RISE User's Guild Manual (Integrated version)

(DRAFT v1.0 Not for distribution)

Sixuan Wang, Gabriel A. Wainer
Dept. of Systems and Computer Engineering
Carleton University, Ottawa, Canada

Abstraction

For the user's point of view, RISE is very easy to use. RISE supports different CD++ versions of DEVS/Cell-DEVS formalism, including DCDpp for normal and distributed simulation and Lopez with different ports/variables. You do not need to install complicated simulation software, the only thing you need is to choose the right RISE API, upload the model file and run it online with standard HTTP requests, then retrieve simulation results from the website.

The website (http://134.117.53.66:8080/cdpp/sim/workspaces) is still under construction. If you have any questions and commons, or if you would use RISE often and want to open an account, please contact with Sixuan swang@sce.carleton.ca.

1. Introduction

Most modeling and simulation methods run on single-user workstations, which normally cost too much time of installation and configuration of all the software and dependencies needed by the simulation. It is better to realize a much easy way to remote access of the simulation resources with web service interfaces, improving data accessibility, interoperability and user experience. In this way, users can reuse and share simulation resources on site without worrying about the capability of their local machine CPU or memory. Plus, with the help of advanced distributed simulation technologies, the simulation can be executed on distributed computers via communication networks, which can further improve interoperability and speed up the execution time.

RESTful WS can solve these issues. It is to transfer message representations of Web resources using a set of uniform stateless operations. The resources here are not only files, but also a query, a table, or any concepts. Access to RESTful WS is through Web resources (URIs) and XML messages using HTTP methods (GET, PUT, POST and DELETE). Its strengths of simplicity, efficiency and scalability make RESTful WS an excellent candidate to perform remote simulation. Based on these ideas, in [1][2], the authors presented the first existing RESTful Interoperability Simulation Environment (RISE) middleware. The main objective of RISE is to support interoperability and mash-ups of distributed simulations regardless of the model formalism, model language or

simulation engine. RISE is accessed through Web resources (URIs) and XML messages using HTTP methods. Implementations can be hidden in resources, which are represented only via URIs. Users can run multiple instances as needed, which are persistent and repeatable by specific URIs. The HTTP methods are typical four types: GET (to read a resource), PUT (to create or update a resource), POST (to append data to a resource), and DELETE (to remove a resource).

Currently, RISE supports different versions of CD++ (Modeling and Simulation language based on DEVS formalism). The uniformed API between seems a classic website URL like "http://134.117.53.66:8080/cdpp/sim/workspaces", attached by the following services (detailed RISE API can be seen from [3]):

- *../{userworkspace}* contains of all simulation services for a given user.
- *../{userworkspace}/{servicetype}* contains of all frameworks for a given user and simulation service type (e.g., a simulation engine CD++).
- ***../{userworkspace}/{servicetype}/{framework}*** allows interacting with a framework (including the simulation's input initial files, the model's configuration and source code).
- ***../{userworkspace}/{servicetype}{framework}/simulation*** allows interacting with the simulator execution.
- ***../{userworkspace}/{servicetype}/{framework}/results*** contains of the simulation outputs.
- *../{userworkspace}/{servicetype}/{framework}/debug* contains of the model-debugging files.

For a specific simulation, we would usually use the bold API mentioned above. We can realize the remote simulation by using these URIs with HTTP methods. For example, after getting initial model and configuration files for simulation, we use PUT to create a framework with the configuration XML file and POST supported initial model files in a ZIP file. Then, this new created simulation environment can be executed by using PUT to *{framework}/simulation*, then we wait the simulation to finish and GET simulation results files from *{framework}/results*.

In each HTTP response (no matter from which URI), the Response Status is very informative. Normally, it means the request is successful if you see 200 (OK) and 201 (Created); otherwise, some error or unexpected things would have happened, such as 400(bad request), 401(unauthorized), 403(forbidden), 404(not found), 406(not acceptable), 501(not implemented).

2. Use API

DEVS is a Modeling and Simulation formalism that has been widely used to study the dynamics of discrete event systems. **Cell-DEVS** is a DEVS-based formalism that defines spatial models as a cell space assembled of a group of DEVS models connected together. **CD++** is a modeling and simulation language/toolkit capable of executing DEVS and Cell-DEVS models that has proven to be useful tools for executing complex models.

In RISE, there are various CD++ versions available, including **DCDpp** for normal and distributed

simulation and **Lopez** with different port/variable values. The use of DCDpp and Lopez are very similar, the differences are only represented in {ServerType} in URI and some XML configuration format (different version needs different configuration and files).

For GET HTTP request, you just need simply access particular website under the URI http://134.117.53.66:8080/cdpp/sim/workspaces. For PUT/POST/DELET HTTP request, you need a client program to encapsulate your request and forward to related URI. We provide this kind of client program in RISE_Test.zip for your convenience (of course you can build your own client program by yourself for specific client requirement).

2.1 DCDpp

DCDpp is the default and normally used version. DCD++ extends the CD++ simulation engine to perform distributed simulation among various geographically distributed machines. Normally, we usually use its one machine version (the distributed machine version is for special purpose).

Preparation:

1. Unzip RISE_Test.zip, assume "E:\RISE_Test"(make sure there are /lib and RESTful_CDppTest.jar)
2. Put your XML and ZIP files under "E:\RISE_Test", or simply copy "life.xml" and "life.zip" from "E:\ RISE_Test\models\dcdpp\life" to "E:\RISE_Test".
3. open command line (cmd in windows or terminal in linux), go to "E:\RISE_Test".

Use API:

Step 1) Create your workspace, DCD++ service and a framework. This can be done in one request, "LifeModel" is the framework defined by you:
>> java -jar RESTful_CDppTest.jar PutXMLFile test test test/dcdpp/**LifeModel life.xml**
Then see http://134.117.53.66:8080/cdpp/sim/workspaces/test/dcdpp/LifeModel

Step 2) Submit the life model files to your framework URI, as follows:
>> java -jar RESTful_CDppTest.jar PostZipFile test test test/dcdpp/**LifeModel**?zdir=**life life.zip**
Then see http://134.117.53.66:8080/cdpp/sim/workspaces/test/dcdpp/LifeModel

Step 3) Run/Re-run Simulation, as follows:
>> java -jar RESTful_CDppTest.jar PutFramework test test test/dcdpp/**LifeModel**/simulation
Then see http://134.117.53.66:8080/cdpp/sim/workspaces/test/dcdpp/LifeModel/simulation

Step 4) Retrieve Simulation results, click "Download Last Simulation Results" or
http://134.117.53.66:8080/cdpp/sim/workspaces/test/dcdpp/LifeModel/results

Check Error

If error happened, see "Download Model Debugging/Statistics Logs" or
http://134.117.53.66:8080/cdpp/sim/workspaces/test/dcdpp/LifeModel/debug

If you want to delete the framework

>> java -jar RESTful_CDppTest.jar DeleteFramework test test test/dcdpp/LifeModel

It is suggested that you delete the framework that is not working, because it takes space in Server.

2.2 Lopez

Lopez is an extended version of CD++ for allowing the cells to use multiple state variables and to use multiple ports for inter-cell communications. Detail specification could be seen in [5]. Additionally, for Cell-DEVS, we can have option to see only Y message and see 2D visualization of simulation result by DrawLog. Now, this version is integrated in RISE.

Preparation:

1. Unzip RISE_Test.zip, assume "E:\RISE_Test" (make sure there are /lib and RESTful_CDppTest.jar)
2. Put your XML and ZIP files under "E:\RISE_Test", or simply copy "life.xml" and "life.zip" from "E:\ RISE_Test\models\ lopez\life-ori" to "E:\RISE_Test".
3. open command line (cmd in windows or terminal in linux), go to "E:\RISE_Test".

Use API:

Step 1) Create your workspace, DCD++ service and a framework. This can be done in one request:

>> java -jar RESTful_CDppTest.jar PutXMLFile test test test/lopez/**LifeModel life.xml**

Then see http://134.117.53.66:8080/cdpp/sim/workspaces/test/lopez/LifeModel

Step 2) Submit the life model files to your framework URI, as follows:

>> java -jar RESTful_CDppTest.jar PostZipFile test test test/lopez/**LifeModel**?zdir=**life life.zip**

Then see http://134.117.53.66:8080/cdpp/sim/workspaces/test/lopez/LifeModel

Step 3) Run/Re-run Simulation, as follows:

>> java -jar RESTful_CDppTest.jar PutFramework test test test/lopez/**LifeModel**/simulation

Then see http://134.117.53.66:8080/cdpp/sim/workspaces/test/lopez/LifeModel/simulation

Step 4) Retrieve Simulation results, click "Download Last Simulation Results" or http://134.117.53.66:8080/cdpp/sim/workspaces/test/lopez/LifeModel/results

Check Error

If error happened, see "Download Model Debugging/Statistics Logs" or http://134.117.53.66:8080/cdpp/sim/workspaces/test/lopez/LifeModel/debug

If you want to delete the framework

>> java -jar RESTful_CDppTest.jar DeleteFramework test test test/lopez/LifeModel

It is suggested that you delete the framework that is not working, because it takes space in Server.

3. Examples

3.1 DCDpp

3.1.1 Cell-DEVS

XML configuration like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ConfigFramework>
    <Doc> This model Simulates Life using Cell-Devs. </Doc>    //description
        <Files>
          <File  ftype="ma">life.ma</File>                  //list all model files in ZIP file (the ftype should be 'ma' for *.ma; 'ev' for *.ev, 'inc' for *.inc)
          <File  ftype="sup">life.val</File>                //NOTE: ftype should be 'sup' for *.val, if you have other type of files to be uploaded, use 'sup' for it.
        …
      </Files>
    <Options>                                   //options, not necessary
        <TimeOp>00:00:02:000</TimeOp>           //stop time
        <ParsingOp>false</ParsingOp>            //print extra info when the parsing occurs
    </Options>
    <DCDpp>
        <Servers>                           //distributed machines, usually one machine of the server
            <Server IP="134.117.53.66" PORT="8080">    //server
           <Zone>life(0,0)..(19,19)</Zone>        //partition, if one machine, put whole cells here
       </Server>
        </Servers>
    </DCDpp>
</ConfigFramework>
```

Model Definition

Detail MA definition could be found in [4], here give some description based on life.ma

```
[top]                                   //top
components : life                       //lists of components, if more than one, needs Links
[life]                                  //life component
type : cell
width : 20                              //width size
height : 20                             //height size
delay : transport                         //transport / inertial
defaultDelayTime : 100
border : wrapped                           //wrapped / nowrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)          //neighbors
neighbors : life(0,-1)    life(0,0)    life(0,1)
```

```
neighbors : life(1,-1)    life(1,0)    life(1,1)
initialvalue : 0                                    //initial values, can be a initial file
initialrowvalue :    0          01111000111100011110
initialrowvalue :    1          00010001111000000000
…
initialrowvalue : 13          00110000011110000010
initialrowvalue : 14          00101111000111100011
localtransition : life-rule
[life-rule]                                                    //rules to obey
rule : 1 100 { (0,0) = 1 and (truecount = 3 or truecount = 4) }          //value to be/delay/conditions
rule : 1 100 { (0,0) = 0 and truecount = 3 }
rule : 0 100 { t }
```

3.1.2 DEVS

XML configuration

DEVS models contains src C++ files. Other configuration are similar with the one in Cell-DEVS.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <ConfigFramework>
  <Doc>This DEVS model simulates Barber shop .</Doc>
- <Files>                                    //list all model files in ZIP file, if you have other
type of files, use 'sup' for ftype.
  <File ftype="ev">barber.ev</File>
  <File ftype="ma">barber.ma</File>
  <File ftype="src">Reception.cpp</File>
  <File ftype="hdr" class="Reception">Reception.h</File>
  <File ftype="src">Checkhair.cpp</File>
  <File ftype="hdr" class="Checkhair">Checkhair.h</File>
  <File ftype="src">Cuthair.cpp</File>
  <File ftype="hdr" class="Cuthair">Cuthair.h</File>
  </Files>
- <Options>
  <TimeOp>null</TimeOp>
  <ParsingOp>false</ParsingOp>
  </Options>
- <DCDpp>
- <Servers>
- <Server IP="134.117.53.66" PORT="8080">          //list all model name in the server
  <MODEL>reception</MODEL>
  <MODEL>cuthair</MODEL>
  <MODEL>checkhair</MODEL>
  </Server>
  </Servers>
  </DCDpp>
```

</ConfigFramework>

Model Definition

Zip file contains all src files list in XML configuration, each file is an atomic model and would specify Exteral/Interal/Output functions with time advancing. The MA file like the one in Cell-DEVS with slight diffenrece.

[top] *//list all components and links*
components : reception@Reception Barber
in : newcust next
out : cust finished
Link : newcust newcust@reception
Link : out@Barber next@reception
Link : out@Barber finished
Link : cust@reception in@Barber
[reception] //list variables used in the model
numberofChairs : 8
preparationTime : 00:00:01:000
openingTime : 09:00:00:000
closingTime : 16:00:00:000
[Barber]
components : checkhair@Checkhair cuthair@Cuthair
in : in
out : out
Link : in cust@checkhair
Link : finished@checkhair out
Link : cutcontinue@checkhair cutcontinue@cuthair
Link : progress@cuthair progress@checkhair
[checkhair]
preparationTime : 00:00:09:000
[cuthair]
preparationTime : 00:00:11:000

3.2 Lopez

3.2.1 Cell-DEVS with multiple variables

XML configuration

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ConfigFramework>
<Doc> This model simulates Life using Cell-Devs. It uses Lopez linux version with multiple neighbor ports (value). </Doc>
<Files>
<File ftype="ma">life.ma</File> *//list all model files in ZIP file (the ftype should be*

*'ma' for *.ma; 'ev' for *.ev, 'inc' for *.inc)*

```
        <File ftype="sup">life.val</File>                  //pay attention to 'sup' for val, if you have other type of files to be uploaded, use 'sup' for it.
        <File ftype="stvalues">life.stvalues</File>    //the initial file for new value
         </Files>
    <Options>
        <TimeOp>00:00:02:000</TimeOp>
        <OnlyYMsgOP>true</OnlyYMsgOP>      //only show Y message in log files, optional
        <ParsingOp>false</ParsingOp>
    </Options>
    <DrawLog>                          //for Cell-DEVS only, see 2D visualization in *.drw file, same directory of log files; if you don't want the drw file, just omit this part
        <CoupledModel>life</CoupledModel>              //the model wanted to draw
        <Width>3</Width>                                  // Width used to represent numeric values
        <Precision>0</Precision>                       //Precision used to represent numeric values
        <NotPrintZero>true</NotPrintZero>          // Don't print the zero value
        <TimeInterval>00:00:00:100</TimeInterval>    // Interval time between each two
    </DrawLog>
</ConfigFramework>
```

MA file

```
[top]
components : life
[life]
type : cell
width : 21
height : 21
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1)    life(0,0)    life(0,1)
neighbors : life(1,-1)    life(1,0)    life(1,1)
initialvalue : 0
initialrowvalue : 4    000011100000000000000
initialrowvalue : 5    000011100010000000000
…
initialrowvalue : 15 000000000000001110001
initialrowvalue : 16 000000000000001110011
localtransition : conrad-rule
statevariables: value          //new state
initialvariablesvalue: life.stvalues          //state initial values
```

```
[conrad-rule]
rule : { $value } { $value := 1; } 100 { $value =1 and (truecount = 3 or truecount = 4) }
rule : { $value } { $value := 0; } 100 { $value =1 and (truecount < 3 or truecount > 4) }
rule : { $value } { $value := 1; } 100 { $value =0 and truecount = 3 }
rule : { $value } { $value := 0; } 100 { $value =0 and truecount != 3 }
```

3.2.2 Cell-DEVS with multiple ports

XML configuration

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ConfigFramework>
    <Doc> This model simulates Life using Cell-Devs. It uses Lopez linux version with multiple neighbor ports (value). </Doc>
        <Files>
       <File ftype="ma">life.ma</File>
        </Files>
    <Options>
        <TimeOp>00:00:02:000</TimeOp>
        <OnlyYMsgOP>true</OnlyYMsgOP>      //only show Y message in log files, optional
        <ParsingOp>false</ParsingOp>
    </Options>
    <DrawLog>                          //for Cell-DEVS only, see 2D visualization in *.drw file, same directory of log files; if you don't want the drw file, just omit this part
        <CoupledModel>life</CoupledModel>              //the model wanted to draw
        <Width>3</Width>                               // Width used to represent numeric values
        <Precision>0</Precision>                       //Precision used to represent numeric values
        <NotPrintZero>true</NotPrintZero>          // Don't print the zero value
        <TimeInterval>00:00:00:100</TimeInterval>   // Interval time between each two
        <NeighborPort>value</NeighborPort>       //the special port to see, none if normal model
    </DrawLog>
</ConfigFramework>
```

MA file

```
[top]
components : life
[life]
type : cell
width : 21
height : 21
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1)    life(0,0)    life(0,1)
```

neighbors : life(1,-1) life(1,0) life(1,1)
initialvalue : 0
initialrowvalue : 4 00001110000000000000
initialrowvalue : 5 00001110001000000000
…
initialrowvalue : 15 00000000000001110001
initialrowvalue : 16 00000000000001110011
localtransition : conrad-rule
neighborports : value //new neighborport, (value1 value2) if more than one
[conrad-rule]
*rule : { **~value** := 1; } 100 { (0,0)**~value** = 1 and (statecount(1, **~value**) = 3 or statecount(1, ~value) = 4) }*
rule : { ~value := 0; } 100 { (0,0)~value = 1 and (statecount(1, ~value) < 3 or statecount(1, ~value) > 4) }
rule : { ~value := 1; } 100 { (0,0)~value = 0 and statecount(1, ~value) = 3 }
rule : { ~value := 0; } 100 { (0,0)~value = 0 and statecount(1, ~value) != 3 }

3.2.3 DEVS

XML configuration

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ConfigFramework>
    <Doc> This model simulates Queue using DEVS. It uses Lopez linux version. For Detailed XML configuration please see User Manual.  </Doc>
        <Files>
            <File ftype="ev">queue.ev</File>
            <File ftype="ma">queue.ma</File>
            <File ftype="src">queue.cpp</File>
            <File ftype="hdr" class="Queue">queue.h</File>
        </Files>
    <Options>
        <TimeOp>null</TimeOp>
        <ParsingOp>false</ParsingOp>
    </Options>
</ConfigFramework>
```

MA file

[top]
components : queue@Queue *//components and links*
out : out
in : in done stop
Link : in in@queue
Link : out@queue out
Link : done done@queue

#Link : stop stop@queue
[queue] *//variables used in queue atomic model*
preparation : 00:00:05:000

References

[1] K. Al-Zoubi and G. Wainer. 2009. Using REST Web Services Architecture for Distributed Simulation. In: Proceedings of Principles of Advanced and Distributed Simulation (PADS 2009). pp. 114-121. Lake Placid, New York, USA.

[2] K. Al-Zoubi and G. Wainer. 2010. Distributed Simulation using RESTful Interoperability Simulation Environment (RISE) Middleware. Intelligence-Based Systems Engineering, pages 129–157.

[3] K. Al-Zoubi and G. Wainer. 2010. RESTful Interoperability Simulation Environment (RISE) Middleware Manual.

[4] G. Wainer. 2001. CD++ parallel version User's Guide.

[5] A López, G. Wainer. 2003. Extending CD++ Specification Language for Cell-DEVS Model Definition.