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

Benjamin Camus[1,2], Julien Vaubourg[2], Yannick Presse[2],
Victorien Elvinger[2], Thomas Paris[1,2], Alexandre Tan[2]
Vincent Chevrier[1,2], Laurent Ciarletta[1,2], Christine Bourjot[1,2]
[1]Universite de Lorraine, CNRS, LORIA UMR 7503,
Vandoeuvre-les-Nancy, F-54506, France.
[2]INRIA, Villers-les-Nancy, F-54600, France.
`mecsyco@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 DDS[1]).

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

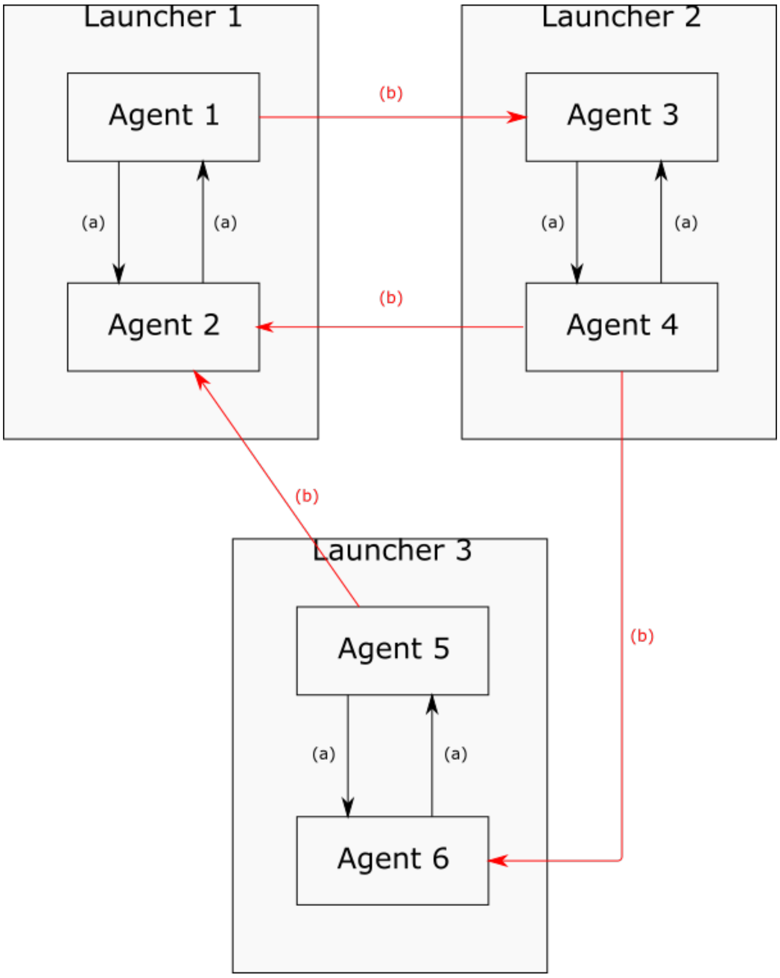

Figure 3.1: Decentralized and distributed multi-model. (a) Link done with usual coupling artifact. (b) Link done with DDS coupling artifact.

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

3.1 Templates

The templates can be use 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)

```java

import mecsyco.communication.dds.coupling.DDSEventCouplingArtifactReceiver;
import mecsyco.communication.dds.coupling.DDSEventCouplingArtifactSender;
import mecsyco.core.agent.EventMAgent;
import mecsyco.core.agent.ObservingMAgent;
import mecsyco.core.coupling.CentralizedEventCouplingArtifact;
import mecsyco.core.exception.CausalityException;
import mecsyco.core.type.SimulData;
import mecsyco.observing.base.comparator.DataComparator;
import mecsyco.observing.base.logging.LoggingArtifact;
import mecsyco.observing.jfreechart.bar.LiveBarGraphic;
import mecsyco.observing.jfreechart.bar.PostMortemBarGraphic;
import mecsyco.observing.jfreechart.event.LiveEventGraphic;
import mecsyco.observing.jfreechart.event.PostMortemEventGraphic;
import mecsyco.observing.jfreechart.pie.LivePieGraphic;
import mecsyco.observing.jfreechart.pie.PostMortemPieGraphic;
import mecsyco.observing.jfreechart.xy.LiveTXGraphic;
import mecsyco.observing.jfreechart.xy.LiveXYGraphic;
import mecsyco.observing.jfreechart.xy.PostMortemTXGraphic;
import mecsyco.observing.jfreechart.xy.PostMortemXYGraphic;
import mecsyco.observing.jfreechart.xy.Renderer;
import mecsyco.observing.jzy3d.graphic.Live3DGraphic;
import mecsyco.observing.jzy3d.graphic.PostMortem3DGraphic;
import mecsyco.observing.swing.dispatcher.SwingDispatcherArtifact;
import mecsyco.observing.swing.r.LogToRProject;

public class Launcher1 {
    //Simulation length:
    public final static double maxSimulationTime = 10;

    public static void main(String args[]) {

        /**********************************/
        /**** AGENTS & MODEL ARTIFACTS ****/
        /**********************************/

        // First agent from first model (Model1)
        //Agent
        EventMAgent agent1 = new EventMAgent("nameAgent1",maxSimulationTime);
        EventMAgent agent2 = new EventMAgent("nameAgent1",maxSimulationTime);

        //Then Model Artifacts
        AModelArtifact Agent1Artifact = new AmodelArtifact( /**parameters**/);
        AModelArtifact Agent2Artifact = new AmodelArtifact(/**parameters**/);

        //Associate agent and artifact
        agent1.setModelArtefact(Agent1Artifact);
        agent2.setModelArtefact(Agent2Artifact);


        /****************************/
        /**** COUPLING ARTEFACTS ****/
        /****************************/

        //Inside communication
        CentralizedEventCouplingArtifact Agent1ToAgent2 = new CentralizedEventCouplingArtifact();
        CentralizedEventCouplingArtifact Agent2ToAgent1 = new CentralizedEventCouplingArtifact();

        //Outside communication
        //Arrows that go to Launcher2
        DDSEventCouplingArtifactSender Agent1ToAgent3sender = new DDSEventCouplingArtifactSender("1To3");
        //Arrows that come from Launcher2
        DDSEventCouplingArtifactReceiver Agent4ToAgent2receiver = new DDSEventCouplingArtifactReceiver("4To2", SimulData.class);
        // "1To3" and "4To2" are DDS topics, it needs to be the same in Model2
        // "SimulData.class" corresponds to the type of the expected value, from the remote model

        //Connection
        //Inside:
        agent1.addOutputCouplingArtifact(Agent1ToAgent2, "OutputPortName1");
        agent2.addOutputCouplingArtifact(Agent2ToAgent1, "OutputPortName2");

        agent2.addInputCouplingArtifact(Agent1ToAgent2, "InputPortName1");
        agent1.addInputCouplingArtifact(Agent2ToAgent1, "InputPortName2");

        //Outside
        // Agent1 will send data from OutputPortName3 via the topic "1To3" (output events)
        agent1.addOutputCouplingArtifact(Agent1ToAgent3sender, "OutputPortName3");
        // Agent2 will receive Data in InputPortName3 with the value received via the topic "4To2" (input events)
        agent2.addInputCouplingArtifact(Agent4ToAgent2receiver, "InputPortName3");


        /***************************************************/
        /******************** Operations *******************/
        /*** Check User Guide: Create your own operations ***/
        /***************************************************/

        /*In the case of internal operation, it does not change
         * We will then see in the cast of the external link
         */

        //In this launcher, we can only apply operation on the link from Agent4 to Agent2
        //event operation
        DataOperationTemplate DataOpe= new DataOperationTemplate();
        Agent4ToAgent2receiver.addEventOperation(DataOpe);
        //time operation
        TimeOperationTemplate TimeOpe= new TimeOperationTemplate();
        Agent4ToAgent2receiver.addTimeOperation(TimeOpe);

        /************************************************/
        /****LOGGING VISUALIZATION OR POST TREATMENT ****/
        /******** Check User Guide: MECSYCO-visu ********/
        /*** Check User Guide section Simulation data ***/
        /************************************************/

        /* Set the agent name for logging if you didn't named it at the creation
        *(otherwise, an unique default number is attributed)
        */
        agent1.setAgentName("Agentl1");
        agent2.setAgentName("Agentl2");

        /*Create observing Agent and the dispatcher
         * Create both for each different display windows you want
         */
        ObservingMAgent obsAgent = new ObservingMAgent ("ObserverName", maxSimulationTime);
        SwingDispatcherArtifact ObsModelArtifact = new SwingDispatcherArtifact ();
        obsAgent.setDispatcherArtifact(ObsModelArtifact);

        /* Coupling Artifact and connection
         * Same rules apply here, if the Port to observe is from Model2, use DDS
         */
        CentralizedEventCouplingArtifact Agent1Port1ToObs = new CentralizedEventCouplingArtifact();
        CentralizedEventCouplingArtifact Agent1Port2ToObs = new CentralizedEventCouplingArtifact();
        CentralizedEventCouplingArtifact Agent2ToObs = new CentralizedEventCouplingArtifact();
        DDSEventCouplingArtifactReceiver Agent3ToObsReceiver = new DDSEventCouplingArtifactReceiver("3ToObs", SimulData.class);
        DDSEventCouplingArtifactReceiver Agent4Port1ToObsReceiver = new DDSEventCouplingArtifactReceiver("4Port1ToObs", SimulData.class);
        DDSEventCouplingArtifactReceiver Agent4Port2ToObsReceiver = new DDSEventCouplingArtifactReceiver("4Port2ToObs", SimulData.class);

        agent1.addOutputCouplingArtifact(Agent1Port1ToObs, "OutputPortName1");
        agent1.addOutputCouplingArtifact(Agent1Port2ToObs, "OutputPortName3");
        agent2.addOutputCouplingArtifact(Agent2ToObs, "OutputPortName2");
        //For easy reading, we named the input port as the port we want to observed
        obsAgent.addInputCouplingArtifact(Agent1Port1ToObs, "OutputPortName1");
        obsAgent.addInputCouplingArtifact(Agent1Port2ToObs, "OutputPortName3");
        obsAgent.addInputCouplingArtifact(Agent2ToObs, "OutputPortName2");
        obsAgent.addInputCouplingArtifact(Agent3ToObsReceiver, "OutputPortName4");
        obsAgent.addInputCouplingArtifact(Agent4Port1ToObsReceiver, "OutputPortName5");
        obsAgent.addInputCouplingArtifact(Agent4Port2ToObsReceiver, "OutputPortName6");

        /*if the same kind of observer is created in Model2
         * you need to create the sender
         */
        DDSEventCouplingArtifactSender Agent1Port1ToObs2Sender = new DDSEventCouplingArtifactSender("1Port1ToObs2");
        DDSEventCouplingArtifactSender Agent1Port2ToObs2Sender = new DDSEventCouplingArtifactSender("1Port2ToObs2");
        DDSEventCouplingArtifactSender Agent2ToObs2Sender = new DDSEventCouplingArtifactSender("2ToObs2");

        agent1.addOutputCouplingArtifact(Agent1Port1ToObs2Sender, "OutputPortName1");
        agent1.addOutputCouplingArtifact(Agent1Port2ToObs2Sender, "OutputPortName3");
        agent2.addOutputCouplingArtifact(Agent2ToObs2Sender, "OutputPortName2");

        /*
         *Visualization in real time  (can slow down the simulation a bit)
         *the name of ports is the one assigned as observer's input port
         *Comment the observing you don't need
         *all real time will be display on the same windows if only one was created
         */
        //Temporal graph (if ports observed are Double)
        ObsModelArtifact.addObservingArtifact (new LiveTXGraphic (
                "Graph name", "Y axis name", Renderer.Line,//or Rendere.Dot or Renderer.Step
                new String [] {"Names for display purpose, one name per port"} ,
                new String [] {"Names of ports you want to display"}));
        //XY graphics (if the port observed is a Tuple2 of Double)
        ObsModelArtifact.addObservingArtifact (new LiveXYGraphic(
                "Graph name", "X axis name", "Y axis name", Renderer.Line, //or Rendere.Dot or Renderer.Step
                "Name for display purpose", "Name of port observed"));
        //Bar chart (if the port observed is a SimulVector of Double)
        ObsModelArtifact.addObservingArtifact (new LiveBarGraphic(
                "Graph name", "X axis name", "Y axis name",
                new String [] {"Names for display purpose, one name vector's component"},
                "Name of port observed"));
        //Pie chart (if the port observed is a SimulVector of Double)
        ObsModelArtifact.addObservingArtifact (new LivePieGraphic(
                "Graph name",
                new String [] {"Names for display purpose, one name vector's component"},
                "Name of port observed"));
        //Factual representation (if the port observed is a Double)
        ObsModelArtifact.addObservingArtifact (new LiveEventGraphic(
                "Graph name", "X axis name", "Y axis name",
                "Name for display purpose", "Name of port observed"));
        //3D graphic (if the port observed is a Tuple3 of Double)
        ObsModelArtifact.addObservingArtifact (new Live3DGraphic(
                "Graph name", "X axis name", "Y axis name", "Z axis name",
                "Name of port observed"));

        /*
         *Visualization in post-mortem
         *same comment as for real time
         */
        //Temporal graph (if ports observed are Double)
        ObsModelArtifact.addObservingArtifact (new PostMortemTXGraphic (
                "Graph name", "Y axis name", Renderer.Line,//or Rendere.Dot or Renderer.Step
                new String [] {"Names for display purpose, one name per port"} ,
                new String [] {"Names of ports you want to display"}));
        //XY graphics (if the port observed is a Tuple2 of Double)
        ObsModelArtifact.addObservingArtifact (new PostMortemXYGraphic(
                "Graph name", "X axis name", "Y axis name", Renderer.Line, //or Rendere.Dot or Renderer.Step
                "Name for display purpose", "Name of port observed"));
        //Bar chart (if the port observed is a SimulVector of Double)
        ObsModelArtifact.addObservingArtifact (new PostMortemBarGraphic(
                "Graph name", "X axis name", "Y axis name",
```

```java
                new String [] {"Names for display purpose, one name vector's component"},
                "Name of port observed"));
        //Pie chart (if the port observed is a SimulVector of Double)
        ObsModelArtifact.addObservingArtifact (new PostMortemPieGraphic(
                "Graph name",
                new String [] {"Names for display purpose, one name vector's component"},
                "Name of port observed"));
        //Factual representation (if the port observed is a Double)
        ObsModelArtifact.addObservingArtifact (new PostMortemEventGraphic(
                "Graph name", "X axis name", "Y axis name",
                "Name for display purpose", "Name of port observed"));
        //3D graphic (if the port observed is a Tuple3 of Double)
        ObsModelArtifact.addObservingArtifact (new PostMortem3DGraphic(
                "Graph name", "X axis name", "Y axis name", "Z axis name",
                "Name of port observed"));

        /*
         *Logging
         *the name of ports is the one assigned as observer's input port
         *the name of files need the extension (.csv or else)
         */
        String path="path to folder/";
        //One file per output port (Work well only with Tuple1)
        ObsModelArtifact.addObservingArtifact (new LoggingArtifact (
                new String [] {path+"name of file1",path+"name of file 2" /**one per port**/},
                new String [] {"Names of ports you want to display"},
                "%time;%value \n"));//column for time, one for value and ";" as column separator
        //One file per output manual fixed for other type(use one method for each file)
        ObsModelArtifact.addObservingArtifact (new LoggingArtifact (
                path+"name of file1", new String [] {"name of port logged in this file"}, "%time;%valuenn"));
        ObsModelArtifact.addObservingArtifact (new LoggingArtifact (
                path+"name of file2", new String [] {"name of port logged in this file"}, "%time;%valuenn"));
        //One file for all output ports, line structure
        ObsModelArtifact.addObservingArtifact (new LoggingArtifact (
                path+"name of file",
                new String [] {"Names of ports you want to display"}, "%time;%value \n"));

        /*
         * Post Treatment
         */
        //Invoke Script R
        ObsModelArtifact.addObservingArtifact (new LogToRProject(
                path+"name of file to log",new String [] {"Names of ports you want to study"}));
        //Data comparator
        ObsModelArtifact.addObservingArtifact (new DataComparator(
                new String [] {path+"name of file to use as reference 1" /**one file per port**/},
                new String [] {"Names of ports you want to study"}));

        /*******************************/
        /**** MODELS INITIALIZATION ****/
        /*******************************/

        // Start the simulation software associated to model1
        // This is not systematically necessary, depending on the simulation software used
        agent1.startModelSoftware();
        agent2.startModelSoftware();
        obsAgent.startModelSoftware();

        // Initialize Model1 parameters
        // e.g. time discretization or constants
        // This is not systematically necessary, depending on the model
        String [] args_agent1 = { "0.001" /**;and other arguments**/ };
        String [] args_agent2 = { "0.001" /**;and other arguments**/};
        agent1.setModelParameters(args_agent1);
        agent2.setModelParameters(args_agent2);

        /***************************************/
        /**** CO-SIMULATION INIT & STARTING ****/
        /***************************************/

        try {
            // Co-initialization with first exchanges
            // This is necessary only when the model initial states are co-dependant
            agent1.coInitialize();
            agent2.coInitialize();

            // Start the co-simulation
            agent1.start();
            agent2.start();
            obsAgent.start();

            // This should never happen
        } catch (CausalityException e) {
            e.printStackTrace();
        }
    }
}
```

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

```java
import mecsyco.communication.dds.coupling.DDSEventCouplingArtifactReceiver;
import mecsyco.communication.dds.coupling.DDSEventCouplingArtifactSender;
import mecsyco.core.agent.EventMAgent;
import mecsyco.core.agent.ObservingMAgent;
import mecsyco.core.coupling.CentralizedEventCouplingArtifact;
import mecsyco.core.exception.CausalityException;
import mecsyco.core.type.SimulData;
import mecsyco.observing.base.comparator.DataComparator;
import mecsyco.observing.base.logging.LoggingArtifact;
import mecsyco.observing.jfreechart.bar.LiveBarGraphic;
import mecsyco.observing.jfreechart.bar.PostMortemBarGraphic;
import mecsyco.observing.jfreechart.event.LiveEventGraphic;
import mecsyco.observing.jfreechart.event.PostMortemEventGraphic;
import mecsyco.observing.jfreechart.pie.LivePieGraphic;
import mecsyco.observing.jfreechart.pie.PostMortemPieGraphic;
import mecsyco.observing.jfreechart.xy.LiveTXGraphic;
import mecsyco.observing.jfreechart.xy.LiveXYGraphic;
import mecsyco.observing.jfreechart.xy.PostMortemTXGraphic;
import mecsyco.observing.jfreechart.xy.PostMortemXYGraphic;
import mecsyco.observing.jfreechart.xy.Renderer;
import mecsyco.observing.jzy3d.graphic.Live3DGraphic;
import mecsyco.observing.jzy3d.graphic.PostMortem3DGraphic;
import mecsyco.observing.swing.dispatcher.SwingDispatcherArtifact;
import mecsyco.observing.swing.r.LogToRProject;

public class Launcher2 {
    //Simulation length:
    public final static double maxSimulationTime = 10;

    public static void main(String args[]) {

        /**********************************/
        /**** AGENTS & MODEL ARTIFACTS ****/
        /**********************************/

        // First agent from first model (Model1)
        //Agent
        EventMAgent agent3 = new EventMAgent("nameAgent3",maxSimulationTime);
        EventMAgent agent4 = new EventMAgent("nameAgent4",maxSimulationTime);

        //Then Model Artifact
        AModelArtifact Agent3Artifact = new AmodelArtifact( /**parameters**/);
        AModelArtifact Agent4Artifact = new AmodelArtifact(/**parameters**/);

        //Associate agent and artifact
        agent3.setModelArtefact(Agent3Artifact);
        agent4.setModelArtefact(Agent4Artifact);


        /****************************/
        /**** COUPLING ARTEFACTS ****/
        /****************************/

        //Inside communication
        CentralizedEventCouplingArtifact Agent3ToAgent4 = new CentralizedEventCouplingArtifact();
        CentralizedEventCouplingArtifact Agent4ToAgent3 = new CentralizedEventCouplingArtifact();

        //Outside communication
        //Arrows that go to Model1
        DDSEventCouplingArtifactSender Agent4ToAgent2sender = new DDSEventCouplingArtifactSender("4To2");
        //Arrows that come from Model1
        DDSEventCouplingArtifactReceiver Agent1ToAgent3receiver = new DDSEventCouplingArtifactReceiver("1To3", SimulData.class);
        // "1To3" and "4To2" are DDS topics, it needs to be the same in Model1
        // "SimulData.class" corresponds to the type of the expected value, from the remote model

        //Connection
        //Inside:
        agent3.addOutputCouplingArtifact(Agent3ToAgent4, "OutputPortName4");
        agent4.addOutputCouplingArtifact(Agent4ToAgent3, "OutputPortName5");

        agent4.addInputCouplingArtifact(Agent3ToAgent4, "InputPortName4");
        agent3.addInputCouplingArtifact(Agent4ToAgent3, "InputPortName5");

        //Outside
        // Agent4 will send data from OutputPortName6 via the topic "4To2" (output events)
        agent4.addOutputCouplingArtifact(Agent4ToAgent2sender, "OutputPortName6");
        // Agent3 will receive Data in InputPortName6 with the value received via the topic "1To3" (input events)
        agent3.addInputCouplingArtifact(Agent1ToAgent3receiver, "InputPortName6");

        /****************************************************/
        /******************** Operations ********************/
        /*** Check User Guide: Create your own operations ***/
        /****************************************************/


        /*In the case of internal operation, it does not change
         * We will then see in the cast of the external link
         */

        //In this launcher, we can only apply operation on the link from Agent1 to Agent3
        //event operation
        DataOperationTemplate DataOpe= new DataOperationTemplate();
        Agent1ToAgent3receiver.addEventOperation(DataOpe);
        //time operation
        TimeOperationTemplate TimeOpe= new TimeOperationTemplate();
        Agent1ToAgent3receiver.addTimeOperation(TimeOpe);


        /************************************************/
        /****LOGGING VISUALIZATION OR POST TREATMENT ****/
        /******** Check User Guide: MECSYCO-visu ********/
        /*** Check User Guide section Simulation data ***/
        /************************************************/

        /* Set the agent name for logging if you didn't named it at the creation
        *(otherwise, an unique default number is attributed)
        */
        agent3.setAgentName("Agentl3");
        agent4.setAgentName("Agentl4");

        /*Create observing Agent and the dispatcher
         * Create both for each different display windows you want
         */
        ObservingMAgent obsAgent2 = new ObservingMAgent ("ObserverName2", maxSimulationTime);
        SwingDispatcherArtifact ObsModelArtifact2 = new SwingDispatcherArtifact ();
        obsAgent2.setDispatcherArtifact(ObsModelArtifact2);

        /* Coupling Artifact and connection
         * Same rules apply here, if the Port to observe is from Model1, use DDS
         */
        CentralizedEventCouplingArtifact Agent3ToObs = new CentralizedEventCouplingArtifact();
        CentralizedEventCouplingArtifact Agent4Port1ToObs = new CentralizedEventCouplingArtifact();
        CentralizedEventCouplingArtifact Agent4Port2ToObs = new CentralizedEventCouplingArtifact();
        DDSEventCouplingArtifactReceiver Agent2ToObs2Receiver = new DDSEventCouplingArtifactReceiver("2ToObs2", SimulData.class);
        DDSEventCouplingArtifactReceiver Agent1Port1ToObs2Receiver = new DDSEventCouplingArtifactReceiver("1Port1ToObs2", SimulData.class);
        DDSEventCouplingArtifactReceiver Agent1Port2ToObs2Receiver = new DDSEventCouplingArtifactReceiver("1Port2ToObs2", SimulData.class);

        agent3.addOutputCouplingArtifact(Agent3ToObs, "OutputPortName4");
        agent4.addOutputCouplingArtifact(Agent4Port1ToObs, "OutputPortName5");
        agent4.addOutputCouplingArtifact(Agent4Port2ToObs, "OutputPortName6");
        //For easy reading, we named the input port as the port we want to observed
        obsAgent2.addInputCouplingArtifact(Agent3ToObs, "OutputPortName4");
        obsAgent2.addInputCouplingArtifact(Agent4Port1ToObs, "OutputPortName5");
        obsAgent2.addInputCouplingArtifact(Agent4Port2ToObs, "OutputPortName6");
        obsAgent2.addInputCouplingArtifact(Agent1Port1ToObs2Receiver, "OutputPortName1");
        obsAgent2.addInputCouplingArtifact(Agent1Port2ToObs2Receiver, "OutputPortName3");
        obsAgent2.addInputCouplingArtifact(Agent2ToObs2Receiver, "OutputPortName2");

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

        agent3.addOutputCouplingArtifact(Agent3ToObsSender, "OutputPortName4");
        agent4.addOutputCouplingArtifact(Agent4Port1ToObsSender, "OutputPortName5");
        agent4.addOutputCouplingArtifact(Agent4Port2ToObsSender, "OutputPortName6");

        /*
         *Visualization in real time  (can slow down the simulation a bit)
         *the name of ports is the one assigned as observer's input port
         *Comment the observing you don't need
         *all real time will be display on the same windows if only one was created
         */
        //Temporal graph (if ports observed are Double)
        ObsModelArtifact2.addObservingArtifact (new LiveTXGraphic (
                "Graph name", "Y axis name", Renderer.Line,//or Rendere.Dot or Renderer.Step
                new String [] {"Names for display purpose, one name per port"} ,
                new String [] {"Names of ports you want to display"}));
        //XY graphics (if the port observed is a Tuple2 of Double)
        ObsModelArtifact2.addObservingArtifact (new LiveXYGraphic(
                "Graph name", "X axis name", "Y axis name", Renderer.Line, //or Rendere.Dot or Renderer.Step
                "Name for display purpose", "Name of port observed"));
        //Bar chart (if the port observed is a SimulVector of Double)
        ObsModelArtifact2.addObservingArtifact (new LiveBarGraphic(
                "Graph name", "X axis name", "Y axis name",
                new String [] {"Names for display purpose, one name vector's component"},
                "Name of port observed"));
        //Pie chart (if the port observed is a SimulVector of Double)
        ObsModelArtifact2.addObservingArtifact (new LivePieGraphic(
                "Graph name",
                new String [] {"Names for display purpose, one name vector's component"},
                "Name of port observed"));
        //Factual representation (if the port observed is a Double)
        ObsModelArtifact2.addObservingArtifact (new LiveEventGraphic(
                "Graph name", "X axis name", "Y axis name",
                "Name for display purpose", "Name of port observed"));
        //3D graphic (if the port observed is a Tuple3 of Double)
        ObsModelArtifact2.addObservingArtifact (new Live3DGraphic(
                "Graph name", "X axis name", "Y axis name", "Z axis name",
                "Name of port observed"));

        /*
         *Visualization in post-mortem
         *same comment as for real time
         */
        //Temporal graph (if ports observed are Double)
        ObsModelArtifact2.addObservingArtifact (new PostMortemTXGraphic (
                "Graph name", "Y axis name", Renderer.Line,//or Rendere.Dot or Renderer.Step
                new String [] {"Names for display purpose, one name per port"} ,
                new String [] {"Names of ports you want to display"}));
        //XY graphics (if the port observed is a Tuple2 of Double)
        ObsModelArtifact2.addObservingArtifact (new PostMortemXYGraphic(
                "Graph name", "X axis name", "Y axis name", Renderer.Line, //or Rendere.Dot or Renderer.Step
                "Name for display purpose", "Name of port observed"));
        //Bar chart (if the port observed is a SimulVector of Double)
        ObsModelArtifact2.addObservingArtifact (new PostMortemBarGraphic(
                "Graph name", "X axis name", "Y axis name",
                new String [] {"Names for display purpose, one name vector's component"},
                "Name of port observed"));
        //Pie chart (if the port observed is a SimulVector of Double)
        ObsModelArtifact2.addObservingArtifact (new PostMortemPieGraphic(
                "Graph name",
                new String [] {"Names for display purpose, one name vector's component"},
                "Name of port observed"));
        //Factual representation (if the port observed is a Double)
        ObsModelArtifact2.addObservingArtifact (new PostMortemEventGraphic(
                "Graph name", "X axis name", "Y axis name",
                "Name for display purpose", "Name of port observed"));
        //3D graphic (if the port observed is a Tuple3 of Double)
        ObsModelArtifact2.addObservingArtifact (new PostMortem3DGraphic(
                "Graph name", "X axis name", "Y axis name", "Z axis name",
                "Name of port observed"));

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

        /*
         * Post Treatment
         */
        //Invoke Script R
        ObsModelArtifact2.addObservingArtifact (new LogToRProject(
                path+"name of file to log",new String [] {"Names of ports you want to study"}));
        //Data comparator
        ObsModelArtifact2.addObservingArtifact (new DataComparator(
                new String [] {path+"name of file to use as reference 1" /**one file per port**/},
                new String [] {"Names of ports you want to study"}));

        /******************************/
        /**** MODELS INITIALIZATION ****/
        /******************************/

        // Start the simulation software associated to model1
        // This is not systematically necessary, depending on the simulation software used
        agent3.startModelSoftware();
        agent4.startModelSoftware();
        obsAgent2.startModelSoftware();

        // Initialize Model1 parameters
        // e.g. time discretization or constants
        // This is not systematically necessary, depending on the model
        String [] args_agent3 = { "0.001" /**;and other arguments**/ };
        String [] args_agent4 = { "0.001" /**;and other arguments**/};
        agent3.setModelParameters(args_agent3);
        agent4.setModelParameters(args_agent4);

        /**************************************/
        /**** CO-SIMULATION INIT & STARTING ****/
        /**************************************/

        try {
            // Co-initialization with first exchanges
            // This is necessary only when the model initial states are co-dependant
            agent3.coInitialize();
            agent4.coInitialize();

            // Start the co-simulation
            agent3.start();
            agent4.start();
            obsAgent2.start();

            // This should never happen
        } catch (CausalityException e) {
            e.printStackTrace();
        }
    }
}
```

3.2 Remarks:

You can notice that DDS is compatible with all other functions of MECSYCO (observing, operation). Becareful, 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.

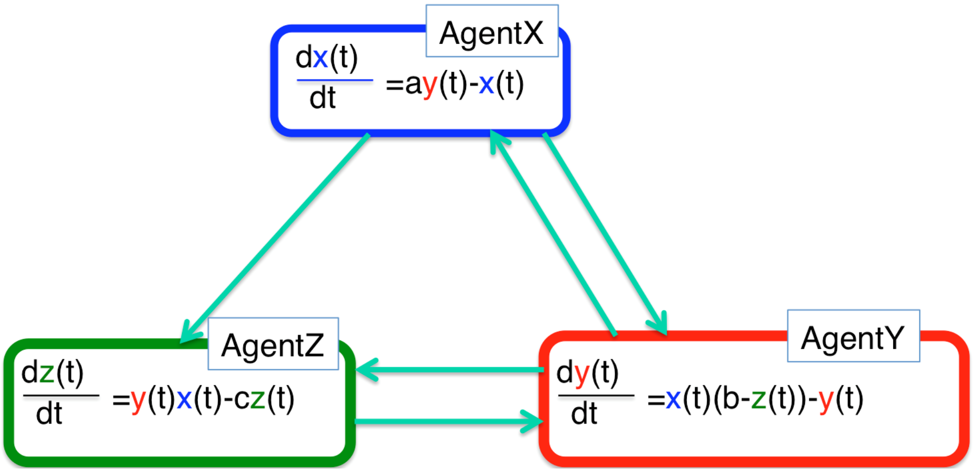
AgentX
$\frac{dx(t)}{dt} = ay(t)-x(t)$
AgentZ
$\frac{dz(t)}{dt} = y(t)x(t)-cz(t)$
AgentY
$\frac{dy(t)}{dt} = x(t)(b-z(t))-y(t)$

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*.