|  |  |
| --- | --- |
|  |  |

**Leo**

**NLP Developer Guide**

Ver. 1.0 of October 2013

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | | Author | Description of Revision |
| Date | Number |
| 10/2013 | 1.0 | Tim Trautman  Thomas Ginter | Original |
|  |  |  |  |

Contents

[1. Introduction 1](#_Toc368401528)

[2. Leo Overview 1](#_Toc368401529)

[3. Implementing Leo 2](#_Toc368401530)

[3.1 Pre-requisites 2](#_Toc368401531)

[3.2 Implementation 2](#_Toc368401532)

[Appendix A - Leo Service Diagram 4](#_Toc368401533)

[Appendix B - LeoAEDescriptor Diagram 5](#_Toc368401534)

[Appendix C - Leo Client Diagram 6](#_Toc368401535)

[Appendix D - Collection Readers Diagram 7](#_Toc368401536)

[Appendix E - About Listeners 8](#_Toc368401537)

[Appendix F - Sample Client 1 code 9](#_Toc368401538)

[Appendix G - Sample Service 1 code 11](#_Toc368401539)

[Appendix H - Sample Client 2 code 14](#_Toc368401540)

[Appendix I - Sample Service 2 code 17](#_Toc368401541)

[Appendix J. Definitions and Acronyms 20](#_Toc368401542)

# Introduction

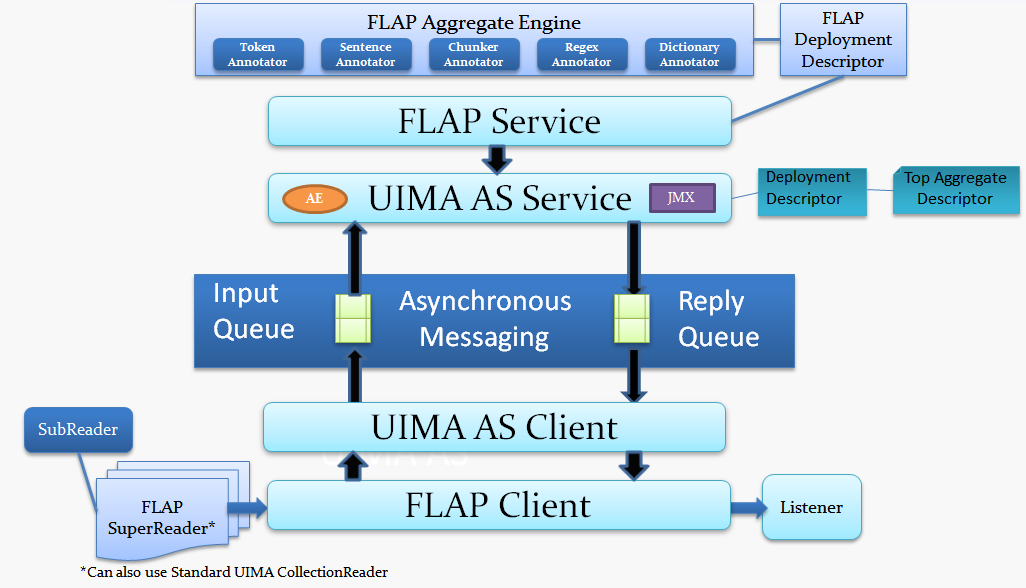
The Leo framework functionality is a Java-based program that utilizes [Unstructured Information Management Architecture (UIMA)](http://uima.apache.org/) - [Asynchronous Scaleout (AS)](http://uima.apache.org/doc-uimaas-what.html) Client - Service architecture for V3NLP development.

Some of the features of Leo are:

* Rapid Natural Language Processing (NLP) Development
* Programmatically Generate:
  + UIMA Descriptors
  + Parameter and Type information for Analysis Engine (AE)
  + Aggregate AE for a Service
  + Configuration and Service Launch
  + Standard and Custom CollectionReaders
* Plug and Play Algorithm Modules
* Launch Services and Clients in UIMA-AS

# Leo Overview

Below is a diagram of the Leo program’s services and client architecture.



Insert more information here.

# Implementing Leo

## Pre-requisites

* UIMA-AS 2.3.1 or higher
* Java 1.6 or higher
* Java Development environment (Eclipse or IntelliJ)
* Maven

## Implementation

1. Set-up parameters
   1. Add *http://decipher.chpc.utah.edu/nexus* as a mirror in Maven *Settings.xml*
   2. Add Leo dependency to Maven *POM.xml* - retrieve from [*http://decipher.chpc.utah.edu/nexus/index.html*](http://decipher.chpc.utah.edu/nexus/index.html)by searching for the latest version of “Leo”.
2. Create the service

38 server = new gov.va.vinci.Leo.Service(); /\* Step 1 - create the service object \*/

39 server.setInputQueueName(SERVICE\_QUEUE\_NAME); /\* Step 2 - name of the input queue for ActiveMQ \*/

40 server.setBrokerURL(BROKER\_URL); /\* Step 3 - set the ActiveMQ broker URL \*/

41 server.setCasPoolSize(CAS\_POOL\_SIZE); /\* Step 4 - set the size of the input and output queues \*/

1. Create the Type system

142 LeoTypeSystemDescription types = new LeoTypeSystemDescription(); /\* Step 1 - create a Type system object \*/

143

144 /\* Step 2 - Add annotation Types to the Type system \*/

145 types

146 .addType("gov.va.vinci.demo.types.Alcohol", "Annotation for alcohol extraction.", "uima.tcas.Annotation")

147 .addType("gov.va.vinci.demo.types.PastUse", "Annotation for past use extraction.", "uima.tcas.Annotation")

148 .addType("gov.va.vinci.demo.types.Current", "Annotation for past use extraction.", "uima.tcas.Annotation")

149 .addType("gov.va.vinci.demo.types.Never", "Annotation for past use extraction.", "uima.tcas.Annotation")

150 .addType("gov.va.vinci.demo.types.Exclude", "Annotation for past use extraction.", "uima.tcas.Annotation")

151 .addType("gov.va.vinci.demo.types.Window", "Annotation for window extraction.", "uima.tcas.Annotation");

152 if (generateTypes) /\* Step 3 - If required, generate the java and class files for the Type system \*/

153 types.jCasGen("src/main/java", "target/classes");

1. Create the aggregate descriptor

84 LeoAEDescriptor pipeline = new LeoAEDescriptor();

1. Create and add the delegates

88 pipeline.addDelegate (

new LeoAEDescriptor() /\* Step 1 - create a new analysis engine descriptor \*/

89 .setName("AlcoholRegexAnnotator") /\* Step 2- give new analysis engine descriptor a name \*/

90 .setImplementationName(RegexAnnotator.class.getCanonicalName()) /\* Step 3 - tell descriptor which Java class will be executed \*/

91 .addParameterSetting("resource", true, false, "String", "src/main/resources/alcohol.regex") /\* Step 4 - Add required parameters \*/

92 .addParameterSetting("outputType", true, false, "String", "gov.va.vinci.demo.types.Alcohol") /\* Step 4 - Add required parameters \*/

93 .addTypeSystemDescription(types)) /\* Step 5 - Add the Type system \*/

1. Deploy the service

53 server.deploy(pipeline);

1. Create a Client

64 gov.va.vinci.Leo.Client myClient = new gov.va.vinci.Leo.Client(); /\* Step 1 - create client object \*/

65 myClient.setBrokerURL(BROKER\_URL); /\* Step 2 - name of the input queue for ActiveMQ \*/

66 myClient.setInputQueueName(SERVICE\_QUEUE\_NAME); /\* Step 3 - set the ActiveMQ broker URL \*/

67 myClient.setCasPoolSize(CAS\_POOL\_SIZE); /\* Step 4 - set the size of the input and output queues \*/

1. Create the Listener

52 XmiUABListener xmiListener = new XmiUABListener(outdir); /\* This listener outputs each CAS to xmi for review \*/

1. Create the Reader

56 CollectionReader myReader = CollectionReaderFactory.generateFileSubReader( /\* From the collection reader factory \*/

57 indir, /\* input directory -- all files in that directory will be processed. \*/

58 "UTF-8", /\* Text encoding \*/

59 false, /\* Recurse the input directory \*/

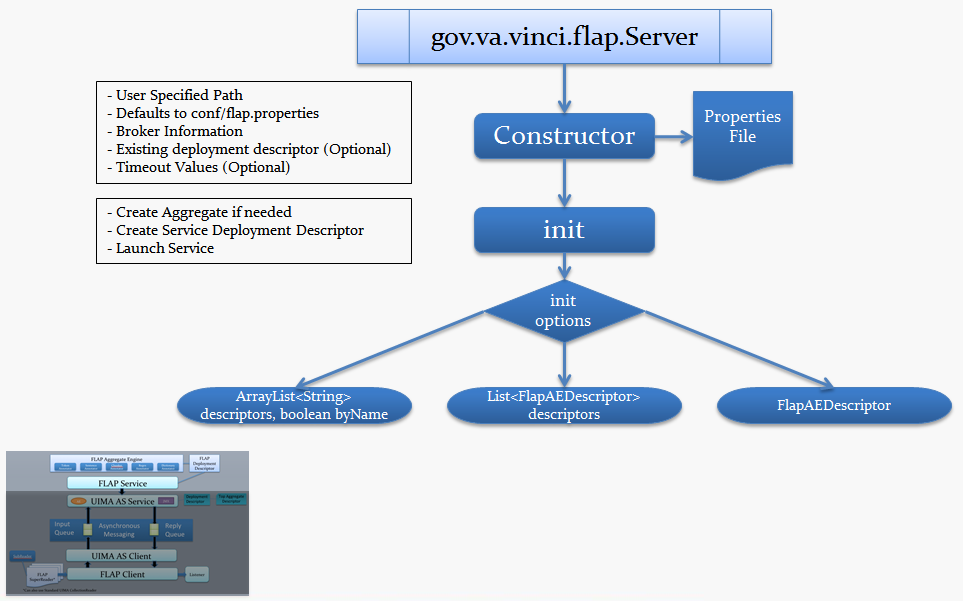
60 "gov.va.vinci.Leo.tools.AsciiFilter", "gov.va.vinci.Leo.tools.XmlFilter"

61 /\* Leo filters to deal with possible unusual characters in text \*/);

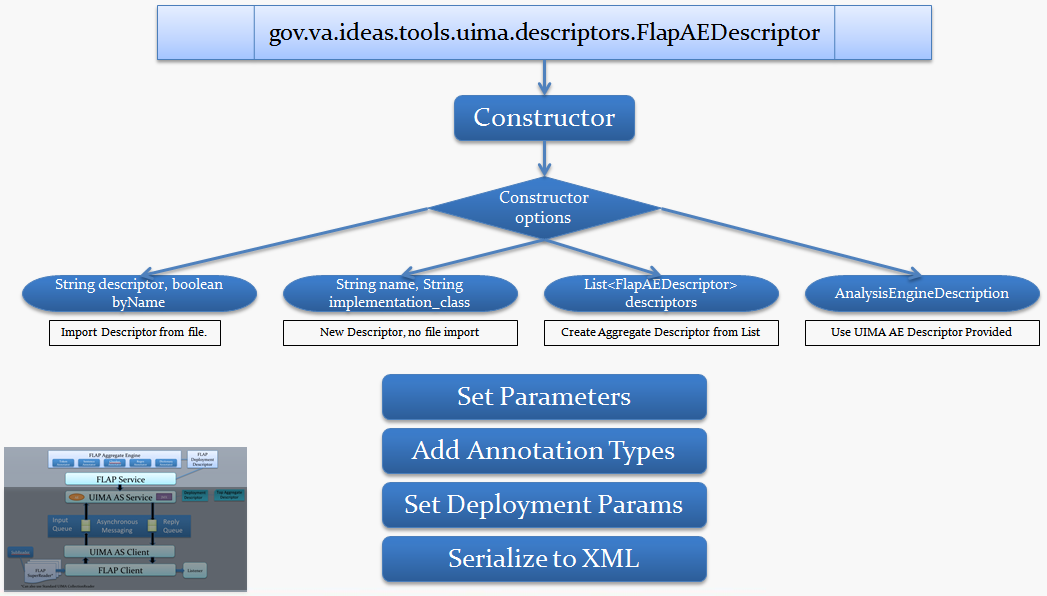
1. Launch the Client

69 myClient.run(myReader, xmiListener);

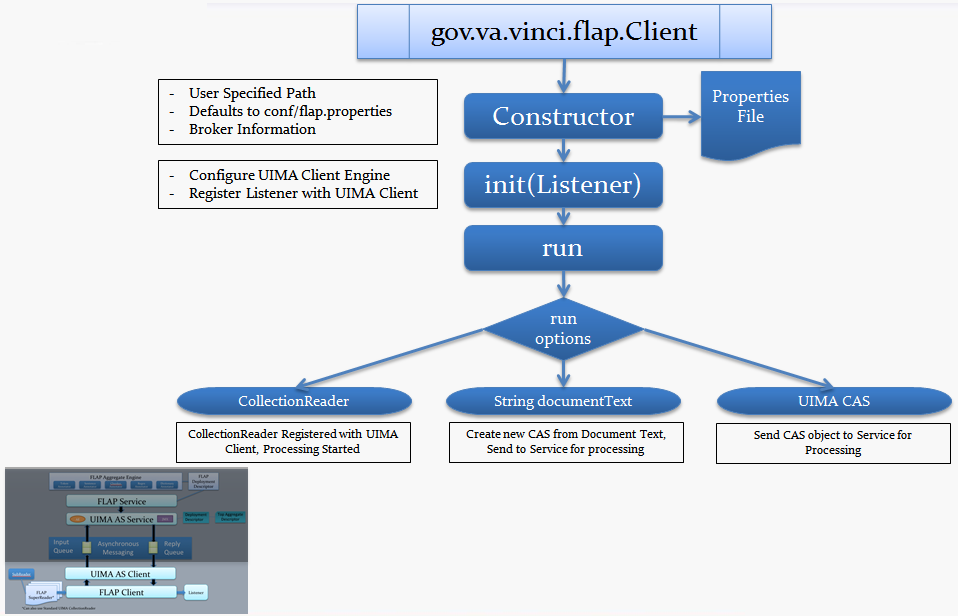
# Appendix A - Leo Service Diagram



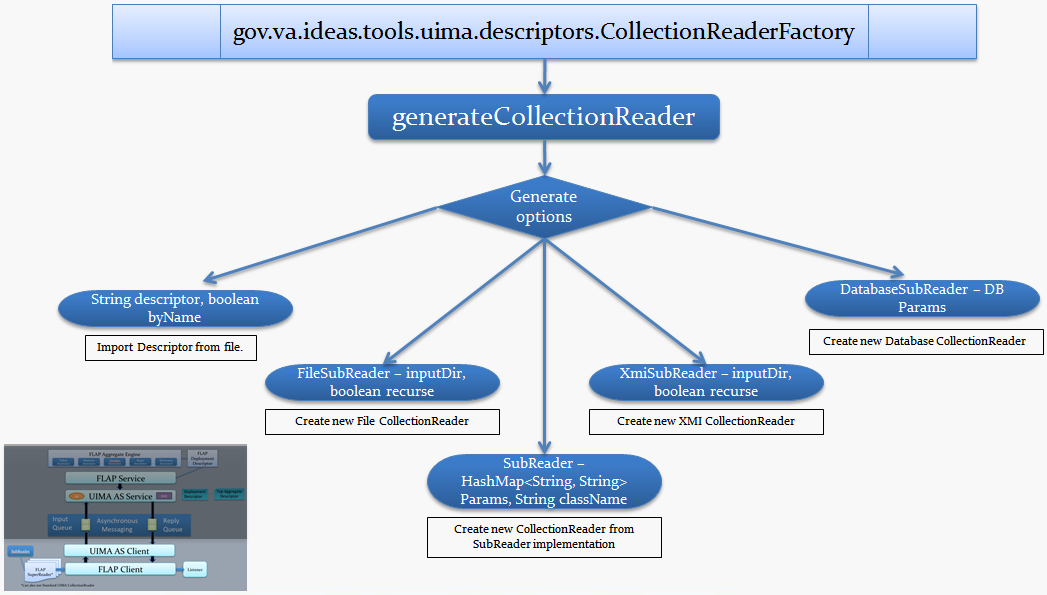
# Appendix B - LeoAEDescriptor Diagram



# Appendix C - Leo Client Diagram



# Appendix D - Collection Readers Diagram



# Appendix E - About Listeners

* UimaAsBaseListener abstract class
* Listener Events:
  + Collection processing starts
  + CAS is being sent up to the service
  + CAS is returned from the service
  + Collection processing complete
* Leo Listeners
  + XmiUABListener,
    - Serializes CAS files to XMI output
    - Launches UIMA Annotation Viewer for collection processing complete event
  + DatabaseUABListener,
    - Output annotations to an annotation DB
    - Internally developed schema

# Appendix F - Sample Client 1 code

Parameters that can or must be changed are highlighted in yellow along with instructions for that parameter as code comments.

1 package gov.va.vinci.demo.task1;

2

3 import gov.va.vinci.demo.listeners.CSVListener;

4 import gov.va.vinci.Leo.client.DataQueryInformation;

5 import gov.va.vinci.Leo.client.DatabaseConnectionInformation;

6 import gov.va.vinci.Leo.client.DatabaseMultiBatchClient;

7 import gov.va.vinci.Leo.descriptors.CollectionReaderFactory;

8 import gov.va.vinci.Leo.listener.XmiUABListener;

9

10 import java.io.File;

11 import java.io.FileWriter;

12 import java.io.IOException;

13 import java.util.Date;

14 import org.apache.commons.lang.time.StopWatch;

15 import org.apache.uima.collection.CollectionReader;

16

17 public class Task1Client {

18 /\* Parameters that link the client to the service \*/

19 protected static String BROKER\_URL = Task1Service.BROKER\_URL;

20 protected static String SERVICE\_QUEUE\_NAME = Task1Service.SERVICE\_QUEUE\_NAME;

21 protected static int CAS\_POOL\_SIZE = Task1Service.CAS\_POOL\_SIZE;

22 /\* The input and output path can be either relative to the root of the project

23 \* or absolute path locally or remotely \*/

24 protected static String indir = "data//input//";

25 protected static String outdir = "data//output//Demo1//xmi//";

26 protected static String outcsv = "data//output//Demo1//AlcoholUseClassification.csv";

27

28 /\*\*

29 \* @param args

30 \* @throws IOException

31 \*/

32 public static void main(String[] args) throws IOException {

33 StopWatch sw = new StopWatch();

34 sw.start();

35 System.out.println(" Starting Task 1 client at " + new Date(sw.getStartTime()));

36

37 if(!(new File(outdir)).exists()) (new File(outdir)).mkdirs();

38 /\* Toggle comment lines to change input method from text file to database readers \*/

39 //runFileReaderClient();

40 runDatabaseReaderClient();

41 sw.stop();

42 System.out.println("Processing time: " + sw.toString() + "\n" +

43 "Processing ended at: " + new Date(System.currentTimeMillis()));

44 }

45

46 /\*\*

47 \*

48 \*/

49 private static void runFileReaderClient() {

50 try {

51 /\* Specify the type of listeners. \*/

52 XmiUABListener xmiListener = new XmiUABListener(outdir);

53 xmiListener.setLaunchAnnotationViewer(true);

54

55 /\* Specify collection reader -- file reader \*/

56 CollectionReader myReader = CollectionReaderFactory.generateFileSubReader(

57 indir, /\* input directory -- all files in that directory will be processed. \*/

58 "UTF-8", /\* Text encoding \*/

59 false, /\* Recurse the input directory \*/

60 "gov.va.vinci.Leo.tools.AsciiFilter", "gov.va.vinci.Leo.tools.XmlFilter"

61 /\* Leo filters to deal with possible unusual characters in text \*/);

62

63 /\* Create a client \*/

64 gov.va.vinci.Leo.Client myClient = new gov.va.vinci.Leo.Client();

65 myClient.setBrokerURL(BROKER\_URL);

66 myClient.setInputQueueName(SERVICE\_QUEUE\_NAME);

67 myClient.setCasPoolSize(CAS\_POOL\_SIZE);

68

69 myClient.run(myReader, xmiListener);

70 } catch (Exception e) {

71 e.printStackTrace();

72 }

73 }

74

75 private static void runDatabaseReaderClient() throws IOException {

C:\Users\vhaisltrautt\Documents\Leo\Task1Client.java: 1/2

76 try {

77 /\* Specify the type of listeners. \*/

78 CSVListener csvListener = new CSVListener(new FileWriter(outcsv));

79 XmiUABListener xmiListener = new XmiUABListener(outdir);

80 xmiListener.setLaunchAnnotationViewer(true);

81 xmiListener.setAnnotationTypeFilter(new String[] { "gov.va.vinci.demo.types.Alcohol" });

82

83 /\* Create a client with batch database collection reader \*/

84 DatabaseMultiBatchClient myClient = new DatabaseMultiBatchClient(

85 (new DatabaseConnectionInformation(

86 "com.mysql.jdbc.Driver", /\* driver class \*/

87 "jdbc:mysql://localhost:3306/demo", /\* dataset url\*/

88 "demo", /\* username \*/

89 "demo2013" /\* password\*/)),

90 (new DataQueryInformation(

91 "select RowNo, NoteText, AlcoholUseStatus from notes order by RowNo;", /\*database query for a limited set\*/

92 //"select RowNo, NoteText, AlcoholUseStatus from fullDataset where RowNo > {min} AND RowNo <= {max}; ", /\*database query for the full

set\*/

93 1, /\* note column \*/

94 0 /\* id column \*/)),

95 0, /\* Starting row number\*/

96 1000, /\* Ending row number \*/

97 1000, /\* Read batch size \*/

98 csvListener, xmiListener

99 );

100

101 /\* Linking the client to the service through the broker \*/

102 myClient.setBrokerURL(BROKER\_URL);

103 myClient.setInputQueueName(SERVICE\_QUEUE\_NAME);

104 myClient.setCasPoolSize(CAS\_POOL\_SIZE);

105

106 myClient.run();

107 } catch (Exception e) {

108 e.getStackTrace();

109 }

110 }

111

1. }

# Appendix G - Sample Service 1 code

Parameters that can or must be changed are highlighted in yellow along with instructions for that parameter as code comments.

1 package gov.va.vinci.demo.task1;

2

3 import gov.va.vinci.Leo.descriptors.LeoAEDescriptor;

4 import gov.va.vinci.Leo.descriptors.LeoTypeSystemDescription;

5 import gov.va.vinci.uima.ae.RegexAnnotator;

6 import gov.va.vinci.uima.ae.WindowAnnotator;

7

8 public class Task1Service {

9 /\*\*

10 \* Broker Variables, shared between client and server

11 \*/

12 protected static String SERVICE\_QUEUE\_NAME = "Task1Service";

13 /\* Local broker \*/

14 public static String BROKER\_URL = "tcp://localhost:61616";

15 /\* Local JAM \*/

16 public static String JAM\_URL = "http://localhost:8080/jam";

17 /\* The number of CASes in the queue \*/

18 public static int CAS\_POOL\_SIZE = 4;

19

20 /\* If you create new or change types, change this flag to TRUE \*/

21 protected static boolean GENERATE\_TYPES = false;

22

23 /\* Flag whether to register this service with JAM \*/

24 protected static boolean REGISTER\_WITH\_JAM = true;

25

26 /\* Flag whether to start the client

27 \* set TRUE for local, simple deployment \*/

28 protected static boolean START\_CLIENT = true;

29

30 /\*\*

31 \* Basic main method to run the service.

32 \* @param args

33 \*/

34 public static void main(String[] args) {

35 gov.va.vinci.Leo.Service server = null;

36 try {

37 /\* Create a service and set the parameters \*/

38 server = new gov.va.vinci.Leo.Service();

39 server.setInputQueueName(SERVICE\_QUEUE\_NAME);

40 server.setBrokerURL(BROKER\_URL);

41 server.setCasPoolSize(CAS\_POOL\_SIZE);

42

43 /\* Setting parameters for the connection with JMX Analysis Module (JAM) \*/

44 if (REGISTER\_WITH\_JAM) {

45 server.setJamServerBaseUrl(JAM\_URL);

46 server.setJamQueryIntervalInSeconds(10);

47 server.setJamResetStatisticsAfterQuery(false);

48 System.out.println("Registering service with JAM: " + SERVICE\_QUEUE\_NAME);

49 }

50

51 /\* Create and deploy the service. \*/

52 LeoAEDescriptor pipeline = createPipeline(GENERATE\_TYPES);

53 server.deploy(pipeline);

54

55 if (START\_CLIENT) {

56 Task1Client.main(null);

57 }

58

59 /\* When a service is initialized, a set of XML configuration files called

60 \* UIMA descriptor files is automatically created.

61 \* The path to the aggregate descriptor file is printed to the console. \*/

62 System.out.println("Aggregate Descriptor File : \r\n" + server.getAggregateDescriptorFile());

63 /\* Start the client while the service is running.

64 \*

65 \* Requiring the user to press enter in the Console,

66 \* allows to gracefully unregister the service with JAM \*/

67 System.out.println("\nService running, press enter in this console to stop.");

68 System.in.read();

69 System.exit(0);

70 } catch (Exception e) {

71 e.printStackTrace();

72 }

73 }

74

75 private static LeoAEDescriptor createPipeline(boolean generateTypes) throws Exception {

C:\Users\vhaisltrautt\Documents\Leo\Task1Service.java: 1/3

76 /\* UIMA service requires a set of type classes.

77 \* If not provided, they can be automatically generated at run-time. \*/

78 LeoTypeSystemDescription types = createTypeSystem(generateTypes);

79

80 /\* Creating the pipeline as a set of steps.

81 \* Each step outlined below will be performed sequentially,

82 \* passing documents from each step to the next in the order

83 \* the steps are specified below. \*/

84 LeoAEDescriptor pipeline = new LeoAEDescriptor();

85

86 pipeline

87 /\* Step1: Find mentions of alcohol related term \*/

88 .addDelegate(new LeoAEDescriptor()

89 .setName("AlcoholRegexAnnotator")

90 .setImplementationName(RegexAnnotator.class.getCanonicalName())

91 .addParameterSetting("resource", true, false, "String", "src/main/resources/alcohol.regex")

92 .addParameterSetting("outputType", true, false, "String", "gov.va.vinci.demo.types.Alcohol")

93 .addTypeSystemDescription(types))

94 /\* Step 2. Mark a window of 10 words before and 5 words after each alcohol related term \*/

95 .addDelegate(new LeoAEDescriptor()

96 .setName("WindowAnnotator")

97 .setImplementationName(WindowAnnotator.class.getCanonicalName())

98 .addParameterSetting("inputType", true, true, "String", new String[] { "gov.va.vinci.demo.types.Alcohol" })

99 .addParameterSetting("outputType", true, false, "String", "gov.va.vinci.demo.types.Window")

100 .addParameterSetting("windowLtSize", true, false, "Integer", 10)

101 .addParameterSetting("windowRtSize", true, false, "Integer", 5)

102 .addTypeSystemDescription(types))

103 /\* Step 3. Mark a term related to past alcohol use that occurs within each window \*/

104 .addDelegate(new LeoAEDescriptor()

105 .setName("PastUseRegexAnnotator")

106 .setImplementationName(RegexAnnotator.class.getCanonicalName())

107 .addParameterSetting("inputType", true, true, "String", new String[] { "gov.va.vinci.demo.types.Window" })

108 .addParameterSetting("resource", true, false, "String", "src/main/resources/past.regex")

109 .addParameterSetting("outputType", true, false, "String", "gov.va.vinci.demo.types.PastUse")

110 .addTypeSystemDescription(types))

111 /\* Step 4. Mark a term related to current alcohol use that occurs within each window \*/

112 .addDelegate(new LeoAEDescriptor()

113 .setName("CurrentUseRegexAnnotator")

114 .setImplementationName(RegexAnnotator.class.getCanonicalName())

115 .addParameterSetting("inputType", true, true, "String", new String[] { "gov.va.vinci.demo.types.Window" })

116 .addParameterSetting("resource", true, false, "String", "src/main/resources/current.regex")

117 .addParameterSetting("outputType", true, false, "String", "gov.va.vinci.demo.types.Current")

118 .addTypeSystemDescription(types))

119 /\* Step 5. Mark a term related to the patient's denying any alcohol use that occurs within each window \*/

120 .addDelegate(new LeoAEDescriptor()

121 .setName("NeverUseRegexAnnotator")

122 .setImplementationName(RegexAnnotator.class.getCanonicalName())

123 .addParameterSetting("inputType", true, true, "String", new String[] { "gov.va.vinci.demo.types.Window" })

124 .addParameterSetting("resource", true, false, "String", "src/main/resources/never.regex")

125 .addParameterSetting("outputType", true, false, "String", "gov.va.vinci.demo.types.Never")

126 .addTypeSystemDescription(types))

127 /\* Step 6. Mark a term related to the patient's denying any alcohol use that occurs within each window \*/

128 .addDelegate(new LeoAEDescriptor()

129 .setName("ExcludeRegexAnnotator")

130 .setImplementationName(RegexAnnotator.class.getCanonicalName())

131 .addParameterSetting("inputType", true, true, "String", new String[] { "gov.va.vinci.demo.types.Window" })

132 .addParameterSetting("resource", true, false, "String", "src/main/resources/exclude.regex")

133 .addParameterSetting("outputType", true, false, "String", "gov.va.vinci.demo.types.Exclude")

134 .addTypeSystemDescription(types));

135

136 pipeline.setIsAsync(false);

137 pipeline.setNumberOfInstances(1);

138 return pipeline;

139 }

140

141 private static LeoTypeSystemDescription createTypeSystem(boolean generateTypes) throws Exception {

142 LeoTypeSystemDescription types = new LeoTypeSystemDescription();

143 // Create UIMA types

144

145 types

146 .addType("gov.va.vinci.demo.types.Alcohol", "Annotation for alcohol extraction.", "uima.tcas.Annotation")

147 .addType("gov.va.vinci.demo.types.PastUse", "Annotation for past use extraction.", "uima.tcas.Annotation")

148 .addType("gov.va.vinci.demo.types.Current", "Annotation for past use extraction.", "uima.tcas.Annotation")

149 .addType("gov.va.vinci.demo.types.Never", "Annotation for past use extraction.", "uima.tcas.Annotation")

150 .addType("gov.va.vinci.demo.types.Exclude", "Annotation for past use extraction.", "uima.tcas.Annotation")

151 .addType("gov.va.vinci.demo.types.Window", "Annotation for window extraction.", "uima.tcas.Annotation");

152 if (generateTypes) {

153 types.jCasGen("src/main/java", "target/classes");

154 }

155 return types;

156 }

1. }

# Appendix H - Sample Client 2 code

Parameters that can or must be changed are highlighted in yellow along with instructions for that parameter as code comments.

1 package gov.va.vinci.demo.task2;

2

3 import gov.va.vinci.Leo.client.DataQueryInformation;

4 import gov.va.vinci.Leo.client.DatabaseConnectionInformation;

5 import gov.va.vinci.Leo.client.DatabaseMultiBatchClient;

6 import gov.va.vinci.Leo.descriptors.CollectionReaderFactory;

7 import gov.va.vinci.Leo.listener.XmiUABListener;

8

9 import java.io.File;

10 import java.io.FileWriter;

11 import java.io.IOException;

12 import java.io.PrintWriter;

13 import java.util.Date;

14 import org.apache.commons.lang.time.StopWatch;

15 import org.apache.uima.collection.CollectionReader;

16

17 public class Task2Client {

18

19 /\* Local broker -- (un)comment the following line to toggle use of a local broker \*/

20 protected static String BROKER\_URL = "tcp://localhost:61616";

21 /\* Remote broker -- (un)comment the following line to toggle use of a remote broker \*/

22 //protected static String BROKER\_URL = tcp://remotehost:61616";

23

24 /\* Service queue name is used to link the client to the service through broker \*/

25 protected static String SERVICE\_QUEUE\_NAME = "Task2Service"; /\* Unique name for each group of services \*/

26

27 protected static int CAS\_POOL\_SIZE = 20; /\* Size of client CAS queue - at least as large as number of services in services cloud \*/

28

29 /\* Input and output variables

30 \* you may specify either path relative to the root of the project

31 \* or an absolute path \*/

32 protected static String indir = "data//input//";

33 protected static String outdir = "data//output//Demo2//xmi//";

34 protected static String outcsv = "data//output//Demo2//Demo.csv";

35 protected static String timeLogFile = "data//output//Demo2//timeLogTask2.csv";

36

37 /\*\*

38 \* @param args

39 \* @throws IOException

40 \*/

41 public static void main(String[] args) throws IOException {

42 if (args.length == 1) SERVICE\_QUEUE\_NAME = args[0];

43 /\* Timing variables to illustrate processing time \*/

44 StopWatch sw = new StopWatch();

45 sw.start();

46 Date startTime = new Date(sw.getStartTime());

47 System.out.println(" Starting Demo Task 2 client at " + SERVICE\_QUEUE\_NAME + " started at " + startTime);

48 if(!(new File(outdir)).exists()) (new File(outdir)).mkdirs();

49 /\* If the input text is in text files, use File reader client \*/

50 runFileReaderClient();

51 /\* if the input text is in a database, use Database Client \*/

52 //runDatabaseReaderClient();

53 sw.stop();

54 Date endTime = new Date(System.currentTimeMillis());

55

56 System.out.println("Processing time: " + sw.toString() + "\n" + "Processing ended at: " + endTime);

57

58 PrintWriter pw = null;

59

60 try {

61 File file = new File(timeLogFile);

62 FileWriter fw = new FileWriter(file, true);

63 pw = new PrintWriter(fw);

64 pw.println(SERVICE\_QUEUE\_NAME + "," + startTime + "," + sw.toString());

65 } catch (IOException e) {

66 e.printStackTrace();

67 } finally {

68 if (pw != null) {

69 pw.close();

70 }

71 }

72 System.out.close();

73 }

74

75 private static void runFileReaderClient() {

C:\Users\vhaisltrautt\Documents\Leo\Task2Client.java: 1/2

76 try {

77 /\* Specify the type of listeners. \*/

78 XmiUABListener xmiListener = new XmiUABListener(outdir);

79 xmiListener.setLaunchAnnotationViewer(false); /\* false - don’t run viewer after running; true - run viewer after running \*/

80

81 /\* Specify collection reader -- file reader \*/

82 CollectionReader myReader = CollectionReaderFactory.generateFileSubReader(

83 indir, /\* input directory -- all files in that directory will be processed. \*/

84 "UTF-8", /\* Text encoding \*/

85 false, /\* Recurse the input directory: false - look in parent directory; true - look through parent and sub-directory \*/

86 "gov.va.vinci.Leo.tools.AsciiFilter", "gov.va.vinci.Leo.tools.XmlFilter"

87 /\* Leo filters to deal with possible unusual characters in text \*/);

88

89 /\* Create a client \*/

90 gov.va.vinci.Leo.Client myClient = new gov.va.vinci.Leo.Client();

91 myClient.setBrokerURL(BROKER\_URL);

92 myClient.setInputQueueName(SERVICE\_QUEUE\_NAME);

93 myClient.setCasPoolSize(CAS\_POOL\_SIZE);

94 myClient.run(myReader, xmiListener); /\* one or more listeners \*/

95 } catch (Exception e) {

96 e.printStackTrace();

97 }

98 }

99

100 /\*\*

101 \* Example of a batch database reader

102 \* @throws IOException

103 \*/

104 private static void runDatabaseReaderClient() throws IOException {

105 try {

106 /\* Specify the type of listeners. \*/

107 XmiUABListener xmiListener = new XmiUABListener(outdir);

108 xmiListener.setLaunchAnnotationViewer(true);

109

110 /\* Create a client with batch database collection reader \*/

111 DatabaseMultiBatchClient myClient = new DatabaseMultiBatchClient(

112 (new DatabaseConnectionInformation(

113 "com.mysql.jdbc.Driver", /\* driver class \*/

114 "jdbc:mysql://localhost:3306/demo", /\* dataset url\*/

115 "", /\* username \*/

116 "" /\* password\*/)),

117 (new DataQueryInformation(

118 "select RowNo, NoteText from fullDataset;", /\*database query\*/

119 1, /\* note column \*/

120 0 /\* id column \*/)),

121 0, /\* Starting row number\*/

122 1000, /\* Ending row number \*/

123 1000, /\* Read batch size \*/

124 xmiListener /\* Listeners \*/

125 );

126

127 myClient.setBrokerURL(BROKER\_URL);

128 myClient.setInputQueueName(SERVICE\_QUEUE\_NAME);

129 myClient.setCasPoolSize(CAS\_POOL\_SIZE);

130 myClient.run();

131 } catch (Exception e) {

132 e.getStackTrace();

133 }

134 }

135

1. }

# Appendix I - Sample Service 2 code

Parameters that can or must be changed are highlighted in yellow along with instructions for that parameter as code comments.

1 package gov.va.vinci.demo.task2;

2

3 import gov.va.vinci.demo.ae.\*;

4 import gov.va.vinci.demo.task1.Task1Client;

5 import gov.va.vinci.Leo.descriptors.LeoAEDescriptor;

6 import gov.va.vinci.Leo.descriptors.LeoTypeSystemDescription;

7

8 public class Task2Service {

9 /\*\*

10 \* Broker Variables, shared between client and server

11 \*/

12 protected static String SERVICE\_QUEUE\_NAME = "Task2Service";

13 /\* Local broker \*/

14 public static String BROKER\_URL = "tcp://localhost:61616";

15 /\* Local JAM \*/

16 public static String JAM\_URL = "http://localhost:8080/jam";

17 /\* The number of CASes in the queue \*/

18 public static int CAS\_POOL\_SIZE = 10; /\* Enter the size of your CAS pool \*/

19

20 /\* The number of instances of the pipeline \*/

21 public static int INSTANCE\_NUMBER = 1;

22

23 /\* If you create new or change types, change this flag to TRUE \*/

24 protected static boolean GENERATE\_TYPES = false;

25

26 /\* Flag whether to register this service with JAM \*/

27 protected static boolean REGISTER\_WITH\_JAM = true;

28

29 /\* Flag whether to start the client

30 \* set TRUE for local, simple deployment \*/

31 protected static boolean START\_CLIENT = true;

32

33 /\*\*

34 \* Basic main method to run the service.

35 \* @param args

36 \*/

37 public static void main(String[] args) {

38 /\* Adjusting the name of the service to account for

39 \* a different number of instances for demonstration purposes \*/

40 SERVICE\_QUEUE\_NAME = SERVICE\_QUEUE\_NAME + INSTANCE\_NUMBER;

41

42 /\* Creating the service object \*/

43 gov.va.vinci.Leo.Service server = null;

44 try {

45 /\* Create a service and set the parameters \*/

46 server = new gov.va.vinci.Leo.Service();

47 server.setInputQueueName(SERVICE\_QUEUE\_NAME);

48 server.setBrokerURL(BROKER\_URL);

49 server.setCasPoolSize(CAS\_POOL\_SIZE);

50

51 /\* Setting parameters for the connection with JMX Analysis Module (JAM) \*/

52 if (REGISTER\_WITH\_JAM) {

53 server.setJamServerBaseUrl(JAM\_URL);

54 server.setJamQueryIntervalInSeconds(10);

55 server.setJamResetStatisticsAfterQuery(false);

56 System.out.println("Registering service with JAM: " + SERVICE\_QUEUE\_NAME);

57 }

58

59 /\* Create and deploy the service. \*/

60 LeoAEDescriptor pipeline = createPipeline(GENERATE\_TYPES);

61 server.deploy(pipeline);

62

63 if (START\_CLIENT) {

64 Task2Client.main(new String[] { SERVICE\_QUEUE\_NAME });

65 }

66

67 /\* When a service is initialized, a set of XML configuration files called

68 \* UIMA descriptor files is automatically created.

69 \* The path to the aggregate descriptor file is printed to the console. \*/

70 System.out.println("Aggregate Descriptor File : \r\n" + server.getAggregateDescriptorFile());

71

72 if (!START\_CLIENT) {

73 /\* Start the client while the service is running.

74 \*

75 \* Requiring the user to press enter in the Console,

C:\Users\vhaisltrautt\Documents\Leo\Task2Service.java: 1/2

76 \* allows to gracefully unregister the service with JAM \*/

77 System.out.println("\nService running, press enter in this console to stop.");

78 System.in.read();

79 }

80 System.exit(0);

81 } catch (Exception e) {

82 e.printStackTrace();

83 }

84 }

85

86 private static LeoAEDescriptor createPipeline(boolean generateTypes) throws Exception {

87 /\* UIMA service requires a set of type classes.

88 \* If not provided, they can be automatically generated at run-time. \*/

89 LeoTypeSystemDescription types = createTypeSystem(generateTypes);

90

91 LeoAEDescriptor pipeline = new LeoAEDescriptor();

92

93 // Add the fastest annotator -- average processing speed 100 cases per sec.

94 LeoAEDescriptor fastestAnnotator = new PauseAnnotator().getLeoAEDescriptor();

95 fastestAnnotator.setName("FastestAnnotator");

96 fastestAnnotator.setParameterSetting(PauseAnnotator.Param.DELAY\_IN\_MS.getName(), "10");

97 fastestAnnotator.setParameterSetting(PauseAnnotator.Param.OUTPUT\_TYPE.getName(), "none");

98 fastestAnnotator.addTypeSystemDescription(types);

99 pipeline.addDelegate(fastestAnnotator);

100

101 // Add the medium speed annotator -- average processing speed = 50 cases per sec.

102 LeoAEDescriptor mediumAnnotator = new PauseAnnotator().getLeoAEDescriptor();

103 mediumAnnotator.setName("MediumAnnotator");

104 mediumAnnotator.setParameterSetting(PauseAnnotator.Param.DELAY\_IN\_MS.getName(), "20");

105 mediumAnnotator.setParameterSetting(PauseAnnotator.Param.OUTPUT\_TYPE.getName(), "none");

106 mediumAnnotator.addTypeSystemDescription(types);

107 pipeline.addDelegate(mediumAnnotator);

108

109 // Add the slowest speed annotator -- average processing speed = 25 cases per sec.

110 LeoAEDescriptor slowestAnnotator = new PauseAnnotator().getLeoAEDescriptor();

111 slowestAnnotator.setName("SlowestAnnotator");

112 slowestAnnotator.setParameterSetting(PauseAnnotator.Param.DELAY\_IN\_MS.getName(), "40");

113 slowestAnnotator.setParameterSetting(PauseAnnotator.Param.OUTPUT\_TYPE.getName(), "none");

114 slowestAnnotator.addTypeSystemDescription(types);

115 pipeline.addDelegate(slowestAnnotator);

116

117 pipeline.setIsAsync(false);

118 pipeline.setNumberOfInstances(INSTANCE\_NUMBER);

119 return pipeline;

120 }

121

122 /\*\*

123 \*

124 \* @param generateTypes

125 \* @return

126 \* @throws Exception

127 \*/

128 private static LeoTypeSystemDescription createTypeSystem(boolean generateTypes) throws Exception {

129 LeoTypeSystemDescription types = new LeoTypeSystemDescription();

130 // Create UIMA types

131 types.addType("gov.va.vinci.Leo.types.CSI", "", "uima.tcas.Annotation"); /\* Examples of addType are: \*/

132 if (generateTypes) {

133 types.jCasGen("src/main/java", "target/classes");

134 }

135 return types;

136 }

137

138 }

# Appendix J. Definitions and Acronyms

|  |  |
| --- | --- |
| AE | Analysis Engine (part of UIMA) |
| AS | Asynchronous Scaleout |
| CAS | Common Analysis System |
| CSI |  |
| DB | Database |
| UAB |  |
| UIMA | Unstructured Information Management Architecture |
| XMI | XML Metadata Interchange |
| XML | Extensible Markup Language |