
**Information technology — MPEG video
technologies —**

**Part 6:
Tools for reconfigurable media coding
implementations**

Technologies de l'information — Technologies vidéo MPEG —

Partie 6: Outils d'implémentation du codage média reconfigurable





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2017, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Foreword	v
Introduction	vi
1 Scope	1
2 Normative references	1
3 Terms, definitions and abbreviated terms	1
4 Overview	3
5 RVC-CAL	3
5.1 General	3
5.2 Installing ORCC tools	3
5.2.1 Java Runtime Environment	3
5.2.2 Eclipse	4
5.2.3 ORCC plug-in for Eclipse	4
5.3 "Hello world"	5
5.3.1 Creating a new project	5
5.3.2 Creating a new package	7
5.3.3 Creating a new actor	7
5.3.4 Creating a network	8
5.3.5 Running simulation	12
5.4 Simple actor	15
5.4.1 Structure of actors	15
5.4.2 Simplest actor	15
5.4.3 Running the examples	16
5.4.4 Other simple actors	17
5.4.5 Network of simple actors	19
5.5 Non-determinism	20
5.6 Guarded actions	20
5.7 State variables	23
5.8 Scheduling	25
5.9 Priorities	28
5.10 Repeat clause	29
5.11 Control flow	31
5.11.1 General	31
5.11.2 Data types	31
5.11.3 Assignments	31
5.11.4 If statement	32
5.11.5 While statement	33
5.11.6 Foreach statement	33
6 Papify and Papify Viewer	34
6.1 General	34
6.2 Using Papify	34
6.2.1 Papify activation	35
6.2.2 Actor assessment	36
6.2.3 Action assessment	37
6.2.4 Output folder	37
6.3 Papify Viewer	37
6.3.1 Chronological visualization	37
6.3.2 Event histograms	40
7 TURNUS	41
7.1 General	41
7.2 Installing the TURNUS framework	42
7.2.1 General	42
7.2.2 Java Runtime Environment	42

7.2.3	Eclipse	42
7.2.4	TURNUS plug-in for Eclipse	42
7.3	Profiling an RVC-CAL HEVC video decoder	43
7.3.1	General	43
7.3.2	Download the design and the conformance bit-streams	43
7.3.3	Import the HEVC design project in the Eclipse IDE workspace	44
7.3.4	Static code profiling	45
7.3.5	Dynamic code programming	49
7.3.6	TURNUS ORCC dynamic interpreter profiler	50
7.3.7	Algorithmic bottleneck analysis	61
7.3.8	Impact analysis	62
7.3.9	Buffer size minimization	63
7.3.10	Partitioning	64
Bibliography		66

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all the parts in the ISO/IEC 23002 series can be found on the ISO website.

Introduction

This document provides a description of a set of tools that are intended to be helpful for developing reconfigurable media coding implementations based on ISO/IEC 23001-4, ISO/IEC 23002-4 and ISO/IEC 23002-5. The description includes the following guidelines:

- guidelines on good practices to implement specifications based on ISO/IEC 23001-4, ISO/IEC 23002-4 and ISO/IEC 23002-5;
- guidelines on usage of a monitoring tool for specifications based on ISO/IEC 23001-4, ISO/IEC 23002-4 and ISO/IEC 23002-5.
- guidelines on usage of a design exploration and optimization tool for specifications based on ISO/IEC 23001-4, ISO/IEC 23002-4 and ISO/IEC 23002-5.

Information technology — MPEG video technologies —

Part 6:

Tools for reconfigurable media coding implementations

1 Scope

This document provides a description of a set of tools that are intended to be helpful for developing reconfigurable media coding implementations based on ISO/IEC 23001-4, ISO/IEC 23002-4 and ISO/IEC 23002-5.

2 Normative references

There are no normative references in this document.

3 Terms, definitions and abbreviated terms

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at <http://www.electropedia.org/>
- ISO Online browsing platform: available at <http://www.iso.org/obp>

3.1 Terms and definitions

3.1.1

Reconfigurable Video Coding RVC

framework defined to support coding standards at the tool level while maintaining interoperability between solutions from different implementers

3.1.2

functional unit

FU

modular tool characterized by its input/output behaviour, consisting of a processing unit

Note 1 to entry: Also referred to as “actor”.

3.1.3

token

data entity exchanged among *functional units* (3.1.2), such that a functional unit performs operations on input tokens, produces output tokens and modifies its state

3.1.4

connection

link from output ports to input ports of *functional units* (3.1.2) that enable *token* (3.1.3) exchange between the corresponding functional units

3.1.5

functional unit network language

FU network language

FNL

language that describes a network of *functional units* ([3.1.2](#))

3.1.6

functional unit network description

FU network description

FND

functional unit ([3.1.2](#)) *connection* ([3.1.4](#)) used to build a decoder and modelled using the *functional unit network language* ([3.1.5](#))

3.1.7

RVC-CAL

dataflow specification language for specifying of *functional units* ([3.1.2](#)) and the reference software

3.1.8

video tool library

VTL

collection of *functional units* ([3.1.2](#))

3.1.9

MPEG video tool library

video tool library ([3.1.8](#)) that contains *functional units* ([3.1.2](#)) drawn from existing MPEG standards

3.2 Abbreviated terms

API	Application Programming Interface
BXDF	Buffer-size XML Dataflow Format
CAL	CAL Actor Language
DSE	Design Space Exploration
ETG	Execution Trace Graph
FIFO	First-In, First-Out
FSM	Finite State Machine
HEVC	High-Efficiency Video Coding
IDE	Integrated Development Environment
L1_DCM	L1 Data Cache Misses
NoL	Number of Lines
PAPI	Performance API
PMC	Performance Monitoring Counter
RMC	Reconfigurable Media Coding
TOT_INS	TOTAL Number of INSTRUCTIONS
XDF	XML Dataflow Format
XML	eXtensible Markup Language

4 Overview

This document describes a set of tools that are intended to be helpful for developing reconfigurable media coding implementations. [Clause 5](#) describes how to implement an RVC-CAL specification using the Open RVC-CAL Compiler (ORCC)¹⁾ tool. [Clause 6](#) details how to use Papify²⁾ and Papify Viewer³⁾. Papify is an implementation of an event-based performance monitoring tool integrated into ORCC. Papify Viewer is a visualization tool to monitor the actions of actors in RVC-CAL specifications. Fired actions can be analysed chronologically from different points of view. Finally, [Clause 7](#) describes how to use TURNUS⁴⁾, an open-source system design exploration and optimization framework especially tailored for CAL dataflow programs.

5 RVC-CAL

5.1 General

RVC-CAL is a standardized version of the CAL Actor Language which implements a dataflow model of computation.

All the examples in this document are written for and tested with the ORCC compiler infrastructure tools.

Open RVC-CAL Compiler (ORCC) is an open-source Integrated Development Environment (IDE) based on Eclipse and dedicated to dataflow programming. Eclipse⁵⁾ is a software package which provides an integrated development environment for Java/C/C++ programs. The primary purpose of ORCC is to provide developers with a compiler infrastructure to allow software/hardware code to be generated from dataflow descriptions.

5.2 Installing ORCC tools

First, all the tools needed for compiling and running the provided examples should be installed. The Open RVC-CAL Compiler (ORCC) requires the Eclipse platform. Therefore, in order to use it, it is necessary to install the Java™ Runtime Environment (JRE)⁶⁾ and Eclipse IDE.

5.2.1 Java Runtime Environment

ORCC requires JRE version 1.6 or higher. The latest JRE release is available at <https://java.com/en/download/>. To use the Eclipse IDE for Java Developers edition, it is necessary to also install the Java™

1) Open RVC-CAL Compiler (ORCC) is an open-source IDE based on Eclipse. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

2) Papify is an open-source tool available on GitHub. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

3) Papify Viewer is an open-source visualization tool available on GitHub. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

4) TURNUS is an open-source system published by IEEE.org. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

5) Eclipse is an open-source software package published by the Eclipse Foundation (www.eclipse.org). This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

6) Java Runtime Environment (JRE) is a trademark of a product supplied by Oracle. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Development Kit (JDK)⁷⁾, which can be downloaded from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

5.2.2 Eclipse

ORCC is compatible with Eclipse versions 4.3 and higher. The Eclipse IDE can be downloaded from <https://www.eclipse.org/downloads/> (for which the Eclipse IDE for Java Developers edition is suggested). To install it, extract the archive into a local directory.

Eclipse should be configured to allocate at least 512 MB of memory for the heap. This can be done by adding the -Xmx512m parameter in the eclipse.ini file.

5.2.3 ORCC plug-in for Eclipse

The Eclipse Software Update Manager can be used to install ORCC. The steps are as follows.

- In Eclipse, go to Help > Install New Software.
- Click Add to add an update site.
- Set its name (e.g. ORCC) and its URL to <http://orcc.sourceforge.net/eclipse> (see Figure 1).

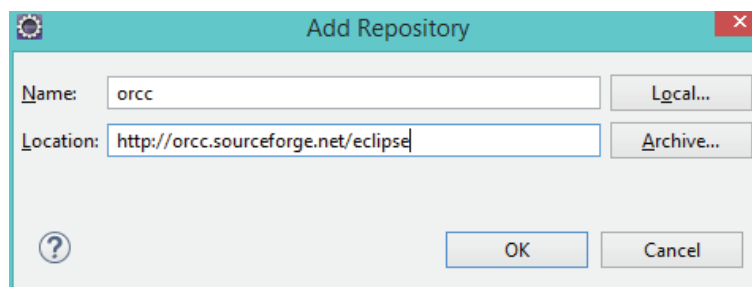


Figure 1 — Repository setting during the installation of the ORCC plug-in for Eclipse

- Once this is done, select Open RVC-CAL Compiler or ORCC (see Figure 2).

7) Java Development Kit (JDK) is a trademark of a product supplied by Oracle. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

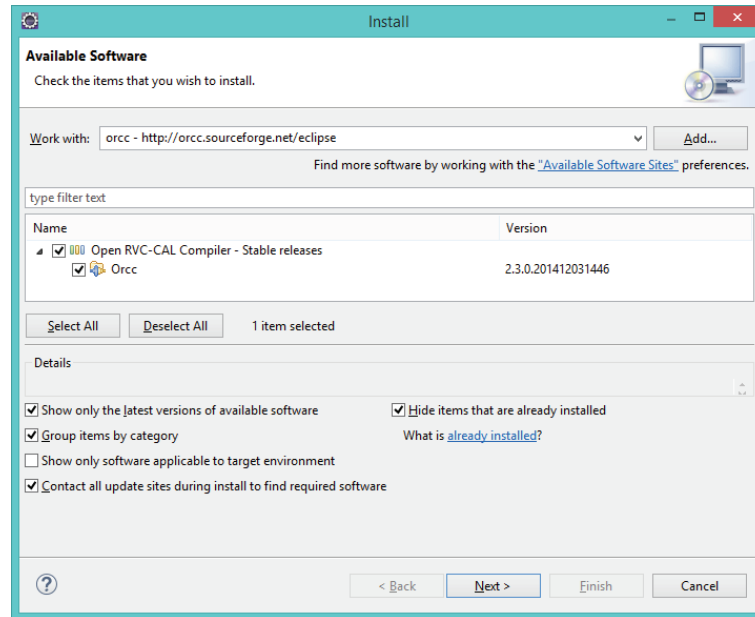


Figure 2 — Selection of the Open RVC-CAL Compiler during the installation of the ORCC plug-in for Eclipse

- Click Next, check and accept the licenses and then click Finish.
- Eclipse will prompt for a confirmation to install an unverified feature. Accept and restart Eclipse.

5.3 “Hello world”

Make sure that the active perspective in Eclipse is Java™ EE⁸⁾; otherwise, the menus will differ slightly and some menu items will not be in the illustrated places.

5.3.1 Creating a new project

A new ORCC project should be created. In the menu File > New > Other, select ORCC > Orcc Project (see [Figure 3](#)).

8) Java EE (JEE) is a trademark of a product supplied by Oracle. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

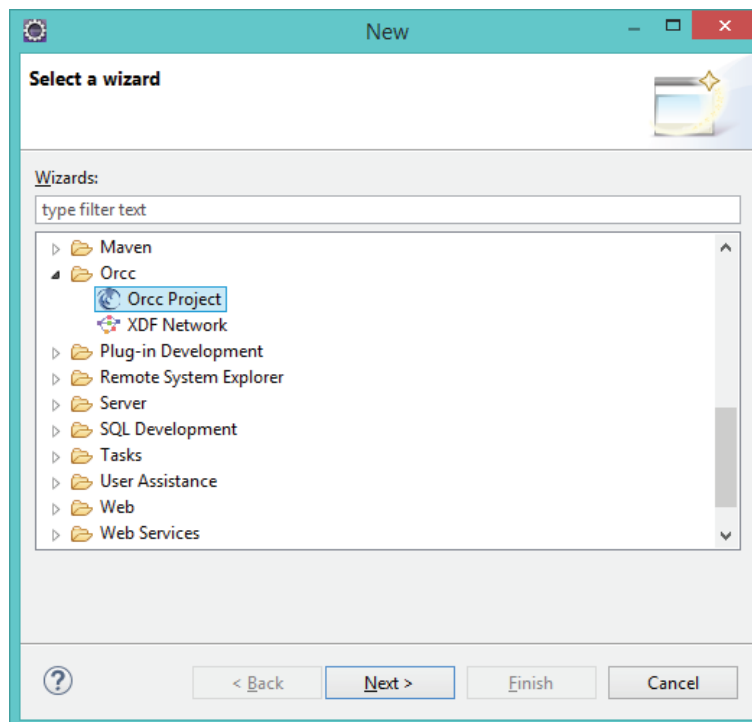


Figure 3 — Wizard selection to create a new project

Specify the name of the project and click Finish (see [Figure 4](#)).

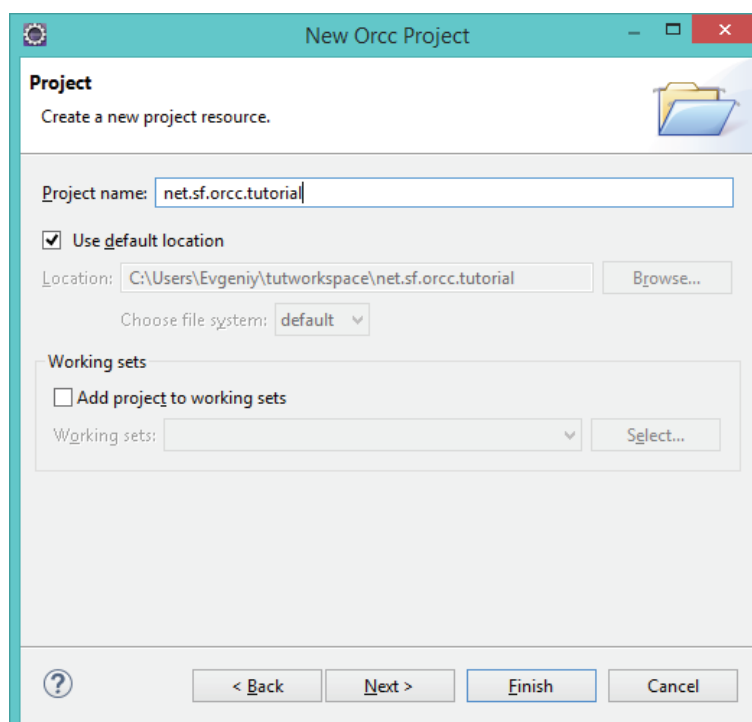


Figure 4 — Project name setting in ORCC

A default *src* directory will be added to the created project by this action.

5.3.2 Creating a new package

Right-click on the *src* folder in the *Project Explorer* pane. Select New > Package (see [Figure 5](#)).

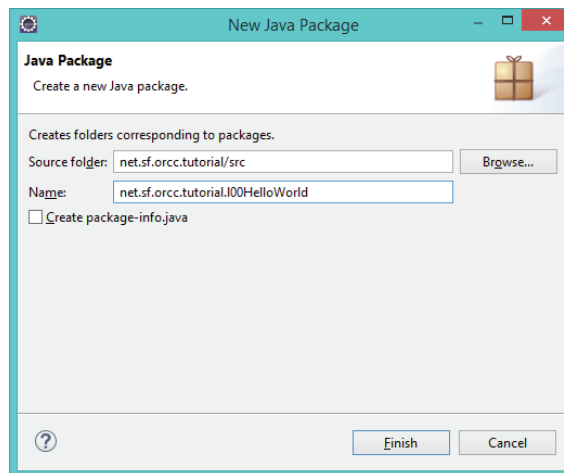


Figure 5 — Package setting in ORCC

5.3.3 Creating a new actor

In the *Project explorer* pane, select the package that was just created. Then click menu File > New > File. Specify the name and the extension *.cal* (see [Figure 6](#)).

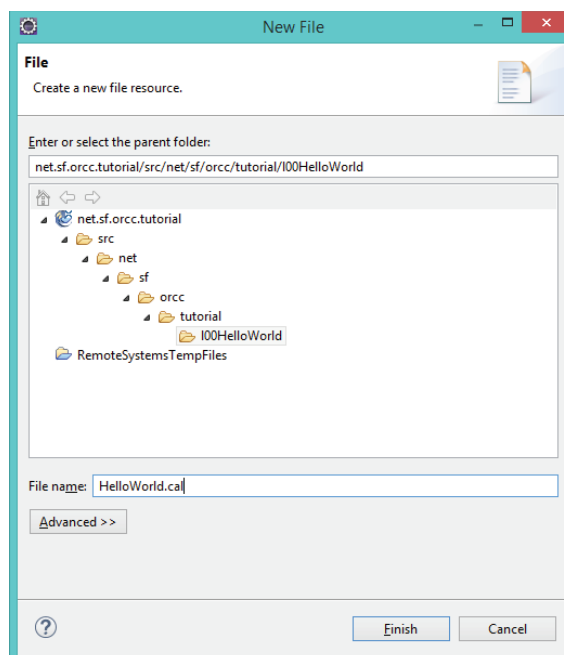


Figure 6 — New CAL file setting in ORCC

After file is created, add the following code and save it. Eclipse will automatically compile the file.

```
package net.sf.orcc.tutorial.HelloWorld;

actor HelloWorld () int In ==> :
  action ==>
  do
    print("Hello World!\n");
  end
end
```

The code above implements the actor named **HelloWorld()**, which takes an input stream of tokens of type **int**.

The action within this actor always executes regardless of any input or other conditions and prints the string to the default output.

A detailed explanation of the syntax is given in the succeeding subclauses.

5.3.4 Creating a network

RVC-CAL is a language that implements the dataflow paradigm. This means that in order to run the application, it will be necessary to build a network of actors. In this case, the network will consist of only one actor.

To build a network, it will be necessary to create a new .xdf file. Go to File > New > Other, then select Orcc > XDF Network (see [Figures 7](#) and [8](#)).

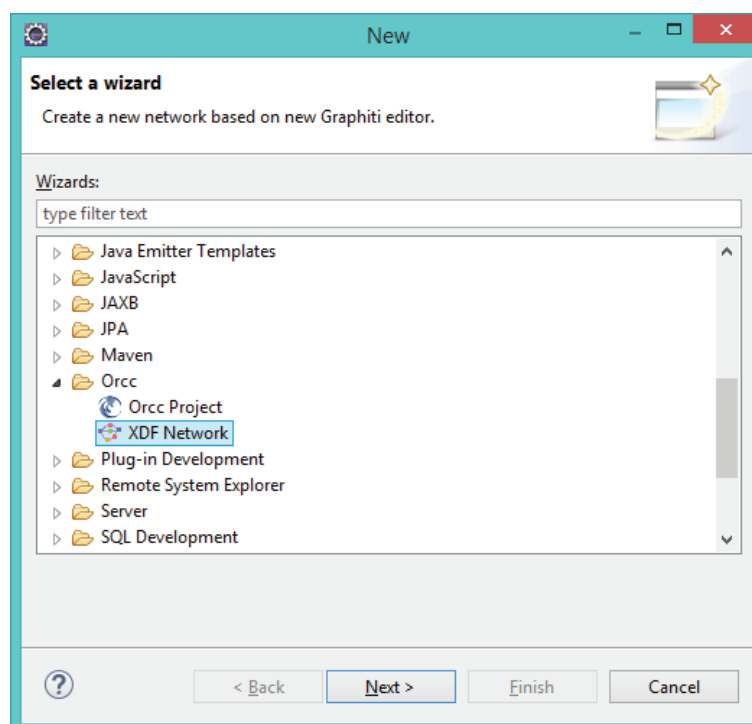


Figure 7 — Network setting in ORCC

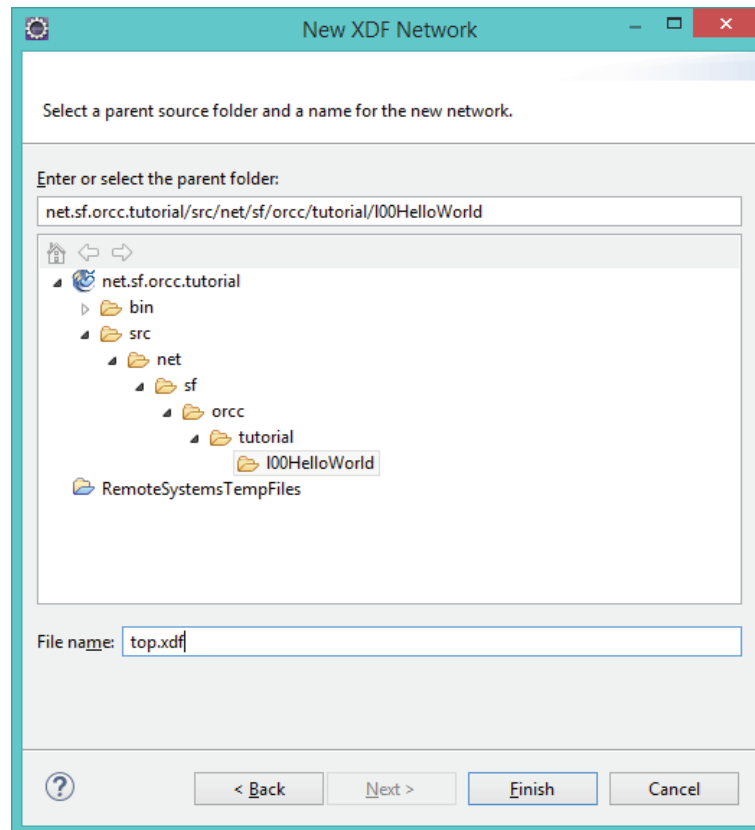


Figure 8 — Network name setting in ORCC

After the empty XDF is created, it will then be necessary to add an instance of an actor (see [Figure 9](#)).

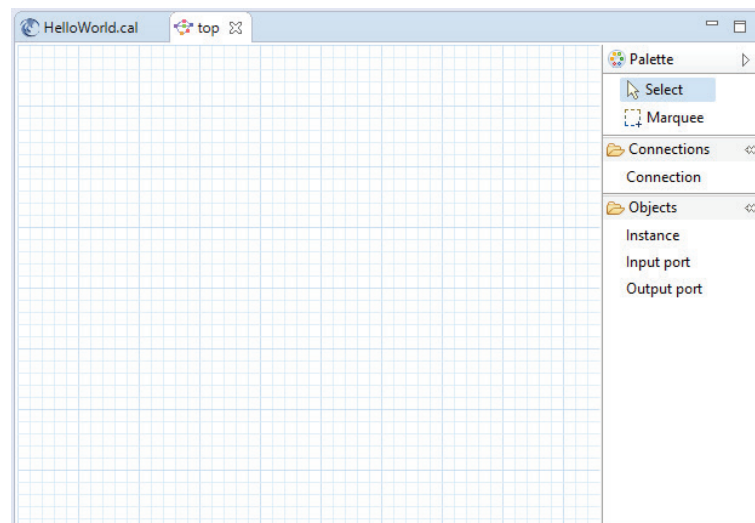


Figure 9 — Empty space for actor instance addition in ORCC

Click on Objects > Instance in the *Palette*, and then click on the .xdf file area to add an instance to the network (see [Figure 10](#)).

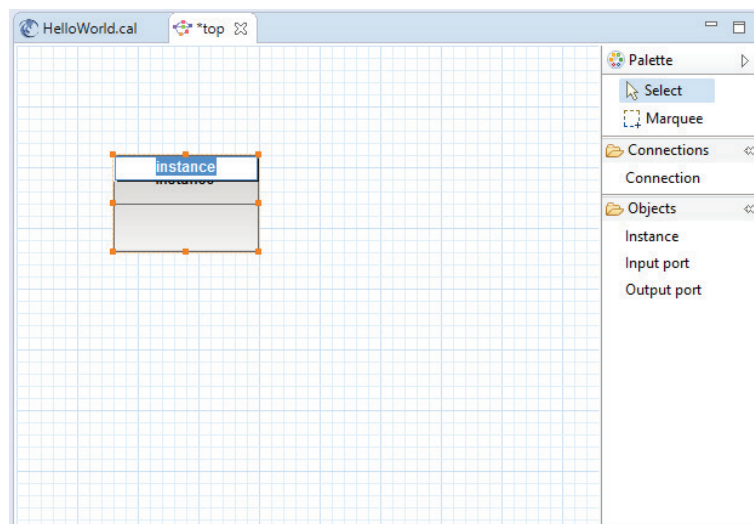


Figure 10 — Actor instance addition in ORCC

Name the instance “Hello”.

Now link this instance to the actor created. Right-click on the instance and select Set/Update Refinement (see [Figure 11](#)).

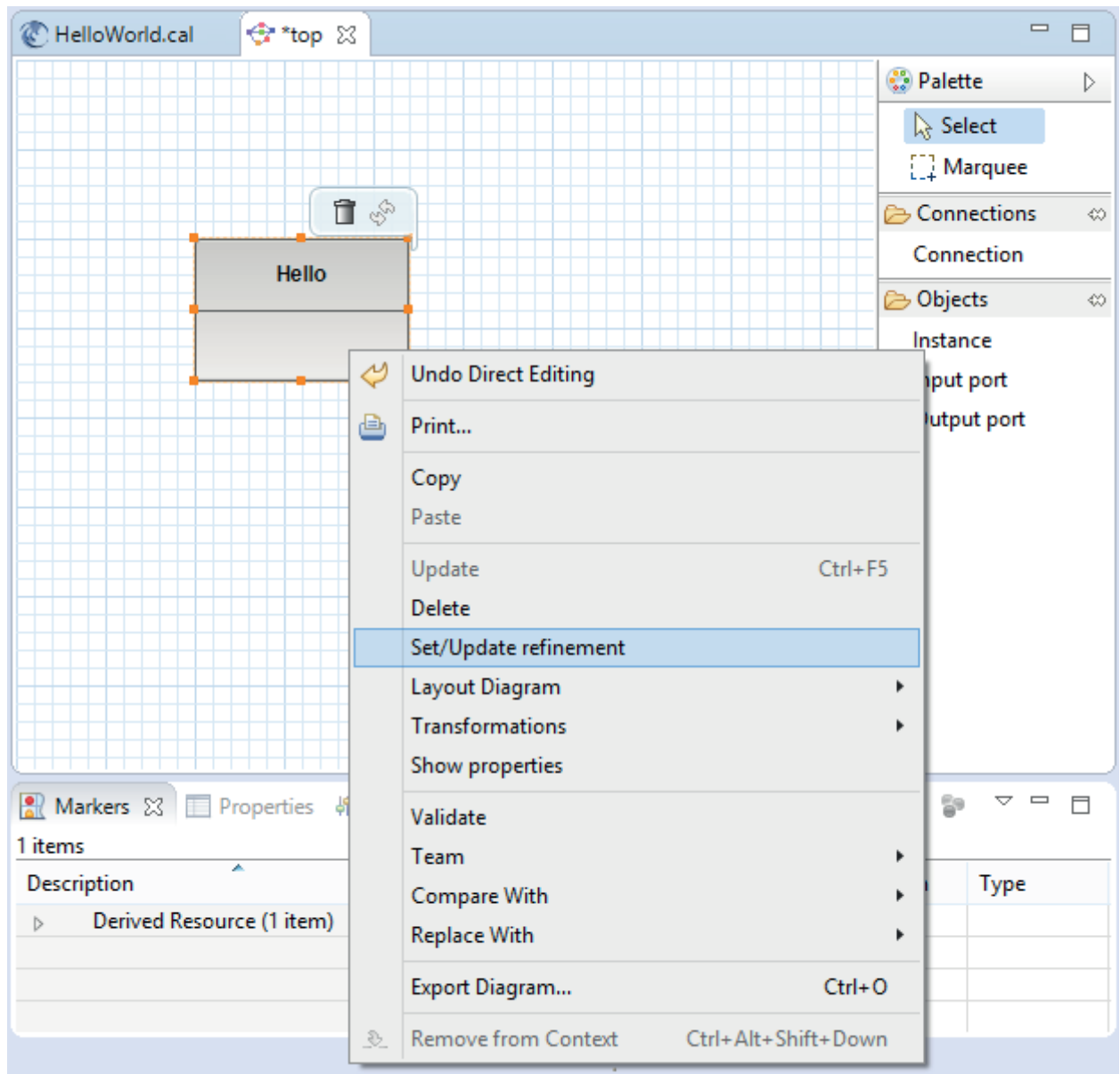


Figure 11 — Refinement of an actor instance to the corresponding CAL file in ORCC

Select the *HelloWorld* actor in the newly opened box (see [Figure 12](#)).

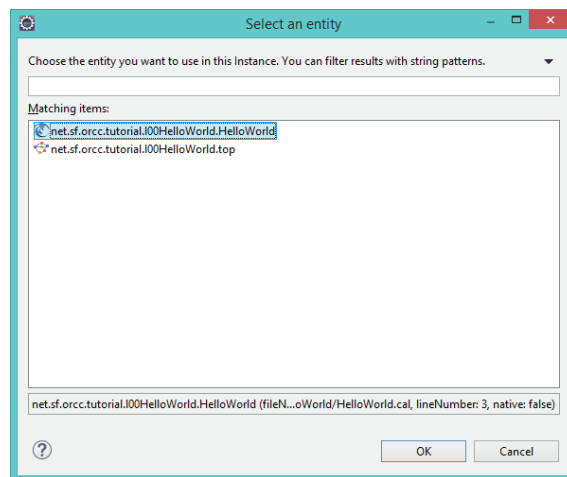


Figure 12 — Selection of a CAL file during an instance refinement process in ORCC

After validation, the *Hello* instance should be displayed in blue (meaning that the instance is assigned to an actor). See [Figure 13](#).

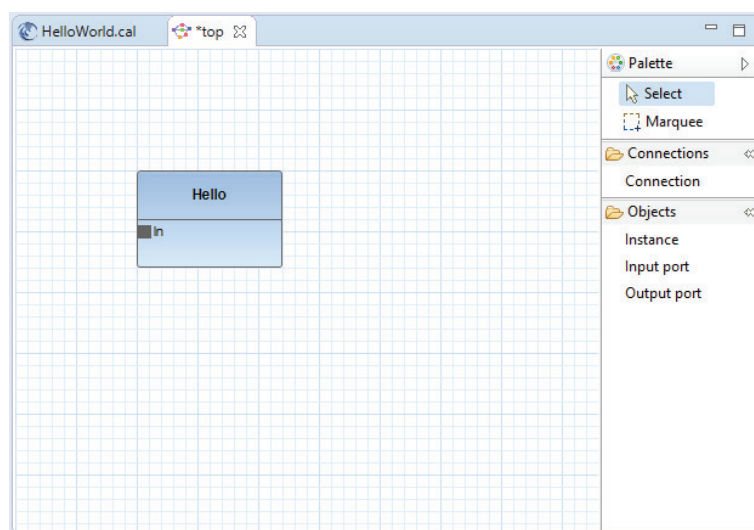


Figure 13 — Hello instance associated to the corresponding CAL file

5.3.5 Running simulation

Right-click on the .xdf file and select Run As > Orcc simulation (see [Figure 14](#)).

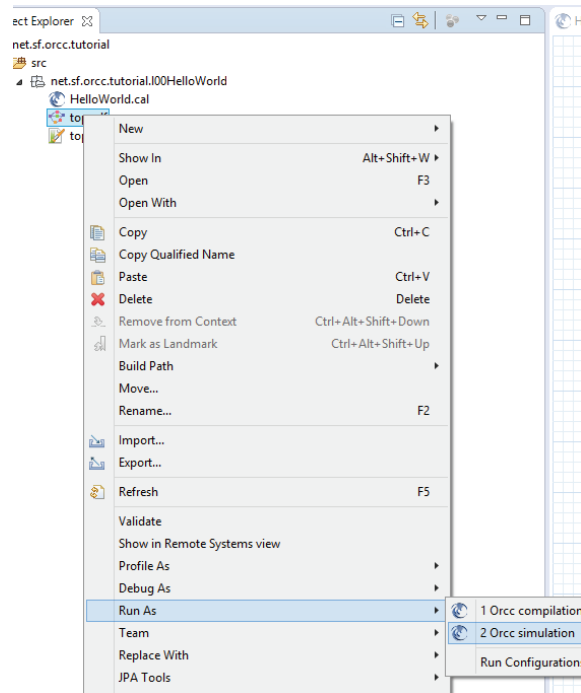


Figure 14 — Simulation selection in ORCC

In the *Select simulator* window, click OK (see [Figure 15](#)).

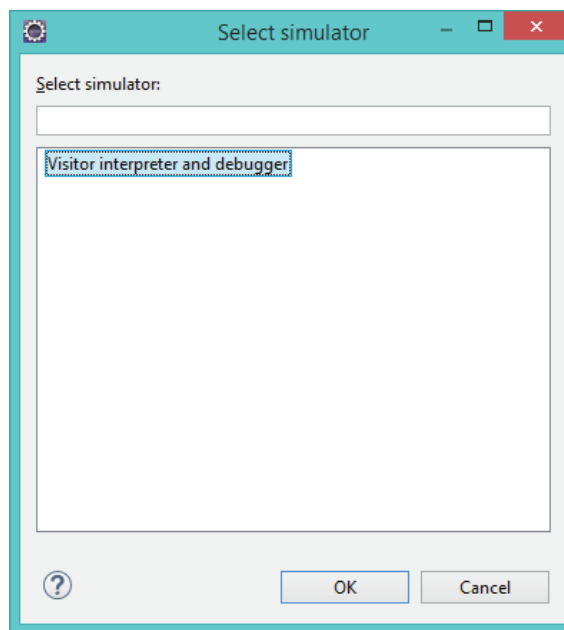


Figure 15 — Interpreter and debugger selection for simulation in ORCC

In the *Select input stimulus* window, select a random file (it will not actually be used by the example). See [Figure 16](#).

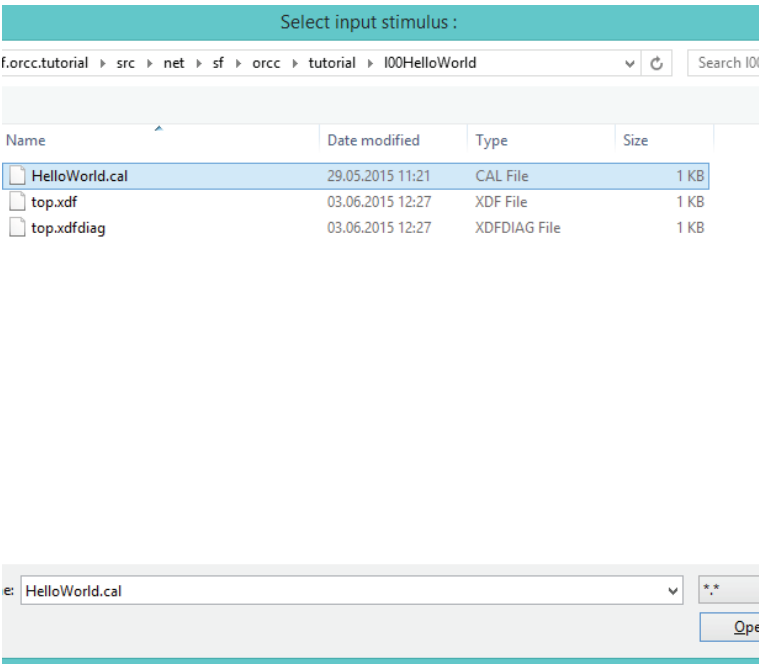


Figure 16 — Input stimuli selection for simulation in ORCC

In the *Run configuration wizard*, click on Run (see [Figure 17](#)).

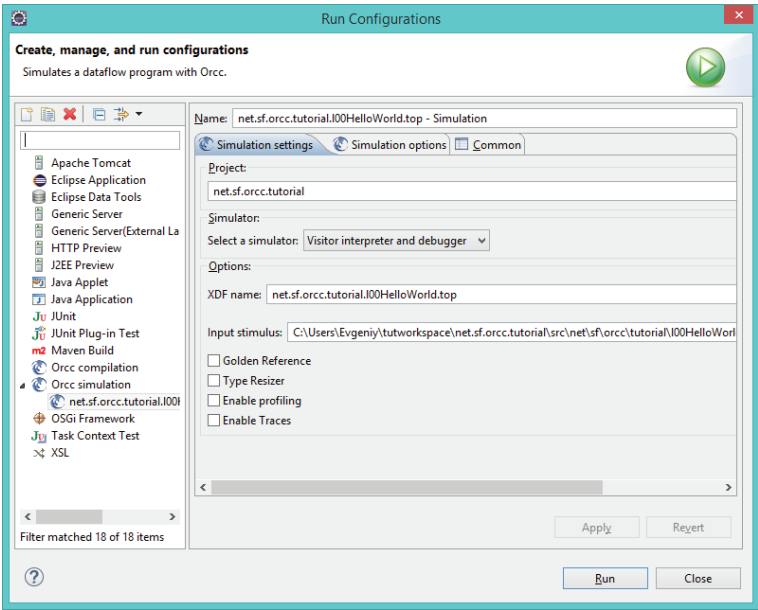


Figure 17 — Simulation configuration in ORCC

The following output (see [Figure 18](#)) should appear in the Eclipse console.

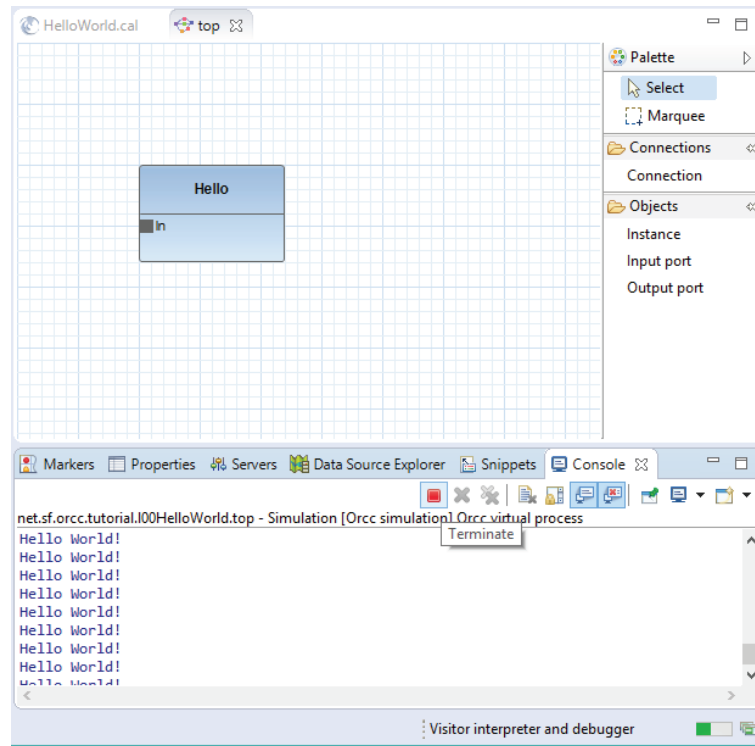


Figure 18 — Simulation console in ORCC

5.4 Simple actor

5.4.1 Structure of actors

Actors perform their computation in a sequence of steps called firings. In each of those steps, the actor

- a) may consume tokens from its input ports,
- b) may produce tokens at its output ports, and
- c) may modify its internal state (this is described in further subclauses).

Describing an actor involves describing its interface, the ports, the structure of its internal state, as well as the steps it can perform, what these steps do (in terms of token production, consumption and actor state update) and how to select the step an actor will perform next.

5.4.2 Simplest actor

The simplest actor just copies a token from the input to the output without changing its value.

```
package net.sf.orcc.tutorial.SimpleActor;

actor ID ( ) int In ==> int Out :
  first: action In: [a] ==> Out: [a] end
end
```

The first line specifies the package.

The main entity in RVC-CAL is an *actor*. In this example, the keyword **actor** is used followed by the name of the actor and the parameters in parentheses (an empty list in this example) to describe the

actor. Input and output ports are specified before and after the `==>` sign, respectively. RVC-CAL is a *statically typed* language, so it is necessary to explicitly define the type for each variable, i.e. in the line **actor ID () int In ==> int Out :** ports **In** and **Out** both are of type **int**. The colon at the end of the line marks the start of the actor body which is bounded by the keyword **end** from the other side.

Each actor has one or more actions within the body section, which execute (*fire*) at one step each. Actions may (or may not) *consume* input tokens and *produce* output tokens at each step. The syntax to describe an action in RVC-CAL is the following:

```
actor <ActorIdentifier> () <input ports> ==> <output ports> :
  [<ActionIdentifier>:] action <input pattern> ==> <output expression>
```

Input pattern specifies how many tokens to consume, from which ports the tokens are consumed and how to call these tokens in the rest of the action. The input pattern for the **ID** actor is **In: [a]**. It tells the action to consume one token from the input **In** and name it **a** within the action body. *Input patterns* realize the idea of *pattern matching*.

The expression following the `==>` sign is an *output expression*. It defines the number and values of output tokens which will be produced on each output port by each *firing* of the action. In this example, **Out: [a]** is an *output expression*. It defines that exactly one token will be produced on the output port **Out** and the value of that token will be **a**.

It is important to understand the difference between *input pattern* and *output expression*. In the input pattern, the local variable **a** is declared and assigned to the value of the input token whenever the action is fired. The output expression uses that variable and send the value of **a** as a produced token to the output port at the end of the action firing.

5.4.3 Running the examples

In the project, create a new package **net.sf.orcc.tutorial.SimpleActor**

Create new CAL file named *ID.cal* and copy the following code there.

```
package net.sf.orcc.tutorial.SimpleActor;

actor ID () int In ==> int Out :
  first: action In: [a] ==> Out: [a] end
end
```

In order to build a network, several additional actors are needed to produce data and print results to the console. This example uses **Source** and **Printer** actors as utilities. These can be downloaded from the GitHub repository (<https://github.com/orcc/rvccaltut/tree/master/net.sf.orcc.tutorial/src/net/sf/orcc/tutorial/utills>) and added to the project.

Source actor is a counter which produces a continuous sequence of numbers. Parameters can be specified for the starting number (default is 1) and the counter upper bound (default is 10). To know how to do that, see [5.4.4.4](#).

Printer prints all the consumed tokens to the console. The name can be specified for each instance. That is useful when there are several of them printing to the same console simultaneously.

The network can be created as indicated in [5.3](#).

After adding instances of actors **ID**, **Source** and **Printer**, the **Source** output should be connected to the **ID** input, and the **ID** output to the **Printer** input.

Note that it is possible to drag-and-drop an actor file from the Project explorer pane to the XDF network diagram to add an instance of an actor.

Now the example can be run as described in [5.3](#) and the result can be seen in the console (see [Figure 19](#)).

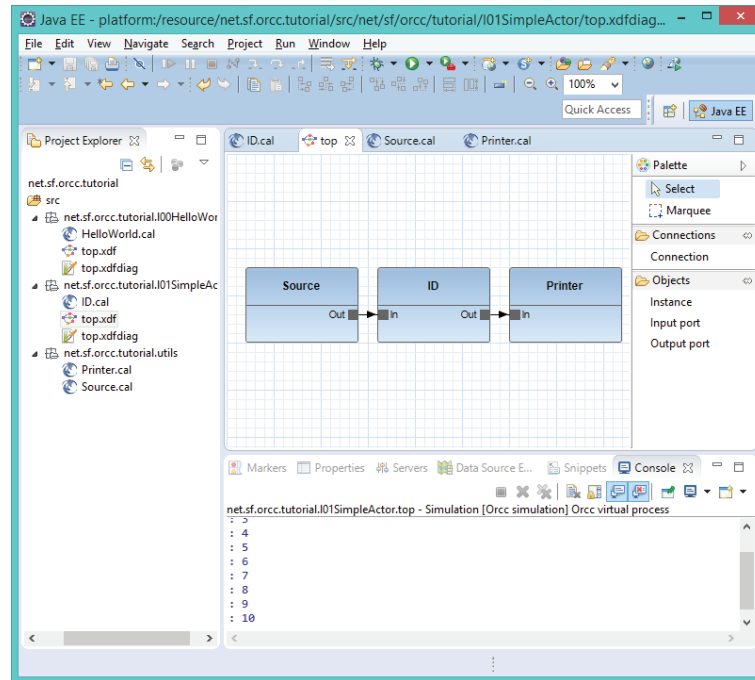


Figure 19 — Simulation of a simple network of actors in ORCC

5.4.4 Other simple actors

5.4.4.1 Add

The next example shows how to make an actor which will be as simple as **ID** but at the same time will perform a real manipulation on the data

```
package net.sf.orcc.tutorial.SimpleActor;

actor Add () int In1, int In2 ==> int Out :
    action In1: [a], In2: [b] ==> Out: [a+b]
end
end
```

This package has two *input ports* separated by a comma, **int In1**, **int In2**, in the actor declaration. Also, *input pattern* changed to **In1: [a]**, **In2: [b]** which means that action will be *fired* only when both ports, **In1** and **In2**, have a valid data on their inputs. Consumed tokens will then be assigned to **a** and **b**, respectively. Moreover, this example clarifies the difference between *input pattern* and *output expression*. **Out: [a+b]**, the *output expression*, includes an actual expression (the sum of two variables), which is calculated after the action is finished and the result is sent to the output port.

5.4.4.2 AddSeq

The example in [5.4.4.1](#) consumes two tokens from two input ports. The following example shows how to add two values with only one input port.

```
package net.sf.orcc.tutorial.SimpleActor;

actor AddSeq () int In1 ==> int Out :
    action In1: [a, b] ==> Out: [a+b]
end
end
```