Distance Generator v1.0 Component Specification

1.Design

The Distance Generator is a custom component for TopCoder's web site. The component will calculate the distance between a member and a set of related members. The component will define at minimum three distances to calculate - rating, geographical, and match overlap. The component is responsible for calculating these distances, as well as weighting them to calculate an overall distance between the two members. After calculation, the component must deliver the results in XML, to provide a data feed for a user interface to display.

When calculation is performed, the component will retrieve a requested member, and a list of related members. For the purposes of any single calculation request, these are the only members that exist. The output of this component will be used in a graphical display, and the intention is to make each individual display as visually appealing and informative as possible. For this reason, this component does not calculate distance in the strict mathematical sense - the component does not define distance functions as symmetrical, for instance.

This component accomplishes the task by separating out the distance calculators from the weighting to allow either to vary independent of the others.

This component provides 3 different distance calculators:

1. A ratings calculator that will calculate the distance based on the ratings of each related member compared to the specified member
2. A geographical calculator that will calculate the distance based on geographical locations between the specified members and related members.
3. A match overlap calculator that will calculate the distance based on the matches that overlap with the specified member.

This component also provides 2 different weighting schemes:

1. An even weighting scheme that will weight all the calculation results evenly
2. A weighted average scheme that will apply a weighted average to each calculation result.

1.1Design Patterns

- Strategy pattern is use to allow the distance generator, member data access, distance calculators and weighting strategies to vary independently.

1.2Industry Standards

- XML

1.3Required Algorithms

1.3.1 Overall Process

The overall processing of the Distance Generator is:

This method will generate the distance from a coder to their related members based on the specified DistanceType(s) and CompetitionType(s). The results will be returned as an XML document. This method will:

1. Create a new List<Double> as memberResults
2. Get our Member by calling memberDataAccess.getMember(coder_id) as coder. Catch and wrap the MemberDataAccessException into a DistanceGeneratorException (creating appropriate ExceptionData) and throw that instead.
3. Get the related members by calling memberDataAccess.getRelatedMembers(coder_id, compTypes) as members. Catch and wrap the MemberDataAccessException into a DistanceGeneratorException (creating appropriate ExceptionData) and throw that instead.
4. Create a new HashMap<DistanceType, List<Double>> called distanceMap
5. For each distancetype in distanceTypes

   a) Get the distance calculator for the distance type

   b) If found and compTypes.size() > 0, call getDistance(coder, members, compTypes) and save the result to distanceMap using the distanceType as the key

   c) If not found or compTypes.size() == 0, create a new ArrayList(members.size()), fill with new Double(-1) and save the result to distanceMap using the distanceType as the key.
6. For each member in members keeping idx as the index in the members list.

   a) Create a new HashMap<DistanceType, Double> as memberMap

   b) For each DistanceType in distanceMap..

      aa) Get the List<Double> value in the distanceMap for the distance type

      ab) Get the double for the idx position within the list

      ac) Save the double to memberMap using the distanceType as the key

   c) Call distanceWeighting.weightDistance(memberMap)

   d) Add new Double(result) to memberResults
7. Call and return writeResult(members, memberResults)

1.3.2 The Rating Calculator:

The rating calculator will get the ratings for all the members for each Competition type. If there is more than one competition type, the resulting distance is calculated as the average of all competition types (ignore ones where all members are unrated):

1) Create an ArrayList<Double>(members.size()) as rcList

2) Create a HashMap<CompetitionType, Double>() as maxRatings

3) For each CompetitionType in compTypes

a) Call getMaxRating(members, compType) and store the result in maxRatings by the comptype if the result is > 0

4) Iterate each Member in members

a) Create a double, set to 0, called totalRating

b) Iterate the CompetitionType's found in maxgeographicallease note that maxRatings only contains those competition types where we have a valid maximum rating (IE it will not contain any all members are unrated types - which should be ignored).

ba) Get our rating via coder.getRating(compType) as ourRating

bb) Get the member rating via member.getRating(compType) as memberRating

bc) If ourRating > 0 and memberRating <= 0, add 1 to totalRating

bd) else If ourRating > 0 and memberRating > 0, add ((ourRating - memberRating) / maxRatings.get(compType))) to totalRating

be) else if ourRating <= 0 and memberRating <= 0, add 0 to totalRating (or simply continue)

bf) else add (memberRating / maxRatings.get(compType)) to the totalRating

c) If totalRating > 0, add new Double(totalRating / maxRating.size()) to rcList.

5) Return rcList.

1.3.3 The GeographicalCalculator:

The geographical calculator will calculate the distance as the member's distance in comparison to all members (ignoring those members that have no geographical location):

1) Create an ArrayList<Double>(members.size()) as rcList

2) Get our geographical distance by calling coder.getGeographicalDistance() as ourDistance

3) If ourDistance >= 0, then Get the maximum geographical distance by calling getMaxGeographicalDistance(members) as maxDistance - else set maxDistance to -1

4) Iterate each Member in members

a) If ourDistance is < 0, then add new Double(-1) to rcList

b) else if memberDistance is < 0, then add new Double(-1) to rcList

c) else add new Double(memberDistance / maxDistance) to rcList

5) Return rcList

1.3.4 The MatchOverlapCalculator

The match overlap calculator will calculate the distance by comparing how many common matches (the overlap) that the member was part of in relation to the all members:

1) Create an ArrayList<Double>(members.size()) as rcList

2) Call getMinMatchOverlap(members) to get the minimum overlap as minOverlap

3) Call getMaxMatchOverlap(members) to get the maximum overlap as maxOverlap

4) Iterate each Member in members

    a) Calculate the distance as (1 - ((member.getMatchOverlap() - minOverlap) / maxOverlap)) as dist

    b) If dist <= 0, set dist equal to 1

    c) Add new Double(dist) to rcList

5) Return rcList.

1.4Component Class Overview

DefaultDistanceGenerator

This class represents the default distance generator for the component. This generator will calculate the distance to all related members and return an XML string that represents the results.

DistanceCalculator

This interface defines the contract for a DistanceCalculator implementation. A distance calculator implementation will take a list of members and a set of CompetitionTypes and will return the distance for each member based on those competition types.

DistanceWeighting

This interface defines the contract for a DistanceWeighting implementation. A distance weighting implementation will take a map of distances (by their DistanceType) and calculate the overall distance for them.

EqualWeighting

This implementation of the Distance Weighting will weight the passed distances using an equal weighting algorithm. An equal weighting algorithm simply weights each distance type element equally.

WeightedAverageWeighting

This implementation of the Distance Weighting will weight the passed distances using an weighted average algorithm. An weighted average will provide an average of weight*distance (with any left over weight evenly distributed amount types).

RatingDistanceCalculator

This implementation of the DistanceCalculator will calculate distance based on the ratings.

GeographicalDistanceCalculator

This implementation of the DistanceCalculator will calculate distance based on the geographical distance from the coder.

MatchOverlapDistanceCalculator

This implementation of the DistanceCalculator will calculate distance based on the number of matches shared with the coder.

1.5Component Exception Definitions

DistanceGeneratorException

This exception represents an exception that occurred during distance generation.

DistanceCalculatorException

This exception represents an exception that occurred during distance calculation.

The developer should note that this exception is NOT thrown by any of the current implementations but is in place for future implementations.

DistanceWeightingException

This exception represents an exception that occurred during distance weighting.

The developer should note that this exception is NOT thrown by any of the current implementations but is in place for future implementations.

1.6Thread Safety

This component is thread safe by employing two strategies:

1. Most of the classes contain no state information and are thread safe because of that.
2. Some classes contain state information. However the state information is immutable both in reference and contents (for maps or lists). Additionally, the state that refers to other classes are thread safe in that those classes are either thread safe themselves or an interface whose contract is to be thread safe.

By employing both of these strategies, this component becomes thread safe.

2.Environment Requirements

2.1Environment

- At minimum, Java1.5 is required for compilation and executing test cases.

2.2TopCoder Software Components

- Base Exception 2.0 is used as the base for all custom exceptions for this component.

NOTE: The default location for TopCoder Software component jars

is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3Third Party Components

- *None*

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3.Installation and Configuration

3.1Package Name

com.topcoder.web.tc.distance – contains the main classes and interfaces

com.topcoder.web.tc.distance.data – contains the classes used for member access

com.topcoder.web.tc.distance.weighting – contains the weighting implementations

com.topcoder.web.tc.distance.calculators – contains the various distance calculators

3.2Configuration Parameters

No configuration needed

3.3Dependencies Configuration

None

4.Usage Notes

4.1Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2Required steps to use the component

Simply include the distribution file in the classpath.

4.3Demo

The following is a short demonstration of how to use the distance generator.

```
// Get the member data access implementations
MemberDataAcces mda = new FlatFileMemberDataAccess(...)

// Create a weighted average weighting scheme to weight
// OVERLAP distance by 20%, the GEOGRAPHICAL of 30%, and the RATING 50%
Map<DistanceType, Integer> weighting = new HashMap<DistanceType, Integer>();
weighting.add(DistanceType.OVERLAP, 20);
```

```
weighting.add(DistanceType.GEOGRAPHICAL, 30);
weighting.add(DistanceType.RATING, 50);

DistanceWeighting dw = new WeightedAverageWeighting(weighting);

// Create the various distance calculators
List<DistanceCalculator> dcs = new ArrayList<DistanceCalculator>();
dcs.add(new RatingDistanceCalculator());
dcs.add(new MatchOverlapDistanceCalculator());
dcs.add(new GeographicalDistanceCalculator());

// Create the distance generator
DistanceGenerator dg = new DefaultDistanceGenerator(mda, dw, dcs);

// Let's calculate distances on all of them
EnumSet<DistanceType> dtes  = EnumSet.Of(DistanceType.OVERLAP,
DistanceType.GEOGRAPHICAL, DistanceType.RATING);

// Let's calculate for only Design and Development
EnumSet<CompetitionType> ctes = EnumSet.Of(CompetitionType.DESIGN,
CompetitionType.DEVELOPMENT);

// Get the distances for coder_id 777
String xml = dg.generateDistance(777, dtes, ctes);

// Print out the results
System.out.println(xml);


// The above would result in something like:
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE catalog SYSTEM "compositions.dtd">
<Distance>
            <map>
                  <coder>
                        <coder_id>777</coder_id>
                        <handle>Stuff</handle>
                        <rating>2929</rating>
                        <image/>
                        <distance>0</distance>
                        <overlap/>
                        <country/>
                  </coder>
                  <coder>
                        <coder_id>2937</coder_id>
                        <handle>Blah</handle>
                        <rating>2939</rating>
                        <image/>
                        <distance>.6</distance>
                        <overlap/>
                        <country/>
                  </coder>
                  ... with other coders ..
            </map>
</Distance>
```

5.Future Enhancements

- Include more distance calculators
- Include more weighting types