Final Report: Friends of the Kiosk

Nathan Timmerman, Jiangnan Fu, Jing Zhang, Tianqi Cai, Yiqi Liu

What we wanted to accomplish

Our project hoped to accomplish creating a way in which frequent users of the Kiosk ("friends" of the Kiosk) can build up profiles of themselves that contains identificatory information and a variety of preferences that the Kiosk can then make suggestions to the users based upon. To this end, our current implementation is a sort of proof of concept that learns a user's major, collegiate level (undergraduate* or graduate), and food preferences through a simple linear conversation, then recommend a preference using SME.

Below is an example conversation:

KIOSK:	Hello. What is your major, Nathan?
USER:	Computer science.
KIOSK:	ComputerScience? Cool! Are you an undergraduate or graduate student?
USER:	Graduate student.
KIOSK:	GraduateStudent! What food or drink do you like, Nathan?
USER:	Apple pie.
KIOSK:	Who doesn't like ApplePie!? Would you like a recommendation?
USER:	Yes.
KIOSK:	I recommend CoconutMilk.

If the user does not want a recommendation, the last two lines could alternatively be:

USER:	No.
KIOSK:	Alright, Nathan! Thanks for chatting.

This conversation saves the following information into the knowledge base:

in BaseKB:
(isa (UserModelMtFn Nathan) Individual)
(isa (UserModelMtFn Nathan) Microtheory)
(modelMtOfUser (UserModelMtFn Nathan) Nathan)
in CyclistsMt:
(isa Nathan HumanCyclist)
in KioskFriendsDataMt:

```
        (isa (SocialModelMtFn Nathan) Microtheory)
in (SocialModelMtFn Nathan):
        (FoodInterest Nathan ApplePie)
        (isa Nathan GraduateStudent)
        (studentMajor Nathan ComputerScience)
in (UserModelMtFn Nathan):
        (learnedModelTopic (UserModelMtFn Nathan) Nathan)
```

*Note that the natural language parser is unable to identify “undergraduate” or “undergraduate student” as a noun. Rather, in both cases, it classifies “undergraduate” as an adjective. Therefore, our implementation is currently unable to accept undergraduate as an input despite the existence of the *UndergraduateStudent* type in the KB. There is no way currently available to classify undergraduate students by grade level (i.e., freshman, sophomore, etc.) instead. When selecting graduate student, the user must include the word “student,” or else the parser identifies the standalone response “graduate” as a verb. These are two restraints put on us by the current knowledge representation and parser, respectively.

As noted, our project implements a proof of concept form of our goal. In the future, it could be expanded to handle a wider variety of conditions such as incorrect types, a user wanting to input multiple preferences at once, and a wider variety of preferences than food and drinks.

Our biggest technical risks

Risk #1
A user not cooperating with the Kiosk and entering irrelevant information.
Response #1
Currently, our implementation is a proof of concept and thus does not have built-in handlers for incorrect types of answers to specific questions. For example, if as a food preference the user answers “Pirate ship,” our methods will not handle this improper response and will cease conversation. For each successive question, the answers must be parsable as types *FieldOfStudy*, *StudentByEducationalOrganizationLevel*, *DefaultDisjointEdibleStuffType*, and either *(NLResponseFn Affirmative-NLResponse)* or *(NLResponseFn Negative-NLResponse)*, respectively.

Risk #2
Writing appropriate HTN plans that will carry out the conversation we want to implement.
Response #2
We “hijacked” the base Kiosk HTN plans at *answerQuestionViaEEs*. This allowed us to inject our own questions at certain times in the conversation, as well as to make use of our narrative function to extract conversation-specific information from general language processing.

Risk #3
We do not have enough users while we need user data to make a recommendation.
Response #3
To successfully make a prediction in the current demo, we stored some example user profiles in KioskFriendsDataMt.krf.

How successful were we?

The project did not go as well as expected. A few reasons include:

1. The KB actually has a lot of powerful items that we can match, like "BiddingToMakeAPurchaseOrSale", but it's really hard to find them, as we have to know pretty much the exact terms before even knowing such a thing exists, especially when the search bar doesn't match the middle of terms, only the start. In the above example, if we try to search "MakeAPurchase", the entry would not appear (like it doesn't exist), and we have to search "BiddingTo" in order to get the entry.
2. Every time we change an already-existing default microtheory (such as Interaction-Manager-KioskMt), we have to reinstall Companions. All other actions don't seem to work, and we can't just delete this entire MT, so reinstalling seems like the only thing that worked. This took a lot of time as we tried different things.
3. Our implementation actually doesn't know what question it asked, and only assumes that the user is answering a question depending on the user's sentence (e.g. you can just name a food and skip major and grade). I think this is partially due to Kiosk's designed purpose (answering questions, not asking them), but I can think of some ways to mitigate this, given more time.

What we learned

1. We should've used Teams more. For projects like this, there's not a lot of available resources, and we should've just asked when we got stuck.
2. Implementing something like this is complicated, and stuff piles up quickly. It's important to organize our files and items better, or we'll get lost.
3. Currently the system requires a lot of manual work to register the "isa"s of all the items. Like you have to enter in the kb, manually, that "coconut milk" is a type of drink. By doing this project, we experienced the difficulty this brings first hand. So a system where users can "teach" it as conversations go on, or one that asks users questions "what is xx", might be a better choice for conversational interfaces.

How to run

1. Load the following krf files:
    a. KioskFriendsDataMt.krf
    b. friends.krf

2. In order to start, run command below in the command line:
   (achieve :receiver interaction-manager :content (startUserProfiles))
3. Answer the questions prompted in a similar manner as shown in the example conversation. Keep in mind the type requirements given in Risk #1 above.
4. In the Commands tab, run:
   (achieve :receiver interaction-manager :content (endUserProfiles))
   to end.