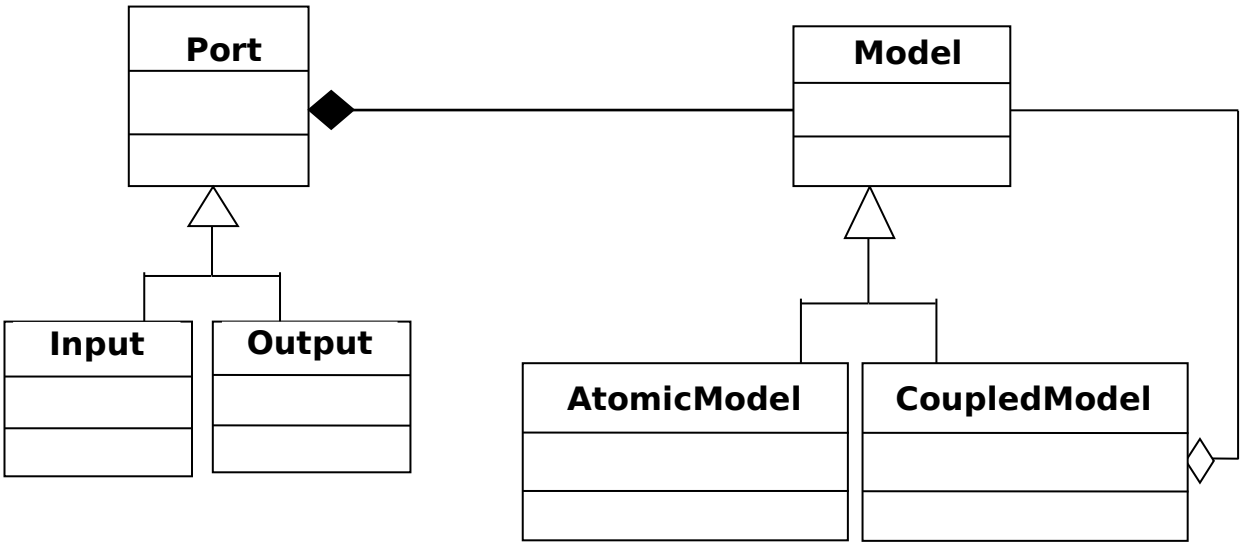
SimStudio REFERENCE GUIDE

PART I

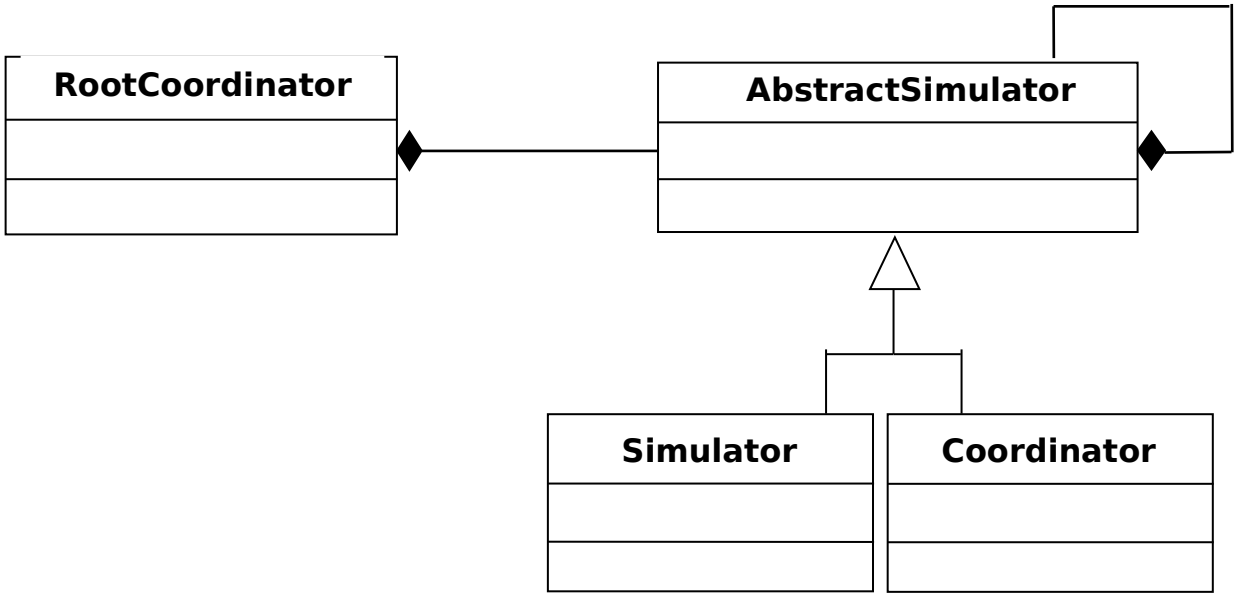
CLASS HIERARCHIES

MODEL



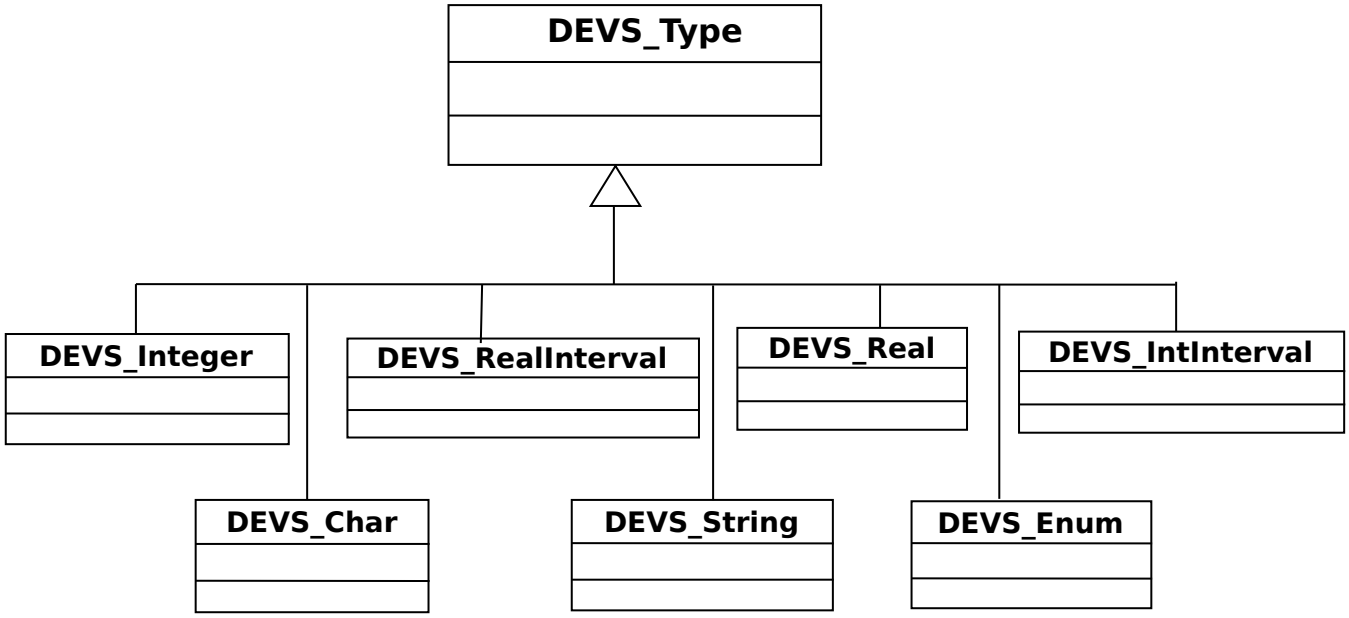
- Model and Port classes are super classes while Input, Output, Atomic Model and Coupled Model classes are sub-classes.
- Atomic and Coupled Models are abstract classes that inherit from the Model (also abstract) class and contain abstract methods.
- While the Port class is composed of Models the Coupled Model aggregates Model.
- Input and Output Classes inherit methods and attributes of the Port Class.
- Each of these classes make use of other packages in the SimStudio.

SIMULATOR



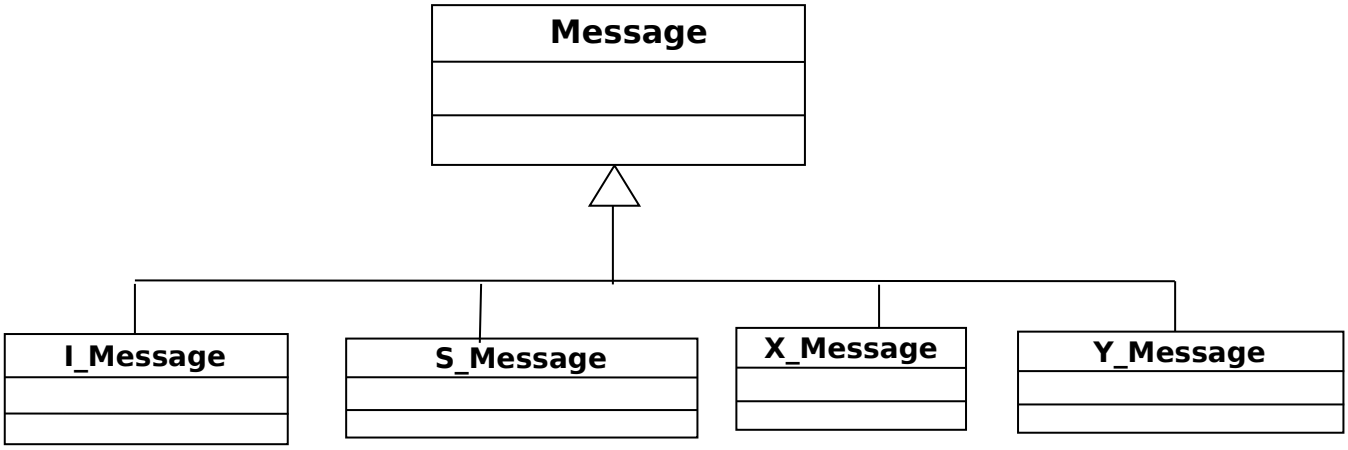
- AbstractSimulator is an abstract class that contains abstract methods and is composed of itself.
- Simulator and Coordinator classes inherits the attributes and methods of the AbstractSimulator class.
- The Coordinator class controls a Coupled Model and the Simulator class controls an Atomic Model.
- The RootCoordinator is composed of the AbstractSimulator class. It initialises and launches the simulator.
- Simulator class initialises and runs the simulation.
- Each of these classes make use of other packages in the SimStudio.

TYPES



- DEVS_Type is an abstract class that contains abstract methods. It is used to define input and output types. It is a super-class that other sub-classes inherits from.
- DEVS_Integer is a class that represents an integer.
- DEVS_Char is a class that represents a character.
- DEVS_RealInterval is a class that represents an interval of real numbers and a value in the interval.
- DEVS_String is a class that represents a string.
- DEVS_Real is a class that represents a real number
- DEVS_Enum is a class that represents an enumeration of objects
- DEVS_IntInterval is a class that represents an interval of integers and a value in the interval

MESSAGES



- Message class is used to define common specifications of the various types of messages. It is a super-class that other sub-classes inherits from.
- I_Message class contains the initialisation message.
- S_Message class contains internal transition message.
- X_Message class contains external transition message.
- Y_Message class contains output message.

PART II

MANUAL

MANUAL

The Discrete Event System Specification (DEVS) formalism provides a means of specifying a mathematical object called a model. A model is used to construct a conceptual framework that describes a system. Basically, a model has a time base, inputs, states, and outputs, and functions for determining next states and outputs given current states and inputs.

SimStudio is a simulator, implemented in Java programming language, that manages the communications between the models, manage time and uses the specifications defined by the modeller. It gives the modeller an opportunity to simulate his model using either the Classic DEVS or the Parallel DEVS formalisms. However, a DEVS model is either atomic or coupled.

ATOMIC MODEL

An atomic model describes a simple system. In addition to ports (input or output), an atomic model has, a set of states, one of which is the initial state, and two types of transitions between states: internal and external. Associated with each state is a time-advance and an output.

SPECIFICATIONS

To be able to to design an atomic model using the Simstudio package the modeller needs to provide the following information:

- the set of input ports through which external events are received
- the set of output ports through which external events are sent
- the set of state variables and parameters: two state variables are usually present, “phase” and “sigma” (in the absence of external events the system stays in the current “phase” for the time given by “sigma”)
- the time advance function which controls the timing of internal transitions – when the “sigma” state variable is present, this function just returns the value of “sigma”.
- the internal transition function which specifies to which next state the system will transit after the time given by the time advance function has elapsed

- the external transition function which specifies how the system changes state when an input is received – the effect is to place the system in a new "phase" and "sigma" thus scheduling it for a next internal transition; the next state is computed on the basis of the present state, the input port and value of the external event, and the time that has elapsed in the current state.
- the confluent transition function which is applied when an input is received at the same time that an internal transition is to occur – the default definition simply applies the internal transition function before applying the external transition function to the resulting sate.
- the output function which generates an external output just before an internal transition takes place.

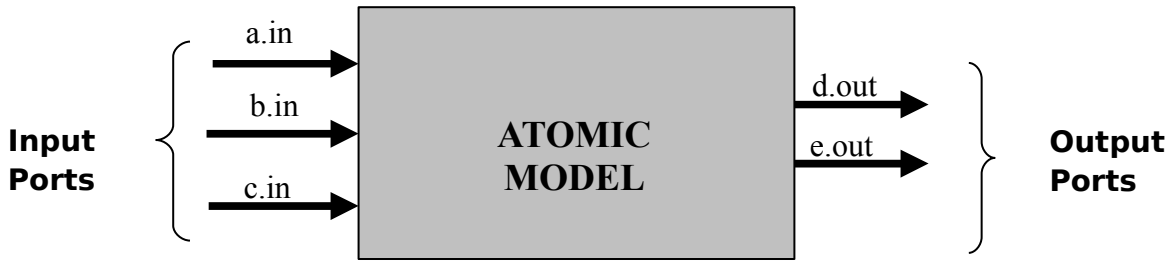


Fig 1 Atomic Model

HOW TO PLUG-IN ATOMIC MODEL SPECIFICATIONS INTO THE SimStudio

The structure below works for both CDEVS and PDEVS formalisms.

- Import the AtomicModel class from the model in the SimStudio.

```
import model.AtomicModel;
```

Any other class to be used will be done this way, for example

```
import types.DEVS_Integer;
import exception.DEVS_Exception;
```

- Give the name of the model to be simulated and extend the AtomicModel class

```
public class MyModel extends AtomicModel
```

- In the constructor specify parameters or operations and initialize them.

```
public MyModel(String name, String desc) {
        super(name, desc);
        addInputPortStructure(DEVS_Type type, String name, String desc);
        addOutputPortStructure(DEVS_Type type, String nam, String desc);
    }
```

The name and description of the model must be specified in the constructor of the super class. Also, for each port on the model, the name, description and data structure should be provided. For example

```
addInputPortStructure(DEVS_Type type, “a.in”, “Input Port”);
addOutputPortStructure(DEVS_Type type, “d.out”, “Output Port”);
```

Other types of data-structures can be used in place of DEVS_Type

- the internal transition function which specifies to which next state the system will transit after the time given by the time advance function has elapsed is defined in the method

```
public void deltaInt()
```

- the output function which generates an external output just before an internal transition takes place.

```
public void lambda() throws DEVS_Exception {
    setOutputPortData(String name, DEVS_Type type);
}
```

Usually the the value for the output port is defined in this method by specifying the name of the output port and the value. For example

```
setOutputPortData(“d.out”, new Grid(grid));
```

- the external transition function which specifies how the system changes state when an input is received

```
public void deltaExt(double e) throws DEVS_Exception
```

- the time advance function which controls the timing of internal transitions – when the “sigma” variable is present, this function just returns the value of “sigma”.

```
public double ta() {
    return sigma;
}
```

If the modeller chooses to use the PDEVS instead, the deltaconf method has to be added to the structure.

- the confluent transition function decides the next state in the case of collision between external and internal events

```
public double deltaconf()
```

Coupled Models

A coupled model is the composition of several sub-models which can be atomic or coupled. Sub-models have ports, which are connected by channels. Ports have a type: they are either input or output ports. Ports and channels allow a model to receive and send signals from and to other models respectively. A channel must go from an output port of some model to an input port of a different model, from an input port in a coupled model to an input port of one of its sub-models, or from an output port of a sub-model to an output port of its parent model.

The coupling specification consisting of:

- the external input coupling which connects the input ports of the coupled to model to one or more of the input ports of the components — this directs inputs received by the coupled model to designated component models
- the external output coupling which connects output ports of components to output ports of the coupled model- thus when an output is generated by a component it may be sent to a designated output port of the coupled model and thus be transmitted externally
- the internal coupling which connects output ports of components to input ports of other components- when an input is generated by a component it may be sent to the input ports of designated components (in addition to being sent to an output port of the coupled model)

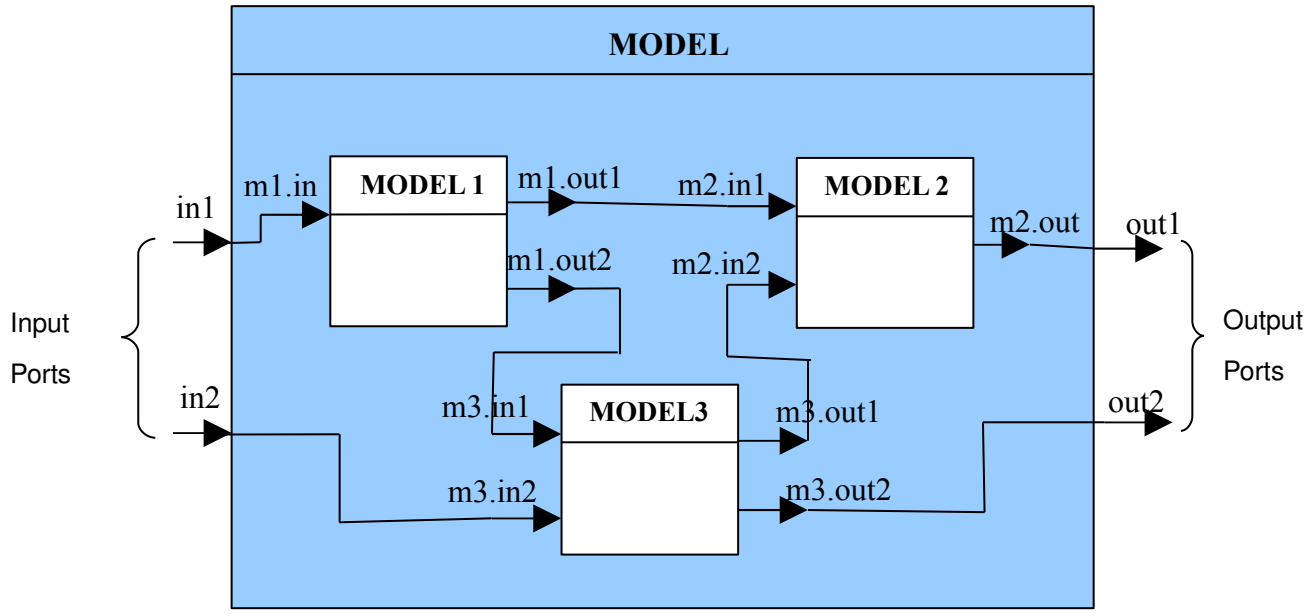


Fig 2 Coupled Model

HOW TO PLUG-IN ATOMIC MODEL SPECIFICATIONS INTO THE SimStudio

The structure below works for both CDEVS and PDEVS formalisms.

- Import the CoupledModel class from the model in the SimStudio.

```
import model.CoupledModel;
```

Any other class to be used will be done this way, for example

```
import model.Model;
import exception.DEVS_Exception;
```

- Give the name of the model to be simulated and extend the CoupledModel class

```
public class MyModel extends CoupledModel
```

- In the constructor specify parameters or operations and initialize them.

```
public MyModel(String name, String desc) {
                super(name, desc);
                model = new ModelName("ModelName", "description");
                addSubModel(model);

addIC(Model1.getOutputPortStructure("m1.out1"),Model2.getInputPortStructure("m2.in1")
);
            }
```

New instances must be created for each of the Atomic models or sub-models

```
Model1 = new Model1("Model1", "Model1's description");
```

and added using the method

```
addSubModel(Model1);
```

Using Fig 2

External Input Coupling (EIC) will be specified as

```
(in1, m1.in) and (in2, m3.in2)
```

External Output Coupling (EOC) will be specified as

```
(m2.out, out1) and (m3.out2, out2)
```

Internal Coupling (IC) will be specified as

```
(m1.out1, m2.in1), (m1.out2, m3.in1), (m3.out2, out2)
```

The EIC,EOC, IC must be specified in SimStudio using the addIC function, for example

```
addIC(Model1.getOutputPortStructure("m1.out1"),Model2.getInputPortStructure("m2.in1")
);
addEIC(Model.getInputPortStructure("in1"),Model1.getInputPortStructure("m1.in1"));
addEOC(Model2.getOutputPortStructure("m2.out"),Model.getIOutputPortStructure("out1"))
;
```

- Select method: has to be defined when implementing a coupled model. It returns a model among a list of given models.

```
public Model select(ArrayList<Model> arg0)
```

If the modeller uses the PDEVs formalism, he does not have to implement the Select Function.

START THE SIMULATOR

To start the simulator a new class has to be created

- Import the RootCoordinator class from the simulator in the SimStudio.

```
import simulator.RootCoordinator;
import exception.DEVS_Exception;
```

- Create a new class

```
public class Simulation
```

- Create a Main method.

```
public static void main(String args[]) throws DEVS_Exception
```

- Create and initialize an object of the Coupled Model
- Create an object of the RootCoordinator class and use it to initialise and run the simulator in the Main method.

```
RootCoordinator root = new RootCoordinator(sys.getSimulator());
root.init(startTime);
sim.root.run(value);
```

The startTime is the time in which the simulation should start from while value represents the number of times the simulation should run.