Software Documentation : Distance Generator (.NET)

This page last changed on May 12, 2008 by adamselene.

1. Scope

1.1 Overview

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.

1.2 Logic Requirements

1.2.1 Distance Calculation

Distance between two members is defined as a floating point value from zero to one, inclusive. Larger values indicate further distance. Distances must always be calculated in a relative manner, to maximize the distinction between related members. The component must support at minimum the following calculations. Note that minimum and maximum functions mentioned below are local and only cover the current list of related members. You do not need to calculate absolute or global minimums and maximums.

1.2.1.1 Rating Distance

The component must support calculation of the distance between two members based on their ratings. Members may have more than one rating (for instance, one member may be rated in design, development, and algorithm competitions). The types of ratings available are: algorithm, development, design, marathon, and high-school. Placeholders will be provided for studio and assembly, but there will be no rating data available for these competitions. All rating data will provided by the data access interface.

When comparing a member to another member, these rules should be followed:

From (A)	To (B)	Value	Reason
Rated Member	Unrated Member	1.0	A rated member is equidistant to all unrated members of the type, with a non-zero distance. To maximize the effect of this distance, it will be considered the maximum distance (1.0).
Unrated Member	Rated member	B.rating / max(rating)	An unrated member is assumed to have a zero rating for determining the distance between

			themselves and rated members.
Unrated Member	Unrated Member	0.0	An unrated member is in the same state as another unrated member and may be assumed to be equidistant. The distance between them is assumed to be zero.
Rated Member	Rated Member	abs(A.rating - B.rating) / max(rating)	This function produces the relative distance between two members.

In the data provided to the component a rating less than or equal to zero should be considered as "unrated".

To compute the overall rating distance between one member and another member, the average (mean) distance must be computed, based on the individual distances per rating. If all provided members are unrated in a competition type, the distance should be dropped from the computation to prevent skewing the mean.

1.2.1.2 Geographical Distance

The component must support calculation of the distance between two members geographically. The geographic distance in uniform units will be provided by the data access interface. A geographical distance of zero indicates no geographical distance between two members, and a distance less than zero indicates an unknown geographical distance between the members.

From (A)	To (B)	Value	Reason
Known Member	Unknown Member	Undefined	The distance between a known and unknown member is undefined.
Unknown Member	Known member	Undefined	The distance between a known and unknown member is undefined.
Unknown Member	Unknown Member	Undefined	The distance between unknown members is undefined and the distance should not be included.
Known Member	Known Member	B.geoDistance / max(geoDistance)	This function produces the relative distance between two members. If max(geoDistance) is zero, the function should return zero, and not undefined.

1.2.1.3 Match Overlap

The component must support calculation of the distance between two members based on the number of shared matches. Note that this function is inverse (higher overlap indicates lower distance). Additionally, there is no "unknown" value for this data - a member will share zero or more matches with another member.

The recommended function for this calculation is 1 - ((B.overlap - min(overlap)) / max(overlap)). If max(overlap) is zero the function should return 1 and not undefined.

1.2.2 Distance Aggregation

The consumer of the component must be able to specify which of the above distances to calculate per request. When more than one type is specified, the distances must be averaged together. The component must use a weighted average to compute the final distance. The weighting should default to equal weight, but the consumer must be able to alter the weighting if desired.

For example, if the following distances were calculated with equal weighting:

Handle	Rating Distance	Geographical Distance	Final Distance
AdamSelene	1.0	0.0	0.5
dok	0.5	0.0	0.25
mess	0.25	undefined	undefined

If we were to give higher weighting to geographical distance, the final distances would decrease. Note that any aggregate distance which includes an undefined distance will be undefined.

Unused weight must be re-distributed evenly. For example, if the default weightings are GEO=40, OVR=40, RAT=20, if we only calculate GEO and OVR, we redistribute the 20 points evenly between the other two weights - which is 40+10, or 50. Similarly, for OVR and RAT, we redistribute GEO to result in 60/40.

1.2.3 Data Access Interface

The data access interface for the component is provided. A default implementation is provided to designers and developers as an attachment. The component must support pluggable data access implementation, which may include updates to the default implementation provided. However, the designer may not change the basic interfaces provided (MemberDataAccess or Member). Source files are provided with the predefined interfaces and implementations. The designers do not need to document the default implementation, nor are they required to include in it their UML diagrams.

1.2.4 Output

At minimum, the component must support the following interface for generating distances for a specific member:

String GenerateDistanceXml(long coderId, DistanceTypes distanceType, CompetitionTypes compType);

coderId is the long, unique identifier for the coder to generate distances for. DistanceTypes is an enumeration of the three required distance calculations described above. CompetitionTypes is an enumeration of the competition types to generate data for (which is handled by the data access interface).

1.2.4.1 Format

The method must return XML. An XSD for the XML output is attached. The first coder element in the output must always be the requested member.

Any member with an undefined distance must not be included in the output.

1.2.4.2 Details

The elements in the XSD map as follows:

Element	Content
coder_id	Member.Id
handle	Member.Handle
rating	Member.MaxRating
image	Member.Image
distance	The calculated distance or zero for the first member.
overlap	Member.MatchOverlap
country	Member.Country

1.2.5 Thread Safety

The distance generator will be used in a web application environment; while instances will likely not be shared across threads the component will be independently called by many threads. Distance generation must not depend on static mutable data between instances.

1.3 Required Algorithms

The algorithms for distance calculation and aggregation must be clearly specified.

1.4 Example of the Software Usage

This component will provide the data for a graphical map of members related to a specific member, and how "close" they are in certain terms.

1.5 Future Component Direction

Future versions will include more distance types - for instance the difference in volatility, the average time to submit an algorithm problem, submission score similarity, and so forth.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

The component must support the interfaces and XSDs attached. A quick summary follows, but please use the attached code files for full documentation.

```
public interface IDistanceGenerator {
String GenerateDistanceXml(long coderId, DistanceTypes distanceType, CompetitionTypes compType);
}
```

```
public interface IMemberDataAccess
{
Member GetMember(long id);
IList<Member> GetRelatedMembers(long id, CompetitionTypes competitionType);
}
```

```
[Flags]
public enum DistanceTypes {
```

```
Overlap,
Country,
Rating
}
```

```
[Flags]
public enum CompetitionTypes {
Algorithm,
Development,
Design,
Studio,
Marathon,
Assembly,
HighSchool
}
```

```
public class Member
{
public String Handle { get; }
public int MaxRating { get; }
public long Id { get; }
public int GeographicalDistance { get; }
public int MatchOverlap ( get; }
public String Country { get; }
public String Image { get; }
public int GetRating(CompetitionTypes type);
}
```

2.1.3 Environment Requirements

- Development language: .NET 2.0
- Compile target: .NET 2.0 and .NET 3.0

2.1.4 Package Structure

TopCoder.Web.Distance
TopCoder.Web.Distance.Data

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

No elements explicitly need to be configurable. The designer may include configuration at their discretion.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None.

3.2.2 TopCoder Software Component Dependencies:

There are no explicit dependencies. The designer may choose to include catalog components at their discretion.

3.2.3 Third Party Component, Library, or Product Dependencies:

None.

3.2.4 QA Environment:

- Solaris 9
- RedHat Linux Enterprise 4
- Windows XP

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of TC UML Tool.