Muhammad Afzal, Syed Imran Ali, Muhammad Sadiq

Service Curation Layer

Integrated Service Curation Layer v2.5

25 DEC 2016

Table of Contents

1. Overview
2. Recommendation Builder
   1. Introduction
   2. Capabilities
   3. Use Case Scenarios
3. Recommendation Interpreter
   1. Introduction
   2. Capabilities
   3. Pre-requisite data files
      1. Physical activity data files
      2. Food item data files
   4. Structure of data files
   5. Food-item recommendation
   6. Admissible rules for Recommendation Interpreter
   7. Running Scenario
4. Service Orchestrator
   1. Introduction
   2. Purpose
5. Communication Workflow
6. Overview

Integrated Service Curation Layer (SCL) is designed to curating recommendation services for health and wellness. It composed of two main components: recommendation builder (RB) and recommendation interpreter (RI). These two component are supported with a third component called service orchestrator (SO) in terms of communication with the outer world

1. Recommendation Builder
   1. Introduction

Recommendation Builder, generates recommendations through reasoning on the user proﬁle and life-log data and the knowledge rules developed in a specific format. RB provided recommendations are considered as initial recommendation because of the fact that the recommendations are yet to be interpreted from the user’s contextual perspective. The initial recommendation may be forwarded as-is or transforming it to a more applicable form.

* 1. Capabilities

Recommendation Builder generates recommendation based on the input rules and facts.

Recommendation builder uses forward chaining algorithm to provide recommendations for the given situation. Forward chaining algorithm is implemented inside *PatternMatcher* class, the PatternMatcher may fire more than one recommendation rules for a single situation, now to resolve this conflict we need a kind of conflict resolution mechanism for this purpose we have provided *ConflictResolver* class which uses *maximum specificity* mechanism to select most appropriate recommendation from the set of recommendations provided by the PatternMatcher. You can find PatternMatcher and ConflictResolver classes inside *org.uclab.scl.framework.recbuilder*

* 1. Use Case Scenario’s
* Fire Rule for Specific Situation

*fireRule(event, rules)* method of PatternMatcher class is used to fire rule/s for specific event or situation. This method requires two parameters

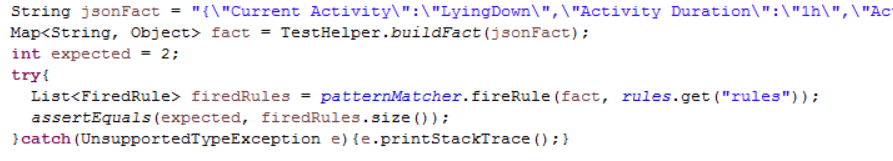
1. *Event – Map<String, Object>*
2. *Rules - List<Map<String, Object>>*

Calling this method with appropriate parameters will return list of FiredRule Objects e.g.

*List<FiredRule> firedRules = patternMatcher.fireRule(fact, rules.get("rules"));*

For more detail usage look over unit test cases in the following package:

*org.uclab.scl.framework.recbuilder*



* Conflict Resolution

As we mentioned above the pattern matcher may fire more than one rule for specific event or situation, so in that case we need to provide the fired rules to the conflict resolver so that it can resolve conflict among the fired rules and select the most applicable rule. You can use conflict resolver class as below:

*resolvedRules = conflictResolver.resolveConflict(firedRules)* for more details refer to unit test cases in the following package:

*org.uclab.scl.framework.recbuilder*

1. Recommendation Interpreter
   1. Introduction

Recommendation Interpreter (RI) is envisioned as a contextual information processing unit of a larger recommendation generating system. In the current, concept-of-proof implementation, RI deals with two kinds of recommendations i.e. physical activity based recommendation and food-items recommendation. The main purpose of RI is to complement the recommendation generated based on user’s physiological requirements or conditions. RI takes into account contextual information of the user in terms of location of the user, user’s current activity, weather conditions, and emotional state of the user. Physical activity recommendations are based on a novel concept called “Contextual Matrix”. Contextual Matrix processes users’ preference data and in turn produces cross-contextual global patterns. This contextual matrix is populated using users’ surveyed data.

* 1. Capabilities

RI in the current implementation has following features:

1. Assess user’s interruptibility
2. Evaluate recommendation for its contextual suitability
3. Provide alternative recommendations if so required
4. Provide explanatory sentences along with audio/visual aids for the generated recommendations
5. Provide food based recommendations
   1. Pre-requisite data files

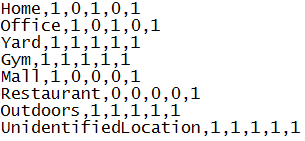
RI requires a number of data files to perform aforementioned activities. Data files are logically categorized into two types based on the nature of recommendation i.e. physical activity data files and food-item data files.

* + 1. Physical activity data files
       - Location.txt
       - HLC.txt
       - Weather.txt
       - Emotion.txt
       - Disable.txt
       - Preferences.txt
       - Explanation.txt
       - URL.txt
       - Postprocessing.txt
       - RecommendationMapper.txt
    2. Food-item data files
       - FoodCatToFooditems.txt
       - FooditemsToNutrient.txt
       - NutrientToCategory.txt
       - UserChoices.txt

Note: For the current implementation these files are required to be provided. However, based on the system requirements, this code can be extended to read/write from any other repository. Moreover, current version provides a proof of concept implementation of the RI, in subsequent versions extendibility of the component along with handling multiple recommendations will be addressed following the best practices of software design.

* 1. Structure of data files

Almost all of the files share the same *key-value* pair structure. Following figure depicts structure of Location.txt file:

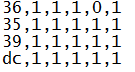


For example first string “*Office,1,0,1,0,1*” depicts a contextual scenario for location. Each bit represents a binary decision. Currently RI deals with five recommendations namely ‘*Walking’*, ‘*Running’*, ‘*Stretching’*, ‘*Cycling’*, and ‘*Sitting’*. Recommendations are treated in the same order as mentioned. For example in ‘Office context ‘*Running’* and ‘*Cycling’* can’t be performed. These data files are based on surveyed data. Users may change these values according to their own suitability and desired results. Contextual matrix will behave as per the data fed to it.

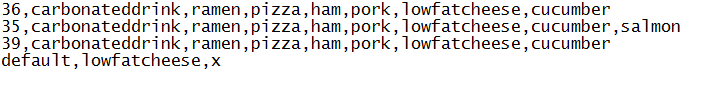
Data files are provided in:



All the pre-requisite data files are in the aforementioned package. It is important to note that in the present build (version 1.0) user’s information has to be entered manually in the *Preferences.txt* file (for physical activity) and *UserChoices.txt* (for food-item recommendation).



Where, *UserChoices.txt* includes those food-items which are disliked by the user in the given list provided in *FooditemsToNutrient.txt.*

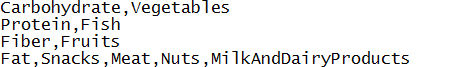


For example userid ‘*36*’ doesn’t like ‘*Cycling’* and doesn’t prefer food-items such as “*Carbonated drink, Ramen, Pizza, Ham, Pork, Low Fat Cheese, and Cucumber*”.

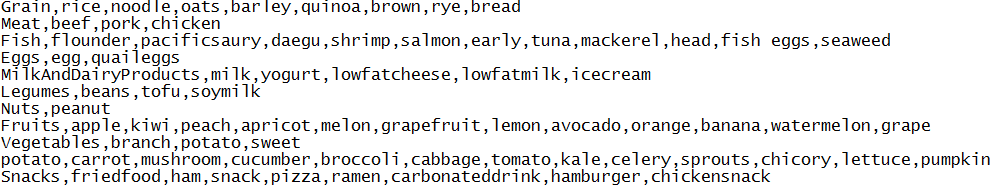
* 1. Food-item recommendation

In the current version RI cater for following nutrient categories:

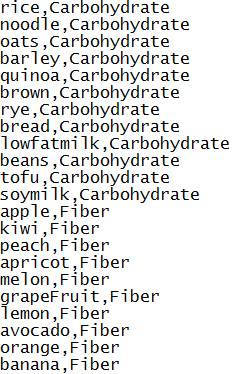
* + 1. Carbohydrate
    2. Protein
    3. Fiber
    4. Fat



Following are the mappings from food categories to food items



Following is a snippet for food-item to nutrient mappings:



Note: In the aforementioned snippets first word is treated as *key* while rest of the string is dealt as *value*, as in the key-value pair. Naming conventions must be followed as depicted in the data files.

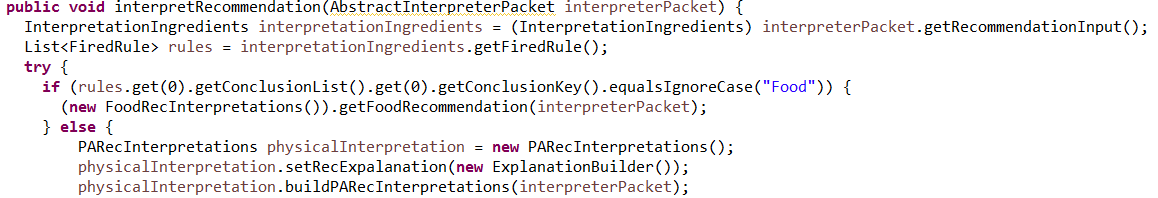
Following is a sample output for food-item recommendation:



Where, SNS trend value is followed by the recommended food-item, which may be used for visualization purposes as well.

* 1. Admissible rules for Recommendation Interpreter

Following code snippet depicts the core logic where rules for food recommendation are differentiated from physical activity rules:



As illustrated in the aforementioned code snippet rules for food recommendation contain “*Food*” is key while rules for physical activity contains the keyword “*Recommended Activity*”. Please consult sample rules, provided in *Main.java* file, for creating new rules.

* 1. Running Scenario
     1. RI receives recommendation in the form of a production rule from Recommendation Builder (please refer to Recommendation Builder for the rule’s structure)
     2. *InterpretRecommendation* method in *RecInterpter* class serves as the starting point for RI processing
     3. Received input as in the form a *packet* (for the structure of packet please refer to *InterpreterPacket* class)
     4. Based on the contents of packet a decision is made whether to activate processing for physical activity recommendation or food recommendation (as mentioned at number 6)
     5. For example rule pertaining to physical activity is received
     6. *PARecInterpretations.java* is the key class which controls logic for processing different aspects (user’s interruptibility, contextual viability of the recommendation, explanation generation) of physical activity
     7. Final result of physical activity recommendation is stored in *InterpretedRecommendations* object
     8. *InterpretedRecommendations* object contains following key properties for physical activity recommendation
        + Recommended activity (string)
        + Recommended activity duration (string)
        + Recommended activity description (string)
        + Recommended activity URL (string)
        + Recommended activity’s current context (string)
     9. For the case of food recommendation following two properties of *InterpretedRecommendations* are set:
        + Recommended food item list (string)
        + General description (which is part of the rule received from Recommendation Builder)

1. Service Orchestrator
   1. Introduction

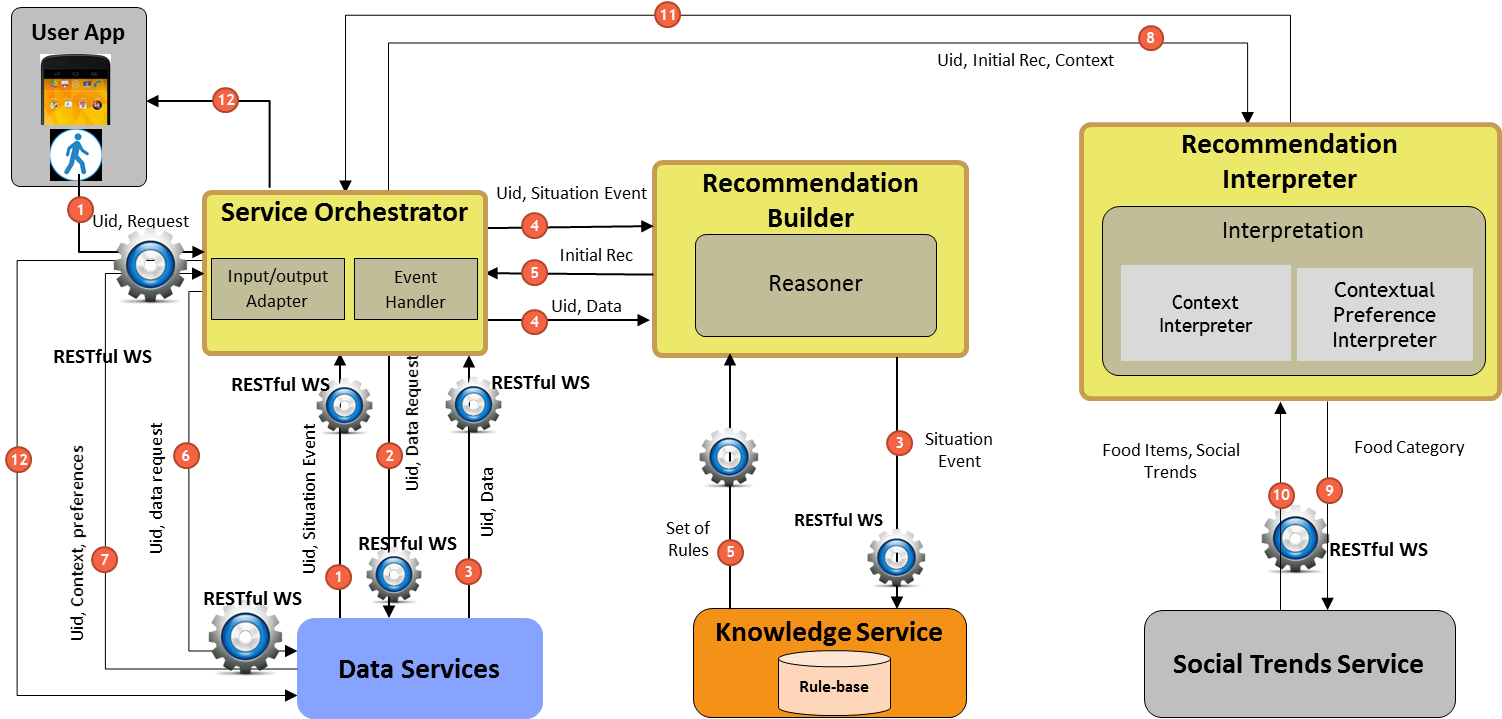
Service Orchestrator, is in charge of handling the potential requests, invoking the necessary services and coordinating with outer world to fulfill the data requirements for the processes involved in the curation of the services. The requests may be of various types, i.e., triggered by direct user queries (“suggest me an exercise plan for today’s workout”) or based on events (e.g., “sitting one hour”).

* 1. Purpose

Service Orchestrator is a wrapper for recommendation builder and recommendation interpreter. Service orchestrator provides an interface for outside world to communicate to the recommendation builder and interpreter through REST web services. It provides REST endpoints for communication. You can find the service class e.g. *SCLServiceResource.java* for the endpoints in *org.uclab.scl.rs* package.

1. Communication Workflow

* The recommendation request is received to SO either through user directly or generated on the basis of some events.
* SO fulfills the data requirements in the form of recommendation ingredients by calling different services from different locations
* SO passes the request along with recommendation ingredients to recommendation builder (RB).
* RB connects to knowledge server in order to find the rules against the triggered situation event
* Using forward chaining mechanism, RB performs reasoning over the rules and data and builds the initial set of recommendations
* RB returns the initial recommendation to SO.
* SO passes the initial recommendation to recommendation interpreter (RI).
* RI checks if the recommendation is related to Food, it connects to social service to get trends for a recommended food category and finalized the recommendations accordingly.
* If the recommendations are physical activity related, the recommendations interpreted based on the different contexts such as location, high level context, etc.
* RI explains the recommendation for further clarification in case the initial recommendation are not included with such descriptions. The explanation may include descriptive
* statements and links to educational tutorials/videos.
* The final contextualized recommendation are first sent to returned to SO.
* SO sends the final recommendation to the persistent layer to persist in the life-log of the user and then sent to the requester.



*Fig: Service Curation Framework*