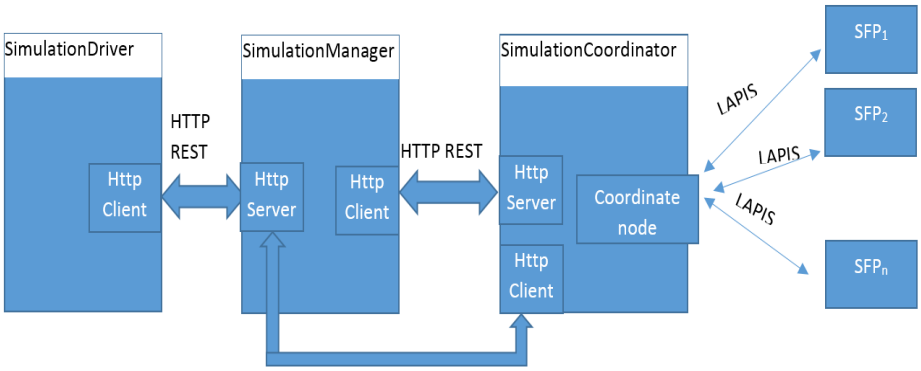
The Architecture of Simulation management



SimulationManager

SimulationManger initialized with a simulation process description XML document which is the configuration of the simulation process. The document describe function of each SFP, include its name, input and output parameter. SimulationManager start the http listening as an http sever. When it receive a request, it will parsing the request and match with the configuration to find out which SFPs need to be called and plan the calling schedule, then as an http client , make a calling request to SimulationCoordinator. For we donot know exactly when the SFP will finished it execution, the Http sever also listing the report of the SimulationCoordinator which will report the SFP execution result. When the SimulationManager get response form the SomulationCoordinator, it will parsing the response, and store the execution result of the SFPs.

Member variable

String configFileName // simulation configuration document name.
String REQUEST_URI // the url of SimulationCoordinator
Map<String, String> inputMap // hash table to store the input parameters
Map<String, String> outputMap // output parameters
List<String> finishedSfpList // store sfp name which has finished
List<String> sfpList // store all the sfp's name

Member function

@post
Public String handlePostRequest(StringRepresentation re)

It is overloaded function as the subclass of SeverResource, which parsing the request string re. If it is a driving request, initialize the outputMap, inputMap, finishedSfpList, then make a call to run simulation process. If it is a result query request, check the results, if it is ready, return the result. If it is the SFP execution report, it parsing the report and store the output of SFPs.

Public String handleRequest(String reqStr)

It is the function parsing the reqStr, then find out the toexecution SFP list, then call the runSimuProcessbyInput to arrange threads to make request to the SimulationCoordinator.

Public Document parseStrXML(String str)
It is a helper function, parsing an xml format string.

Public List<String> findSfpToRunByInput()

According the inputMap, it traversal the all the SFP nodes to find which need and satisfied to be executed at next step. It return the SFP to execute list.

Public List<String> findSfpToRunByOutput()

According the output specification of SimulationDriver's request, it traversal the all the SFP nodes to find which need and satisfied to be executed at next step. It return the SFP to execute list.

Public String oneRunSfp(String sfpName, int timestamp)

It make a request to SimulationCoordnator to run a SFP. The request string is also a XML format string which embedded the instance of input parameters.

Public Boolean isGetAllOutput()

It judge weather the output which are specified in the SimulationDriver's request are collected, through searching in the return result which stored in outputMap.

Public void runSimProcessbyInput(String input,
Public Boolean is RunAllSfp()

Weather all the SFPs which need to execute in the list are all been forked.

Public void runSimuProcessbyInput(String input, String output, int timestamp)

Start threads to request to the SimulationCoordinator to fork SFP which is in the return by findSfpToRunByInput.

Inner Classes

sfpWorker implement Runable interface, call the oneRunSfp to make a request to SimulationCoordinator, is a the worker part of a thread.

SimulationCoordinator

It is the coordinator between SimulationManager and SFP nodes. As a http server, It is listening the request form SimulationManager. When a request come, it parsing the input XML string, instace the input paramenters, then use the coordinator LAPI to set the input parameter's value of a SFP, then make a loop query for the return output value. If the values returned, use the http client API to post the XML packed output to the SimulationManager. It also maintain the active SFP nodes information.

Member variable

String lapisNodeName //coordinator node name
String coordinatorAddress // url
String REQUEST_URI // the simulationManager's url
String lapisApi // coordinator's handle
Map<String, String> onlineSfpListMap

Member function

@post
Public String handlePostRequest(StringRepresentation re)
It is overloaded function as the subclass of SeverResource, which parsing the

request string re to get the SFP name, input parameters and output parameters. Then to find if the SFP is online, if it is online, execute the SFP. For we donot know when the result return form SFP, an “ok” is return immediately as response to the SimulationManager.

Public void init()

It initialize the lapisApi and register the callback to deal with the SFP nodes’ statue.

Private void executeSfp(String input, String output, String funcionName)

It start a thread to fork the SFP.

Private String wrapReturnValue(String name, String output, String returnValue)

For the response to the SimulationManager need to be XML style string. We prepare the output string for the named parameter.

Inner Classes

LapisNetworkCallback implements NetworkChangeCallback interface.

It deal with the statues of SFP nodes. When they log on the internet, we add it on the online list, when they exit, delete it.

LapisWorker implements Runable interface.

The run function of the LapisWorker takes charge of parsing the input and output specification. And make parameter instance according the type definition in the input and output string, then loop requiring the SFP nodes until get the result. Once it get the result, pack it as a XML string then report the result to the SimulationManager.

MotorModelDriver

MotorModelDriver is the client side; it can be any web browser or other kind of apps which send http post to the SimulationManger.

The post data format is XML format, but not a integrate XML document strictly, the format include two parts: input and output (it can be empty). Input part package the initial parameter values. Output part describes what are the expected outputs by the simulation process. So we have two ways to drive the simulation process: driven by output and driven by input.

Driven by output:
1 Build an input parameters set and output parameter set which contain the instance value for parameters.
2 Search the configuration file to find a subset of SFP nodes which produces those output.
3 In the subset of SFP nodes, invoke one by one
4 If the SFP node need set input parameters, and can find those parameters in the input parameter sets, set those. Then invoke SFP node to execute, update the input parameters set and output parameters set.
5 If the SFP node need set input parameters, and cannot find those parameters in the input parameter sets. Those parameter need wait other SFP output the values for those parameter, then go to 2
6 if all the outputs need are produced, the loop is end, else go to 3

Driven by input:

1 Build an input parameters set and output parameter set, which contain the instance value for parameters.
2 Search the configuration file to find a subset of SFP nodes whose input parameters are all ready.
3 in the subset of lapis node, invoke one by one, get output, update input and output parameter set.
4 if all the SFP nodes had been executed, end loop, else go to 2

The example of post data in MotorDriver.java
"<output><parameter><name>speed</name><type>int</type></parameter></output><timeStamp>1</timeStamp>";