Logo, company name

Description automatically generated

**Cell-DEVS: Web-Application for CD++**

**Developer Manual**

*Advanced Real-Time Simulation Laboratory,*

*Department of System and Computer Engineering,*

*Carleton University, Ottawa, Canada*

Date: 11th April 2022

**Front-end Application:**

* **Framework:** HTML5, Vanilla JavaScript, CSS, BootStrap5
* **External library:** Ace.js and JSZip.js

The major component of the front-end logic consists of the editor windows. For the code editor windows Ace.js (<https://ace.c9.io/>) library is used. To create a code editor object using JavaScript with number and brackets highlighted and dark theme enabled following snippet is used,

1. **Editor windows:** In HTML create an empty div for editor window,

<div id="editor"></div>

var editor = ace.edit("editor");

editor.setTheme("ace/theme/monokai");

editor.session.setMode("ace/mode/javascript");

editor.getSession().setUseWorker(false);

Similarly, other editors are also created.

1. **Multiple files as request body:** For running the simulation, fetch API is used, and all the files are passed as form data. The request body is created using following snippet,

Array.from(files).forEach(file => {

        formData.append('files', file);

    });

Here, “files” is the array object which contains all the files that needs to be sent to the service.

1. **Zip the files to download:**

To zip the files after the simulation runs, the JSZip library is used,

zip = new JSZip();

zip.file(filename, new File([fileData],filename, {

                                type: "text/plain"

                            })

To return a zip object when a user clicks on “Download”,

function downloadZIP(){

    zip.generateAsync({type:"blob"})

    .then(function(content) {

        saveAs(content, projectName+".zip");

    });

}

1. **Visualization:** For visualization, modified app-embed project is used,

import WDSV from '../app-embed/WDSV.js'

On the successful simulation run (status code of the API call is 200 OK),

var viewer = new WDSV(vis, {},files);

viewer.On("Error", (error) => alert(error.toString()));

Here, files is an file array which contains, “.ma”, “.pal”,”.val” and “.log” files.

**Backend Service:**

* **Framework: Spring-Boot**

The backend services read files from two different folders, for sample projects the files are read from log-files folder and the simulators are kept in a run-time folder to execute the models. The location of this folders is set as a property in the application.properties file,

log.folder=/opt/tomcat/webapps/CD++/log-files

runtime.folder=/opt/tomcat/webapps/CD++/run-time

1. **Add Sample Projects:** To add sample project, simply create a project folder with the model, value, macro, palette and state values files. The project folder name should follow the format “{{project-name}} (Simulator name)”, for instance “Life (CD++)”. Use FileZilla to transfer folder from local machine to server.
2. The run-time folder will contain the simulators that are used by the service, currently, two simulators are configured CD++ and santi.
3. The web-application’s flow starts from controller file, WebController.java. The controller has three endpoints,
   1. Path: /cell-devs/upload

Method: POST

Function: Upload the files to run-time folder under a subfolder with unique id (UUID) and calls the service method to run the simulation.

* 1. Path: /cell-devs/projects

Method: GET

Function: Returns a list of strings, the project names that are currently available in log-files folder.

* 1. Path: /cell-devs/project/{project-name}

Method: GET

Function: Returns a zipped project folder from the log-files named {{project-name}}.

1. The services contain three components,
   1. **ZipFile.java**: Converts list of files to byte array output stream and returns the zip file data as byte array.
   2. **FileResponse**.**java:** Constructs a Response entity with the file name and file content as byte array.
   3. **Simulation**.**java:** Contains two methods, runLinux and runWindows,
      1. **runLinux**: This method runs the linux version of simulator, santi. ProcessBuilder is used to run the simulation using the santi executable from the run-time folder.
         1. On the successful run the file named “output.log01” is copied to the file “messages.log” and all the files that match the pattern “output.\*” is deleted. All the remaining files are zipped using ZipFile.java and response entity is returned using FileResponse.java class.
         2. On failure all the output.\* files are deleted and the remaining files are zipped and returned with the status code of 400 Bad Request.

Before returning the response entity in both cases the temporary project folder is deleted.

* + 1. **runWindows:** This method is used to run the windows simulator CD++. Since the application is deployed on the linux server, “wine” utility is used. For any command that runs in windows, if we want to run it with linux add wine before the command. In windows,

SIMU.exe -mmodel.ma -lmessages.log -t00:01:00:000

In linux this command will be,

wine SIMU.exe -mmodel.ma -lmessages.log -t00:01:00:000

**Deployment:**

To deploy the web-application on a Apache Tomcat server, follow the steps.

1. Create a new user called “tomcat” for /opt/tomcat directory by running the following command,

sudo useradd -m -d /opt/tomcat -U -s /bin/false tomcat

1. Install JDK package,

sudo apt update

sudo apt install default-jdk

1. Check if java is installed correctly,

Java -version

1. Download the Apache tomcat bundle,

wget <https://dlcdn.apache.org/tomcat/tomcat-10/v10.0.20/bin/apache-tomcat-10.0.20.tar.gz>

1. Extract the bundle to /opt/tomcat,

sudo tar xzvf apache-tomcat-10\*tar.gz -C /opt/tomcat --strip-components=1

1. Grant the user created in step 1 owner’s permission,

sudo chown -R tomcat:tomcat /opt/tomcat/

sudo chmod -R u+x /opt/tomcat/bin

1. To configure admin user,

sudo nano /opt/tomcat/conf/tomcat-users.xml

Edit this file’s username, password values.

<role rolename="manager-gui" />

<user username="manager" password="manager\_password" roles="manager-gui" />

<role rolename="admin-gui" />

<user username="admin" password="admin\_password" roles="manager-gui,admin-gui" />

1. To remove the restriction from the manager page,

sudo nano /opt/tomcat/webapps/manager/META-INF/context.xml

Edit this file by commenting the lines with <Valve> tag.

...

<Context antiResourceLocking="false" privileged="true" >

<CookieProcessor className="org.apache.tomcat.util.http.Rfc6265CookieProcessor"

sameSiteCookies="strict" />

<!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve"

allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->

<Manager sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|Integer|Long|Number|String)|org\.apache\.catalina\.filters\.Csr>

</Context>

1. Repeat step 8 for host-manager file as well by opening and editing the context.xml file,

sudo nano /opt/tomcat/webapps/host-manager/META-INF/context.xml

1. Creating a systemd service,

To get the running JDK path use following command,

sudo update-java-alternatives -l

The output should look like following,

java-1.11.0-openjdk-amd64 1111 /usr/lib/jvm/java-1.11.0-openjdk-amd64

Here, /usr/lib/jvm/java-1.11.0-openjdk-amd64 is the path of the running JDK.

1. Create and edit the file called tomcat.service,

sudo nano /etc/systemd/system/tomcat.service

Add following content in this file,

[Unit]

Description=Tomcat

After=network.target

[Service]

Type=forking

User=tomcat

Group=tomcat

Environment="JAVA\_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64"

Environment="JAVA\_OPTS=-Djava.security.egd=file:///dev/urandom"

Environment="CATALINA\_BASE=/opt/tomcat"

Environment="CATALINA\_HOME=/opt/tomcat"

Environment="CATALINA\_PID=/opt/tomcat/temp/tomcat.pid"

Environment="CATALINA\_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC"

ExecStart=/opt/tomcat/bin/startup.sh

ExecStop=/opt/tomcat/bin/shutdown.sh

RestartSec=10

Restart=always

[Install]

WantedBy=multi-user.target

1. To start the server,

cd /opt/tomcat/bin

sudo ./shutdown.sh

sudo ./startup.sh

Use step 12 to restart the server after each change in the deployment.

**Using FileZilla to Transfer files from/to Server**

1. Download FileZilla client from <https://filezilla-project.org/>
2. Go to files->Site manager to configure our server for transfer.

Graphical user interface

Description automatically generated

1. Fill the details as mentioned in the above image, use different hostname and port as required. For some servers Logon type will be ssh key, use private key file for authentication.
2. Clone the project files to local machine from github,

git clone <https://github.com/lamrin13/cell-devs-web-ui>

git clone <https://github.com/lamrin13/cell-devs-service>

1. Create a new folder on the server in the webapps folder named “CD++”,
2. Copy all the files from the cell-devs-web-ui to the opt/tomcat/webapps/CD++ folder, after the transfer is done, the filezilla application window will look like following.

A screenshot of a computer

Description automatically generated

Restart the server after the changes.

1. To deploy the service, go to cell-devs-services folder in local machine. Use any command line based on the OS, go to the cell-devs-services directory and run following command to generate executable WAR file of the spring-boot project,

mvn install

1. The WAR file will be created in the cell-devs-services/target folder.
2. Create a new folder on the server for java application named webapps-javaee,

mkdir /opt/tomcat/webapps-javaee

1. Transfer the WAR file from the local to webapps-javaee folder,

Graphical user interface, text, application

Description automatically generated

1. Again, restart the server and wait for 1 minute to synchronize the files on the server.
2. Access the url http://{IP of the server}}:8080/CD++/Cell-DEVS