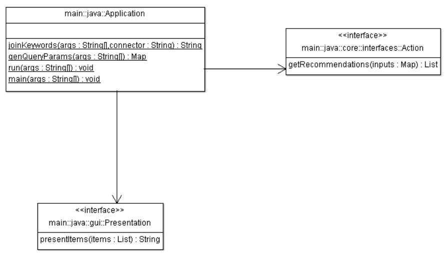
Solution Overview

Xiaocheng Zou

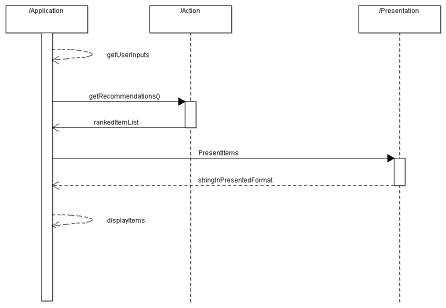
January 13, 2016

The high-level overview design is shown in the figure 1. Basically, this application is divided into three components, as shown in subfigure 1(a). Note that there are two components shown as interfaces for the convenience. This design is a MVC design architecture. As we know, one of the benefits of MVC architecture is that it separates the "view" (represented by the **Presentation** interface in the figure 1) from the "model" (represented by the **Action** interface in the figure 1), which has the business logic. These two components are glued together by the "controller" (represented by the **Application** class in the figure 1). Besides gluing the "view" and "model" components, the "controller" also deals with user inputs.

A sequence diagram (shown in subfigure 1(b)) shows the interaction between these three components. The overall sequence is relatively straightforward. After the **Application** class collects the user inputs, it calls the *getRecommendation* function from the "model" component (i.e., the **Action** interface). The *getRecommendation* function returns a list of ordered recommended items to the **Application** class, who later gives them to the "view" component (i.e., the **Presentation** interface). The **Presentation** interface returns a specific presentation of the recommended items, which is finally displayed by the **Application** class.


main::java::Application
joinKeywords(args : String[],connector : String) : String
genQueryParams(args : String[]) : Map
run(args : String[]) : void
main(args : String[]) : void
<<interface>>
main::java::core::interfaces::Action
getRecommendations(inputs : Map) : List
<<interface>>
main::java::gui::Presentation
presentItems(items : List) : String

(a)

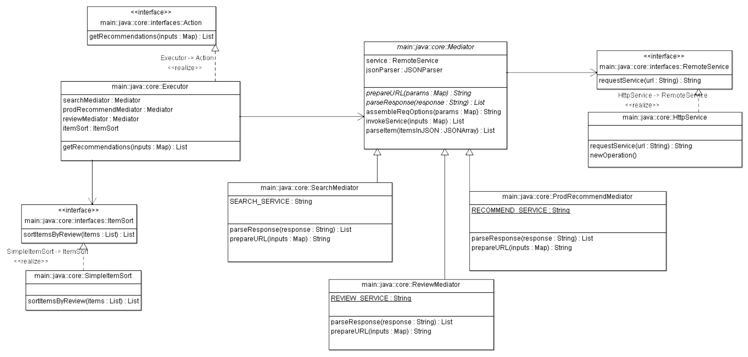


(b)

Figure 1: High-level design with class diagram shown in subfigure 1(a) and sequence diagram shown in subfigure 1(b). The sequence diagram shows the overall flow in this application.

The "controller" and "view" components are relatively simple in this application as the user interface is not required. The relatively complicated component is from the "model" component. Now, lets discuss how the *getRecommendation* function is implemented inside of the "model" component.

Figure 2 shows class relationships in the "model". The **Executor** class implements the **Action** interface. It has three concrete **Mediator** classes to fulfil the search, product recommendation and review services. In addition, it also as a **ItemSort** interface, which is responsible for the item sorting. These three *Mediator* classes, all inherited from an abstract class **Mediator**, which abstracts http services to a three-step procedure. These three steps are prepare the request URL, invoke http request, and parse the response as shown in the figure 3. After these three services are finished, the returned recommended items with reviews are then handled to the **ItemSort** interface, which returns the sorted item list.



<<interface>>
main::java::core::interfaces::Action
getRecommendations(inputs : Map) : List
Executor -> Action
<<realize>>
main::java::core::Executor
searchMediator : Mediator
prodRecommendMediator : Mediator
reviewMediator : Mediator
itemSort : ItemSort
getRecommendations(inputs : Map) : List
<<interface>>
main::java::core::interfaces::ItemSort
sortItemsByReview(items : List) : List
SimpleItemSort -> ItemSort
<<realize>>
main::java::core::SimpleItemSort
sortItemsByReview(items : List) : List
main::java::core::Mediator
service : RemoteService
jsonParser : JSONParser
prepareURL(params : Map) : String
parseResponse(response : String) : List
assembleReqOptions(params : Map) : String
invokeService(inputs : Map) : List
parseItem(itemsInJSON : JSONArray) : List
<<interface>>
main::java::core::interfaces::RemoteService
requestService(url : String) : String
HttpService -> RemoteService
<<realize>>
main::java::core::HttpService
requestService(url : String) : String
newOperation()
main::java::core::SearchMediator
SEARCH_SERVICE : String
parseResponse(response : String) : List
prepareURL(inputs : Map) : String
main::java::core::ProdRecommendMediator
RECOMMEND_SERVICE : String
parseResponse(response : String) : List
prepareURL(inputs : Map) : String
main::java::core::ReviewMediator
REVIEW_SERVICE : String
parseResponse(response : String) : List
prepareURL(inputs : Map) : String

Figure 2: Class diagram in the "model" component.

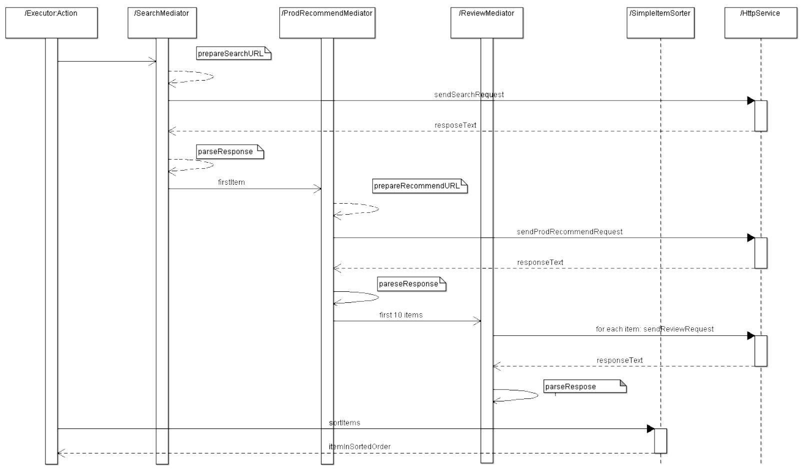


Figure 3: A sequence diagram in the “model” component, illustrating how the *getRecommendation* function is implemented.