UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

EPL 343 SOFTWARE ENGINEERING

Design Document

Foody Recommendation Agent

Valentinos Pariza

Marios Pafitis

Demetris Shimitras

Leonidas Achilleos

Stephanos Pantziaros

<10.11.2019>

Table of Contents

REVISION CHART

Version	Primary Author(s)	Description of Version	Date Completed
Draft	Valentinos Pariza Marios Pafitis	Initial draft for separating the jobs giving some ideas and recommendations to the rest of the team members as Project Managers.	1/11/19
Preliminary	Valentinos Pariza Marios Pafitis Demetris Shimitras Leonidas Achilleos Stephanos Pantziaros	Research and analysis of all the parts from each member of the team. Creation of all the diagrams and comment on them in a very sufficient state. Corrections from the feedback of the second deliverable.	10/11/19
Final	Valentinos Pariza Marios Pafitis	Final review of the content. Corrections and improvements for certain parts. Final touches and furthermore analysis in some cases.	11/11/19

1. Introduction

1.1 Purpose

The Design Documentation targets the specification of the "Food Recommendation Agent" system's design. In particular, the document's purpose is to specify the architectural designments for the system along with the designation of its components that will be utilized. Moreover, the classes defined in the previous deliverable will be presented individually accompanied with the interactions they have with possibly other classes used for implementing the system. At last, the numeral functionalities that will be provided to the users are demonstrated in sequence and/or state diagrams with the possible interactions can be realized at every state.

1.2 Scope

The "Food Recommendation Agent" is a software system, will be providing services and additional functionalities to the existing system of Foody (the "Foody Online Orders" system). The aforementioned services targets to recommend specific-calculated foods to its customers by using their designated and/or possible preferences and constantly estimating and updating them. FRA is going to be used not only for processing of data purposes (estimating user's preferences) but also as a data storage of data related to the offered services. Reaching at the functional part of the software system, the system is going to be built upon five main operations:

1. Collection of data from Foody's database, fed into the Recommendation System.
2. Recommender System calculation of the recommendation attributes for each user and their storing in our database.
3. Everyday collection of substantial data of the users like their history order, rankings and other information taken from web pages, in which the users will show their preferences by setting liked or disliked items and creating groups of their preferred combinations of foods (so-called Journeys) being stored in our database.
4. Production of recommendation attributes through the Recommender System for the users of Foody, using the information gathered for them and store them in the Database.
5. The use of the recommendation attributes produced and stored in the Database to recommend targeted foods and drinks to users of Foody.

1.3 Definitions, Acronyms, and Abbreviations

- Foody Recommendation Agent (FRA): The system that we are implementing which is consisted of the Recommender System, Likes/Dislikes and Journeys features.
- Foody: A company in Cyprus in which you can order online from different restaurants,
- Recommender System: A system that is going to calculate the recommendation attributes of a user.
- Recommendation attributes: Values that define the preferences of a user.
- Likes: Foods that a user like.
- Dislikes: Foods that a user doesn't like.
- Journey: A collection of liked foods for a user.
- My Journeys: A collection of Journeys for a user.
- Food: Any kind of food or beverage. Products that Foody's restaurants sell.
- Item: Any sellable provided in Foody's website.
- Picture: An image of an item in the website.
- Database: A place to store the data calculated from our system.
- MVC: Model View Controller architecture pattern
- Repository: Repository architecture pattern
- Foody's Recommender System: The system that Foody is going to implement to make use of the outputs of the Foody Recommendation Agent.
- Foody's Database: The Database that Foody already uses.
- API: Application Programming Interface.

1.4 References

- Material.io for User Interface Design: https://material.io/
- Foody's Website: https://foody.com.cy/gr/2202?filters=1
- Amazon's Website: https://www.amazon.com/
- User Interface Design: https://en.wikipedia.org/wiki/User_interface_design
- Responsive User Interface: https://en.wikipedia.org/wiki/Responsive_web_design
- Use Case Diagrams: https://www.uml-diagrams.org/use-case-diagrams.html
- UML Diagrams: https://tallyfy.com/uml-diagram/
- Modelio: https://www.modelio.org/
- Project Libre: https://www.projectlibre.com/
- UML Language: https://en.wikipedia.org/wiki/Unified_Modeling_Language

- Software Engineering Ian Sommerville: https://www.academia.edu/31140292/Ian_Sommerville_Software_Engineering_9th_Edition_2011_1_
- Using UML: software engineering with objects and components: http://homepages.inf.ed.ac.uk/perdita/Book/index.html
- Class diagrams: https://en.wikipedia.org/wiki/Class_diagram
- Use Case Diagrams: https://en.wikipedia.org/wiki/Use_case_diagram
- MySQL Data Type Storage Requirements : https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html & https://www.techonthenet.com/mysql/datatypes.php
- ERDplus : https://erdplus.com/
- Software Lifecycle Processes: https://standards.ieee.org/standard/12207-2017.html
- System Interfaces: https://deseng.ryerson.ca/dokuwiki/design:system_interface?fbclid=IwAR3M2mEcuOhjhTSC2gJjJcVaPzkBB5RgHTwCoSosNiRAbJ3VEiuoUCrhhhg
- Software Requirements: https://belitsoft.com/blog/software-requirements-specification-document-example-international-standard?fbclid=IwAR2nFX0f5kFs6hB7EvZBi3DEb83fhqXTMv8kW78q8JxqADt7gZtINejeq8Y
- Software Engineering Ian Sommerville: https://www.academia.edu/31140292/Ian_Sommerville_Software_Engineering_9th_Edition_2011_1_
- Component Diagram: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/
- MVC Framework: https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm
- Repository Architecture Pattern: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design
- Sequence Diagrams: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/
- Modelio Diagrams: https://www.modeliosoft.com/en/resources/diagram-examples.html

2. Architecture

2.1 Major Design Decisions

For the development of our system, we are going to utilize the pattern of the Model-View-Controller (MVC), combined with the Repository architecture pattern.

The Foody Recommendation Agent it is split into two major Systems, the Journeys, Likes/ Dislikes System and the Recommender System. The Journeys, Likes/ Dislikes System interacts with the user through a User Interfaces. So, we decided to use the architecture pattern Model-View-Controller for this system, as it is going to provide us an abstraction level between the front-end and back-end functionality.

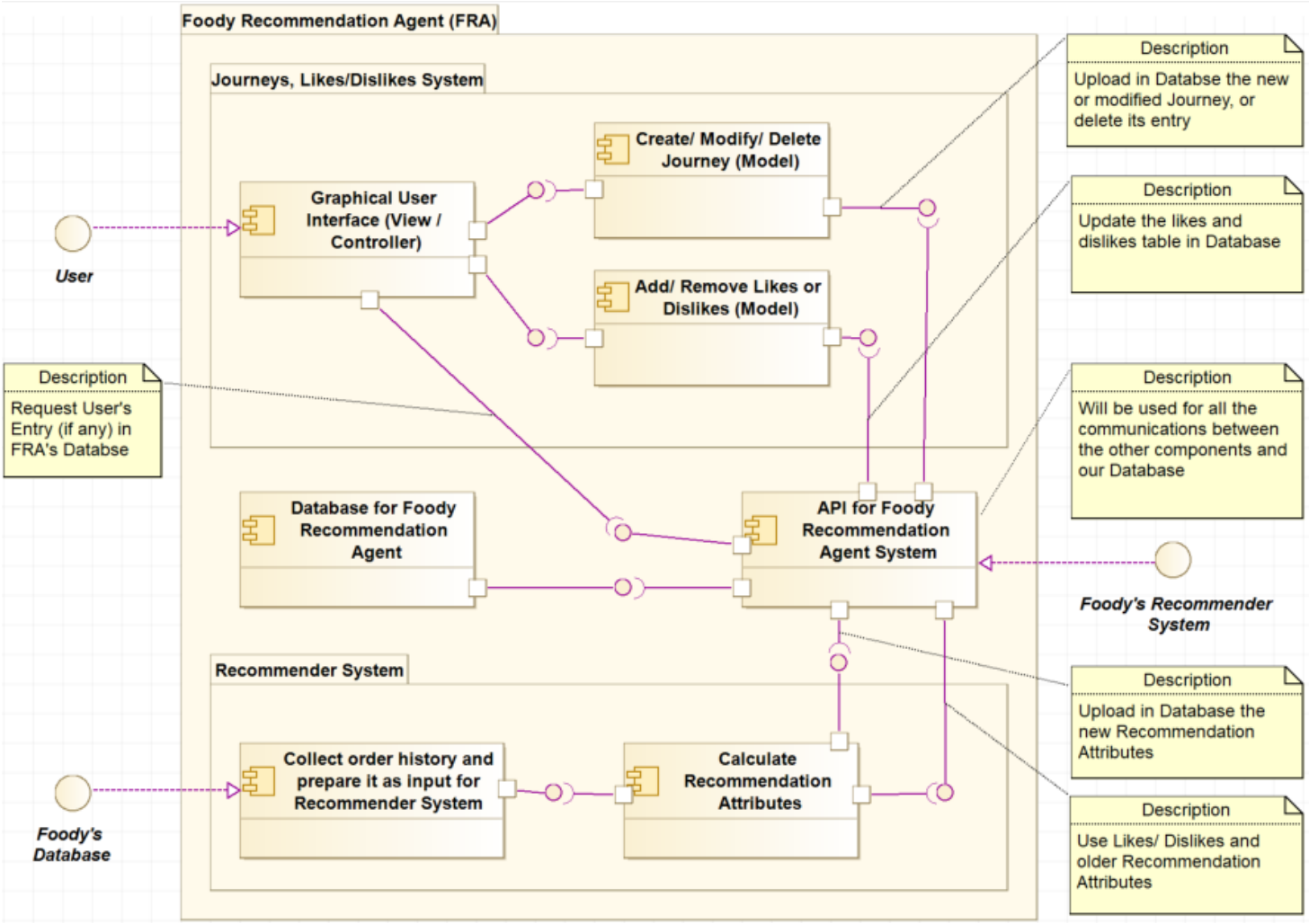
More specific, our system is going to be using lists of foods for the different food categories. The foods should be easily modified, added or deleted in the lists without needing to reprogram the user interface, is our case the html part, but change dynamically. Through the MVC we will be able to modify the Model (the content of the lists) and the View (the user interface of the web page) is going to be updated dynamically.

Of course, the MVC requires more code and it may increase the complexity, but our system can handle this extra overhead. As the system is planned to be easily expandable and portable between other possible systems, it is necessary for it to have a structure by separating the front-end and back-end functionalities in the Journeys, Likes/Dislikes System.

The second architecture pattern that our system is going to be using is the Repository Architecture. Both the Journeys, Likes/ Dislikes System and the Recommender System are depended from a centralized Database which interconnects the two systems.

The Recommender System uses the Database to collect older recommendation attributes, likes and dislikes in order to generate through the machine learning algorithm the new recommendation attributes for each user. Moreover, the Journeys, Likes/Dislikes System uses the Database when a user enters the web page to load his already created Journeys and Likes/Dislikes. After, if the user modifies, creates or deletes any existing Journeys, Likes/Dislikes the system update the Database accordingly.

2.2 Architectural Diagrams



2.2.1 Actors

2.2.1.1 User

The user represents each user of Foody. The user will be able to access only the Graphical User Interface of the Journeys, Likes/Dislikes System which is part of the Foody Recommendation System. The user can create, modify and delete Journeys. Also, the user can add or remove likes and dislikes.

2.2.1.2 Foody's Database

The Recommender System collects data from the Foody's Database such as the order history of each user in order to calculate the recommendation attributes.

2.2.1.3 Foody's Recommender System

The Foody's Recommender System is the System that Foody is going to implement in order to make use of the recommendation attributes for each user. Also, this actor can collect data from our database such as the Journeys, Likes and Dislikes for each user.

2.2.2 Components

2.2.2.1 Graphical User Interface (View/Controller)

The user interface that the user is going to interact with the Journeys, Likes/Dislikes System, is the web page that will contain four different tabs, the "My Journeys", "Likes", "Dislikes" and "Explore". The user will be able to navigate through these tabs and perform specific actions such as the Creation, Modification and Deletion of a Journey or the Addition and Removal of Likes/Dislikes. More detailed description of the User Interface can be found in the first deliverable. (section 2.1.2) To load the user interface, if the user had already used the Journeys, Likes/Dislikes System before, the component requests from the API component for the Database the entry of the specific user. This component also interacts with the Create/ Modify/ Delete Journey and the Add/ Remove Likes or Dislikes component. More specific it provides the input of the user (Controller) to the Model in order for the two components to perform their actions. After each action the User Interface (View) is being updated.

2.2.2.2 Create/ Modify/ Delete Journey (Model)

The user through the User Interface can create, modify or delete an already created Journey. The user controls which action is going to perform and the result updates the Database through the API component. To be specific. if a user creates/ modifies or delete a Journey, the entry of a Journey is created, edited or deleted respectively in the Database of FRA System.

2.2.2.3 Add/ Remove Likes or Dislikes (Model)

The user through the User Interface can also add or remove liked and disliked products. For each action the Database of FRA System is updated too through its API.

2.2.2.4 Database for Foody Recommendation Agent

This is the main Database of Foody Recommendation Agent. The Database is going to keep the data for the Journeys, Likes/ Dislikes and Recommendation Attributes for each user. The Database only provides data to its API and there is no other way to access it for security reasons. Also, only through the API its content can be modified. Because our Database is essential for any functionality of our System, we decided to implement it based on the Repository architecture pattern (more in section 2.2.3)

2.2.2.5 API for Foody Recommendation Agent System

This component is going to be used for the connection between the database and the rest of the components of the system. The Database is going to provide the functionality to the API to store and update its data. The component will be

able to receive data from Create/Modify/Delete Journey, Add/Remove Likes or Dislikes and Calculate Recommendation Attributes components. The user will be able to store the data from the three components to the Database. Moreover, the component will be able to provide data to the Graphical User Interface and to the Calculate Recommendation Attributes components. The component will provide the User's Entry from the Database for the Graphical User Interface and older recommendation attributes to the Calculate Recommendation Attributes component.

2.2.2.6 Collect order history and prepare it as input for Recommender System

This component collects from Foody's Database the history of orders for each user of Foody. Then it modifies and prepares the data to be imported in the Calculate Recommendation Attributes.

2.2.2.7 Calculate Recommendation Attributes

This component uses as input the history of orders for a user, the previously calculated recommendation attributes, the likes and dislikes in order to calculate the recommendation attributes. After the calculation of the recommendation attributes the component provides the data to the API components in order to update the Database of FRA.

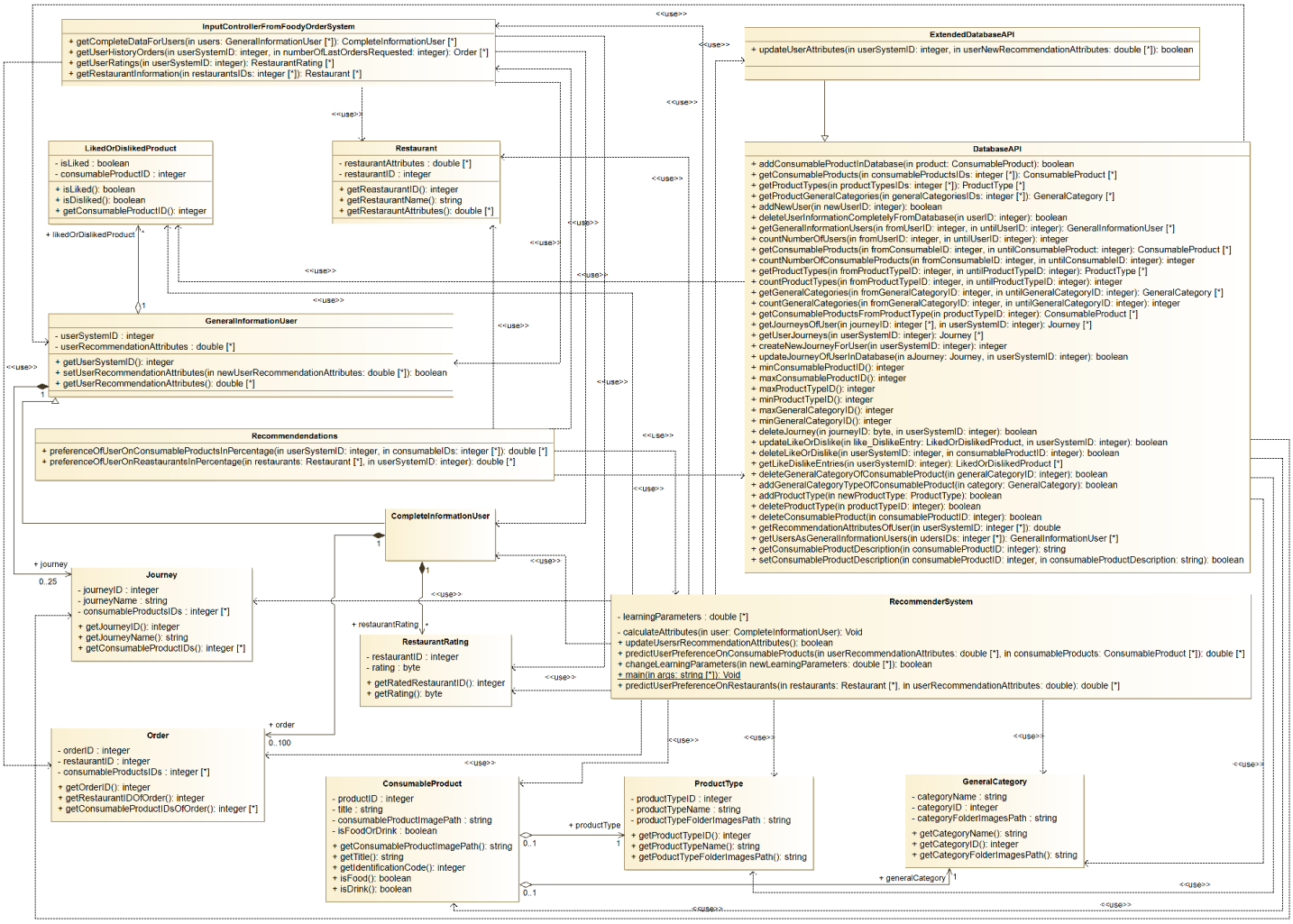
2.2.3 Architecture Patterns

The MVC can be clearly be observed in the Journeys, Likes/Dislikes System components. The Graphical User Interface component uses the View to present the html content dynamically to the user. Through the Controller the input of the User is being handled to be provided into the Model. The Model is part of the Creation, Modification and Deletion of the Journeys, as the content of the Journey doesn't affect itself the functionality of the action. The same is happening with the Add or Remove Likes and Dislikes. More specific, the Model is going to be used to dynamically update the Graphical User Interface and to provide the functionalities without caring about the View or how the input from the user is going to be handled.

The Repository architecture pattern is represented in the API for Foody Recommendation Agent System component. All the components of the system that make calculations on data interact with this component as a centralized repository. This is the component which is being connected with the Database of FRA, so it is the only component that has direct access to the Database. This API offers an abstraction level and provides more security to the data of our Database. Also, the Foody itself (Foody's Recommender System), the only way to extract data through our system is through this component.

The Repository architecture pattern helps in the large scale of data that our system is going to be handled. Moreover, a change in the Journeys, Likes/Dislikes System we would like to be easily accessible to the Recommender System in order to calculate new recommendation attributes for a user that are more accurate. The negative of this architecture is that it is a single point of failure. A problem in the database could block all the Foody Recommendation Agent system. In order to solve this, a backup of the database should be up and running fully updated and easily swappable in case of a problem.

3. Analytical Class Diagrams



The UML Class Diagram is the same as in the second deliverable (section 4).

4. Sample Scenarios and diagrams

4.1 <Calculate User's Recommendation Attributes>

This scenario refers to the calculation of the Recommendation attributes of each user, which the system, at first. retrieves from the database of Foody Recommendation Agent System (the new system) all the information stored about the user. Secondly, the system retrieves some other information about the user from the Foody Online Orders System and thirdly calculating the new Recommendation Attributes of the user based on the information collected.

More specifically this scenario depicts the execution of a routine in the main() method of the RecommenderSystem object of the system, which routine is executed every day from 11:00 p.m. Firstly, the routine takes a number of users — their IDs, and collects their Journeys, their liked and disliked consumable products, from the database of the Foody Recommendation Agent System, as objects of type GeneralInformationUser which contains the above.

Secondly, the routine collects for each of the user who has been chosen, their order history of the last 100 orders and their Restaurant Ratings and creates an Object of type CompleteInformationUser which contains the latter as well as all information retrieved about the user from the previous step.

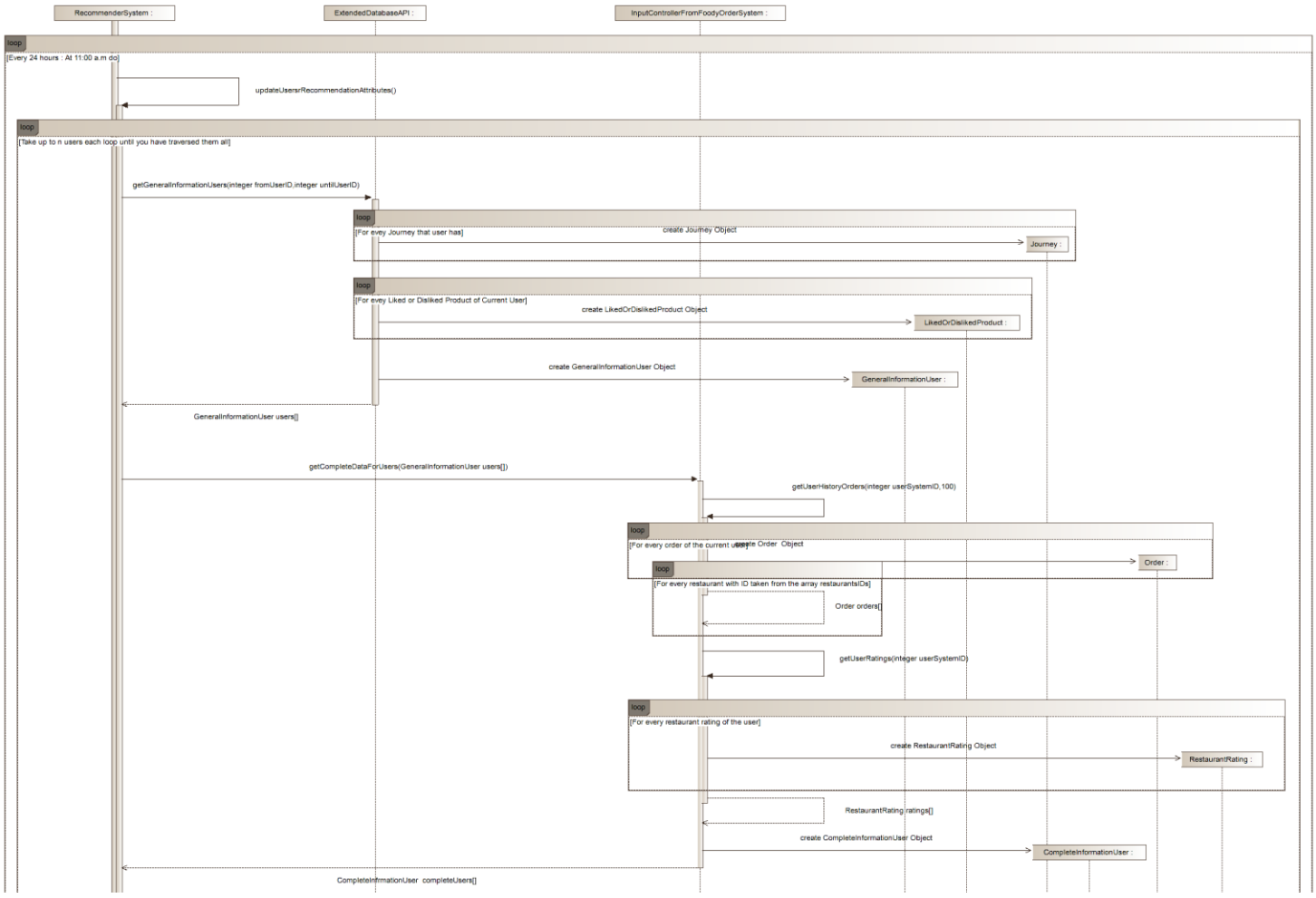
Thirdly, for each of the users that have been selected, an asynchronous invocation (creation of a new thread) of the method calculateAttributes (CompleteInformationUser user) from class RecommenderSystem is made. During the execution of that method all restaurants (stored in the Database of Foody Online Orders system) and Consumable Products (stored in the Database of the Foody Recommendation Agent system) that are associated with the orders, RestaurantRatings, Journeys, Liked or Disliked Consumable Products of the user are retrieved from the appropriate Databases. All the above are used to calculate the new Recommendation attributes of the user and after they have been calculated, they are stored in the Database of the Foody Recommendation Agent System again. Afterwards, the objects associated with the information of the user are destroyed with the correct sequence by invoking the appropriate function-destructor (__del__) in python. (Note that all the objects are destroyed explicitly by invoking the destructor except the last one, object of type GeneralInformationUser which is collected by the Garbage collector without invoking explicitly the destructor). Then it moves back to the third step where it takes the next collected available users and follows the same routine. When the available collected users are traversed, it returns to the second step where it chooses a group of users which haven't been checked yet and then goes to step three again. It follows the same behavior until it has updated the attributes of all the users in the Database of the Foody Recommendation Agent System. Last step before it finishes with all the users, it can update the learning parameters of the RecommenderSystem class which is optional and depends on whether there is a new optimization in learning that can be done.

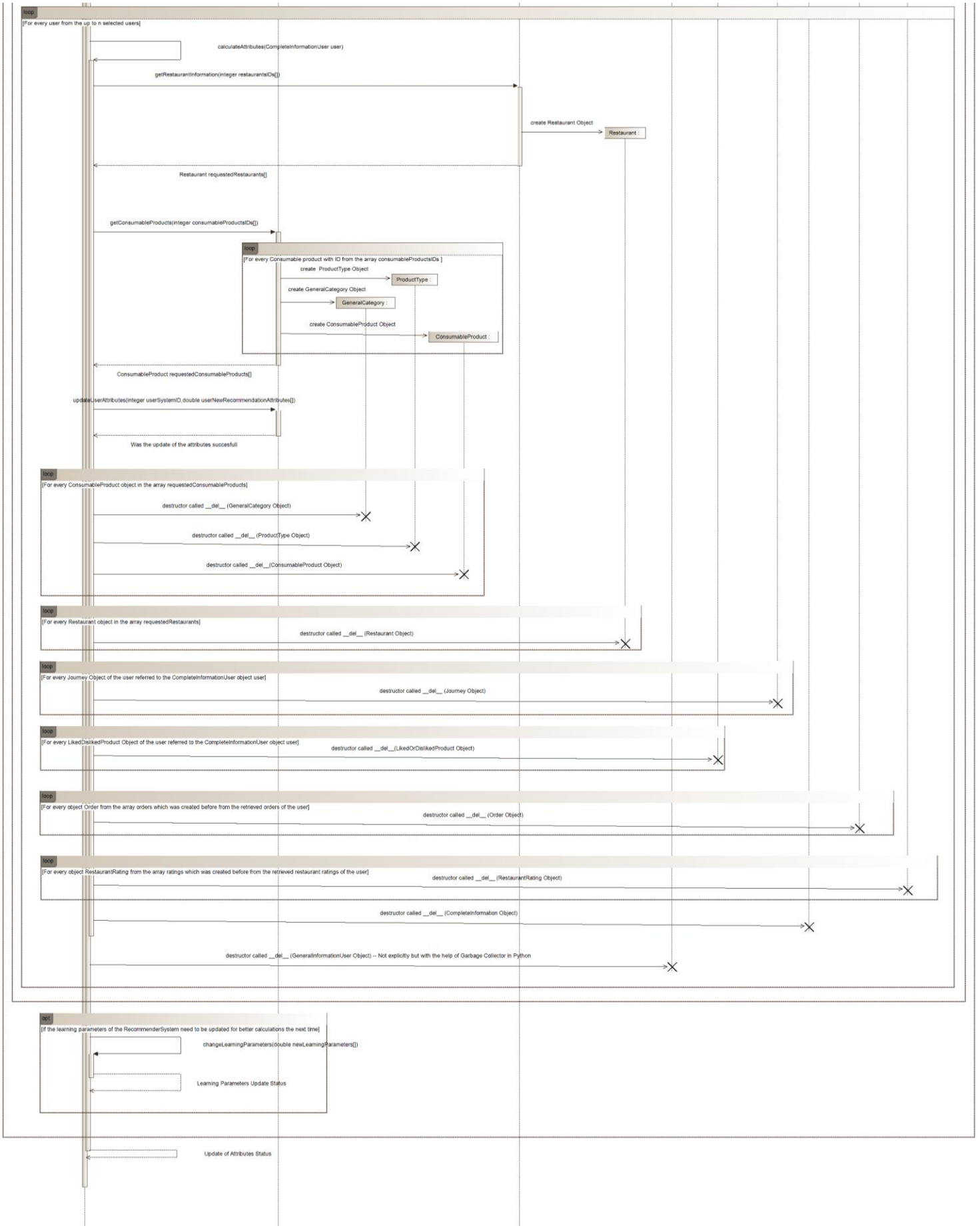
4.1.1 Sequence diagram

In the diagram below the difference between

1. "destructor called __del__ (... Object) -- Not explicitly but with the help of Garbage Collector in Python"
2. "destructor called __del__ (Journey Object)"

is that the first one is an explicit call of the deconstructor in Python and the second is when the Garbage Collector takes care of the objects that remain useless.




loop
[For every user from the up to n selected users]
calculateAttributes(CompleteInformationUser user)
getRestaurantInformation(integer restaurantsIDs[])
create Restaurant Object
Restaurant :
Restaurant requestedRestaurants[]
getConsumableProducts(integer consumableProductsIDs[])
loop
[For every Consumable product with ID from the array consumableProductsIDs]
create ProductType Object
ProductType :
create GeneralCategory Object
GeneralCategory :
create ConsumableProduct Object
ConsumableProduct :
ConsumableProduct requestedConsumableProducts[]
updateUserAttributes(integer userSystemID,double userNewRecommendationAttributes[])
Was the update of the attributes succesfull
loop
[For every ConsumableProduct object in the array requestedConsumableProducts]
destructor called __del__ (GeneralCategory Object)
destructor called __del__ (ProductType Object)
destructor called __del__(ConsumableProduct Object)
loop
[For every Restaurant object in the array requestedRestaurants]
destructor called __del__ (Restaurant Object)
loop
[For every Journey Object of the user referred to the CompleteInformationUser object user]
destructor called __del__ (Journey Object)
loop
[For every LikedDislikedProduct Object of the user referred to the CompleteInformationUser object user]
destructor called __del__(LikedOrDislikedProduct Object)
loop
[For every object Order from the array orders which was created before from the retrieved orders of the user]
destructor called __del__ (Order Object)
loop
[For every object RestaurantRating from the array ratings which was created before from the retrieved restaurant ratings of the user]
destructor called __del__ (RestaurantRating Object)
destructor called __del__ (CompleteInformation Object)
destructor called __del__ (GeneralInformationUser Object) -- Not explicitly but with the help of Garbage Collector in Python
opt
[If the learning parameters of the RecommenderSystem need to be updated for better calculations the next time]
changeLearningParameters(double newLearningParameters[])
Learning Parameters Update Status
Update of Attributes Status

4.2. <Create Journey>

4.2.1 Scenario Description

This scenario describes how a user can create a new Journey using a website of the Foody Recommendation Agent system, but also what are the transitions they can do, while they are interacting with the website for the creation of a new journey.

Firstly, the user manages to connect to their account (this is not part of the Foody Recommendation Agent system but of the Foody Online Orders system) and goes to the webpage where their Journeys can be viewed (retrieved from the database of FRA and displayed to the user) as well as some of the Consumable Products of each Journey (also retrieved from the database of FRA and displayed to user).

Secondly, the user proceeds to create the new Journey. If they haven't created so far 25 Journeys, the requested Journey will be created as empty and will be kept with the other loaded Journeys of the user and also will automatically be saved in Database as an empty Journey. Otherwise it isn't created and of course nothing is saved.

Thirdly, the user can later modify their Journey or any other Journey that they have. By modifying it, they can find new Consumable Products and add them in the Journeys, as well as like or dislike the consumable products they add, with respect to their option.

Either the user choose to modify their Journey or not (both from the Modification step of the Journey and from the Creation step of the Journey, the transition to the Likes/Dislikes is possible), the user can jump to the Like/Dislike section where they can search for Consumable Products and update them as liked or disliked consumable products. In that section, the Liked/Disliked consumable products of the user are loaded and then the user can update their likes/dislikes on consumable products.

If the user doesn't want to leave but traverse to the section of Likes/Dislikes but wants to save or delete their updates in Journeys, they can proceed to that subsection, where they can choose whether to make a deletion of their Journey or update the changes made to that Journey.

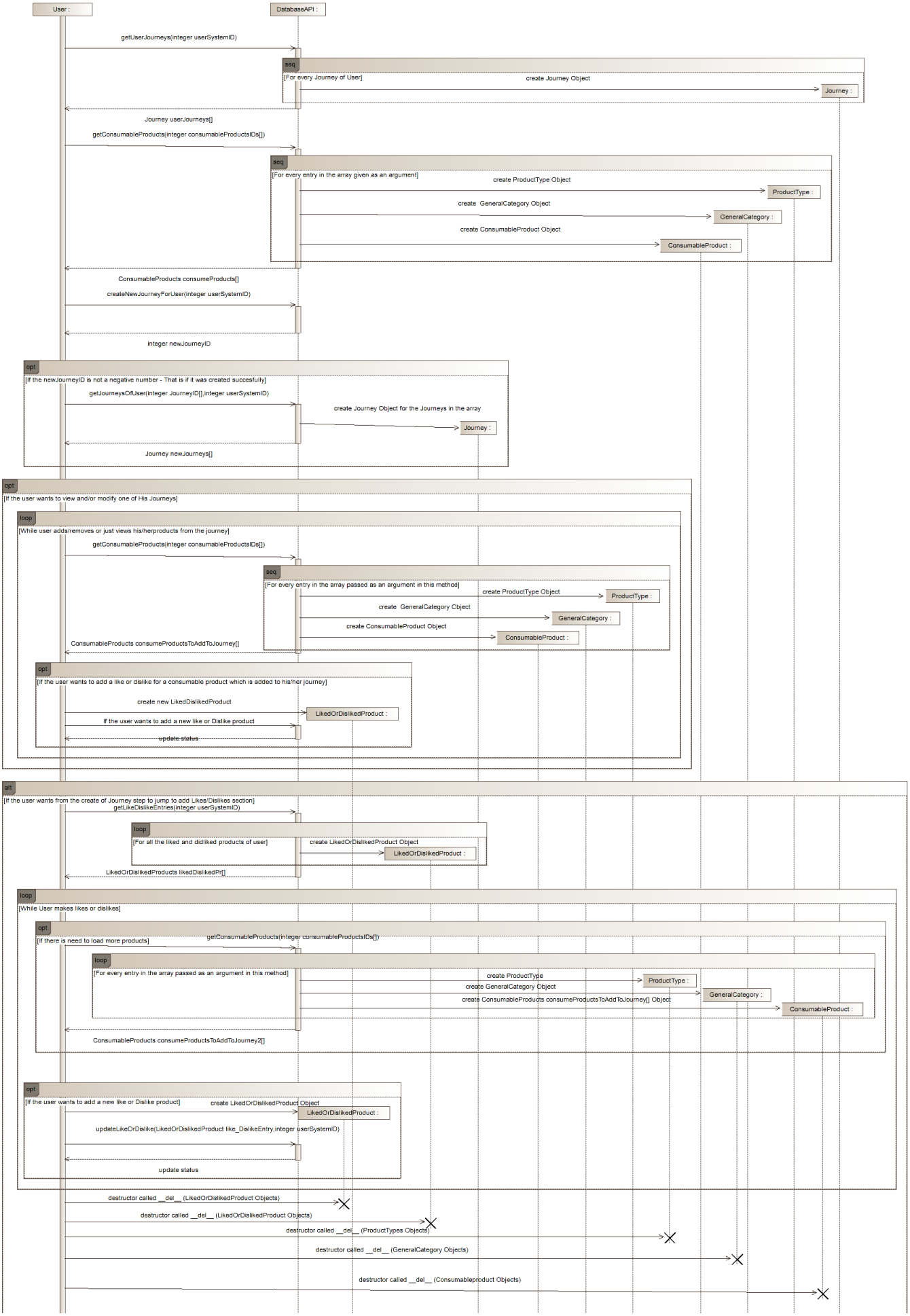
4.2.2 Sequence diagram

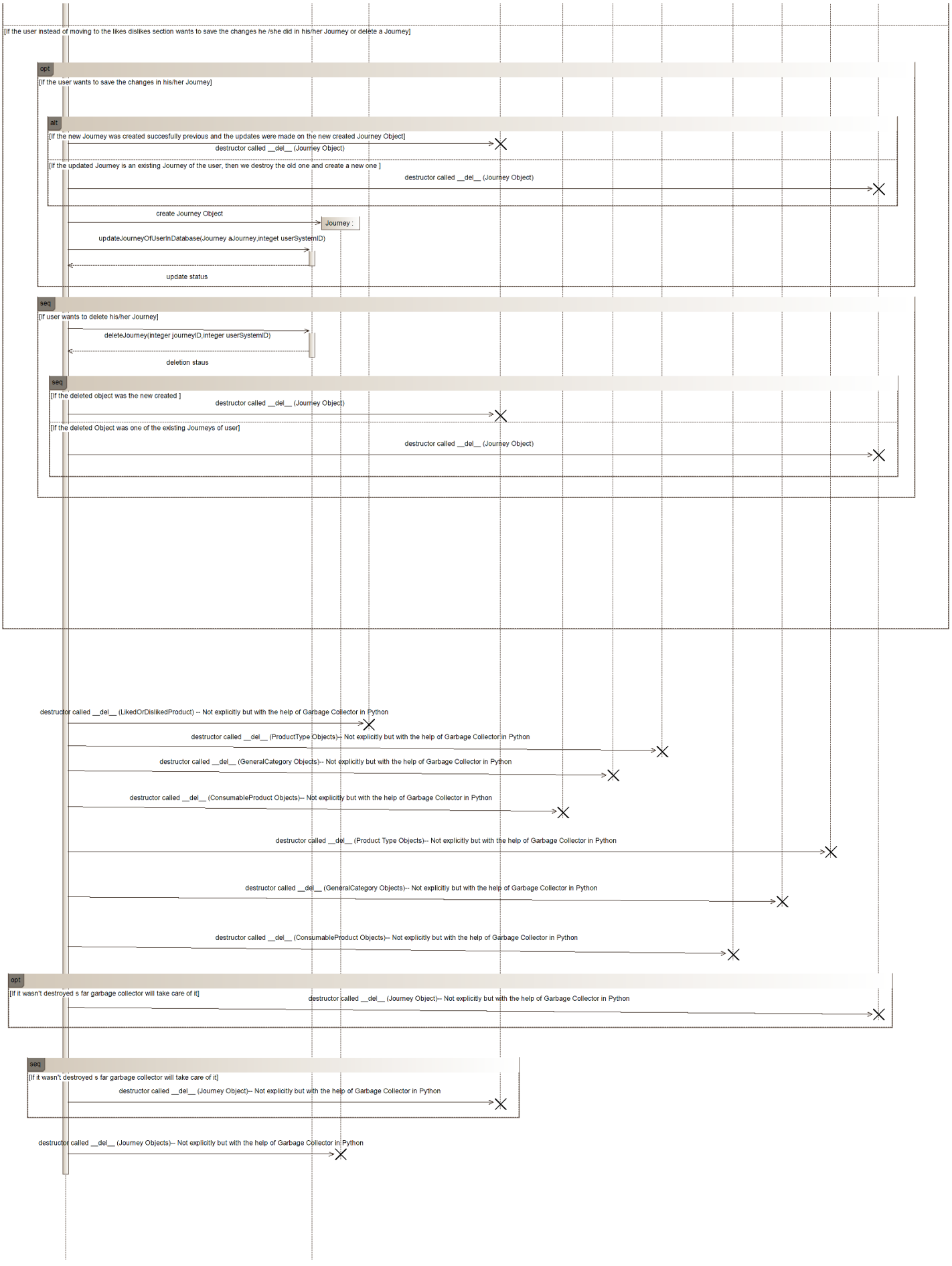
In the diagram below the difference between

3. "destructor called __del__ (... Object) -- Not explicitly but with the help of Garbage Collector in Python"
4. "destructor called __del__ (Journey Object)"

is that the first one is an explicit call of the deconstructor in Python and the second is when the Garbage Collector takes care of the objects that remain useless.

Note: Note the difference when used the word "Object" in "destructor called __del__ (... Object)" and the word "Objects in "destructor called __del__ (... Objects)". The case when the word "Object" is used, indicates that only one object is destroyed and the case when word "Objects" is used is when many objects of the indicating type are destroyed.


User :
DatabaseAPI :
getUserJourneys(integer userSystemID)
seq
[For every Journey of User]
create Journey Object
Journey :
Journey userJourneys[]
getConsumableProducts(integer consumableProductsIDs[])
seq
[For every entry in the array given as an argument]
create ProductType Object
ProductType :
create GeneralCategory Object
GeneralCategory :
create ConsumableProduct Object
ConsumableProduct :
ConsumableProducts consumeProducts[]
createNewJourneyForUser(integer userSystemID)
integer newJourneyID
opt
[If the newJourneyID is not a negative number - That is if it was created succesfully]
getJourneysOfUser(integer JourneyID[],integer userSystemID)
create Journey Object for the Journeys in the array
Journey :
Journey newJourneys[]
opt
[If the user wants to view and/or modify one of His Journeys]
loop
[While user adds/removes or just views his/herproducts from the journey]
getConsumableProducts(integer consumableProductsIDs[])
seq
[For every entry in the array passed as an argument in this method]
create ProductType Object
ProductType :
create GeneralCategory Object
GeneralCategory :
create ConsumableProduct Object
ConsumableProduct :
ConsumableProducts consumeProductsToAddToJourney[]
opt
[If the user wants to add a like or dislike for a consumable product which is added to his/her journey]
create new LikedDislikedProduct
LikedOrDislikedProduct :
If the user wants to add a new like or Dislike product
update status
alt
[If the user wants from the create of Journey step to jump to add Likes/Dislikes section]
getLikeDislikeEntries(integer userSystemID)
loop
[For all the liked and didliked products of user]
create LikedOrDislikedProduct Object
LikedOrDislikedProduct :
LikedOrDislikedProducts likedDislikedPr[]
loop
[While User makes likes or dislikes]
opt
[If there is need to load more products]
getConsumableProducts(integer consumableProductsIDs[])
loop
[For every entry in the array passed as an argument in this method]
create ProductType
ProductType :
create GeneralCategory Object
GeneralCategory :
create ConsumableProducts consumeProductsToAddToJourney[] Object
ConsumableProduct :
ConsumableProducts consumeProductsToAddToJourney2[]
opt
[If the user wants to add a new like or Dislike product]
create LikedOrDislikedProduct Object
LikedOrDislikedProduct :
updateLikeOrDislike(LikedOrDislikedProduct like_DislikeEntry,integer userSystemID)
update status
destructor called __del__ (LikedOrDislikedProduct Objects)
destructor called __del__ (LikedOrDislikedProduct Objects)
destructor called __del__ (ProductTypes Objects)
destructor called __del__ (GeneralCategory Objects)
destructor called __del__ (Consumableproduct Objects)


[If the user instead of moving to the likes dislikes section wants to save the changes he /she did in his/her Journey or delete a Journey]
opt
[If the user wants to save the changes in his/her Journey]
alt
[If the new Journey was created succesfully previous and the updates were made on the new created Journey Object]
destructor called __del__ (Journey Object)
[If the updated Journey is an existing Journey of the user, then we destroy the old one and create a new one]
destructor called __del__ (Journey Object)
create Journey Object
Journey :
updateJourneyOfUserInDatabase(Journey aJourney,integet userSystemID)
update status
seq
[If user wants to delete his/her Journey]
deleteJourney(integer journeyID,integer userSystemID)
deletion staus
seq
[If the deleted object was the new created]
destructor called __del__ (Journey Object)
[If the deleted Object was one of the existing Journeys of user]
destructor called __del__ (Journey Object)
destructor called __del__ (LikedOrDislikedProduct) -- Not explicitly but with the help of Garbage Collector in Python
destructor called __del__ (ProductType Objects)-- Not explicitly but with the help of Garbage Collector in Python
destructor called __del__ (GeneralCategory Objects)-- Not explicitly but with the help of Garbage Collector in Python
destructor called __del__ (ConsumableProduct Objects)-- Not explicitly but with the help of Garbage Collector in Python
destructor called __del__ (Product Type Objects)-- Not explicitly but with the help of Garbage Collector in Python
destructor called __del__ (GeneralCategory Objects)-- Not explicitly but with the help of Garbage Collector in Python
destructor called __del__ (ConsumableProduct Objects)-- Not explicitly but with the help of Garbage Collector in Python
opt
[If it wasn't destroyed s far garbage collector will take care of it]
destructor called __del__ (Journey Object)-- Not explicitly but with the help of Garbage Collector in Python
seq
[If it wasn't destroyed s far garbage collector will take care of it]
destructor called __del__ (Journey Object)-- Not explicitly but with the help of Garbage Collector in Python
destructor called __del__ (Journey Objects)-- Not explicitly but with the help of Garbage Collector in Python

4.3 <RecommenderSystem processing cycle>

4.3.1 <Scenario Description>

In this scenario we can see the different states in which an object of the class Recommender System can be in.

There are 9 states in which this class can be:

1. The state where it waits until an event occurs or until it is time to calculate the attributes of the users. State: **WaitingUntilTimeForRecommendationAlgorithmExecution**

2. The state where it executes the Recommender algorithm; that is, method updateUsersRecommendationAttributes() only. State: **ExecutingRecommendationAlgorithmOnly**

3. The state where it executes the Recommender algorithm on a set of users; that is, method calculateAttributes() only on some users (Execute many parallel threads of that method on different users —> In order to move, all threads have to wait until all of the parallel threads are finished with the execution of the method).
   State: **CalculatingAttributesForSpecificUserOnly**

4. The state where it waits until it's time to calculate the attributes of the users but at the same time services the invocation of a method for predicting User preferences on consumable products; that is, method predictUserPreferenceOnConsumableProducts(). State:
   PredictingUserPreferencesOnConsumableProductsWhileWaitingUntilTimeForRecommendationAlgorithmExecution

5. The state where it executes the Recommender algorithm and also services the invocation of a method for predicting User preferences on consumable products; that is, method updateUsersRecommendationAttributes() and also method predictUserPreferenceOnConsumableProducts(). State:
   PredictingUserPreferencesOnConsumableProductsWhileExecutingRecommendationAlgorithm

6. The state where it executes the Recommender algorithm on a set of users and also services the invocation of a method for predicting User preferences on consumable products; that is, execution of the method calculateAttributes() on some users and method predictUserPreferenceOnConsumableProducts() (Execute many parallel threads of that method on different users —> In order to move, all threads have to wait until all of the parallel threads are finished with the execution of the method). State:
   PredictingUserPreferencesOnConsumableProductsWhileCalculatingAttributesForSpecificUser

7. The state where it waits until it is time to calculate the attributes of the users but at the same time services the invocation of a method for predicting User preferences on Restaurants; that is, method predictUserPreferenceOnRestaurants(). State:
   PredictingUserPreferencesOnRestaurantsWhileWaitingUntilTimeForRecommendationAlgorithmExecution

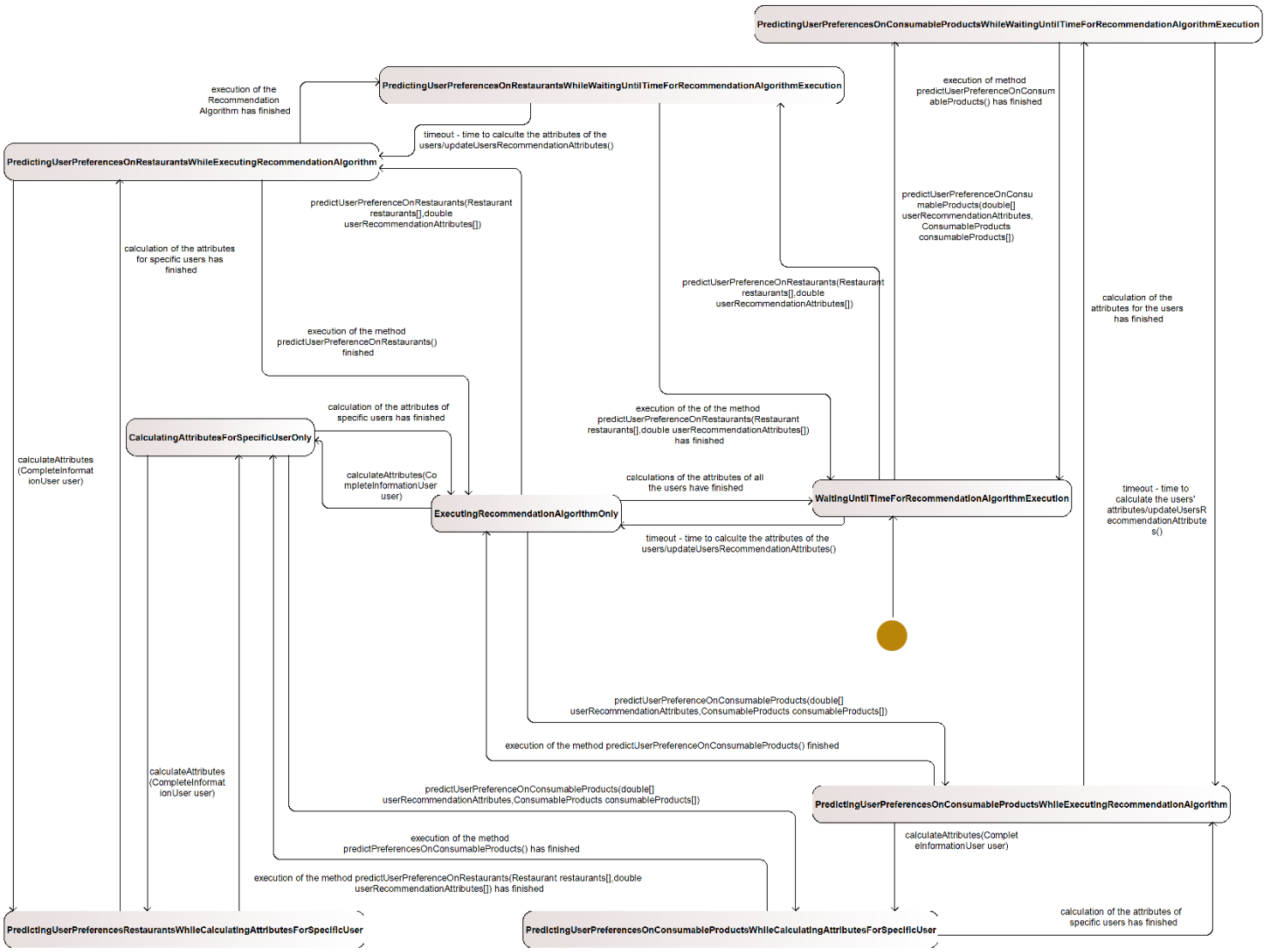
8. The state where it executes the Recommender algorithm and also services the invocation of a method for predicting User preferences on Restaurants; that is, method updateUsersRecommendationAttributes() and also method predictUserPreferenceOnRestaurants() .State:
   PredictingUserPreferencesOnRestaurantsWhileExecutingRecommendationAlgorithm

9. The state where it executes the Recommender algorithm on a set of users and also services the invocation of a method for predicting User preferences on Restaurants; that is, it executes the method calculateAttributes() on some users

and the method predictUserPreferenceOnRestaurants() (Execute many parallel threads of that method on different users —> In order to move , all threads have to wait until all of the parallel threads are finished with the execution of the method). State: **PredictingUserPreferencesRestaurantsWhileCalculatingAttributesForSpecificUser**

4.3.2 State Diagram

In the state diagram the events on the transitions are omitted by the UML Modelio tool. The transitions as well as the actions that happen on each transition can be seen by double left clicking on each transition on the state machine diagram in the UML Modelio. So, please refer to the diagram for a complete understanding of the transitions of the object.



5. Appendices

5.1 Correction in Use Case Diagram

In the Use Case Diagram, the actor Foody's Recommender System was named Foody Recommendation Agent by mistake. Foody's Recommender Agent it the system of Foody which is going to use all the data that we generate for them such as the Journeys, the Likes/ Dislikes and the Recommendation Attributes for the users. Through all these data, the Foody's Recommender System will use them to make appropriate recommendations to his customers.