THE CYCLUS REMOTE EXECUTION SOFTWARE ARCHITECTURE

Tevaganthan Veluppillai and Robert E. Hiromoto

University of Idaho

Abstract

This document describes the Cyclus remote execution software architecture. The software is designed to support a distributed client/server architecture that provides Cyclus simulation users with the ability to executed simulation on remote designated machines. Under this architectural design, laptop users can dispatch a Cyclus simulation request to a remote designated server machine. The requested simulation is executed on the remote server and it corresponding outputs are saved on an SQLite database and, if requested, sent back to the user.

Table of Contents

Section 1 — Cyclus Invocations in Cyclist

1.1 GUI interface to Cyclist

Cyclist is the GUI interface for invoking the Cyclus simulator. The Cyclus architecture allows for several execution configurations as listed in Table 1. These execution configurations are illustrated in the screen shots of the User's GUI environment.

Physical location of Cyclus	Physical location of XML (Input scenario files)	Simulation Output (SQLITE format)
Local System (Cyclus on local user system)	Local System	Local System
Remote Server with META DATA services	Remote location	Remote and Local System
Remote Server with no META DATA services	Local files	Remote and Local System
Remote Server with input data via Metadata Server information	Input scenario files located on the remote server.	Remote and Local System

Table 1.

1.2 Screen shots of User's GUI environment

The main user's GUI is illustrated in the Fig. 1. It has the following three choices: 1) Local System 2) Meta-data Server 3) Remote Server (Special User). These options are selected by clicking on the *Method of Simulation Invocation* panel

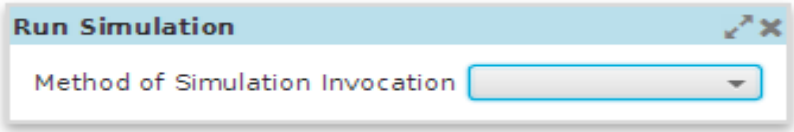
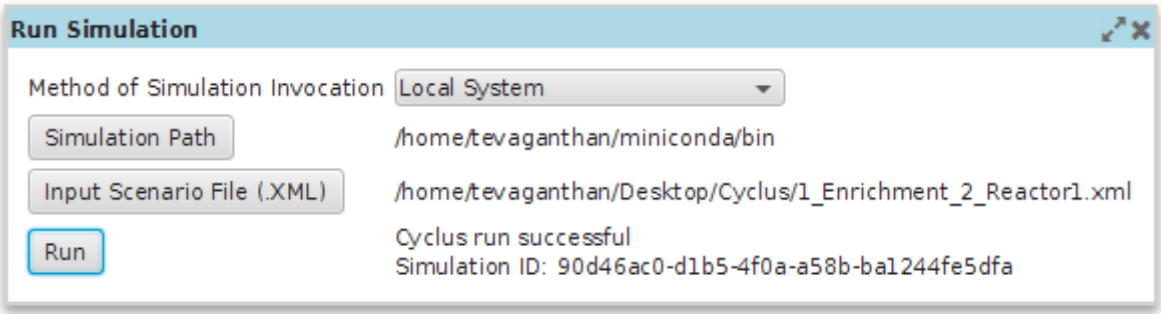


Fig. 1.

1.2.1 Local User Interface

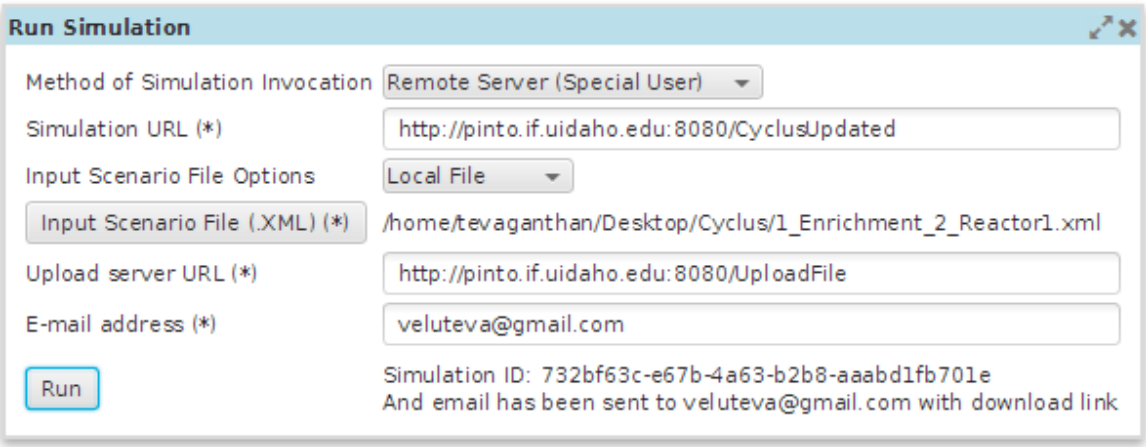


1.2.2 Remote Special User Interface

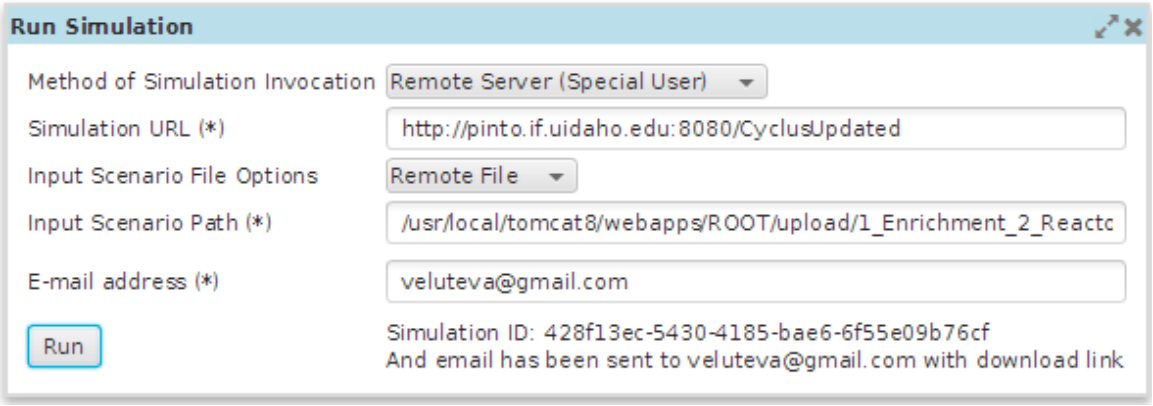
The 'Special User' designates a user that has privileged access to a remote server for Cyclus development activities. The activities may involve debugging, verifying and validating a new Cyclus modification, etc., that maybe required before registering the final version onto the Metadata server. The Special User in coordination with the program developer(s) is provided a special 'Simulation URL' (as depicted in the GUI panel below) that allows access to the development server for testing.

1.2.2a Uploading *Local* Input File (Non Metadata Service)

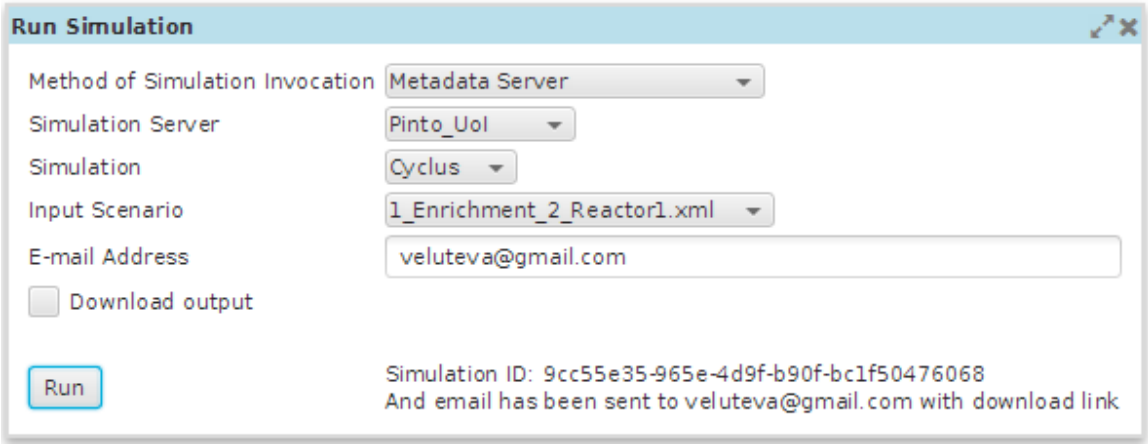
(Describe special user)



1.2.2b Requesting *Remote* Input File (Non Metadata Service)



1.2.3 Remote User Interface: Using Metadata Services



Section 2 — Details of the Cyclus Invocation in Cyclist

2.1 Cyclus Invocation Options

The main objective of this view is to run the Cyclus simulation from local computers or from remote simulation servers by using remote or local input scenario files.

Invocation of simulation has the following three major components:

1. Cyclus Simulation (./cyclus executable)
2. Input Scenario files (XML files)
3. Output of the simulation (SQLITE file)

Users can run the simulations under the following four options:

2.1.1 local User Execution Interface

The following are the main steps of this operation:

Step 1: On the Server dropdown list select "Local System"

Step 2: As soon as user chooses the "Local System" from drop down list and "Choose Cyclus Location" file explorer button will appear and user has to select the path to cyclus file.

Step 3: From the Input Scenario Files drop down list "local input file" is selected by default and user needs to give the location for the input scenario files.

Step 4: User chooses run simulation and the output (SQLITE format) will be stored in the local system.

Flow of the operations:

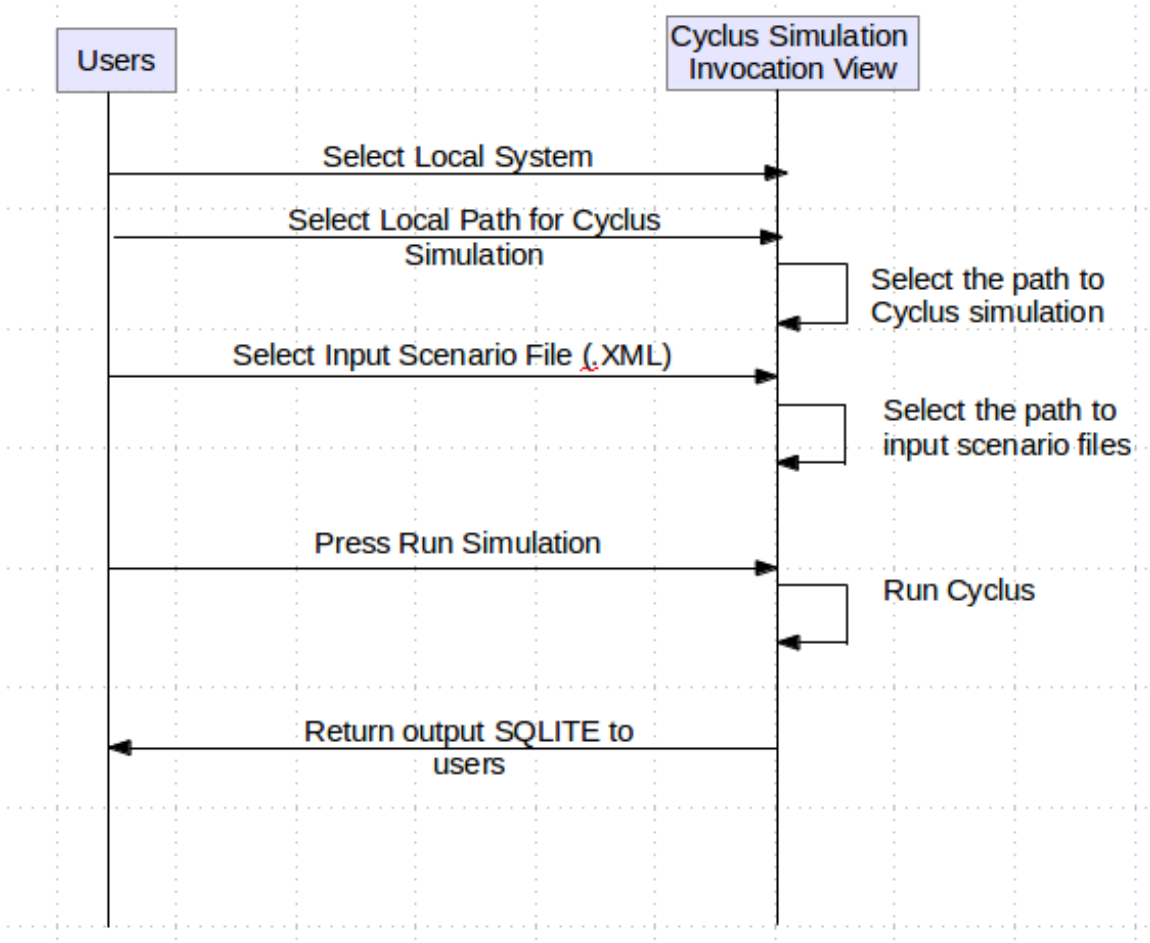


Fig. 1. Sequence diagram for local system operations

2.1.2 Remote Special User Execution Interface

2.1.2a Uploading *Local* Input File (Non Metadata Service)

In this scenario, a user has the access URL of the remote simulation server.

[NOTE: Here the access URL is **NOT** obtained from the "metadata server")

This has the following steps.
Step 1: From the Server dropdown list select "Add Remote Simulation Server"

Step 2: Next the "Add Cyclus Simulation Server" view will appear allowing the user to add the remote server access URL and input scenario file on the local system.

Step 3: from the Input Scenario Files dropdown list, the user selects a local file.

Step 4: Since user is running the remote simulation with local input scenario files before the remote invocation, the input scenario file will be uploaded on the remote server to complete the remote invocation of Cyclus.

Step 5: The user selects the download output check box that downloads the output of the execution invocation.

Flow of the operations:

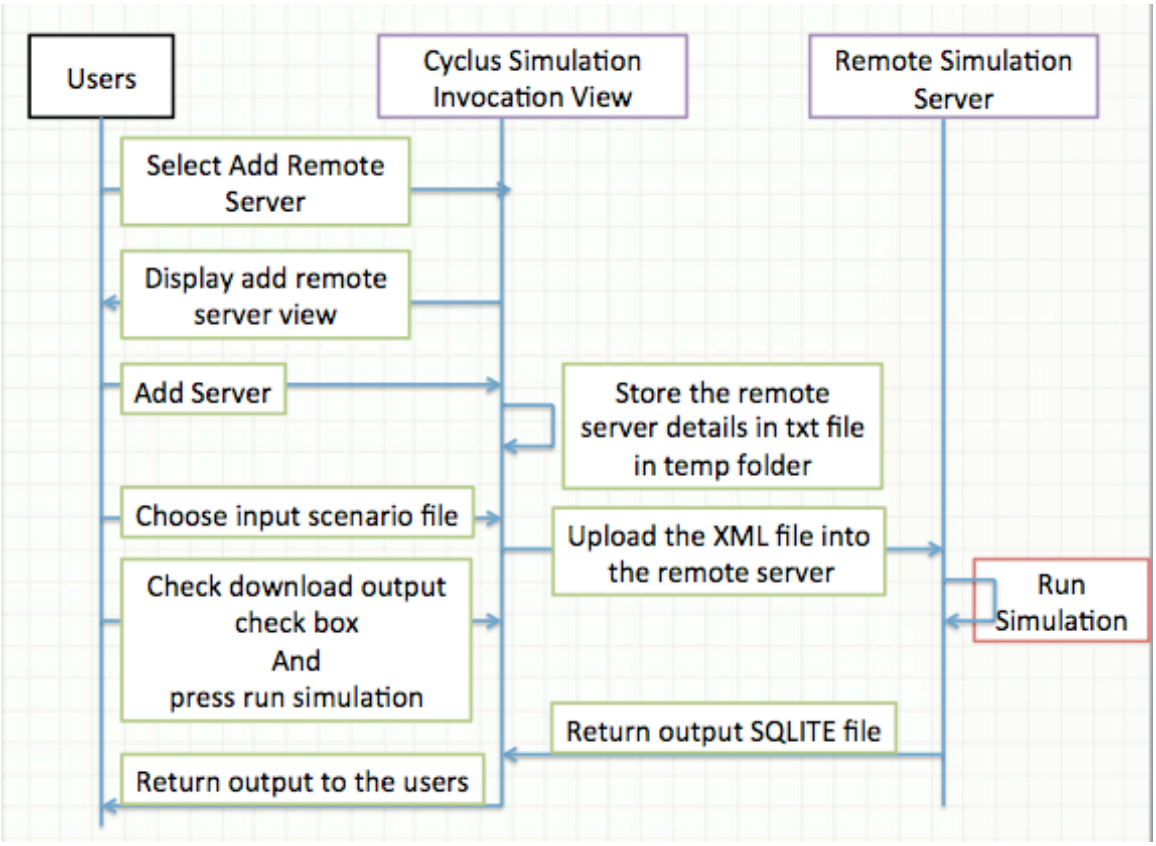


Fig. 2a. Sequence diagram for remote system operations

2.1.2b Requesting *Remote* Input File (Non Metadata Service)

This method is similar to method 2 except that the input file locations are given by remote system.

Step 1: From the Server dropdown list select the "Add Remote Simulation Server"

Step 2: Next the "Add Cyclus Simulation Server" view will appear and the user is then able to add the remote server access URL and input scenario file on the system.

Step 3: From the Input Scenario Files dropdown list, the remote input scenario file is selected.

Step 4: Users give the remote path location of the input scenario files.

Step 5: The user selects the download output check box that downloads the output of the execution invocation on the local computer from remote server.

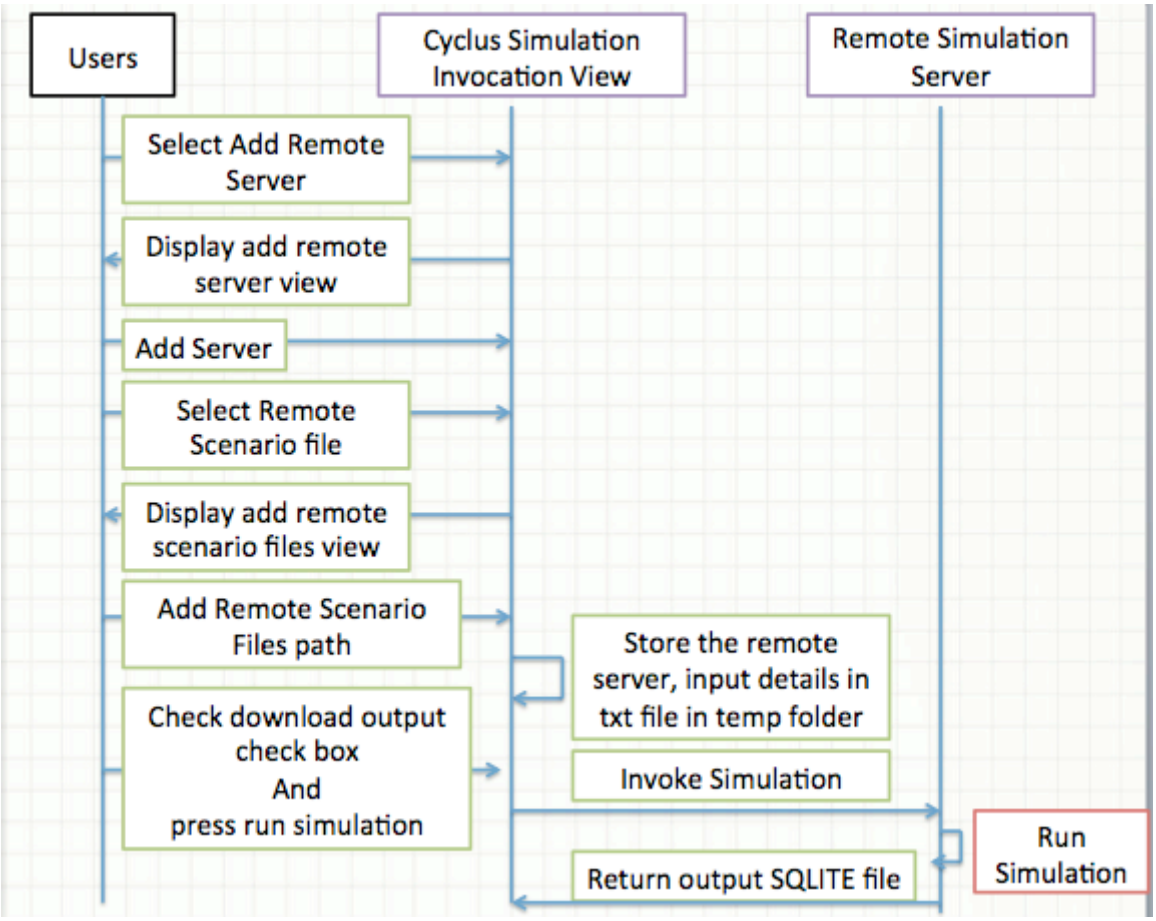


Fig. 2b. Sequence diagram for remote system operations

2.1.3 Option 3: Remote User Execution Interface

In this scenario users are able to access the simulation servers from the metadata server and also users are able to view input scenario files from the metadata servers.

This has the following steps.

Step 1: From the Server dropdown list, select the "Meta Data Server."

Step 2: Next the user will see another dropdown list that contains registered simulation servers.

Step 3: Choose a simulation server

Step 4: Based on the selected simulation server, the input scenario files will be displayed. The input files are linked with the selected simulation server.

Step 5: The user selects an input scenario file (Note: Here the input scenario files are remote files)

Step 6: The user selects the download output check box that downloads the output of the execution invocation.

The following figure shows the operation sequence:

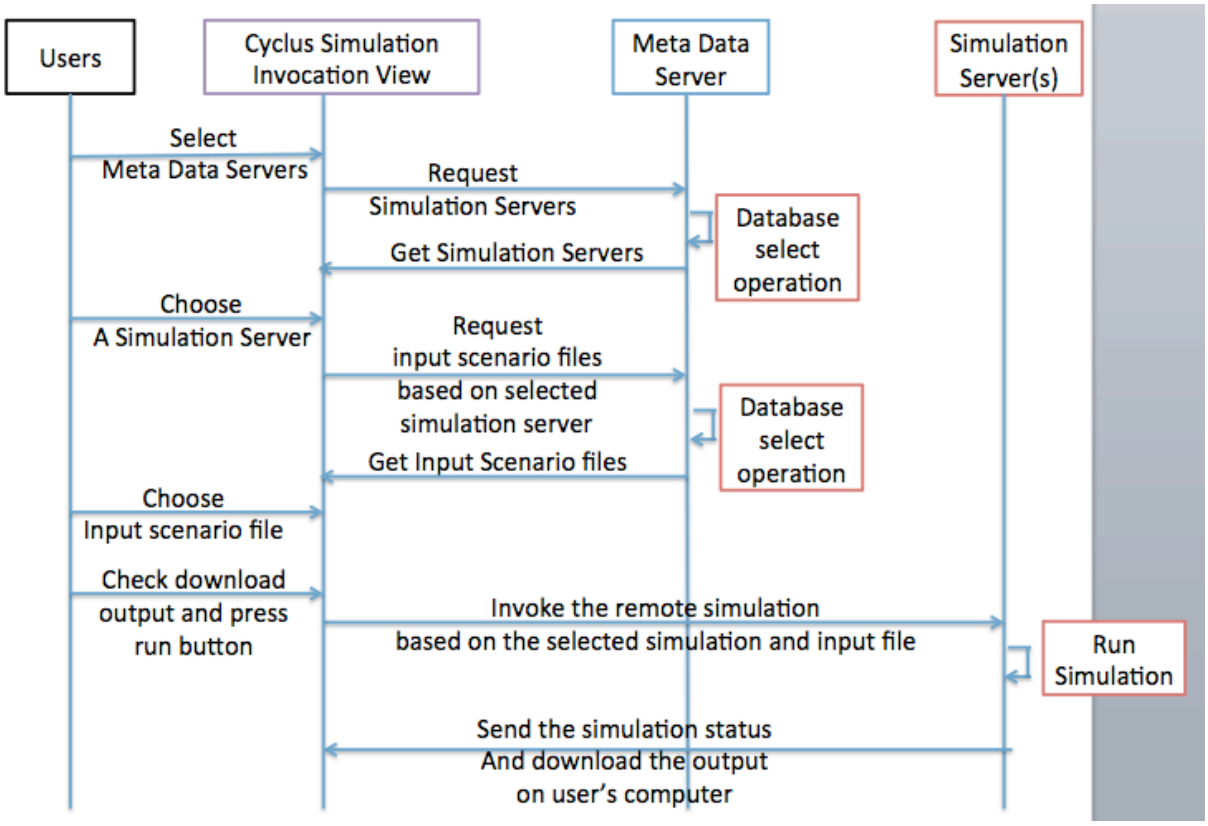


Fig. 3. Sequence diagram for remote system operations via Meta Data server

Section 3 — Configuration and Registration of the Cyclus Remote Server on the Meta Data Server: A Developers Document

This document describes how to setup the Cyclus Simulation Server and its registration in the metadata server.

Cyclus Simulation Tomcat Server requires the following 3 servlets:

a. to run the simulation and email the output
b. to let users to upload their local file
c. to let users to download the output file (SQLITE)

3.1 Cyclus Server Setup

Prerequisite: *Apache tomcat server, Cyclus simulation*

Step1: Installed Cyclus simulation
URL: http://fuelcycle.org/user/install.html

Step 2: Download and configure Apache Tomcat Server
URL: http://tomcat.apache.org/

Step 3: Copy the Cyclus servlet file under tomcat installed directory in the following path:
tomcat8/webapps/ROOT/WEB-INF/classes/

Servlet code is attached: (CyclusUpdated)

You need to change the following information from the code based on your file system

a. String exePath
b. Authenticator auth = new SMTPAuthenticator("your-Gmail", "yourpassword");

Step 4: After the modification you need to compile the code again.

4.1) before compiling, you need to have the following 2 jar files in your lib folder under the tomcat location:

a. javax.mail.jar
b. actiavtion.jar

The jar files are attached with document folder.

4.2) go to the folder where the source code exists. In our case the following place:

/usr/local/tomcat8/webapps/ROOT/WEB-INF/classes

Run the following command to compile the program:

javac -classpath /usr/local/tomcat8/lib/servlet-api.jar:/usr/local/tomcat8/lib/javax.mail.jar:/usr/local/lib/tomcat8/activation.jar CyclusUpdated.java

Step 5: Append the following contents in the web.xml file as the follows from the following location:

/usr/local/tomcat8/webapps/ROOT/WEB-INF/

```
<servlet>
 <servlet-name>CyclusUpdated</servlet-name>
  <servlet-class>CyclusUpdated</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>CyclusUpdated</servlet-name>
 <url-pattern>/CyclusUpdated</url-pattern>
</servlet-mapping>
```

So the final web.xml file looks like following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
        <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"  version="3.1"  metadata-
        complete="true">
<display-name>Welcome to Tomcat</display-name>
 <description>
Welcome to Tomcat
 </description>

<servlet>
 <servlet-name>CyclusUpdated</servlet-name>
 <servlet-class>CyclusUpdated</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>CyclusUpdated</servlet-name>
<url-pattern>/CyclusUpdated</url-pattern>
</servlet-mapping>
</web-app>
```

Step 6: Restart the TOMCAT Server

a. Shutdown Tomcat Server => teva@pinto:/usr/local/tomcat8/bin$ sudo ./shutdown.sh

b. Startup Tomcat Server => teva@pinto:/usr/local/tomcat8/bin$ sudo ./startup.sh

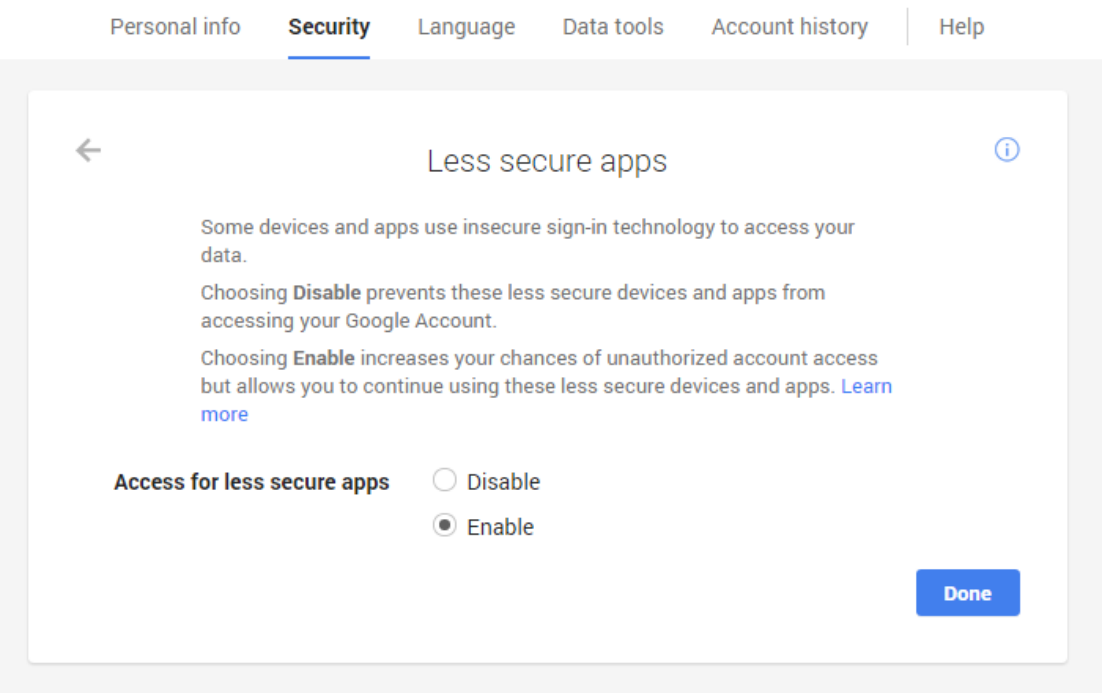
Step 7: Congratulations that you're completed the configuration of Cyclus Server

Your access URL should look like the following format

ServerIP:TomcatPort/CyclusUpdated

E.g.,: http://129.101.194.157:8080/CyclusUpdated

However, there is one more thing you have to do that's related to your Gmail security. Go to the relevant email change the following properties as follows:



3.2 Steps for Configuring Upload File Access URL

Prerequisite: *Apache tomcat server*

Step 1: Download and configure Apache Tomcat Server

URL: http://tomcat.apache.org/

[Note: If you have completed the Cyclus Server Configuration, ignore the Step 1]

Step 2: Copy the Cyclus servlet file under tomcat installed directory in the following path:

tomcat8/webapps/ROOT/WEB-INF/classes/

UploadFile code is attached: (UploadFile)

Step 3: Compile the code.

3.1) before compilation you need to have the following 2 jar files in your lib folder under the tomcat location.

a. commons-fileupload-1.3.1.jar
b. commons-io-2.4.jar

The jar files are attached with document folder.

3.2) go to the folder where the source code exists. In our case the following place:

/usr/local/tomcat8/webapps/ROOT/WEB-INF/classes

Run the following command to compile the program:

javac -classpath /usr/local/tomcat8/lib/servlet-api.jar:/usr/local/tomcat8/lib/commons-fileupload-1.3.1.jar:/usr/local/lib/tomcat8/commons-io-2.4.jar UploadFile.java

Step 4: Append the following contents in the web.xml file as the follows from the following location:

/usr/local/tomcat8/webapps/ROOT/WEB-INF/

```
<servlet>
 <servlet-name>UploadFile</servlet-name>
  <servlet-class>UploadFile</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>UploadFile</servlet-name>
 <url-pattern>/UploadFile</url-pattern>
</servlet-mapping>
```

So the final web.xml file looks like following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"  version="3.1"  metadata-
    complete="true">
<display-name>Welcome to Tomcat</display-name>
 <description>
```

```
Welcome to Tomcat
 </description>

<servlet>
 <servlet-name>CyclusUpdated</servlet-name>
 <servlet-class>CyclusUpdated</servlet-class>
</servlet>
<servlet>
 <servlet-name>UploadFile</servlet-name>
 <servlet-class>UploadFile</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>CyclusUpdated</servlet-name>
<url-pattern>/CyclusUpdated</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>UploadFile</servlet-name>
<url-pattern>/UploadFile</url-pattern>
</servlet-mapping>
</web-app>
```

Step 6: Restart the TOMCAT Server

a. Shutdown Tomcat Server => teva@pinto:/usr/local/tomcat8/bin$ sudo ./shutdown.sh

b. Startup Tomcat Server => teva@pinto:/usr/local/tomcat8/bin$ sudo ./startup.sh

Step 7: Congratulations that you're completed the configuration of Cyclus Server

Your access URL should look like the following format

ServerIP:TomcatPort/ UploadFile

E.g.,: http://129.101.194.157:8080/UploadFile

3.3 Configure Download Access URL Setup

Prerequisite: *Apache tomcat server, Cyclus simulation*

Step1 : Installed Cyclus simulation

URL: http://fuelcycle.org/user/install.html

[Note: If you've performed the previous steps (A, and B) ignore this step

Step 2: Download and configure Apache Tomcat Server
URL: http://tomcat.apache.org/
[Note: If you've performed the previous steps (A, and B) ignore this step

Step 3: Copy the Cyclus servlet file under tomcat installed directory in the following path:
tomcat8/webapps/ROOT/WEB-INF/classes/

Servlet code is attached: (DownloadSim)

You need to change the following information from the code based on your file system

a. filePath = "/home/teva/miniconda/bin/cyclus.sqlite";

Here use need to enter the path for the SQLITE file from Cyclus

Step 4: After the modification you need to compile the code again.

4.1) go to the folder where the source code exists. In our case the following place:

/usr/local/tomcat8/webapps/ROOT/WEB-INF/classes

Run the following command to compile the program:

javac -classpath /usr/local/tomcat8/lib/servlet-api.jar DownloadSim.java

Step 5: Append the following contents in the web.xml file as the follows from the following location:
/usr/local/tomcat8/webapps/ROOT/WEB-INF/

```
<servlet>
 <servlet-name>DownloadSim</servlet-name>
  <servlet-class>DownloadSim</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>DownloadSim</servlet-name>
 <url-pattern>/DownloadSim</url-pattern>
</servlet-mapping>
```

So the final web.xml file looks like following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
        <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"  version="3.1"  metadata-
        complete="true">
<display-name>Welcome to Tomcat</display-name>
 <description>
Welcome to Tomcat
 </description>
<servlet>
 <servlet-name>CyclusUpdated</servlet-name>
 <servlet-class>CyclusUpdated</servlet-class>
</servlet>
<servlet>
 <servlet-name>UploadFile</servlet-name>
 <servlet-class>UploadFile</servlet-class>
</servlet>
<servlet>
 <servlet-name>DownloadSim</servlet-name>
  <servlet-class>DownloadSim</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>CyclusUpdated</servlet-name>
<url-pattern>/CyclusUpdated</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>UploadFile</servlet-name>
<url-pattern>/UploadFile</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>DownloadSim</servlet-name>
 <url-pattern>/DownloadSim</url-pattern>
</servlet-mapping>
</web-app>
```

Step 6: Restart the TOMCAT Server

a. Shutdown Tomcat Server => teva@pinto:/usr/local/tomcat8/bin$ sudo ./shutdown.sh

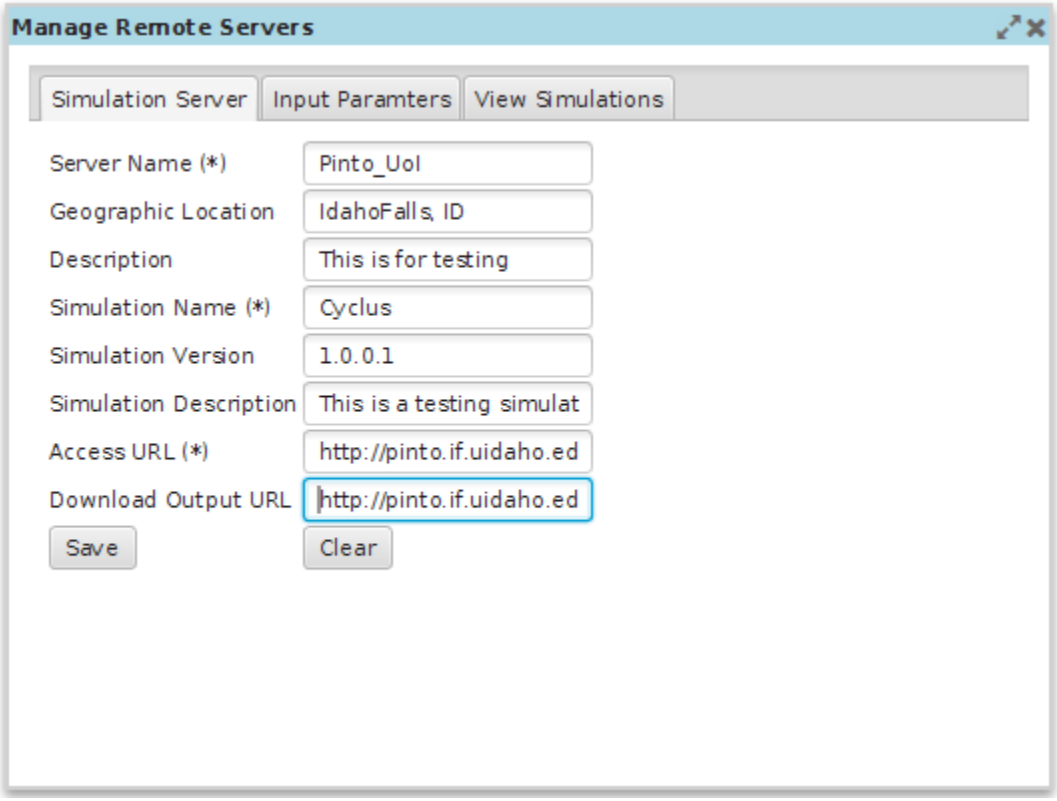
b. Startup Tomcat Server => teva@pinto:/usr/local/tomcat8/bin$ sudo ./startup.sh

Step 7: Congratulations you've completed the configuration of the Cyclus Server
Your access URL should look like the following format
ServerIP:TomcatPort/ DownloadSim
E.g.,: http://129.101.194.157:8080/DownloadSim

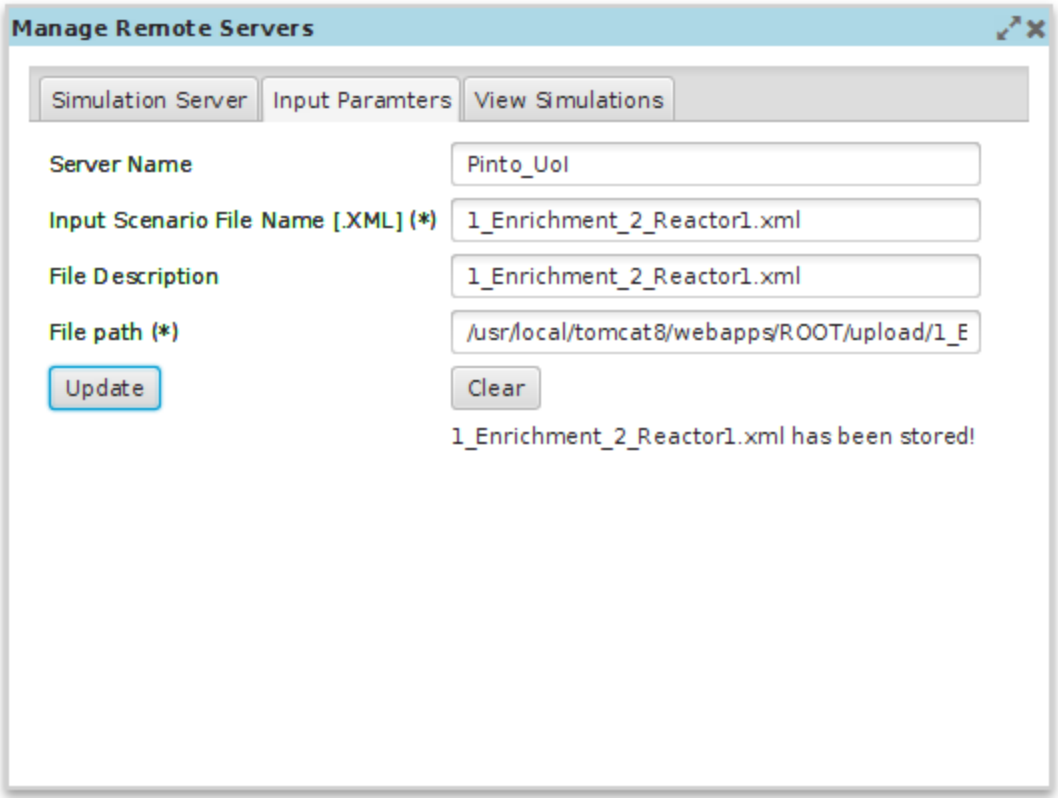
After the configuration of the 3 servlets, the server can now be registered in the metadata server.

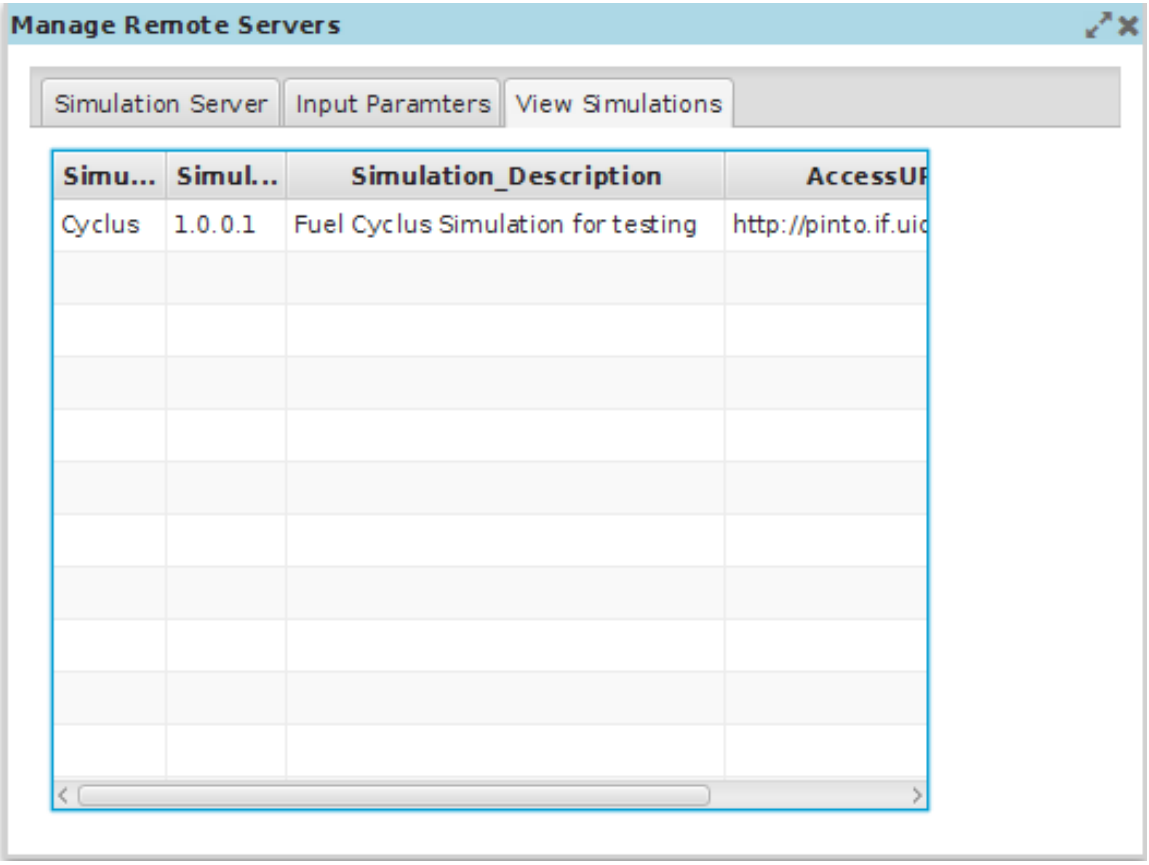
The following describes the registration steps of the server onto the Metadata server.

1. Add Server: Select the Manage Remote Server view from Cyclus and navigate to Simulation Server tab and enter your information. The following shows the sample input from my server.



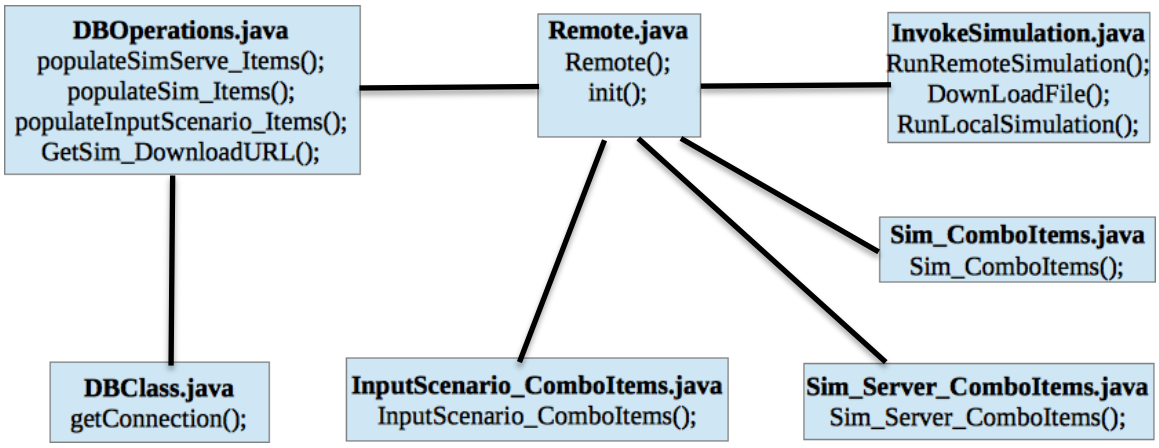
2. Register your input scenario files on the server.
   Here you can add more than one input parameter files (.XML) per server.
3. As a final step, the registration information can be verified by selecting the 'View Simulations' tab as shown below.


Manage Remote Servers
Simulation Server
Input Paramters
View Simulations
Server Name
Pinto_UoI
Input Scenario File Name [.XML] (*)
1_Enrichment_2_Reactor1.xml
File Description
1_Enrichment_2_Reactor1.xml
File path (*)
/usr/local/tomcat8/webapps/ROOT/upload/1_E
Update
Clear
1_Enrichment_2_Reactor1.xml has been stored!


Manage Remote Servers
Simulation Server
Input Paramters
View Simulations
Simu...
Simul...
Simulation_Description
AccessUR
Cyclus
1.0.0.1
Fuel Cyclus Simulation for testing
http://pinto.if.uic

Section 4 — Java Execution Classes and Methods

The following UML class diagram shows the Java execution class diagram.



4.1 Class: Remote.java class

Purpose: Implements the Cyclus simulation in the following ways: Local System, Meta Data Server, and Remote Server [Special Users]

Methods:

Remote(): Initializes the grid pane

init(): Initializes the GUI and implements the Cyclus execution calls.

4.2 Class: InvokeSimulation.java class

Purpose: Executes Cyclus simulator from local system and remote system servlet calls.

Methods:

4.2.1 RunRemoteSimulation(): To run the remote simulations based on the servlet calls.

```
public static String RunRemoteSimulation(String AccessURL, String InputPath)
throws IOException
    {
        String output = "";
```

```
        URL url = new URL(AccessURL);
        URLConnection conn = url.openConnection();
        conn.setDoOutput(true);
        BufferedWriter out = new BufferedWriter( new OutputStreamWriter(
                                conn.getOutputStream() ) );
        out.write("InputParamter=" + InputPath);
        out.flush();
        out.close();
        BufferedReader in = new BufferedReader( new InputStreamReader(
                                conn.getInputStream() ) );
        String response;
        while ( (response = in.readLine()) != null )
        {
            output = response;
        }
        in.close();
        return output;
    }
```

4.2.2 RunLocalSimulation(): To run the remote simulation from the local system based on the selected input Cyclus and input scenario file

```
public static String RunLocalSimulation(String exePath, String InputParamter)
throws IOException
    {
        ProcessBuilder process = new ProcessBuilder();
        process.directory(new File(exePath));
        process.command("./cyclus", InputParamter);


        Process p = process.start();
        String output = "";
        BufferedReader input = new BufferedReader(new
                                InputStreamReader(p.getInputStream()));
            String line;
            while((line=input.readLine()) != null)
```

```
            {
                output = "Cyclus run successful" + "\n" + line;
            }
            input.close();
            return output;
    }
```

4.2.3 DownLoadFile(): To download the output file

```
public static String DownLoadFile(String FileName, String DownloadURL) throws
IOException
    {
        String Download_status = null;
        System.out.println("opening connection");
        URL url = new URL(DownloadURL);
        InputStream in = url.openStream();
        FileOutputStream fos = new FileOutputStream(new File(FileName));

        System.out.println("reading file...");
        int length = -1;
        byte[] buffer = new byte[1024];// buffer for portion of data from
        // connection
        while ((length = in.read(buffer)) > -1)
        {
            fos.write(buffer, 0, length);
        }
        os.close();
        in.close();
        Download_status = "file was downloaded";
        return Download_status;
    }
```

4.3 Class: DBClass. java class

Purpose: To create Metadata server database connection object

Methods:

4.3.1 getConnection(): Creates the connection object

```
public Connection getConnection() throws ClassNotFoundException, SQLException
    {
        Connection conn = null;
        Class.forName("com.mysql.jdbc.Driver");
        //Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //Open a connection
        conn = DriverManager.getConnection(DB_URL, USER, PASS);
        return conn;
    }
```

4.4 Class: DBOperations. java class

Purpose: To interact with databases from Metadata server [MD server] based on the mysql connection object

Methods:

4.4.1 populateSimServe_Items(): Populates simulation servers from MD server

```
public static ChoiceBox<Sim_Server_ComboItems> populateSimServe_Items()
    {
        ChoiceBox<Sim_Server_ComboItems> cmb = null;
        Connection conn = null;
        Statement stmt = null;
        try
        {
          //Execute a query
          DBClass objDBConnection = new DBClass();
          conn = objDBConnection.getConnection();
```

```
            stmt = conn.createStatement();
            String sql = "select Server_ID, Server_Name from Sim_Servers;";
            ResultSet rs = stmt.executeQuery(sql);
            cmb = new ChoiceBox<Sim_Server_ComboItems>();
            while(rs.next())
            {
              int Ser_ID = rs.getInt("Server_ID");
              String ServerName = rs.getString("Server_Name");
              cmb.getItems().add(new Sim_Server_ComboItems(Ser_ID, ServerName));
            }
          }
          catch(Exception ex)
          {
            System.out.print(ex.toString());
          }
    return cmb;
    }
```

4.4.2 populateSim_Items(): Populates Simulations from MD server

```
public static ChoiceBox<Sim_ComboItems> populateSim_Items(String
SimulationServer)
    {
        ChoiceBox<Sim_ComboItems> cmb = null;
        Connection conn = null;
        Statement stmt = null;
        try
        {
            //Execute a query
            DBClass objDBConnection = new DBClass();
            conn = objDBConnection.getConnection();
            stmt = conn.createStatement();
```

```
        String sql = "select Simulation_Name,AccessURL from Simulations where
    Simulation_ID IN (Select Simulation_ID from Server_Simulation inner join
    Sim_Servers on Server_Simulation.Server_ID = Sim_Servers.Server_ID where
    Sim_Servers.Server_Name = '" + SimulationServer + "');";
        ResultSet rs = stmt.executeQuery(sql);
        cmb = new ChoiceBox<Sim_ComboItems>();
        while(rs.next())
        {
            String Sim_Name = rs.getString("Simulation_Name");
            String Access_URL = rs.getString("AccessURL");
            cmb.getItems().add(new Sim_ComboItems(Sim_Name, Access_URL));
        }
    }
    catch(Exception ex)
    {
        System.out.print(ex.toString());
    }
    return cmb;
}
```

4.4.3 populateInputScenario_Items(): Populates Simulations from MD server

```
public static ChoiceBox<InputScenario_ComboItems>
populateInputScenario_Items(String SimulationServer)
    {
        ChoiceBox<InputScenario_ComboItems> cmb = null;
        Connection conn = null;
        Statement stmt = null;
        try
        {
          //Execute a query
          DBClass objDBConnection = new DBClass();
          conn = objDBConnection.getConnection();
          stmt = conn.createStatement();
```

```
        String sql = "select InputData_Name,Path from InputData where
        InputData_ID IN (Select InputData_ID from InputData_Server inner join
        Sim_Servers on InputData_Server.Server_ID = Sim_Servers.Server_ID where
        Sim_Servers.Server_Name = '" + SimulationServer + "');";
          ResultSet rs = stmt.executeQuery(sql);
          cmb = new ChoiceBox<InputScenario_ComboItems>();
          while(rs.next())
          {
           String InputData_Name = rs.getString("InputData_Name");
           String Path = rs.getString("Path");
           cmb.getItems().add(new InputScenario_ComboItems(InputData_Name,
                                    Path));
          }
        }
        catch(Exception ex)
        {
           System.out.print(ex.toString());
        }
        return cmb;
    }
```

4.4.4 GetSim_DownloadURL(): Populates download URL based on the selected Simulation Server

```
public static String GetSim_DownloadURL(String Sim_Name)
    {
        String outputURL = null;
        Connection conn = null;
        Statement stmt = null;
        try
        {
          //Execute a query
          DBClass objDBConnection = new DBClass();
          conn = objDBConnection.getConnection();
          stmt = conn.createStatement();
```

```
            String sql = "select DownloadURL from Simulations where
                    Simulation_Name = '" + Sim_Name + "';";
            ResultSet rs = stmt.executeQuery(sql);
            while(rs.next())
            {
              outputURL = rs.getString("DownloadURL");
            }
        }
        catch(Exception ex)
        {
            System.out.print(ex.toString());
        }
        return outputURL;
    }
```

4.5 Class: Sim_ComboItems.java

Purpose: Populates the choicebox with Simulation Name, and Access URL

Methods:

4.5.1 Sim_ComboItems(): Initializes the Simulation Name, and Access URL objects

```
public class Sim_ComboItems
    {
        private String Simulation_Name;
        private String AccessURL;
        public Sim_ComboItems(String S_Name, String S_AccessURL)
        {
            this.Simulation_Name = S_Name;
            this.AccessURL = S_AccessURL;
        }
        @Override
        public String toString()
        {
```

```
        return Simulation_Name;
    }
    public String getAccessURL()
    {
        return AccessURL;
    }
}
```

4.6 Class: Sim_Server_ComboItems.java

Purpose: Populates the choicebox with Simulation Server Name, and its IDs

Methods:

4.6.1 Sim_Server_ComboItems(): Initializes the Simulation Server Name, and Server ID objects

```
public class Sim_Server_ComboItems
    {
        private int Server_ID;
        private String Server_Name;
        public Sim_Server_ComboItems(int S_ID, String S_Name)
          {
                this.Server_ID = S_ID;
                this.Server_Name = S_Name;
          }
        @Override
        public String toString()
          {
                return Server_Name;
          }
        public int getKey()
          {
                return Server_ID;
```

```
        }
    }
```

4.7 Class: InputScenario_ComboItems.java

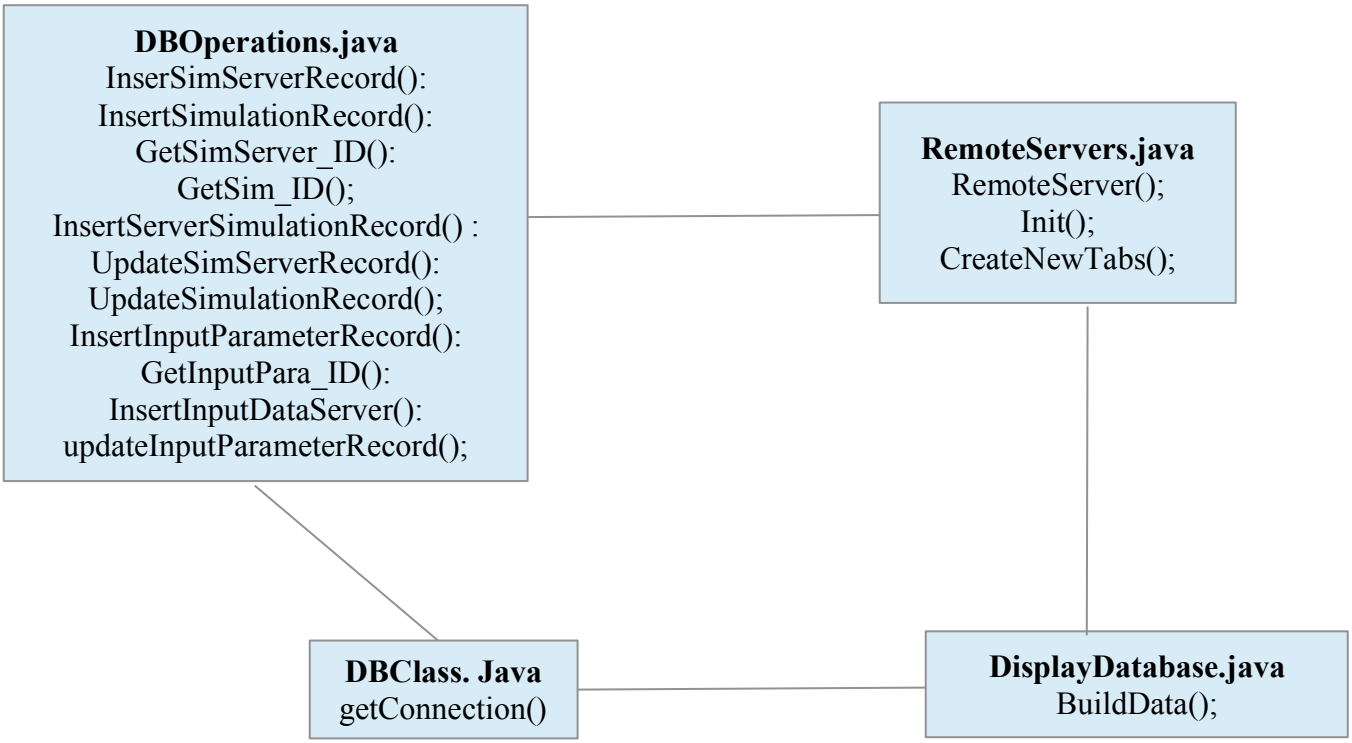
Purpose: Populates the choicebox with input data and its local paths

Methods:

4.7.1 InputScenario_ComboItems(): Initializes the input data and its local paths

```
public class InputScenario_ComboItems
    {
        private String InputData_Name;
        private String Path;
        public InputScenario_ComboItems(String I_Name, String I_Path)
        {
                this.InputData_Name = I_Name;
                this.Path = I_Path;
        }
        @Override
        public String toString()
        {
                return InputData_Name;
        }
        public String getPath()
        {
        return Path;
        }
    }
```

Section 5 — A Java Class diagram for registering a simulation server on the Metadata server



5.1. Class: RemoteServers.java

Purpose: Implements the Cyclus manages server(s) views with 3 tabs: Simulation Server, Input Parameters, and View Simulations

Methods:

5.1.1 Remote(): Initializes the grid pane

5.1.2 init(): Initializes the GUI and implements the database related calls to store and update the relevant tables.

5.1.3 CreateNewTabs(): To create tabs on the GUI.

public TabPane CreateNewTab(TabPane rootTabPan, ArrayList<String> TabMemNames)

```
{
        for(int i=0;i<TabMemNames.size();i++)
        {
                Tab NewTab = new Tab();

                HBox hbox = new HBox();

                NewTab.setText(TabMemNames.get(i));

                NewTab.setId(String.valueOf(i));

                NewTab.setClosable(false);

                hbox.getChildren().add(new Label(TabMemNames.get(i)));

                hbox.setAlignment(Pos.CENTER);

                NewTab.setContent(hbox);

                rootTabPan.getTabs().add(NewTab);
        }

        return rootTabPan;
}
```

Clear(): To clear the textbox text values from different text boxes.

5.2. Class: DisplayDatabase.java

Purpose: This is for displaying available simulations on a tabview

Methods:

5.2.1 buildData(): The following is for building data to display.

```
public static ObservableList<SimulationServermaster> buildData()
{
        ObservableList<SimulationServermaster> data =
FXCollections.observableArrayList();

        Connection con = null;

        Statement stmt = null;

        try
        {
```

```
                DBClass objDBConnection = new DBClass();

                con = objDBConnection.getConnection();

                stmt = con.createStatement();

                String SQL = "Select * from Sim_Servers";

                ResultSet rs = stmt.executeQuery(SQL);

                while(rs.next())
                {
                        SimulationServermaster sm = new SimulationServermaster();
                        sm.Server_ID.set(rs.getInt("Server_ID"));
                        sm.Server_Name.set(rs.getString("Server_Name"));
                        sm.Location.set(rs.getString("Location"));
                        sm.Description.set(rs.getString("Description"));
                        data.add(sm);
                }
        }
        catch(Exception e)
        {
    e.printStackTrace();
    System.out.println("Error on Building Data");
  }
        finally
        {
            //finally block used to close resources
            try
            {
                  if(stmt!=null)
              con.close();
            }catch(SQLException se)
            {
            }// do nothing
        }
        return data;
}
```

5.3. Class: DBOperations.java

Purpose: To interact with Datatables from database. Mainly for storing and updating data values.

Methods:

5.3.1 InserSimServerRecord(): To insert records to Simulation Server Table
5.3.2 InsertSimulationRecord(): To insert records to Simulation Table
5.3.3 GetSimServer_ID(): Update records from Simulation Server Table
5.3.4 GetSim_ID(): Get the simulation ID bases on the simulation name
5.3.5 InsertServerSimulationRecord(): To insert records to Simulation Server Table
5.3.6 UpdateSimServerRecord(): To Update SimulationServer Table records
5.3.7 UpdateSimulationRecord(): To Update Simulation Table
5.3.8 InsertInputParameterRecord(): To insert records to Simulation Server Table
5.3.9 GetInputPara_ID(): Get the input parameter ID based on the name of the input paramter
5.3.10 InsertInputDataServer(): To insert records to Input Data Server Table
5.3.11 updateInputParameterRecord(): To Update the Input parameter table

Few example codes of the above methods:

```
// this method is used to insert records into Simulations Table
public static void insertSimulationRecord(String Sim_Name, String Sim_Version,
String Sim_Des, String AccessURL, String OutputURL)
{
        Connection conn = null;

        Statement stmt = null;

        try
        {
                //Execute a query
                 DBClass objDBConnection = new DBClass();

                 conn = objDBConnection.getConnection();

            stmt = conn.createStatement();

            String sql = "INSERT INTO Simulations " +
                    "VALUES (LAST_INSERT_ID(),"+ "'" + Sim_Name + "'" + "," + "'" +
Sim_Version + "'"+ "," + "'" + Sim_Des + "'" + "," + "'" + AccessURL + "'" + "," + "'" +
OutputURL + "'" + ")";

            stmt.executeUpdate(sql);

           }catch(SQLException se){
             //Handle errors for JDBC
             se.printStackTrace();
           }catch(Exception e){
             //Handle errors for Class.forName
             e.printStackTrace();
```

```
        }finally{
          //finally block used to close resources
          try{
            if(stmt!=null)
              conn.close();
          }catch(SQLException se){
          }// do nothing
          try{
            if(conn!=null)
              conn.close();
          }catch(SQLException se){
            se.printStackTrace();
          }//end finally try
        }//end try
}
```

5.4. Class: DBClass. java class

Purpose: To creates Metadata server database connection object

Methods:

5.4.1 getConnection(): Creates the connection object

```
public Connection getConnection() throws ClassNotFoundException,
SQLException{

            Connection conn = null;

            Class.forName("com.mysql.jdbc.Driver");

            //Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //Open a connection
        conn = DriverManager.getConnection(DB_URL, USER, PASS);

        return conn;
 }
```