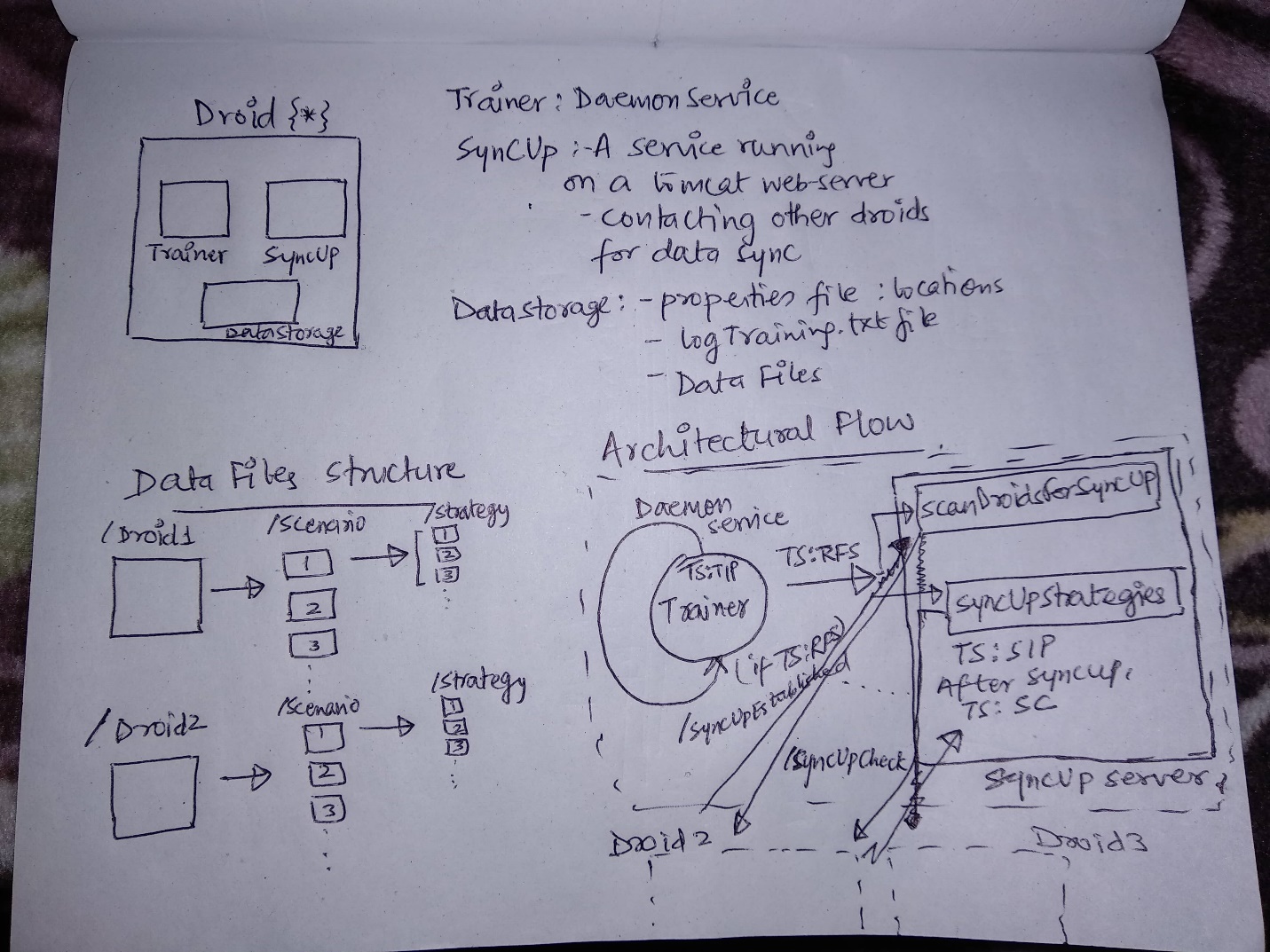
Hi Sifo-Dyas,

I am Ayush Kumar, one of the engineers here at Planet Earth. Prime Minister Lama Su has given me the responsibility of looking into the the droid training simulation that you stumbled upon. Please find my understanding of the simulation along with a sample application representing the design flow of the data.



A Droid consists of 3 components 🡪 1. Trainer

2. SyncUpService running on a Tomcat server

3. DataStorage/DataCentre

Trainer : A daemon service with responsibilities –

1. Constantly scanning the datastorage before and after training
2. Recording the changes in logTraining.txt file.
3. Initiating the scan for sync up through the SyncUpSevice.

A Droid at any time, can be in any of the below training status(TS):

TIP: TrainingInProgress , RFS: ReadyForSyncUp, SIP: SyncUpInProgress, SC: SyncUpCompleted

SyncUpService:

1. This is a Spring MVC framework based service running on Apache Tomcat 7 server
2. Responsibilities include:
3. Scanning all other droids to check if they are ready for syncup(RFS) or sync up completed (SC) status for other droids. If a contacted droid is in any of the above status, then the contacted droid changes its status to SyncUpInProgress and sends the JSON request

along with the multipart files.

1. Once SyncUpEstablished, it reads the incoming trainingLog file and makes changes accordingly to its data center.

Assumption: All Droids have the same Data Strage structure

i.e. Droid1/Scenario1/strategy1.strategy….

c. Once the syncUp is completed, the trainingstatus is changed to SyncUpCompleted and the droid is ready for any other droid to contact and syncup.

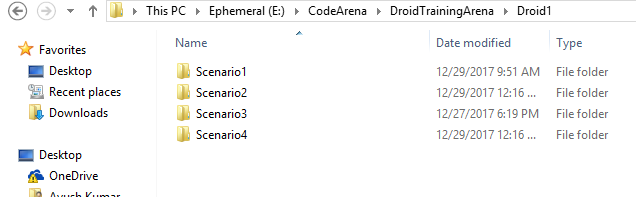
DataStorage:

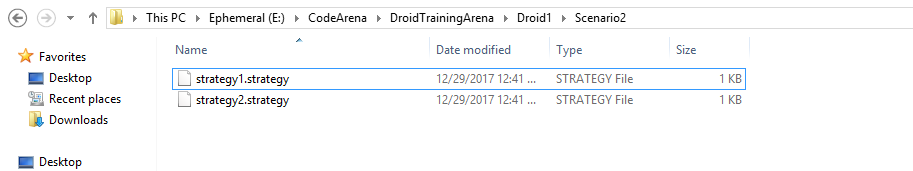
1. Under the Droid name parent folder, the droids have several scenario based folders and each folder consists of several .strategy files.
2. All Droids maintain the same structure.
3. Whenever a new scenario is encountered, a new folder is added under the parent and then the .strategy learnt are added under that scenario.
4. New strategies can also be added/updated under an existing scenario.

I have developed a sample application to simulate the way the droids are behaving: <https://github.com/ayushkumar-1/DroidTrainingSimulation>

**Assumptions:**

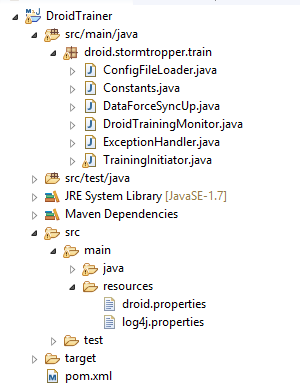
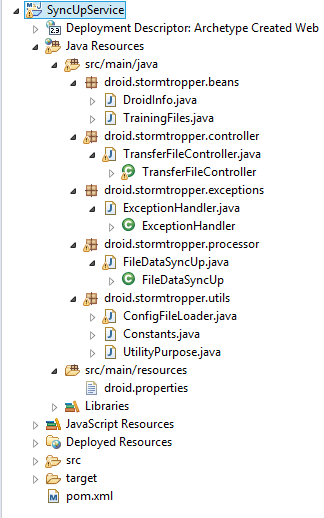
1. All droids have the same data storage structure as shown below:





1. An individual droid cannot do training while doing syncup. They are sequential processes instead of parallel.
2. Complete process (training and sync up) is automatic and would run indefinitely, till the servers support the applications, if not disturbed by external interference. I have used a check for a “training.stop” file which would stop the trainer daemon service.

**Application Structure**

**Design Flow Description:**

1. A Droid switched on for first time starts with trainingstatus as “trainingNotStarted”.
2. DroidTrainer reads the initial file structure of the dataStorage, using timestamp as parameter to check for changes.
3. After Training, The files are checked again and the changes(new scenario, new strategy, update strategy, delete scenario, delete strategy) is checked and published in the logTrainings.txt file.

This file will be used by target droids to understand the exact changes they need to do at their end.

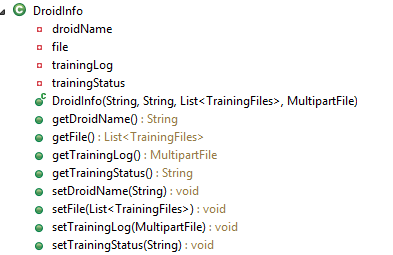
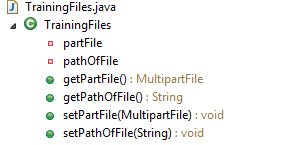
1. After the training, the status is put as “ReadyForSyncUp”

And a HTTP Call is made to the service SyncUpService over the server for mapping: /searchForSyncUp

1. The Trainer has a connection time out of 1 sec, after which it goes back to its infinite loop, checking for next training schedule.
2. The SyncUpService receives the request from Trainer and initiates a “/syncUpCheck” to the other droids one by one. This can be done using multi-threading, but here for simplicity only one is considered.
3. If the droid receives a response with droidStatus as “readyForSyncUp” 🡪 the service is redirected to “/syncUpEstablished”.

The point to be noted is that the contacted droid does the same and after sending the response for “readyForSyncUp”, it changes its status to “syncUpInProgress”. The same is done here by the requestor droid.

1. The droids receive a POST request with the DroidInfo:

1. The File is read using the trainingLog file and the data is synced up using the below logic:
2. After reading the triningLog file, the files along with the paths are taken as a list.
3. As, the data storage has same structure for all droids. Hence, that can be used to reach to the file in the requestor droid.

**e.g. logTraining.txt reads 🡪 Scenario2\startegy1.strategy:updated (from Droid2)**

So, Requestor (Droid1)🡪 should have Scenario2\startegy1.strategy

So, the new data is appended to existing file.

**e.g. logTraining.txt reads 🡪 Scenario2\startegy1.strategy:newStrategy (from Droid2)**

So, Requestor (Droid1)🡪checks for Scenario2\startegy1.strategy

If not present, then creates the file with the content.

**e.g. logTraining.txt reads 🡪**

**Scenario2:newScenario(from Droid2)**

**Scenario2\startegy1.strategy:newStrategy (from Droid2)**

So, Requestor (Droid1)🡪

Checks for Scenario2 🡪 creates if not exists

checks for Scenario2\startegy1.strategy

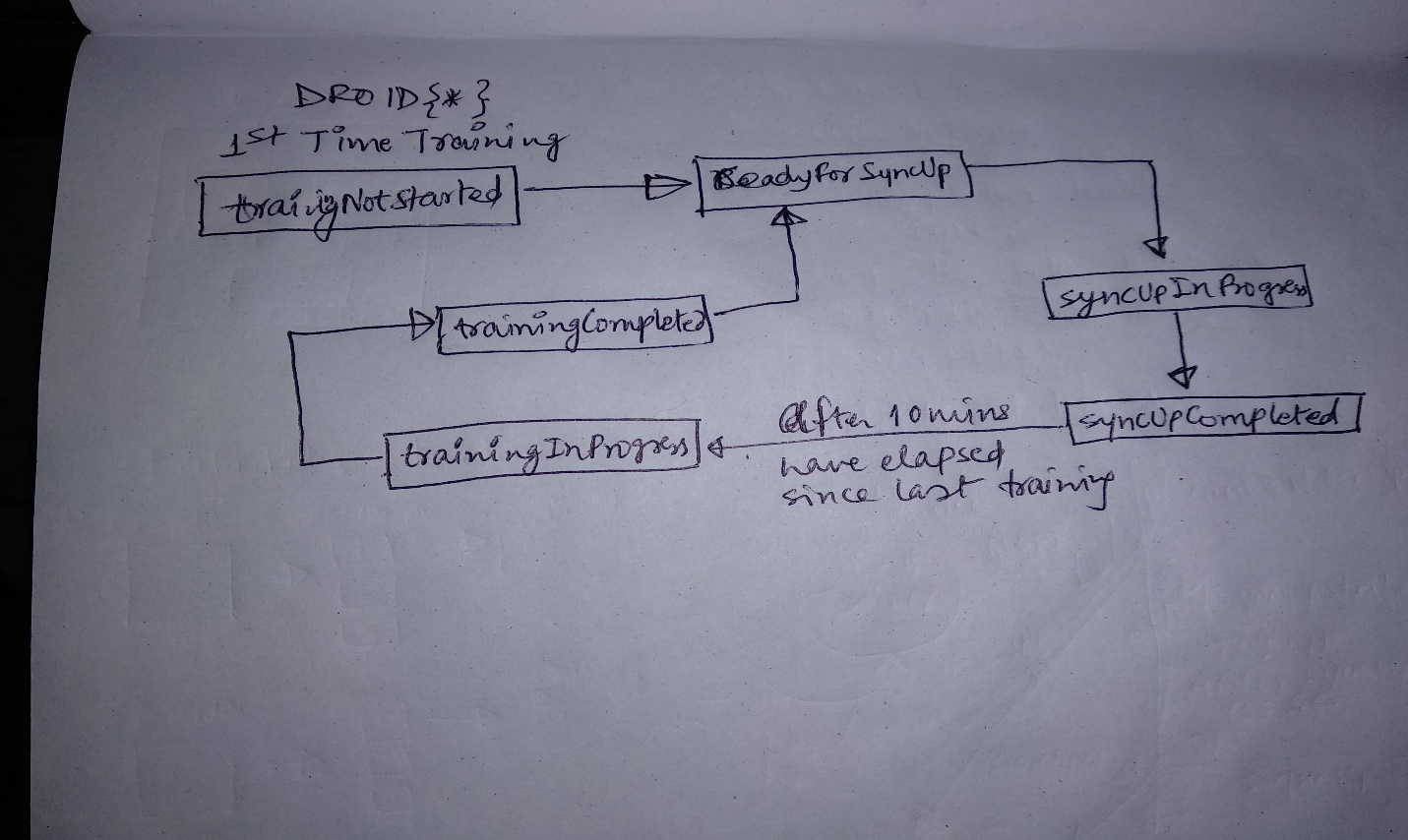
If not present, then creates the file. Now this is treated as an updated existing file and using same method data is appended in the file.

**e.g. LogTraining.txt reads 🡪 Scenario2\startegy1.strategy:deleted**

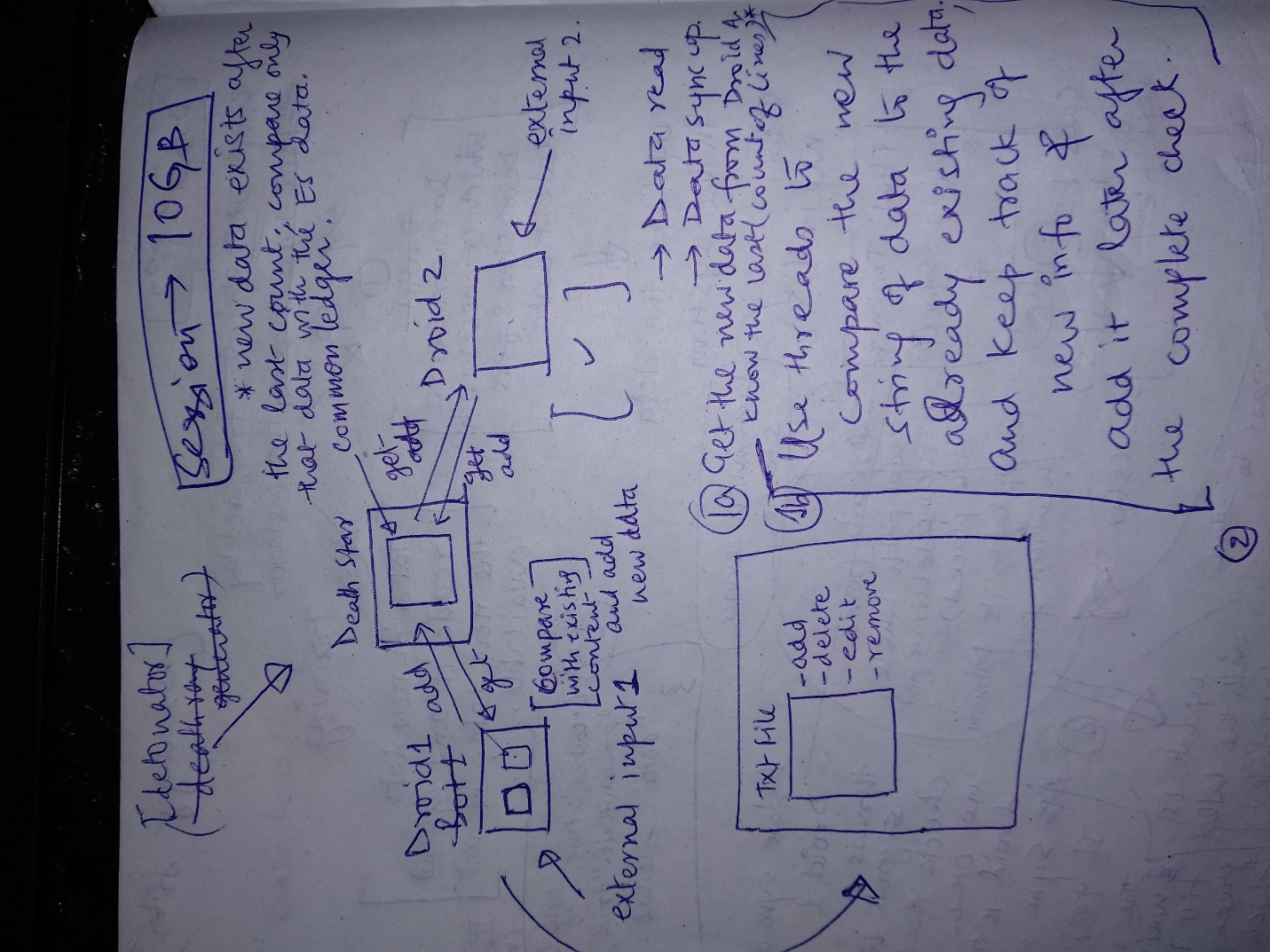
So, Requestor (Droid1) 🡪 delete the strategy if it exists here.

1. Once the syncup is completed, the status for droid is changed to “syncUpCompleted”.
2. Now the trainer can resend the droid for training provided 10 mins(assumed rest duration) since last completed training.
3. The process keeps repeating itself.

The Flow of DroidStatus is shown below:



I also thought about few other scenarios, which seemed must more coherent as far as design is concerned, but as per requirement, I have tried not to include the centralized server (**Death Star**). Yet, here is a rough diagram of what I was thinking:



That’s all from my research on the droid simulation. Please let me know in case of any further clarifications or discussion over the Death Star architecture.

Long Live and Prosper,

Jedi Ayush Kumar