inteliops

# Online Dating Decision Services

## Problem

"Business logic" behind the online dating services is as follows:

Each person creates a profile defining his or her preferences.

The rules check the profiles to determine all the possible matches for a person.

The matches are scored.

Higher scores indicate a better match.

Scoring (once the matching criteria are met) is based on the age difference and the number of matching interests.

Each profile includes:

1. Name
2. Gender
3. City
4. Age
5. List of interests
6. Minimum and Maximum acceptable age
7. Acceptable genders
8. Minimum number of matching interests.

And here are the rules (applied to both persons):

Gender of the other person must be one of the acceptable genders.

Age of the other person must be within the acceptable range.

City must match exactly.

Matching interests of the other person must match at least the number specified.

An example of a compatible match:

Jane (age 26, lives in Seattle, interests are skydiving, knitting, reading) is looking for a male age 28-32 with at least one of those interests.

Jim (age 29, lives in Seattle, interests are skydiving, soccer, knitting) is looking for a female age 24-29 with at least two of those interests.

## InteliOps' Solution

### Overview

This solution is based on data-driven production system. Working memory is defined as the heap space used by the program to run the productions (rules). Facts

are instances of objects; in our case instances of class 'Person'. We create facts of type 'Person' and insert into the working memory so the rules can match on the pattern (left hand side), and when activated run the action (right hand side).

The fact type Person has 11 attributes that model the profile description provided in the problem section above. Note that this is a design decision. The match making rule looks finds two different profiles (facts) in the working memory, if any, and computes the scores. Next, this rule inserts a fact of type 'Match' into the working memory.

All the profile facts are inputs to the rules engine, and the match facts are the outputs.

### Facts

The fact type Profile is defined below. Note that most attributes are self-explanatory. However, we used Boolean values for gender preferences. The idea is that a bi-sexual person may prefer a partner from either gender.

```
declare Profile
        name : String
        gender : String
        city : String
        age : int
        minAcceptableAge : int
        maxAcceptableAge : int
        maleAcceptable : boolean
        femaleAcceptable : boolean
        interests : String
        minInterests : int
        minimalMathes : int
end
```

The fact type Match is defined below:

```
declare Match
        nameA : String
        nameB : String
        score : double
end
```

## Input data

Each row below represents a fact that is inserted into the working memory. Note that we decided to capture interests as a string with pipe (|) as delimiter. We assume the name is the unique key.

| name | gender | minimalMathes | femaleAcceptable | city | minAcceptableAge | maxAcceptableAge | minInterests | interests | maleAcceptable | age |
|---|---|---|---|---|---|---|---|---|---|---|
| Jane | F | 1 | FALSE | Seattle | 28 | 32 | 1 | sky diving\|knitting\|reading | TRUE | 26 |
| Jim | M | 2 | TRUE | Seattle | 24 | 29 | 2 | sky diving\|soccer\|knitting | FALSE | 29 |
| Natalie | F | 2 | FALSE | Aliso Viejo | 35 | 45 | 3 | soccer\|paddle boarding\|kayaking | TRUE | 31 |
| Ashley | F | 1 | FALSE | Aliso Viejo | 29 | 40 | 2 | soccer\|paddle board\|hiking | TRUE | 29 |
| Nathan | M | 2 | TRUE | Aliso Viejo | 24 | 29 | 2 | paddle boarding\|kayaking\|skiing | FALSE | 34 |
| Lisa | F | 1 | FALSE | Aliso Viejo | 25 | 30 | 3 | soccer\|skiing\|reading\|kayaking | TRUE | 23 |
| Ken | M | 1 | TRUE | Aliso Viejo | 21 | 25 | 1 | soccer\|kayaking\|skiing\|reading | FALSE | 29 |
| Flavio | M | 1 | TRUE | Aliso Viejo | 21 | 40 | 1 | soccer\|skiing\|reading | FALSE | 41 |
| Andrew | M | 1 | TRUE | Aliso Viejo | 21 | 35 | 1 | soccer\|kayaking\|paddle boarding\|reading | FALSE | 43 |

## Functions

We define two functions. The first one is a predicate that tests for common interests and returns a Boolean value. The other function computes the score. Note that interests string is split into an array. Both functions primarily iterate over the two arrays to compare. In more sophisticated implementations we could use Rete pattern matcher for improved performance.

```
function boolean commonInterests(String interests1, String interests2, int min) {
        boolean commonInterest = false;
        String [] firstSet = interests1.split("\\|");
        String [] secondSet = interests2.split("\\|");
         int count = 0;
        for(int i=0; i < firstSet.length; i++){
                for(int j=0; j < secondSet.length; j++){
                        if(firstSet[i].equals(secondSet[j])){
                                count++;
                                if(count >= min){
                                        commonInterest = true;
                                        break;
                                }
                        }
                }
        }
   return commonInterest;
}
```

```
function double score(Profile p1, Profile p2) {
        int age1 = p1.getAge();
        int age2 = p2.getAge();
        int ageDiff = java.lang.Math.abs(age1 - age2);
        String interests1 = p1.getInterests();
        String interests2 = p2.getInterests();
        String [] firstSet = interests1.split("\\|");
        String [] secondSet = interests2.split("\\|");
        int count = 0;
        for(int i=0; i < firstSet.length; i++){
                for(int j=0; j < secondSet.length; j++){
                        if(firstSet[i].equals(secondSet[j])){
                                count++;
                        }
                }
        }
        double score = 0.0;
        if(ageDiff != 0){
                score = count*100.0/ageDiff;
        }
        else{
                score = count*30.0;
        }
        @log.println("ageDiff = " + ageDiff + " count = " + count + " score = " + score);
         return score;
}
```

Note that the formula for score is random and may be replaced with a different one.

## Rules

The main rule matches two different profiles. As mentioned earlier, our fact model assumes name of a profile is unique; this will change in future implementations to handle a very large set of profiles, where names are likely to be repeated.

The left hand side of the rule matches on two fats of type Profile and binds them to variables 'p1' and 'p2'. Also, the first profile attributes' name, city, minAcceptableAge, and maxAcceptableAge are constrained in the second profile. The 'eval' construct invokes the predicate 'commonInterests' to match minimum number of interests from the first profile. Also, the pattern starts with any one profile and finds zero or more matches. Whenever a match is found the right hand side of the rule computes the score, creates a fact type 'Match' and inserts into the working memory.

```
rule "Match two profiles"
        when
                p1 : Profile(
                        nm : name,
                        ct : city,
                        min : minAcceptableAge,
                        max : maxAcceptableAge,
                        maleAcceptable,
                        int1 : interests,
                        minInt : minInterests)
                p2 : Profile(
                        name != nm,
                        gender == "M",
                        city == ct,
                        age >= min,
                        age <=max,
                        int2 : interests)
                eval( commonInterests(int1, int2, minInt) )
        then
                @log.println(drools.getRule().getName() + ": " + nm + " matches " +
p2.getName());
                Match match = new Match();
                match.setNameA(p1.getName());
                match.setNameB(p2.getName());
                match.setScore(score(p1, p2));
                 insert( match );
end
```

The second rule just prints out all the matches.

```
rule "Show all matches"
salience -1
  when
                allMatches : java.util.ArrayList() from collect (Match())
        then
                @log.println(drools.getRule().getName());
```

```
    for(Object o : allMatches){
      Match m = (Match) o;
      @log.println(m.getNameA() + " matches " + m.getNameB() + " with a score of
" + m.getScore());
    }
end
```

## Testing the solution

In order to test the solution the best way is to log into InteliOps AI platform. We can provide you a 30-day free access. Once you receive a promotion code from us, follow the instructions below.

1. Click on the link https://ai.inteliops.com/cloud/login
2. On the right hand side under "New to InteliOps" click 'Register' button.
3. Enter you first and last name, email address, and password of your choice, confirm the password, read terms and conditions, and once you agree click 'Continue' button.
4. On payment page check the box 'I have a promotion code', and click OK.
5. You will get to the Welcome page with information to login.
6. Upon logging in you will enter the dashboard for InteliOps AI platform; the dashboard shows following icons from left to right:
    a. Interactively develop rule engine applications
    b. Manage my RESTful web services
    c. Build and manage data mining and machine learning strategies
    d. View my account information
    e. Manage my profile (currently not enabled)

You will begin with the first link to start developing AI applications. Once you develop and test at least one project, you will be able to access the second and third features. You may access your account information (#4) any time.

### Trying our online dating solution

When you click the first icon you will enter the online IDE. The user guide is available on the right hand side pane.

#### Create project

1. Click File → New → Project
2. Enter "OnlineDating" as your project name, or any name you prefer. Note that the name must begin with upper case and should not have any spaces or special characters. Click "Create" button to create a vanilla project.
3. Once the project is created expand it by clicking the "+" sign preceding the project name.

### Create fact types

1. Click File → New → Declare Type
2. Enter "Profile" and click "Create" button.
3. In the editor, copy and paste the attributes of Profile from page #2 of this document; make sure you only copy the body (between 'declare' and 'end' lines) in the editor.
4. Click Build → compile to ensure that there are no errors.

### Upload data

1. Expand 'Data' under the project by clicking the "+" sign preceding the folder.
2. Right-click 'Profile.csv' and select 'Truncate'.
3. Expand 'Declare Types' under the project by clicking the "+" sign preceding the folder.
4. Right-click 'Profile' and select 'Import'.
5. When the window pops up click 'Choose File' button, select Profile.csv file (use the one we have provided) from the location on your computer file system, and click 'Upload' button. You will see the success message in the console below.

### Write functions

1. Click File → New → Function
2. Enter "commonInterests" and click 'Create' button.
3. In the editor first change the return type to 'boolean'.
4. Next, copy and paste the function definition from page#3 in this document.
5. Click Build → compile to ensure that there are no errors.
6. Repeat steps 1 through 5 for "score"; in step #3 instead of 'boolean' enter 'double'.

### Write rules

1. Click File → New → Rule
2. Enter "Match two profiles" and click 'Create' button.
3. In the editor, copy and paste after the line 'when' the entire definition from page#5 in this document; do not include 'end' in your copied lines because the editor enforces the rules syntax strictly.
4. Click Build → compile to ensure that there are no errors.
5. Repeat steps 1 through 4 for "Show all matches"; note that the definition id on page #6 instead of page #5.

### Run the project

In the top menu bar click the run (green arrow) button. You will see the rules on the agenda. Note that you have a choice to fire one or more rules at a time (Fire Next or Fire N) or fire all (Fire All). Depending on your data in Profile.csv files the rules will appear on the 'Agenda' (right hand side pane in the IDE.

You may inspect the working memory contents by clicking 'Search Facts' tab in the lower portion of the IDE. 'Console' tab will show debug outputs created by the 'Show all matches rule'.

## Conclusion

This solution is a simple pattern matching approach based on single fact type. It is very efficient. Using Rete algorithm based rules engine will be very fast when the number of facts grow to 10s of million, which will be the case for a successful online dating site. Also, maintaining a state-full working memory will produce real time matching decisions.