A Summary of the Discussion at the Planning Meeting for Sprint 6

Overview of the Sprint

KitchIntel is now starting to create the distributed version of the system. The sales and marketing team is in the process of trying to sell an early release of the system to a university dining facility. Hence, as part of this spring the team needs to create a proof of concept demonstration.

Overview of My Commitments

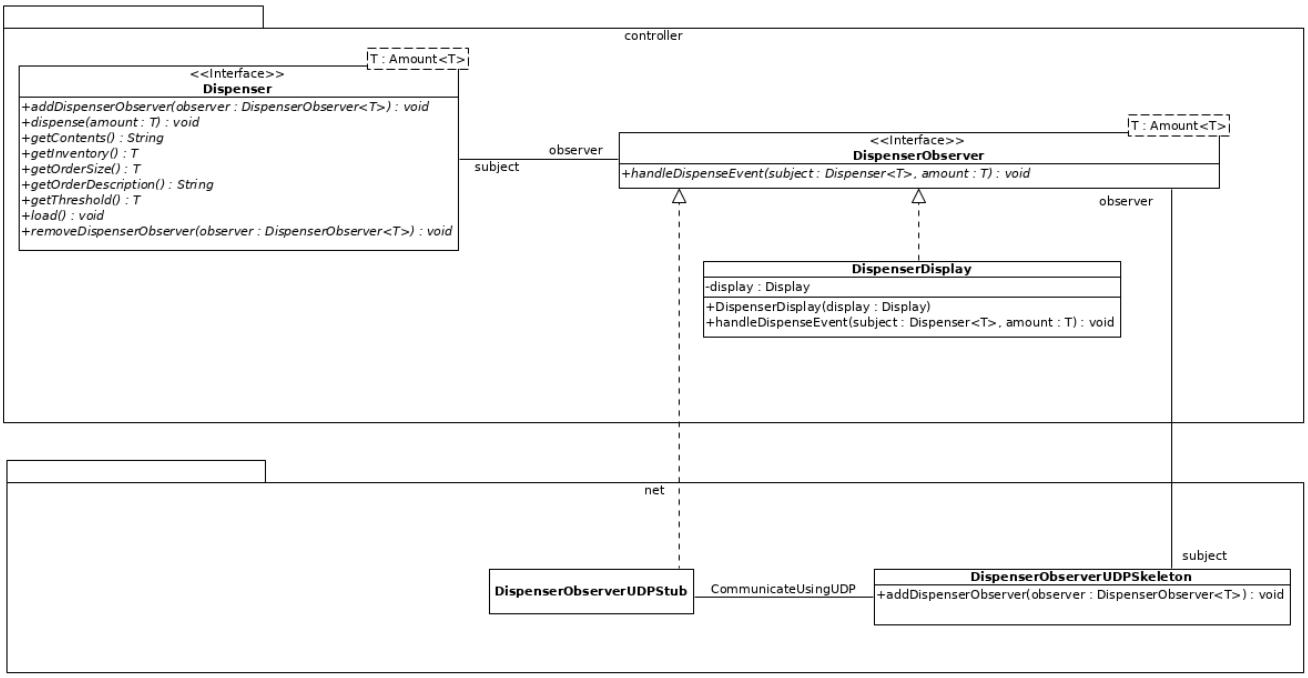
I have agreed to add the ability to have a remote device receive information about “dispense events”. The messages need not be received reliably, since this is only intended to give some indication of activity. Hence, we have agreed that UDP seems appropriate. The remote device must be able to receive messages from multiple dispensers, but they can be "queued" (i.e., they needn't be handled simultaneously). The design we agreed upon is summarized below.

I have also agreed to add the ability to have the `Store` run on a remote device. In this case, all messages between the `Store` and the `Restocker` (in both directions) must be transmitted and received, so we agreed to use TCP. It was pointed out that **the connection must not be maintained while the order is being "processed"** since, when the system is actually deployed, the "processing" could take multiple days.

I have also agreed to create a proof of concept demonstration for these portions of the system. The specifications for this demo are summarized below.

The Design of the System

The team agreed to the following design for the portion of the system that deals with “dispense events”. The essence of the design is to use the Proxy Pattern. This involves creating a `DispenserObserverUDPStub` that implements the `DispenserObserver` interface. It’s `handleDispenseEvent()` method sends a `String` representation of the relevant information to a `DispenserObersverUDPSkeleton` on the remote machine, that forwards it to an actual `DispenserObserver` running on the remote machine. This is illustrated in the following UML class diagram.



This should enable us to use both local and remote `DispenserObserver` objects without making changes to the existing code.

Specifications for My Portion of the Demonstration

For the demo, we agreed that the dining facility must have:

- One machine with `Canister` containing Cheerios.
- One machine with a `Canister` containing Raisin Brans.
- One machine with a `Canister` containing Cap’n Crunch.
- One machine with a `BreadBox` containing wheat bread.
- One machine with a `BreadBox` containing white bread.

Each of these machines must also have it’s own `DispenserDisplay` and `Restocker`. In addition, each of these machines must have it’s own `DispenserObserverUDPStub`.

There must also be:

- One machine running a `DispenserObserverUDPSkeleton` (and associated `DispenserDisplay`). This machine will be used by the manager of the dining facility to roughly track the activity in the facility.
- One machine running a `Store`.

So, in total, the demo will involve seven different machines. The five machines running `Dispenser` objects will all communicate with one remote machine using UDP and with one remote machine using TCP.