CSC 555 - Social Computing S-d

Introduction

I had already implemented a simulation of the ringer manager application in S-b and S-c. I have now updated my application following the specifications stated here. The service simulator is described here. The application is coded in `Node.js` and the output can be seen over the command line. I will document the steps to run my program as well as an explanation (a detailed flow can be seen along the comments in my code) of different parts of the same.

Instructions to run my program

1. Install `Node.js`.
2. Run `npm install` at the root directory (`src`) to install any required package (`sync-request`) that my application uses.
3. Run `node main.js` on the command line to execute my application.
4. The program will print out important simulation steps and metrics on the console. Particularly, it will show response options with scores, response decision made, feedback for calls at the place that I'm in.
5. The code runs continuously with a `setInterval` callback method set to execute `simulateVisits` and `sendFeedback` every 45 seconds.
6. If you want to close the program, just press `CTRL + C` on the command line.
7. The requirement, of course, is that the server at http://yangtze.csc.ncsu.edu:9090/csc555sd/services.jsp should be up and running.

Workflow of my program

1. It parses relationships from **relationships.txt**. This file contains known relationships (present in the service page) and their corresponding relationship types and relationship IDs. The neighbours as well as the callers will affect the decision of the response and the feedback based on their relationship types.
2. Then, it parses known places from **places.txt** which also contains the noise levels shown in the service root page. The noise levels also contribute in the social benefit function.
3. Next, it parses the relationship types from **relationshipTypes.txt** which contains the weight for each relationship type. This will be later used in my social benefit function. Previously, this contribution factor was multiplied with a separate weight which changed dynamically over the lifetime of the application. Now, it is just one value which is initialized to different numbers based on the `relationshipType`.
4. It starts simulating visits in a timed interval of 45 seeconds.
   1. It goes through all places available from the **places.txt** file.
   2. It enters that place through the web service call.
   3. It lists all the neighbors there.
   4. It requests a call in that location.
   5. Calculates the social benefit fucntion based on the neighbors, the call and the place.
   6. Provides the response and gets back a feedback.
   7. Accommodates the feedback into the contribution factors for each relationship.
   8. Exits the place.

5. It starts sending feedbacks in the same timed interval of 45 seconds.
   1. Enters a place.

2. It starts to give feedback to calls in that place as a neighbor.
3. Lists all the current calls in that place.
4. Works with the list and finds appropriate feedback and updated feedback for all the calls.
5. Posts all the feedbacks to the API.
6. Exits the place.

Social benefit function

This function has been completely been implemented along with comments in the `main.js` file. For a detailed workflow, please refer to that file. To give a brief overview of what it does:

1. Iterate over all neighbors.
2. If the neighbor is not present in the relationship ID map from before (this means this neighbor was not listed as a known neighbor in the services page), I insert this neighbor in the map with a relationship type of `stranger`. A stranger starts with a weight of `0.2`, colleague starts with `0.6`, friend starts with `0.4` and family starts with `1.0`.
3. Add the weight component for that relationship to the expected ringer mode property. Thus initially, family contributes more than stranger to deciding the response type.
4. Add a component to the ringer mode property based on the place as well, depending upon the noise level of the place. If the noise level is from 0 to 2, then the `silent` ringer property gets some points added, 3 to 7 adds to `vibrate` and 8 to 10 adds to `loud`.
5. Find the maximum among the three ringer mode properties. If it is `silent` or `vibrate`, add a component for `vibrate` and `loud` ringer modes respectively based on the urgency of the caller. If the ringer mode chosen is loud, then no need to change it.
6. Find the maximum again and return that response.
7. Basically, in this way every neighbor and caller and place votes with specific weights to the decision of the social ringer manager. And ultimately the highest votes among the three ringer modes wins.

Feedback for weights

Feedback is taken into account by incorporating the same into the weights of the relationships. The following things can happen:

1. If the feedback was positive and the final decision was the same as the expected ringer mode of that user, then nothing is done.
2. If the feedback was positive, but the final decision was not the same, then the weight is reduced a bit, because the user seems unpredictable.
3. If the feedback was negative, but the final decision was the same as the expected ringer mode of that user, then the weight reduced a little more than the last case, because in this case the user is more unpredictable.
4. If the feedback was negative, and the final decision was not the same, then the weight is increased a little bit, so that next time he/she contributes more towards deciding the final response. Basically, giving him a chance to factor into the decision more.
5. If the feedback was neutral, then the weight is reduced a little, because this guy might not have much effect into deciding any response.

Rationale

My implementation for the social benefit function is based on weighted majority and not just plain majority. So when I choose the maximum among the three ringer modes, I choose the maximum among the weghted scores built by all the neighbours around me. I have a `responseRationale` variable which I build as I calculate my social benefit function. It is almost built the same way, i.e. I keep making it, unaware of which response is actually going to win ultimately. When I make the final decision, I extract the rationales for that particular ringer mode. My rationales include:

1. Majority of expected modes. Note that this is not a plain majority but rather weighted majority, as I had mentioned before. That's how I had built my social benefit function. Atleastone predicates are not included as that is not how I calculate my social benefit function.
2. Caller urgency.
3. Place and noise level.

Feedbacks for other calls

I enter a place and get the list of current calls in that place. If there is a call for which I am the caller or callee, I ignore it. Otherwise, I take the call and simulate as if I had received the call. So I pass it through my social benefit function and see if my response would have been the same as what the callee's response was. If not, then my first feedback becomes negative, otherwise it's positive. If either the `ringermode` or the `rationale` is `null` then my first and second feedback are accordingly neutral. My second feedback comes into play when the callee's response and mine do not match. This is an interesting case, if not a little troublesome to give feedback for, because even though we have the rationale for the callee, we might not know what are the exact values that his/her social beenfit funciton depends on, which sways the decision of a response quite a lot. So I settled on one particulae case - if the rationale contains caller urgency, this means his/her relationship type weights are different than mine. Probably I have fixed a very large or very small value for a particular relationship type. And that had caused the difference between my response and theirs. So, I give a positive second feedback in that case.

Updating norms

This directly follows from the last section. If I get to the last condition that I had showcased, where I gave a positive second feedback, then I update my relationship type weights accordingly. If the callee's response was `silent` or `vibrate` and mine was `vibrate` or `loud` respectively, then I decrease my relationship type weights for that particular relationship type. If it was vice versa, then I increment the weights.

Conclusion

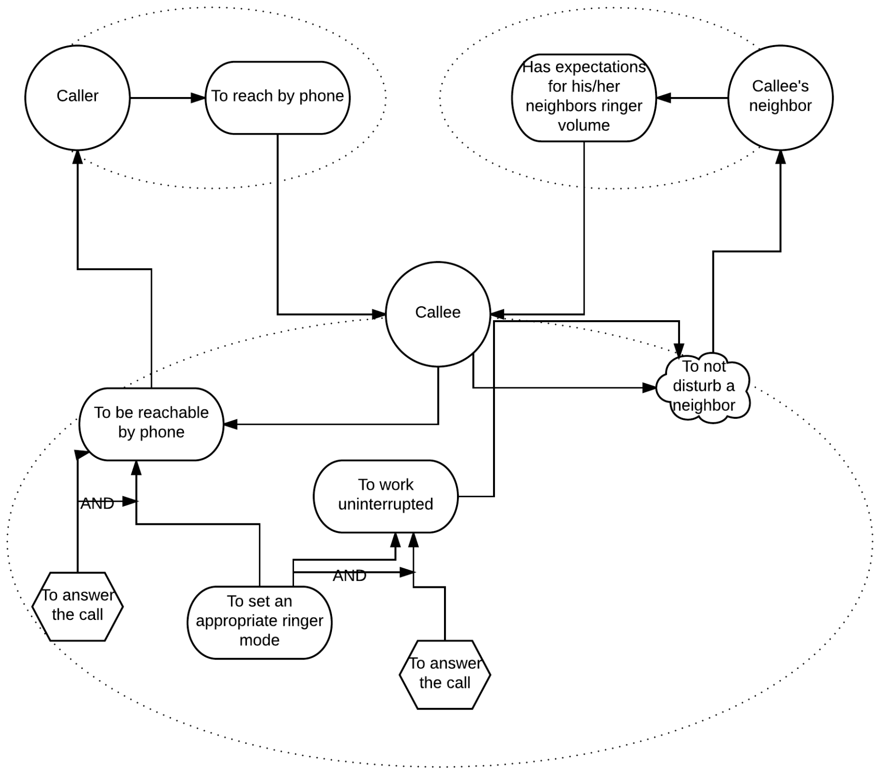
The overall description and workflow of my program and implementation is mentioned in the above parts as well described through the comments in my code. I will also give a snippet of my program output here:

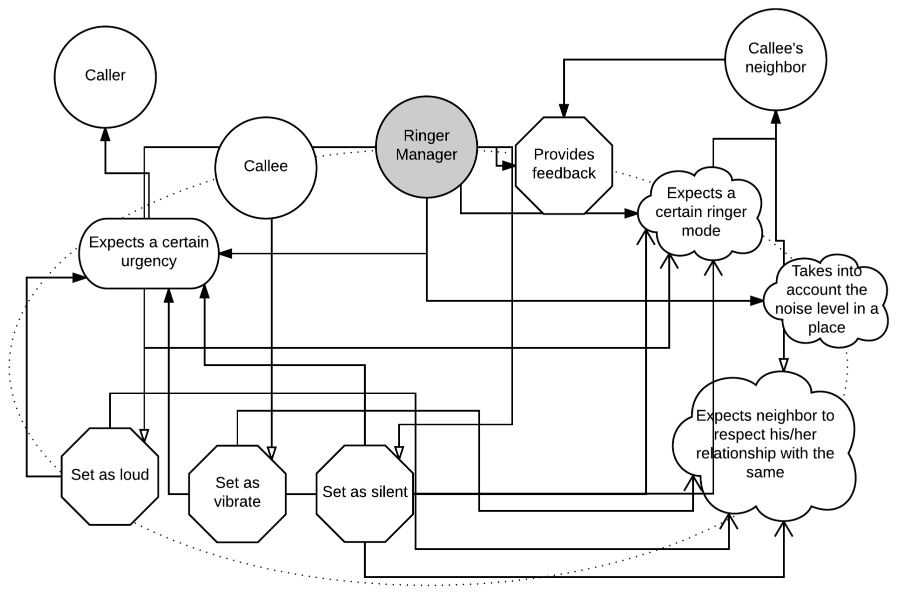
```
Entering hunt.
Getting all neighbors in hunt.
Got call from Faramir.
Options are: {"loud":0.6000000000000001,"silent":2.5,"vibrate":3.5999999999999996}
Ringer manager responded with mode vibrate.
Rationale is: ArgInFav(ringermode-IS-vibrate+WHEN+caller-relationship-IS-2+AND+call-reason-IS-
urgent).
Exiting hunt.

Entering eb2.
Getting all neighbors in eb2.
Got call from Orophin.
Options are: {"loud":2.2,"silent":1.2,"vibrate":1.7}
Ringer manager responded with mode loud.
Rationale is: ArgInFav(ringermode-IS-loud+WHEN+caller-relationship-IS-1+AND+call-reason-IS-urgent).
Exiting eb2.

Entering meeting.
```

```
Getting all neighbors in meeting.
Got call from Denethor.
Options are: {"loud":0.4,"silent":0.6,"vibrate":1.3}
Ringer manager responded with mode vibrate.
Rationale is: ArgInFav(ringermode-IS-vibrate+WHEN+Majority(expected_mode)-IS-vibrate+AND+place-IS-
meeting+AND+noise-IS-5).
Exiting meeting.

...

Entering lab for feedbacks.
Exiting lab for feedbacks.

Entering meeting for feedbacks.
Exiting meeting for feedbacks.

Entering party for feedbacks.
Exiting party for feedbacks.
```

As you can see, it entered `Hunt` and then got the list of neighbours, requested a call, exited the place and moved to the next one. The same holds for feedbacks as well, only getting a call being requested at the same time that I am going through the loop is incredibly difficult. I believe my code is correct and is in the right place, and based on any call events it will work correctly.

This forms my final project report. Below I have also included the `system-as-is` and `system-to-be` models, but they have remained the same as the last submission.




Caller
Callee
Ringer Manager
Provides feedback
Callee's neighbor
Expects a certain ringer mode
Expects a certain urgency
Takes into account the noise level in a place
Expects neighbor to respect his/her relationship with the same
Set as loud
Set as vibrate
Set as silent