Basic POOSL tutorial

1 Introduction

This document contains a short introduction to the language POOSL (Parallel Object-Oriented Specification Language) with a simple example written using the POOSL IDE in Eclipse.

2 POOSL Overview

POOSL is based on processes that can communicate through ports. For a port P, there are two communication statements:

- P ! M(p1, …, pn) : send message M with parameters p1, …, pn via port P
- P? M(x1, …,xn) : receive message M via port P and store the parameters in variables x1, …, xn.

The ports of processes can be connected using channels. Communication is synchronous, that is, a send statement is blocked until a matching receive statement can be executed via a connecting channel in another process. Similarly, a receive statement is blocked until a matching send statement can be executed on a connecting channel. When a matching statement is available, the send and the receive statement are executed simultaneously.

Processes can be grouped into clusters that allow for hierarchy definition. Clusters may be part of other clusters. In addition to process classes and cluster classes, also data classes can be defined.

3 A producer-consumer example

Figure 3-1 presents the schematic overview of a producer/consumer example. The components are a Producer of packages, a receiver or Consumer for the packages and a Network.

Producer and Consumer are part of an Application layer. Each component of the Application layer exchanges information through the network. The Producer creates packets that are

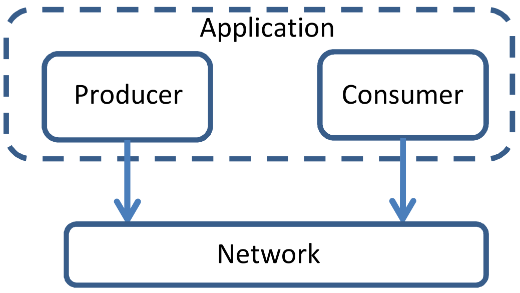

Figure 3-1

identified by an ID and contain a text. The Consumer just receives packets.

4 Data

A data object contains its own data and has the possibility to perform operations upon this data. This data is stored in variables that may contain (references to) other data objects. The variables of an object are encapsulated by that object; they are not directly accessible by other objects. They can only be read and modified by the object itself. A data class defines a number of methods that can be used access of change the private data variables.

Besides user-defined data classes, there are a number of so-called primitive data classes, which have been predefined. Examples of primitive data objects are integers (3, 10, 32769), reals (2.33), logical values (true, false) and characters (‘a’, ‘b’).

4.1 Example

For our example we created one data class for Packets sent through the network. A Packet object contains an identifier and a text of type Integer and String, respectively. The data object has four methods: two methods to set the internal variables, one to obtain the identifier, and one to print the content of the Packet.

```
data class Packet extends Object
variables
      Identifier : Integer,
      Text : String
methods
      setIdentity(I : Integer) : Packet
            Identifier := I ;
            return(self)
      setMessage(S : String) : Packet
            Text := S ;
            return(self)
      GetIdentifier() : Integer
            return(Identifier)
      printString() : String
      /* This method succinctly visualises a packet  when inspecting it */
            return("[" + Identifier printString + ": " + Text + "]")
```

5 Process

Processes may contain internal variables; these variables are only accessible by the process itself. Processes do not share data. A process contains a number of methods that can call each other. A process can only call its own methods or methods of data classes. Each process, has an initial method.

5.1 Example

Our example contains three processes: Network, Producer and Consumer. Each process defines the ports, the messages, the internal variables, and the methods.

The Network waits for a Packet on port In and next immediately sends it on port Out.

```
process class Network()
ports
      Out,
      In
messages
```

```
      In ? Message(Packet),
      Out ! Message(Packet)
variables
      p : Packet
init
      InitialiseNetwork()()
methods
      InitialiseNetwork()()
            TransferData()()
      TransferData()()
            In ? Message(p) ;
            Out ! Message(p) ;
            TransferData()()
```

The Producer generates Packets at constant time intervals. The time interval is given by parameter IdleTime when a Producer instance is created.

```
process class Producer(IdleTime : Integer)
ports
      Out
messages
      Out ! Message(Packet)
variables
      NextIdentifier : Integer
init
      InitialiseProducer()()
methods
      SendPackages()() | p : Packet |
            p := new(Packet) setIdentity(NextIdentifier) setMessage("Message with
number "+ NextIdentifier printString);
            NextIdentifier := NextIdentifier + 1 ;
            Out ! Message(p) ;
            Idle()()
      InitialiseProducer()()
            NextIdentifier := 1 ;
            SendPackages()()
      Idle()()
            delay(IdleTime) ;
            SendPackages()()
```

The Consumer just receives Packets.

```
process class Consumer()
ports
      In
messages
      In ? Message(Packet)
variables
      LastReceivedIdentifier: Integer
init
      ReceivePacket()()
methods
      ReceivePacket()() | p : Packet |
            In ? Message(p) ;
            LastReceivedIdentifier := p GetIdentifier();
            ReceivePacket()()
```

6 Cluster

A cluster in POOSL consists of processes and other clusters and behaves as an abstraction of these.

6.1 Example

The cluster Application contains instances of Producer and Consumer and two channels: one that connects the Producer and the Out port of the cluster and one that connects the In port and the Consumer. The cluster is instantiated with a parameter for the delay between messages that is passed to the Producer instance.

```
cluster class Application(IdleTime:Integer)
ports
	In, Out
instances
	producerInstance : Producer(IdleTime := IdleTime)
	consumerInstance : Consumer()
channels
	{ producerInstance.Out, Out }
	{ In, consumerInstance.In }
```

7 System specification

The system specification defines the process and cluster instances and their communication channels. The system syntax is similar to that of a cluster one except that it is self-contained, i.e. there are no input and output ports.

7.1 Example

Our system has two instances, of the Network process and the Application cluster. For the application instance we have to specify the delay parameter. The two instances are connected by two channels.

```
system
instances
	networkInstance     : Network()
	applicationInstance : Application(IdleTime:=2)
channels
	{ networkInstance.Out, applicationInstance.In }
	{ networkInstance.In, applicationInstance.Out }
```

8 Complete code of the example

```
/* Data Classes */
data class Packet extends Object
variables
      Identifier : Integer,
      Text : String
methods
      setIdentity(I : Integer) : Packet
            Identifier := I ;
            return(self)
      setMessage(S : String) : Packet
            Text := S ;
            return(self)
      GetIdentifier() : Integer
            return(Identifier)
      printString() : String
      /* This method succinctly visualises a packet  when inspecting it */
            return("[" + Identifier printString + ": " + Text + "]")



/* Process Classes */
process class Network()
ports
      Out,
      In
messages
      In ? Message(Packet),
      Out ! Message(Packet)
variables
      p : Packet
init
      InitialiseNetwork()()
methods
      InitialiseNetwork()()
            TransferData()()
      TransferData()()
            In ? Message(p) ;
            Out ! Message(p) ;
            TransferData()()

process class Producer(IdleTime : Integer)
ports
      Out
messages
      Out ! Message(Packet)
variables
      NextIdentifier : Integer
init
      InitialiseProducer()()
methods
      SendPackages()() | p : Packet |
            p := new(Packet) setIdentity(NextIdentifier) setMessage("Message with
number "+ NextIdentifier printString);
            NextIdentifier := NextIdentifier + 1 ;
            Out ! Message(p) ;
            Idle()()
      InitialiseProducer()()
            NextIdentifier := 1 ;
            SendPackages()()
      Idle()()
            delay(IdleTime) ;
            SendPackages()()

process class Consumer()
ports
      In
messages
      In ? Message(Packet)
variables
      LastReceivedIdentifier: Integer
init
      ReceivePacket()()
methods
      ReceivePacket()() | p : Packet |
            In ? Message(p) ;
            LastReceivedIdentifier := p GetIdentifier();
            ReceivePacket()()



/* Cluster Classes */
cluster class Application(IdleTime:Integer)
ports
      In, Out
instances
      producerInstance : Producer(IdleTime := IdleTime)
      consumerInstance : Consumer()
channels
      { producerInstance.Out, Out }
      { In, consumerInstance.In }



/* System Specification */
system
instances
      networkInstance    : Network()
      applicationInstance: Application(IdleTime:=2)
channels
      { networkInstance.Out, applicationInstance.In }
      { networkInstance.In, applicationInstance.Out }
```