

Multi-agent Environment for Complex SYstems
COsimulation (MECSYCO) - User Guide:
MECSYCO-com-dds

Benjamin Camus^{1,2}, Julien Vaubourg², Yannick Presse²,
Victorien Elvinger², Thomas Paris^{1,2}, Alexandre Tan²
Vincent Chevrier^{1,2}, Laurent Ciarletta^{1,2}, Christine Bourjot^{1,2}

¹Universite de Lorraine, CNRS, LORIA UMR 7503,
Vandoeuvre-les-Nancy, F-54506, France.

²INRIA, Villers-les-Nancy, F-54600, France.
mecsycocom-dds@inria.fr

June 21, 2016

Contents

Introduction	2
1 Installation	3
2 DDS CouplingArtifact	4
2.1 DDSEventCouplingArtifactSender	4
2.2 DDSEventCouplingArtifactReceiver	4
3 Model building	6
3.1 Templates	7
3.1.1 Java Example Template: run configuration (decentralized - Launcher1) . .	7
3.1.2 Java Example Template: run configuration (decentralized - Launcher2) . .	9
3.2 Remarks:	12
4 Example	13

Introduction

The communication package gathers the implementation of means to connect several machines or platforms from the network. As a consequence, MECSYCO enables distributed and decentralized simulations in Java, C++, or hybrid code. In order to do that, this package adapt MECSYCO to the use of OpenSlice DDS¹).

MECSYCO-com-dds is used instead of the usual *CouplingArtifact* (User Guide, section *The coupling artifact*), that is to say that it replace the usual link between agent.

All primitives and classes needed for communication are in *MECSYCO-com-dds 2.0.0*. Two templates are also provided in order to help building DDS based model.

The example is just a little part taken from the case Lorenz created in the *Getting Started*.

¹<http://www.prismtech.com/dds-community>

Chapter 1

Installation

MECSYCO-com-dds has a lightly different install than the other libraries (*User Guide: section MECSYCO's installation guide*). In order to work properly, you need to install *Openslice DDS community edition 6.4* in your computer, and manipulate your computer's environment variables.

Download the last version of *DDS Community edition*¹. Take care of downloading the archive that matches your operating system. You can use the 64bits version for Linux and install both 32bits and 64bits for Windows. Windows requires *Visual C++ Runtime*² for running DDS.

For installing it:

- Extract the folder at the place you want to install it
- In Linux, in the terminal type: source /path/to/dds/release.com
- In Windows, create environment variable (access through computer's properties – > Advanced tab – > Environment Variables button):
 - OSPL_HOME: path to OpenSlice's folder
 - OSPL_PATH: %OSPL_HOME%\bin;%OSPL_HOME%\lib;%OSPL_HOME%\examples\lib
 - OSPL_TMPL_PATH: %OSPL_HOME%\etc\idlpp
 - OSPL_URI: file:///%OSPL_HOME%\etc\config\ospl.xml
 - PATH: %OSPL_PATH%

if the variable already exist, just add the new path

- The jar associated to OpenSlice is provided in the folder (HDE – >... – >jar). For an easier use, copy paste "dcpssaj.jar" in the libs folder of your project, then add it to the build path
- Do not forget the dependencies: MECSYCO-re; Jackson-jar

¹<http://www.prismtech.com/dds-community/software-downloads>

²<https://www.microsoft.com/fr-FR/download/details.aspx?id=48145&lc=1033>

Chapter 2

DDS CouplingArtifact

As said, in order to use DDS model, the usual models need to use special coupling artifact. **DDSEventCouplingArtifactSender** is a writer artifact while **DDSEventCouplingArtifactReceiver** is a reader artifact. For each instance of **DDSEventCouplingArtifactSender** it should exist an instance of **DDSEventCouplingArtifactReceiver**. These instances can be on separate computers. Two instances are matched by the use of a common identifier. The identifier is the sharing information which enables the linking of the sender and the receiver.

2.1 DDSEventCouplingArtifactSender

As its name says, this coupling artifact is used for the output port of an agent. Its constructor uses one parameter:

- **topic:** the name of link. For easy reading, try to indicate which data you are sending and to who (*PortOfDataSendToPortOfReception*)

DDSEventCouplingArtifactSender constructor in Java implementation
public DDSEventCouplingArtifactSender (String topic)
DDSEventCouplingArtifactSender constructor in C++ implementation
Not distributed yet

As a consequence, when you want to use this coupling artifact, you use the method *addOutputCouplingArtifact* of the agent

Example:

- **Creation:** DDSEventCouplingArtifactSender ZOutputToYSender=new DDSEventCouplingArtifactSender("ZOutputToY");
- **Link:** ZAgent.addOutputCouplingArtifact(ZOutputToYSender,"Z");

2.2 DDSEventCouplingArtifactReceiver

This coupling artifact is used for the input port of an agent. Its constructor uses two parameter:

- **topic:** the name of link. For easy reading, try to indicate which data you are sending and to who (*PortOfDataSendToPortOfReception*)
- **aDataType:** type of expected data to receive. It has to be a SimulData type (*User Guide section Simulation Data*)

DDSEventCouplingArtifactReceiver constructor in Java implementation
DDSEventCouplingArtifactReceiver (String aTopic, Type aDataType)
DDSEventCouplingArtifactReceiver constructor in C++ implementation
Not distributed yet

As a consequence, when you want to use this coupling artifact, you use the method `addInputCouplingArtifact` of the agent

Example:

- **Creation:** `DDSEventCouplingArtifactReceiver ZOutputToYReceiver=new DDSEventCouplingArtifactReceiver("ZOutputToY", Tuple1.of(Number.class));`
- **Link:** `YAgent.addInputCouplingArtifact(ZOutputToYReceiver,"Z");`

Chapter 3

Model building

MECSYCO-com-dds is used for communication purpose, as said, decentralized model. As a consequence, there is not only one, but multiple launchers for the simulation. Each of these launchers are part of the whole multi-model where communication are done with DDS bewtween launchers, and with usual coupling artifact inside them.(Figure 3.1)

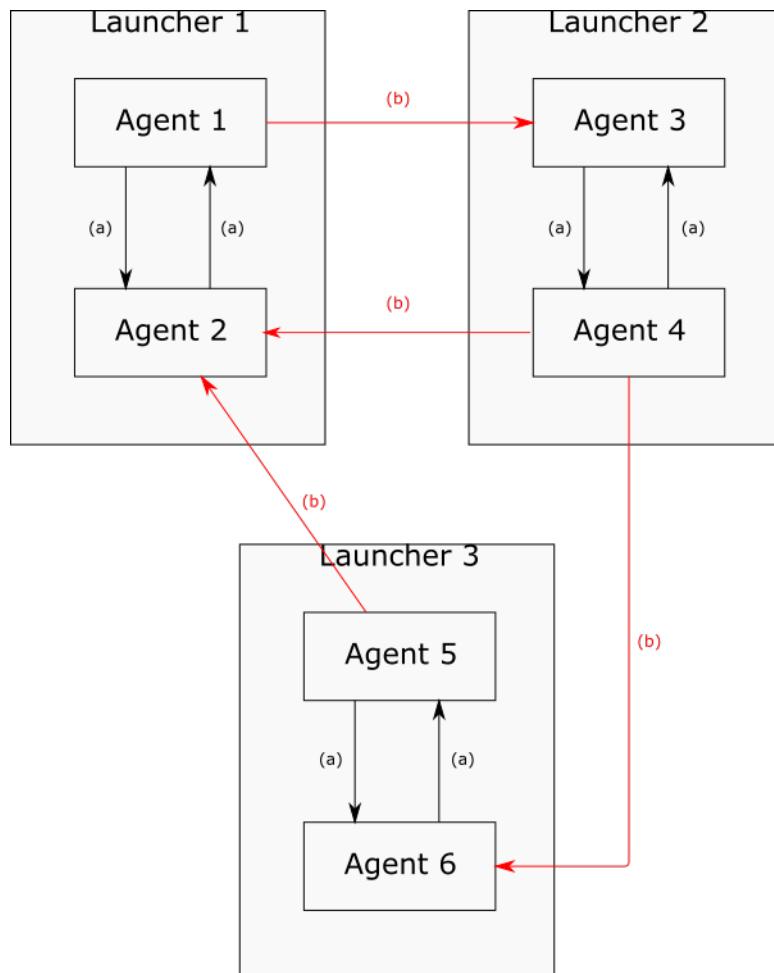


Figure 3.1: Decentralized and distributed multi-model. (a) Link done with usual coupling artifact.
(b) Link done with DDS coupling artifact.

In the case of link (a), only one coupling artifact is used per link, but in the case of (b), an arrow is defined by two coupling artifacts. The start of the arrow with *DDSEventCouplingArtifactSender* and the end with *DDSEventCouplingArtifactReceiver*.

3.1 Templates

The templates can be used for connecting model 1 to model 2 in the previous figure (3.1) without the presence of Model 3.

3.1.1 Java Example Template: run configuration (decentralized - Launcher1)

```

1 import mecsyco.communication.dds.coupling.DDSEventCouplingArtifactReceiver;
2 import mecsyco.communication.dds.coupling.DDSEventCouplingArtifactSender;
3 import mecsyco.core.agent.EventAgent;
4 import mecsyco.core.agent.ObservingMagent;
5 import mecsyco.core.coupling.CentralizedEventCouplingArtifact;
6 import mecsyco.core.exception.CausalityException;
7 import mecsyco.core.type.SimulData;
8 import mecsyco.observing.base.comparator.DataComparator;
9 import mecsyco.observing.base.logging.LoggingArtifact;
10 import mecsyco.observing.jfreechart.bar.LiveBarGraphic;
11 import mecsyco.observing.jfreechart.bar.PostMortemBarGraphic;
12 import mecsyco.observing.jfreechart.event.LiveEventGraphic;
13 import mecsyco.observing.jfreechart.event.PostMortemEventGraphic;
14 import mecsyco.observing.jfreechart.pie.LivePieGraphic;
15 import mecsyco.observing.jfreechart.pie.PostMortemPieGraphic;
16 import mecsyco.observing.jfreechart.xy.LiveXYGraphic;
17 import mecsyco.observing.jfreechart.xy.PostMortemXYGraphic;
18 import mecsyco.observing.jfreechart.xy.PostMortemTXGraphic;
19 import mecsyco.observing.jfreechart.xy.PostMortemXYGraphic;
20 import mecsyco.observing.jfreechart.xy.Renderer;
21 import mecsyco.observing.jzy3d.graphic.Live3DGraphic;
22 import mecsyco.observing.jzy3d.graphic.PostMortem3DGraphic;
23 import mecsyco.observing.swing.dispatcher.SwingDispatcherArtifact;
24 import mecsyco.observing.swing.r.LogToRProject;

25
26 public class Launcher {
27     //Simulation length:
28     public final static double maxSimulationTime = 10;
29
30     public static void main(String args[]) {
31
32         /*****
33         **** AGENTS & MODEL ARTIFACTS ****
34         **** ****
35         **** ****
36
37         // First agent from first model (Model1)
38         //Agent
39         EventMagent agent1 = new EventMagent("nameAgent1",maxSimulationTime);
40         EventMagent agent2 = new EventMagent("nameAgent1",maxSimulationTime);
41
42         //Then Model Artifacts
43         AModelArtifact Agent1Artifact = new AmodelArtifact( /**parameters**/);
44         AModelArtifact Agent2Artifact = new AmodelArtifact(/**parameters**/);
45
46         //Associate agent and artifact
47         agent1.setModelArtefact(Agent1Artifact);
48         agent2.setModelArtefact(Agent2Artifact);
49
50
51         /*****
52         **** COUPLING ARTEFACTS ****
53         **** ****
54
55         //Inside communication
56         CentralizedEventCouplingArtifact Agent1ToAgent2 = new CentralizedEventCouplingArtifact();
57         CentralizedEventCouplingArtifact Agent2ToAgent1 = new CentralizedEventCouplingArtifact();
58
59         //Outside communication
60         //Arrows that go to Launcher2
61         DDSEventCouplingArtifactSender Agent1ToAgent3sender = new DDSEventCouplingArtifactSender("1To3");
62         //Arrows that come from Launcher2
63         DDSEventCouplingArtifactReceiver Agent4ToAgent2receiver = new DDSEventCouplingArtifactReceiver("4To2", SimulData.class);
64         // "1To3" and "4To2" are DDS topics, it needs to be the same in Model2
65         // "SimulData.class" corresponds to the type of the expected value, from the remote model
66
67         //Connection
68         //Inside:
69         agent1.addOutputCouplingArtifact(Agent1ToAgent2, "OutputPortName1");
70         agent2.addOutputCouplingArtifact(Agent2ToAgent1, "OutputPortName2");
71
72         agent2.addInputCouplingArtifact(Agent1ToAgent2, "InputPortName1");
73         agent1.addInputCouplingArtifact(Agent2ToAgent1, "InputPortName2");
74
75         //Outside
76         // Agent1 will send data from OutputPortName3 via the topic "1To3" (output events)
77         agent1.addOutputCouplingArtifact(Agent1ToAgent3sender, "OutputPortName3");
78         // Agent2 will receive Data in InputPortName3 with the value received via the topic "4To2" (input events)
79         agent2.addInputCouplingArtifact(Agent4ToAgent2receiver, "InputPortName3");
80
81
82         /*****
83         **** Operations ****
84         *** Check User Guide: Create your own operations ***
85         **** ****
86
87         /*In the case of internal operation, it does not change
88         * We will then see in the cast of the external link

```

```

89     */
90
91     //In this launcher, we can only apply operation on the link from Agent4 to Agent2
92     //event operation
93     DataOperationTemplate DataOpe= new DataOperationTemplate();
94     Agent4ToAgent2receiver.addEventOperation(DataOpe);
95     //time operation
96     TimeOperationTemplate TimeOpe= new TimeOperationTemplate();
97     Agent4ToAgent2receiver.addTimeOperation(TimeOpe);
98
99     /*************************************************************************/
100    /******LOGGING VISUALIZATION OR POST TREATMENT *****/
101    /****** Check User Guide: MECSCD-visu *****/
102    /*** Check User Guide section Simulation data ***/
103    /*************************************************************************/
104
105    /* Set the agent name for logging if you didn't named it at the creation
106     * (otherwise, an unique default number is attributed)
107     */
108    agent1.setAgentName("Agent1");
109    agent2.setAgentName("Agent2");
110
111    /*Create observing Agent and the dispatcher
112     * Create both for each different display windows you want
113     */
114    ObservingMAgent obsAgent = new ObservingMAgent ("ObserverName", maxSimulationTime);
115    SwingDispatcherArtifact ObsModelArtifact = new SwingDispatcherArtifact ();
116    obsAgent.setDispatcherArtifact(ObsModelArtifact);
117
118    /* Coupling Artifact and connection
119     * Same rules apply here, if the Port to observe is from Model2, use DDS
120     */
121    CentralizedEventCouplingArtifact Agent1Port1ToObs = new CentralizedEventCouplingArtifact();
122    CentralizedEventCouplingArtifact Agent1Port2ToObs = new CentralizedEventCouplingArtifact();
123    CentralizedEventCouplingArtifact Agent2ToObs = new CentralizedEventCouplingArtifact();
124    DDSEventCouplingArtifactReceiver Agent3ToObsReceiver = new DDSEventCouplingArtifactReceiver("3ToObs", SimulData.class);
125    DDSEventCouplingArtifactReceiver Agent4Port1ToObsReceiver = new DDSEventCouplingArtifactReceiver("4Port1ToObs", SimulData.class);
126    DDSEventCouplingArtifactReceiver Agent4Port2ToObsReceiver = new DDSEventCouplingArtifactReceiver("4Port2ToObs", SimulData.class);
127
128    agent1.addOutputCouplingArtifact(Agent1Port1ToObs, "OutputPortName1");
129    agent1.addOutputCouplingArtifact(Agent1Port2ToObs, "OutputPortName3");
130    agent2.addOutputCouplingArtifact(Agent2ToObs, "OutputPortName2");
131    //For easy reading, we named the input port as the port we want to observed
132    obsAgent.addInputCouplingArtifact(Agent1Port1ToObs, "OutputPortName1");
133    obsAgent.addInputCouplingArtifact(Agent1Port2ToObs, "OutputPortName3");
134    obsAgent.addInputCouplingArtifact(Agent2ToObs, "OutputPortName2");
135    obsAgent.addInputCouplingArtifact(Agent3ToObsReceiver, "OutputPortName4");
136    obsAgent.addInputCouplingArtifact(Agent4Port1ToObsReceiver, "OutputPortName5");
137    obsAgent.addInputCouplingArtifact(Agent4Port2ToObsReceiver, "OutputPortName6");
138
139    /*if the same kind of observer is created in Model2
140     * you need to create the sender
141     */
142    DDSEventCouplingArtifactSender Agent1Port1ToObs2Sender = new DDSEventCouplingArtifactSender("1Port1ToObs2");
143    DDSEventCouplingArtifactSender Agent1Port2ToObs2Sender = new DDSEventCouplingArtifactSender("1Port2ToObs2");
144    DDSEventCouplingArtifactSender Agent2ToObs2Sender = new DDSEventCouplingArtifactSender("2ToObs2");
145
146    agent1.addOutputCouplingArtifact(Agent1Port1ToObs2Sender, "OutputPortName1");
147    agent1.addOutputCouplingArtifact(Agent1Port2ToObs2Sender, "OutputPortName3");
148    agent2.addOutputCouplingArtifact(Agent2ToObs2Sender, "OutputPortName2");
149
150    /*
151     *Visualization in real time (can slow down the simulation a bit)
152     *the name of ports is the one assigned as observer's input port
153     *Comment the observing you don't need
154     *all real time will be display on the same windows if only one was created
155     */
156    //Temporal graph (if ports observed are Double)
157    ObsModelArtifact.addObserveringArtifact (new LiveTXGraphic (
158        "Graph name", "Y axis name", Renderer.Line, //or Renderere.Dot or Renderer.Step
159        new String [] {"Names for display purpose, one name per port"}, 
160        new String [] {"Names of ports you want to display"}));
161    //XY graphics (if the port observed is a Tuple2 of Double)
162    ObsModelArtifact.addObserveringArtifact (new LiveXYGraphic(
163        "Graph name", "X axis name", "Y axis name", Renderer.Line, //or Renderere.Dot or Renderer.Step
164        "Name for display purpose", "Name of port observed"));
165    //Bar chart (if the port observed is a SimulVector of Double)
166    ObsModelArtifact.addObserveringArtifact (new LiveBarGraphic(
167        "Graph name", "X axis name", "Y axis name",
168        new String [] {"Names for display purpose, one name vector's component"}, 
169        "Name of port observed"));
170    //Pie chart (if the port observed is a SimulVector of Double)
171    ObsModelArtifact.addObserveringArtifact (new LivePieGraphic(
172        "Graph name",
173        new String [] {"Names for display purpose, one name vector's component"}, 
174        "Name of port observed"));
175    //Factual representation (if the port observed is a Double)
176    ObsModelArtifact.addObserveringArtifact (new LiveEventGraphic(
177        "Graph name", "X axis name", "Y axis name",
178        "Name for display purpose", "Name of port observed"));
179    //3D graphic (if the port observed is a Tuple3 of Double)
180    ObsModelArtifact.addObserveringArtifact (new Live3DGraphic(
181        "Graph name", "X axis name", "Y axis name", "Z axis name",
182        "Name of port observed"));
183
184    /*
185     *Visualization in post-mortem
186     *same comment as for real time
187     */
188    //Temporal graph (if ports observed are Double)
189    ObsModelArtifact.addObserveringArtifact (new PostMortemTXGraphic (
190        "Graph name", "Y axis name", Renderer.Line, //or Renderere.Dot or Renderer.Step
191        new String [] {"Names for display purpose, one name per port"}, 
192        new String [] {"Names of ports you want to display"}));
193    //XY graphics (if the port observed is a Tuple2 of Double)
194    ObsModelArtifact.addObserveringArtifact (new PostMortemXYGraphic(
195        "Graph name", "X axis name", "Y axis name", Renderer.Line, //or Renderere.Dot or Renderer.Step
196        "Name for display purpose", "Name of port observed"));
197    //Bar chart (if the port observed is a SimulVector of Double)
198    ObsModelArtifact.addObserveringArtifact (new PostMortemBarGraphic(
199        "Graph name", "X axis name", "Y axis name",

```

```

201     new String [] {"Names for display purpose, one name vector's component"},  

202     "Name of port observed");  

203     //Pie chart (if the port observed is a SimulVector of Double)  

204     ObsModelArtifact.addObservingArtifact (new PostMortemPieGraphic(  

205         "Graph name",  

206         new String [] {"Names for display purpose, one name vector's component"},  

207         "Name of port observed");  

208     //Factual representation (if the port observed is a Double)  

209     ObsModelArtifact.addObservingArtifact (new PostMortemEventGraphic(  

210         "Graph name", "X axis name", "Y axis name",  

211         "Name for display purpose", "Name of port observed"));  

212     //3D graphic (if the port observed is a Tuple3 of Double)  

213     ObsModelArtifact.addObservingArtifact (new PostMortem3DGraphic(  

214         "Graph name", "X axis name", "Y axis name", "Z axis name",  

215         "Name of port observed"));  

216  

217     /*  

218      *Logging  

219      *the name of ports is the one assigned as observer's input port  

220      *the name of files need the extension (.csv or else)  

221      */  

222     String path="path to folder/";  

223     //One file per output port (Work well only with Tuple1)  

224     ObsModelArtifact.addObservingArtifact (new LoggingArtifact (  

225         new String [] {path+"name of file1",path+"name of file 2" /**one per port**/},  

226         new String [] {"Names of ports you want to display"},  

227         "%time;%value \n");//column for time, one for value and ";" as column separator  

228     //One file per output manual fixied for other type(use one method for each file)  

229     ObsModelArtifact.addObservingArtifact (new LoggingArtifact (  

230         path+"name of file1", new String [] {"name of port logged in this file"}, "%time;%valuenn"));  

231     ObsModelArtifact.addObservingArtifact (new LoggingArtifact (  

232         path+"name of file2", new String [] {"name of port logged in this file"}, "%time;%valuenn"));  

233     //One file for all output ports, line structure  

234     ObsModelArtifact.addObservingArtifact (new LoggingArtifact (  

235         path+"name of file",  

236         new String [] {"Names of ports you want to display"}, "%time;%value \n"));  

237  

238     /*  

239      * Post Treatment  

240      */  

241     //Invoke Script R  

242     ObsModelArtifact.addObservingArtifact (new LogToRProject(  

243         path+"name of file to log",new String [] {"Names of ports you want to study"}));  

244     //Data comparator  

245     ObsModelArtifact.addObservingArtifact (new DataComparator(  

246         new String [] {path+"name of file to use as reference 1" /**one file per port**/},  

247         new String [] {"Names of ports you want to study"}));  

248  

249     /*****  

250     **** MODELS INITIALIZATION ****  

251     ****  

252  

253     // Start the simulation software associated to model1  

254     // This is not systematically necessary, depending on the simulation software used  

255     agent1.startModelSoftware();  

256     agent2.startModelSoftware();  

257     obsAgent.startModelSoftware();  

258  

259     // Initialize Model1 parameters  

260     // e.g. time discretization or constants  

261     // This is not systematically necessary, depending on the model  

262     String [] args_agent1 = { "0.001" /*;and other arguments*/ };  

263     String [] args_agent2 = { "0.001" /*;and other arguments*/ };  

264     agent1.setModelParameters(args_agent1);  

265     agent2.setModelParameters(args_agent2);  

266  

267     /*****  

268     **** CO-SIMULATION INIT & STARTING ****  

269     ****  

270  

271     try {  

272         // Co-initialization with first exchanges  

273         // This is necessary only when the model initial states are co-dependant  

274         agent1.coInitialize();  

275         agent2.coInitialize();  

276  

277         // Start the co-simulation  

278         agent1.start();  

279         agent2.start();  

280         obsAgent.start();  

281  

282         // This should never happen  

283     } catch (CausalityException e) {  

284         e.printStackTrace();  

285     }  

286 }
}

```

3.1.2 Java Example Template: run configuration (decentralized - Launcher2)

```

2 import mecsyco.communication.dds.coupling.DDSEventCouplingArtifactReceiver;  

3 import mecsyco.communication.dds.coupling.DDSEventCouplingArtifactSender;  

4 import mecsyco.core.agent.EventMAgent;  

5 import mecsyco.core.agent.ObservingMAgent;  

6 import mecsyco.core.coupling.CentralizedEventCouplingArtifact;  

7 import mecsyco.core.exception.CausalityException;  

8 import mecsyco.core.type.SimulData;  

9 import mecsyco.observing.base.comparator.DataComparator;  

10 import mecsyco.observing.base.logging.LoggingArtifact;  

11 import mecsyco.observing.jfreechart.bar.LiveBarGraphic;  

12 import mecsyco.observing.jfreechart.event.LiveEventGraphic;  

13 import mecsyco.observing.jfreechart.event.PostMortemEventGraphic;  

14 import mecsyco.observing.jfreechart.pie.LivePieGraphic;  

15 import mecsyco.observing.jfreechart.pie.PostMortemPieGraphic;  

16 import mecsyco.observing.jfreechart.xy.LiveXYGraphic;  

17 import mecsyco.observing.jfreechart.xy.PostMortemXYGraphic;  

18 import mecsyco.observing.jfreechart.xy.PostMortemTXGraphic;

```

```

20 import mecsyco.observing.jfreechart.xy.PostMortemXYGraphic;
21 import mecsyco.observing.jfreechart.xy.Renderer;
22 import mecsyco.observing.jzy3d.graphic.Live3DGraphic;
23 import mecsyco.observing.jzy3d.graphic.PostMortem3DGraphic;
24 import mecsyco.observing.swing.dispatcher.SwingDispatcherArtifact;
25 import mecsyco.observing.swing.r.LogToRProject;
26
27 public class Launcher2 {
28     //Simulation length:
29     public final static double maxSimulationTime = 10;
30
31     public static void main(String args[]) {
32
33         /*****
34         **** AGENTS & MODEL ARTIFACTS ****
35         *****/
36
37         // First agent from first model (Model1)
38         //Agent
39         EventMAgent agent3 = new EventMAgent("nameAgent3",maxSimulationTime);
40         EventMAgent agent4 = new EventMAgent("nameAgent4",maxSimulationTime);
41
42         //Then Model Artifact
43         AModelArtifact Agent3Artifact = new AmodelArtifact( /*parameters*/ );
44         AModelArtifact Agent4Artifact = new AmodelArtifact( /*parameters*/ );
45
46         //Associate agent and artifact
47         agent3.setModelArtifact(Agent3Artifact);
48         agent4.setModelArtifact(Agent4Artifact);
49
50
51         /*****
52         **** COUPLING ARTEFACTS ****
53         *****/
54
55         //Inside communication
56         CentralizedEventCouplingArtifact Agent3ToAgent4 = new CentralizedEventCouplingArtifact();
57         CentralizedEventCouplingArtifact Agent4ToAgent3 = new CentralizedEventCouplingArtifact();
58
59         //Outside communication
60         //Arrows that go to Model1
61         DDSEventCouplingArtifactSender Agent4ToAgent2sender = new DDSEventCouplingArtifactSender("4To2");
62         //Arrows that come from Model1
63         DDSEventCouplingArtifactReceiver Agent1ToAgent3receiver = new DDSEventCouplingArtifactReceiver("1To3", SimulData.class);
64         // "1To3" and "4To2" are DDS topics, it needs to be the same in Model1
65         // "SimulData.class" corresponds to the type of the expected value, from the remote model
66
67         //Connection
68         //Inside:
69         agent3.addOutputCouplingArtifact(Agent3ToAgent4, "OutputPortName4");
70         agent4.addOutputCouplingArtifact(Agent4ToAgent3, "OutputPortName5");
71
72         agent4.addInputCouplingArtifact(Agent3ToAgent4, "InputPortName4");
73         agent3.addInputCouplingArtifact(Agent4ToAgent3, "InputPortName5");
74
75         //Outside
76         // Agent4 will send data from OutputPortName6 via the topic "4To2" (output events)
77         agent4.addOutputCouplingArtifact(Agent4ToAgent2sender, "OutputPortName6");
78         // Agent3 will receive Data in InputPortName6 with the value received via the topic "1To3" (input events)
79         agent3.addInputCouplingArtifact(Agent1ToAgent3receiver, "InputPortName6");
80
81         /*****
82         **** Operations ****
83         *** Check User Guide: Create your own operations ***
84         *****/
85
86
87         /*In the case of internal operation, it does not change
88         * We will then see in the cast of the external link
89         */
90
91         //In this launcher, we can only apply operation on the link from Agent1 to Agent3
92         //event operation
93         DataOperationTemplate DataOpe= new DataOperationTemplate();
94         Agent1ToAgent3receiver.addEventOperation(DataOpe);
95         //time operation
96         TimeOperationTemplate TimeOpe= new TimeOperationTemplate();
97         Agent1ToAgent3receiver.addTimeOperation(TimeOpe);
98
99
100        /*****
101        ****LOGGING VISUALIZATION OR POST TREATMENT ****
102        **** Check User Guide: MECSYCO-visu ****
103        *** Check User Guide section Simulation data ***
104        *****/
105
106        /* Set the agent name for logging if you didn't named it at the creation
107        *(otherwise, an unique default number is attributed)
108        */
109        agent3.setAgentName("Agent13");
110        agent4.setAgentName("Agent14");
111
112        /*Create observing Agent and the dispatcher
113        * Create both for each different display windows you want
114        */
115        ObservingMAgent obsAgent2 = new ObservingMAgent ("ObserverName2", maxSimulationTime);
116        SwingDispatcherArtifact ObsModelArtifact2 = new SwingDispatcherArtifact ();
117        obsAgent2.setDispatcherArtifact(ObsModelArtifact2);
118
119        /* Coupling Artifact and connection
120        * Same rules apply here, if the Port to observe is from Model1, use DDS
121        */
122        CentralizedEventCouplingArtifact Agent3ToObs = new CentralizedEventCouplingArtifact();
123        CentralizedEventCouplingArtifact Agent4Port1ToObs = new CentralizedEventCouplingArtifact();
124        CentralizedEventCouplingArtifact Agent4Port2ToObs = new CentralizedEventCouplingArtifact();
125        DDSEventCouplingArtifactReceiver Agent2ToObs2Receiver = new DDSEventCouplingArtifactReceiver("2ToObs2", SimulData.class);
126        DDSEventCouplingArtifactReceiver Agent1Port1ToObs2Receiver = new DDSEventCouplingArtifactReceiver("1Port1ToObs2", SimulData.class);
127        DDSEventCouplingArtifactReceiver Agent1Port2ToObs2Receiver = new DDSEventCouplingArtifactReceiver("1Port2ToObs2", SimulData.class);
128
129        agent3.addOutputCouplingArtifact(Agent3ToObs, "OutputPortName4");
130        agent4.addOutputCouplingArtifact(Agent4Port1ToObs, "OutputPortName5");

```

```

132     agent4.addOutputCouplingArtifact(Agent4Port2ToObs, "OutputPortName6");
133     //For easy reading, we named the input port as the port we want to observed
134     obsAgent2.addInputCouplingArtifact(Agent3ToObs, "OutputPortName4");
135     obsAgent2.addInputCouplingArtifact(Agent4Port1ToObs, "OutputPortName5");
136     obsAgent2.addInputCouplingArtifact(Agent4Port2ToObs, "OutputPortName6");
137     obsAgent2.addInputCouplingArtifact(Agent1Port1ToObs2Receiver, "OutputPortName1");
138     obsAgent2.addInputCouplingArtifact(Agent1Port2ToObs2Receiver, "OutputPortName3");
139     obsAgent2.addInputCouplingArtifact(Agent2ToObs2Receiver, "OutputPortName2");

140     /*if the same kind of observer is created in Model1
141      * you need to create the sender
142      */
143     DDSEventCouplingArtifactSender Agent3ToObsSender = new DDSEventCouplingArtifactSender("3ToObs");
144     DDSEventCouplingArtifactSender Agent4Port1ToObsSender = new DDSEventCouplingArtifactSender("4Port1ToObs");
145     DDSEventCouplingArtifactSender Agent4Port2ToObsSender = new DDSEventCouplingArtifactSender("4Port2ToObs");
146

147     agent3.addOutputCouplingArtifact(Agent3ToObsSender, "OutputPortName4");
148     agent4.addOutputCouplingArtifact(Agent4Port1ToObsSender, "OutputPortName5");
149     agent4.addOutputCouplingArtifact(Agent4Port2ToObsSender, "OutputPortName6");
150

151     /*
152      *Visualization in real time (can slow down the simulation a bit)
153      *the name of ports is the one assigned as observer's input port
154      *Comment the observing you don't need
155      *all real time will be display on the same windows if only one was created
156      */
157     //Temporal graph (if ports observed are Double)
158     ObsModelArtifact2.addObservingArtifact (new LiveTXGraphic (
159         "Graph name", "Y axis name", Renderer.Line, //or Renderere.Dot or Renderer.Step
160         new String [] {"Names for display purpose, one name per port"} ,
161         new String [] {"Names of ports you want to display"});
162     //XY graphics (if the port observed is a Tuple2 of Double)
163     ObsModelArtifact2.addObservingArtifact (new LiveXYGraphic(
164         "Graph name", "X axis name", "Y axis name", Renderer.Line, //or Renderere.Dot or Renderer.Step
165         "Name for display purpose", "Name of port observed");
166     //Bar chart (if the port observed is a SimulVector of Double)
167     ObsModelArtifact2.addObservingArtifact (new LiveBarGraphic(
168         "Graph name", "X axis name", "Y axis name",
169         new String [] {"Names for display purpose, one name vector's component"},
170         "Name of port observed");
171     //Pie chart (if the port observed is a SimulVector of Double)
172     ObsModelArtifact2.addObservingArtifact (new LivePieGraphic(
173         "Graph name",
174         new String [] {"Names for display purpose, one name vector's component"},
175         "Name of port observed");
176     //Factual representation (if the port observed is a Double)
177     ObsModelArtifact2.addObservingArtifact (new LiveEventGraphic(
178         "Graph name", "X axis name", "Y axis name",
179         "Name for display purpose", "Name of port observed");
180     //3D graphic (if the port observed is a Tuple3 of Double)
181     ObsModelArtifact2.addObservingArtifact (new Live3DGraphic(
182         "Graph name", "X axis name", "Y axis name", "Z axis name",
183         "Name of port observed"));

184     /*
185      *Visualization in post-mortem
186      *same comment as for real time
187      */
188     //Temporal graph (if ports observed are Double)
189     ObsModelArtifact2.addObservingArtifact (new PostMortemTXGraphic (
190         "Graph name", "Y axis name", Renderer.Line, //or Renderere.Dot or Renderer.Step
191         new String [] {"Names for display purpose, one name per port"} ,
192         new String [] {"Names of ports you want to display"});
193     //XY graphics (if the port observed is a Tuple2 of Double)
194     ObsModelArtifact2.addObservingArtifact (new PostMortemXYGraphic(
195         "Graph name", "X axis name", "Y axis name", Renderer.Line, //or Renderere.Dot or Renderer.Step
196         "Name for display purpose", "Name of port observed");
197     //Bar chart (if the port observed is a SimulVector of Double)
198     ObsModelArtifact2.addObservingArtifact (new PostMortemBarGraphic(
199         "Graph name", "X axis name", "Y axis name",
200         new String [] {"Names for display purpose, one name vector's component"},
201         "Name of port observed");
202     //Pie chart (if the port observed is a SimulVector of Double)
203     ObsModelArtifact2.addObservingArtifact (new PostMortemPieGraphic(
204         "Graph name",
205         new String [] {"Names for display purpose, one name vector's component"},
206         "Name of port observed");
207     //Factual representation (if the port observed is a Double)
208     ObsModelArtifact2.addObservingArtifact (new PostMortemEventGraphic(
209         "Graph name", "X axis name", "Y axis name",
210         "Name for display purpose", "Name of port observed");
211     //3D graphic (if the port observed is a Tuple3 of Double)
212     ObsModelArtifact2.addObservingArtifact (new PostMortem3DGraphic(
213         "Graph name", "X axis name", "Y axis name", "Z axis name",
214         "Name of port observed"));

215     /*
216      *Logging
217      *the name of ports is the one assigned as observer's input port
218      *the name of files need the extension (.csv or else)
219      */
220     String path="path to folder";
221     //One file per output port (Work well only with Tuple1)
222     ObsModelArtifact2.addObservingArtifact (new LoggingArtifact (
223         new String [] {path+"name of file1",path+"name of file 2" /**one per port**/},
224         new String [] {"Names of ports you want to display"},
225         "%time;%value \n"); //column for time, one for value and ";" as column separator
226     //One file per output manual fixed for other type(use one method for each file)
227     ObsModelArtifact2.addObservingArtifact (new LoggingArtifact (
228         path+"name of file1", new String [] {"name of port logged in this file"}, "%time;%valuenn");
229     ObsModelArtifact2.addObservingArtifact (new LoggingArtifact (
230         path+"name of file2", new String [] {"name of port logged in this file"}, "%time;%valuenn"));
231     //One file for all output ports, line structure
232     ObsModelArtifact2.addObservingArtifact (new LoggingArtifact (
233         path+"name of file",
234         new String [] {"Names of ports you want to display"}, "%time;%value \n"));

235     /*
236      * Post Treatment
237      */
238     //Invoke Script R

```

```

242     ObsModelArtifact2.addObservingArtifact (new LogToRProject(
243         path+"name of file to log",new String [] {"Names of ports you want to study"}));
244     //Data comparator
245     ObsModelArtifact2.addObservingArtifact (new DataComparator(
246         new String [] {path+"name of file to use as reference 1" /**one file per port**/},
247         new String [] {"Names of ports you want to study"}));
248
249     /***** MODELS INITIALIZATION ****/
250     /*****
251
252     // Start the simulation software associated to modell
253     // This is not systematically necessary, depending on the simulation software used
254     agent3.startModelSoftware();
255     agent4.startModelSoftware();
256     obsAgent2.startModelSoftware();
257
258     // Initialize Model1 parameters
259     // e.g. time discretization or constants
260     // This is not systematically necessary, depending on the model
261     String [] args_agent3 = { "0.001" /*; and other arguments*/ };
262     String [] args_agent4 = { "0.001" /*; and other arguments*/ };
263     agent3.setModelParameters(args_agent3);
264     agent4.setModelParameters(args_agent4);
265
266     /***** CO-SIMULATION INIT & STARTING ****/
267     /*****
268
269     try {
270         // Co-initialization with first exchanges
271         // This is necessary only when the model initial states are co-dependant
272         agent3.coInitialize();
273         agent4.coInitialize();
274
275         // Start the co-simulation
276         agent3.start();
277         agent4.start();
278         obsAgent2.start();
279
280         // This should never happen
281         } catch (CausalityException e) {
282             e.printStackTrace();
283         }
284     }
285 }
```

3.2 Remarks:

You can notice that DDS is compatible with all other functions of MECSYCO (observing, operation). Be careful, the use of operations are not done randomly. Operations are applied in the receiver side, that is why when using DDS, they could be add only on *DDSEventCouplingArtifactReceiver*.

In order to be transmitted, the expected type of data should be a *Jackson* based one. It is then easier to use *SimulData* or to create one by yourself (see "*User Guide: SimulData manipulation*").

Chapter 4

Example

Figure 4.1 present the case of Lorenz as a multi-model.

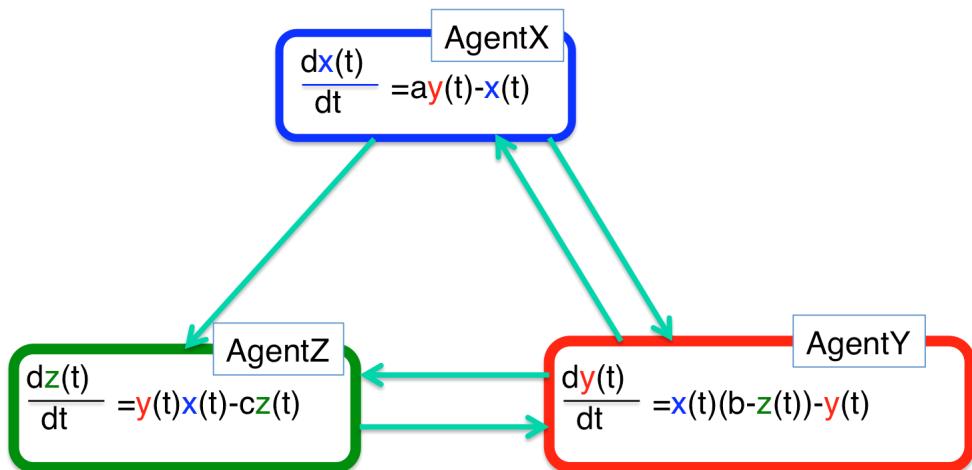


Figure 4.1: Lorenz system as a multi-model.

When using DDS, each agent will have its proper launcher and all links will then be with DDS only. Here is how we did it for each launcher:

- **AgentXLauncher:**

- First, create the coupling artifact:


```
DDSEventCouplingArtifactSender XOutputToZSender=new DDSEventCouplingArtifactSender("XOutputToZ");
DDSEventCouplingArtifactSender XOutputToYSender=new DDSEventCouplingArtifactSender("XOutputToY");
DDSEventCouplingArtifactReceiver YOutputToXReceiver=new DDSEventCouplingArtifactReceiver("YOutputToX", Tuple1.of(Number.class));
```
- if it is a "sender" then it is an output:


```
XAgent.addOutputCouplingArtifact(XOutputToZSender,"X");
XAgent.addOutputCouplingArtifact(XOutputToYSender,"X");
```
- else, it is an input:


```
XAgent.addInputCouplingArtifact(YOutputToXReceiver,"Y");
```

- **AgentYLauncher:**

- First, create the coupling artifact:

```
DDSEventCouplingArtifactReceiver XOutputToYReceiver=new DDSEventCouplingArtifactReceiver("XOutputToY", Tuple1.of(Number.class));
DDSEventCouplingArtifactSender YOutputToXSender=new DDSEventCouplingArtifactSender("YOutputToX");
DDSEventCouplingArtifactSender YOutputToZSender=new DDSEventCouplingArtifactSender("YOutputToZ");
DDSEventCouplingArtifactReceiver ZOutputToYReceiver=new DDSEventCouplingArtifactReceiver("ZOutputToY", Tuple1.of(Number.class));
```
- if it is a "sender" then it is an output:

```
YAgent.addOutputCouplingArtifact(YOutputToXSender,"Y");
YAgent.addOutputCouplingArtifact(YOutputToZSender,"Y");
```
- else, it is an input:

```
YAgent.addInputCouplingArtifact(XOutputToYReceiver,"X");
YAgent.addInputCouplingArtifact(ZOutputToYReceiver,"Z");
```

- **AgentZLauncher:**

- First, create the coupling artifact:

```
DDSEventCouplingArtifactReceiver XOutputToZReceiver=new DDSEventCouplingArtifactReceiver("XOutputToZ", Tuple1.of(Number.class));
DDSEventCouplingArtifactReceiver YOutputToZReceiver=new DDSEventCouplingArtifactReceiver("YOutputToZ", Tuple1.of(Number.class));
DDSEventCouplingArtifactSender ZOutputToYSender=new DDSEventCouplingArtifactSender("ZOutputToY");

- if it is a "sender" then it is an output:
ZAgent.addOutputCouplingArtifact(ZOutputToYSender,"Z");
- else, it is an input:
ZAgent.addInputCouplingArtifact(XOutputToZReceiver,"X");
ZAgent.addInputCouplingArtifact(YOutputToZReceiver,"Y");
```

Do not forgot that one sender implies a receiver with the exact same topic! For the whole construction, check the *Getting Started*.

