wearable in the analysis model: Ready (Idle), Delivering, Delivered

Your OO design model

Turn the analysis classes listed above into design classes. This mapping is not necessarily one-to-one, e.g., an analysis class might be turned into multiple design classes. Identify class relationships and introduce additional classes that handle storage, user interfaces, network connections, etc. Specify the data types of the class attributes. Point out the parameters of the class methods. Draw a UML class diagram.

Detail the state machine of the waiter’s device. The operations represented for its states should match those represented in the class diagram. Draw a statechart diagram in UML.

Draw a UML activity diagram.