# Table of Content

1. **Introduction** ............................................................................................................................... 2  
   1.1. Overview ..................................................................................................................................... 2  
   1.2. Purpose ....................................................................................................................................... 2  
   1.3. Scope ......................................................................................................................................... 2  
   1.4. References ............................................................................................................................... 3  
2. **Architectural Design** .................................................................................................................. 4  
   2.1. Rationale .................................................................................................................................... 4  
   2.2. Software Architecture Model .................................................................................................... 4  
3. **Software Interface Design** .......................................................................................................... 4  
4. **Packages and Classes** .................................................................................................................. 5  
5. **XML File Description** ................................................................................................................ 10  
   5.1. XML File Format ....................................................................................................................... 10  
   5.2. ROM XML Example .................................................................................................................. 11  
6. **The Matrices- Base Text-Partitioning Algorithm Pseudo Code** ............................................... 14  
   6.1. I. General steps of the Matrices- Base Text-Partitioning Algorithm ........................................... 14  
   6.2. II. Equivalent Clauses ............................................................................................................... 15  
   6.3. Glossary ..................................................................................................................................... 16  
7. **Acceptable Criteria** ..................................................................................................................... 16  
   7.1. Test Cases .................................................................................................................................. 16  
8. **Glossary** ..................................................................................................................................... 17
1. Introduction

1.1. Overview

This Software Design Document (SDD) gives a general outline of the architecture and design of the Metric-based text-partitioning algorithm which facilitate the identification of Summary-level Use-cases description from the requirement text.

1.2. Purpose

This SDD describes the overall design of the Summary-level Use-Cases descriptions extracted from requirement text by using text partitioning algorithm. It is primarily written for the developers of the READ Project and future references as well as the software users.

1.3. Scope

The goal is to group the sentences in the requirement text by the Metric-based text-partitioning algorithm in order to facilitate the identification of Summary-level Use-cases description from the text.

The Program essential functionality is getting the improved requirement text from which the ambiguity has been removed with the help of the client [4], saving it in XML format and generating the similarity and dissimilarity tables as well as measuring the distances. It is worth mentioning that the text has already undergone the NLP text Quality assessment.
Such a grouping increases the visibility of a service in the text paragraphs of the original text. This will facilitate the job of the analysts in ensuring the completeness of the use-case descriptions and verifying the text for possible inconsistencies between otherwise scattered statements.

1.4. References


2. Architectural Design

2.1. Rationale

As we continue with the development of the visualization functionality of the READ project [2], the eclipse open development platform was chosen to develop the Matrices-base Text-partitioning Algorithm using Object-oriented methodology and the Java programming language.

2.2. Software Architecture Model

The Metric-based text-partitioning algorithm is divided into four packages:

- `com.Shuang.read.ucip`
- `com.ishrar.adetect.common`
- `com.ishrar.adetect.factory`
- `com.ishrar.adetect.parser`

3. Software Interface Design

This portion of the project is internal to the system and therefore does not considerably involve interaction with the user and as a result there is no direct interface. Once it gets integrated, the interface will be that of the EVP [2].
4. Packages and Classes

- com.ishrar.adetect.common

<table>
<thead>
<tr>
<th>SentenceDelimiter Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inherits from</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SentenceParser Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inherits from</strong></td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
<tr>
<td><strong>Methods</strong></td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>public</td>
</tr>
</tbody>
</table>
• **com.ishrar.adetect.factory**

<table>
<thead>
<tr>
<th>SentenceParserFactory Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherits from</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Visibility</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td></td>
<td>makeSentenceParser</td>
<td></td>
</tr>
</tbody>
</table>

• **com.ishrar.adetect.parser**

<table>
<thead>
<tr>
<th>ParseSentence Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inherits from</td>
</tr>
<tr>
<td><strong>Description</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Visibility</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td></td>
<td>initialize</td>
<td>Implements LexicalizedParser and initializes the maximum length of the input text</td>
</tr>
<tr>
<td>private</td>
<td></td>
<td>process</td>
<td>Process a sentence using POS tagger, and return the text with POS tagger.</td>
</tr>
<tr>
<td>public</td>
<td></td>
<td>parse</td>
<td>Returns the parsed text with POS tag.</td>
</tr>
<tr>
<td>public</td>
<td></td>
<td>getwordCount</td>
<td>Returns the number of words being processed.</td>
</tr>
</tbody>
</table>

### Distance class

<table>
<thead>
<tr>
<th>Inherit from</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Stores the number of the sentences and the distances</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Visibility</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>setKey</td>
<td></td>
<td>Sets the number of sentences.</td>
</tr>
<tr>
<td>public</td>
<td>setDistance</td>
<td></td>
<td>Sets the distances.</td>
</tr>
<tr>
<td>public</td>
<td>getKey</td>
<td></td>
<td>Returns the number of sentences.</td>
</tr>
<tr>
<td>public</td>
<td>getDistance</td>
<td></td>
<td>Returns the distances.</td>
</tr>
</tbody>
</table>

### equivalenceClass class

<table>
<thead>
<tr>
<th>Inherit from</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>stores number of the main sentences and the equivalence classes of the remaining sentences.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Visibility</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>addSet</td>
<td></td>
<td>Adds a remaining sentence to the corresponding equivalence class.</td>
</tr>
<tr>
<td>public</td>
<td>setKey</td>
<td></td>
<td>Sets the number of the main sentences.</td>
</tr>
<tr>
<td>public</td>
<td>getKey</td>
<td></td>
<td>Returns the number of sentences.</td>
</tr>
<tr>
<td>public</td>
<td>getSet</td>
<td></td>
<td>Returns the set of the number of remaining sentences.</td>
</tr>
</tbody>
</table>
### output class

<table>
<thead>
<tr>
<th>Inherits from</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Visibility</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Main</td>
<td></td>
<td>Calls other methods in UCIdentify.</td>
</tr>
</tbody>
</table>

### UCIdentify class

<table>
<thead>
<tr>
<th>Inherits from</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Includes all the operations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Visibility</th>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>readFromTxt</td>
<td></td>
<td>Reads the original description to the memory.</td>
</tr>
<tr>
<td>public</td>
<td>improve</td>
<td></td>
<td>Ignores the redundant words in each sentence.</td>
</tr>
<tr>
<td>public</td>
<td>getActors</td>
<td></td>
<td>Gets the primary actor from primaryActors.xml and stores in the memory.</td>
</tr>
<tr>
<td>public</td>
<td>getSupportActors</td>
<td></td>
<td>Gets the secondary actor from supportActors.xml and store in the memory.</td>
</tr>
<tr>
<td>public</td>
<td>getTriggerEvents</td>
<td></td>
<td>Gets the Triggering Events from triggerEvents.xml and stores in the memory.</td>
</tr>
<tr>
<td>public</td>
<td>getPredefined</td>
<td></td>
<td>Gets the Predefined</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public getUC</td>
<td>Gets the potential use case context.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public getSet</td>
<td>Divides all the sentences into main sentences and remaining sentences.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public containCompare</td>
<td>Compares the potential use case context with the original text.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public getDistance</td>
<td>Calculates the distances between two sentences.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public getSubject</td>
<td>Gets the subject from the input sentence.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public getAllDistance</td>
<td>Gets the pair between the main and remaining sentence’s distance and places the remaining sentences in the corresponding equivalence class.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public writeXML</td>
<td>Writes the equivalence class to UCContext.xml file.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public minValue</td>
<td>Returns the minimum distance.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public isMin</td>
<td>Checks to see whether the input vector has a minimum distance.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Keywords from triggerEvents.xml and stores in the memory.
5. XML File Description

5.1. XML File Format

- Any file name can be used as long as the extension is .xml.

- The root element is used to indicate the model name.

- The children of the root element are all <Entity> elements.

- The possible attributes of an Entity element are:
  - Ent-Name: The name of a particular entity.
  - Ent-Attr: An attribute associated with the entity.
  - Relation: A relationship between different entities.

- There are six different relation types (see following section for examples):
  - association-1
    - e.g. `<Relation type="association-1" target="Order">create</Relation>`
  - association-2
    - e.g. `<Relation type="association-2" d-target="Order" itarget="InvoiceOrderSystem" ref-attr="ID">enter</Relation>`
  - association-3
    - e.g. `<Relation type="association-3" d-target="InvoiceOrderSystem" itarget="Order" ref-rel="cancel">request</Relation>`
  - generalization
    - e.g. `<Relation type="generalization" target="Invoice"> </Relation>`
  - modification
    - e.g. `<Relation type="modification" target="ID">modifies</Relation>`
  - of-preposition
    - e.g. `<Relation type="of-preposition" target="Product">of</Relation>`
5.2. ROM XML Example

```xml
<?xml version="1.0" encoding="UTF-8" ?>

<CBMsys>
  <Entity>
    <Ent-Name>customer</Ent-Name>
    <Relation type="association-3" d-target="CBMSys" i-target="order" ref-rel="place">request</Relation>
    <Relation type="association-2" d-target="payment" i-target="CBMSys" ref-attr="information">provide</Relation>
    <Relation type="association-2" d-target="order" i-target="CBMSys" ref-attr="status">view</Relation>
    <Relation type="association-2" d-target="order" i-target="CBMSys" ref-attr="Id">enter</Relation>
    <Relation type="association-2" d-target="profile" i-target="CBMSys" ref-attr="">update</Relation>
  </Entity>

  <Entity>
    <Ent-Name>publisher</Ent-Name>
    <Relation type="association-2" d-target="profile" i-target="CBMSys" ref-attr="">update</Relation>
  </Entity>

  <Entity>
    <Ent-Name>CBMSys</Ent-Name>
    <Relation type="association-1" target="order">place</Relation>
    <Relation type="association-2" d-target="credit record" i-target="Customer Persistent Storage">retrive</Relation>
    <Relation type="association-2" d-target="order" i-target="publisher">create</Relation>
    <Relation type="association-2" d-target="order" i-target="publisher">send</Relation>
    <Relation type="association-2" d-target="invoice" i-target="publisher">receive</Relation>
    <Relation type="association-2" d-target="purchase invoice" i-target="Account Payable System">accept</Relation>
  </Entity>
</CBMsys>
```
<Relation type="association-2" d-target="purchase invoice" i-target="Account Payable System">send</Relation>

<Relation type="association-1" target="payment">prepare</Relation>

<Relation type="association-2" d-target="check" i-target="publisher">send</Relation>

<Relation type="association-2" d-target="shipment" i-target="order">assign</Relation>

<Relation type="association-2" d-target="sale invoice" i-target="customer">generate</Relation>

<Relation type="association-2" d-target="payment" i-target="Account Receivable System">send</Relation>

<Relation type="association-1" target="order">show</Relation>

</Entity>

- <Entity>
  <Ent-Name>order</Ent-Name>
  <Ent-Attr>status</Ent-Attr>
  <Ent-Attr>id</Ent-Attr>
  <Relation type="association-1" target="Id">has</Relation>
</Entity>

- <Entity>
  <Ent-Name>credit record</Ent-Name>
  <Ent-Attr>good</Ent-Attr>
  <Relation type="of-preposition" target="customer">of</Relation>
</Entity>

- <Entity>
  <Ent-Name>customer persistent storage</Ent-Name>
</Entity>

- <Entity>
  <Ent-Name>invoice</Ent-Name>
</Entity>

- <Entity>
  <Ent-Name>detail</Ent-Name>
</Entity>
<Ent-Attr>correct</Ent-Attr>

</Entity>

- <Entity>
  <Ent-Name>Account Payable System</Ent-Name>
  </Entity>

- <Entity>
  <Ent-Name>payment</Ent-Name>
  <Ent-Attr>information</Ent-Attr>
  <Relation type="of-preposition" target="publisher">of</Relation>
  </Entity>

- <Entity>
  <Ent-Name>check</Ent-Name>
  </Entity>

- <Entity>
  <Ent-Name>shipment</Ent-Name>
  </Entity>

- <Entity>
  <Ent-Name>Account Receivable System</Ent-Name>
  </Entity>

- <Entity>
  <Ent-Name>profile</Ent-Name>
  <Ent-Attr>personal</Ent-Attr>
  <Relation type="modification" target="information">modifies</Relation>
  </Entity>

- <Entity>
  <Ent-Name>purchase invoice</Ent-Name>
  <Relation type="of-preposition" target="detail">of</Relation>
  </Entity>
6. The Matrices- Base Text-Partitioning Algorithm Pseudo Code

6.1. I. General steps of the Matrices- Base Text-Partitioning Algorithm

**Step A:** Find the main sentences list $SS$.

**Step B:** Find the equivalent classes related to Step A sentences.

**Procedure for Step A**

**Input:**  
- IA1. Text  
- IA2. XML file of the improved view  
- IA3. XML Primary Actors  
- IA4. XML Triggering Events  
- IA5. XML Attributes

**Output:**  
- OA1. List of the main sentences $SS$  
- OA2. List of the remaining sentences $RR$

**Procedure A:**

**StepA1. Begin**

If Pattern 1($Si$) or Pattern 2($Si$) then add to main sentences list $SS$

Else add $Si$ to the remaining sentences list

**End**
6.2. II. <Equivalent Clauses>

Procedure for Step B

Input:  
IB1. SS  
IB2. RR

Output:  
OB1. Set of Partitions

Procedure B:

Step B1. Map each sentence to tables, Similarity and Dissimilarity

Step B2. If SS was not empty then choose one of the main sentences, MSn, put it in SS' and remove from SS.  
Else do to Step B5.

Step B3. For each remaining sentences, RSj, in the remaining sentences list, RR, Do calculate distance between MSn and RSj

End for

Step B4. Go to step B2

Step B5. For each RSj in RR Do

If there exist a Unique MIN distance between RSj and MSn then

Begin

1. Add RSj to the Partition of MSn  
2. Remove RSj from RR

End if

End for

Step B6. For each RSj in RR do //Not a Unique MIN

1. Identify a set T in MS to which the RSj has equal MIN distance.  
2. Identify the correspondences to T partitions, Set P.  
3. Find MIN distance between RSj and Partition i in P, MINij  
4. Repeat step 3 for all Partitions in P.
5. Add $RS_j$ to the Partition of $MS_n$ with $MIN_i$

End for

6.3. Glossary

- **MS_i**: Main Sentence #n
- **MIN**: Minimum Distance
- **MIN_{ji}**: Minimum Distance between Partition i and RS_j
- **P**: Set corresponding to T Partition
- **RR**: Remaining Sentences Set
- **RS_i**: Remaining Sentence #j
- **Si**: Sentence #i
- **SS**: Main Sentences Set
- **T**:

7. Acceptable Criteria

7.1. Test Cases

I. **Generate Improved Text.**

   **Purpose**: To ignore redundant words of the sentence.

   **Input**: Original recruitment text.

   **Output**: Improved text.
II. Generate Potential Use Case.

Purpose: To obtain potential use case.

Input: CBMSYs.xml.

Output: Potential use case.

III. Generate Two Sets.

Purpose: To generate the main sentences set and the remaining sentences set.

Input: Improved text and potential use case.

Output: Two sets which separately store the sentences number.

IV. Obtain Equivalence Class.

Purpose: To calculate all the distances between the two sets and place the remaining sentence into the corresponding equivalence class.

Input: Two sets and improved text.

Output: Equivalence class.

8. Glossary

- ECC: Expert Comparable Contextual
- EVP: Eclipse Visualization Plug-in
- READ: Requirement Engineering Assistance Diagnostic
- SDD: Software Design Document
- SUD: System Under Development
- XML: Extensible Markup Language