

Sharif University of Technology



Xmulator

A listener based, multi-layered simulator

Abas Nayebi
Sina Meraji
Arash Shamaei

spring 2006

1- INTRODUCTION	3
1-1 BASICS OF SIMULATION	3
1-2 PRIMARY CONCEPTS	4
1-3 SOME SOFTWARE ENGINEERING POINTS	4
2- XMULATOR ARCHITECTURE	5
2-1 LISTENER-BASED INTEGRATION	5
3- BASIC CLASSES	9
3-1 XOBJECT	9
3-2 XEVENT	9
3-3 XENGINE	10
3-4 BUFFER	10
3-5 METACOMPONENT	10
3-6 MIDDLEPORT & ENDPOR	11
3-7 BASEXEVENETGENERATOR	12
3-8 TIMEINTERVALXEVENTGENERATOR	12
3-9 BASENODE	12
4- INTERCONNECTIONNETWORK PACKAGE	12
4-1 INTERCONNECTIONBUFFER	12
4-2 INPUTVCBUFFER & OUTPUTVCBUFFER	12
4-3BASEMESSAGE	13
4-4 PHYSICALCHANNELSENDER & PHYSICALCHANNELRECEIVER	13
4-5 SWITCH	13
4-6 NODE	13
4-7 VCARBITER	14
4-8 INJECTIONCHANNEL	14
4-9 PE	14
4-10 FLIT, HEADERFLIT, HEADERFLITSINGLE	14
4-11 SWITCHINGINFO	15
4-12 MESSAGECONSUMER	15
4-13 PHYSICALEJECTIONCHANNEL, PHYSICALINJECTIONCHANNEL	15
4-14 RE	15
4-16 SIMPLEMESSAGEGENERATOR	16
4-17 NETWORKGENERATION PACKAGE	16
5- QUEUING NETWORK PACKAGE	18
5-1 A SIMPLE EXAMPLE	18

1- Introduction

1-1 Basics of simulation

There are three general methods for simulation:

Time quantum method

In this method, we divide the time to small intervals and then we run the program. the problem with this method is that if we suppose the time intervals too small, accuracy is increase but the simulation time is increased too, and if suppose the time intervals large then accuracy is decreased. Another problem with this method is time continuity in simulation so we have to convert the real time to quantized discrete time and actually, we lose the time accuracy. We can use this method when the system is synchronous with a single clock and all the activities run at the beginning of time intervals.

Process based method

In this method, each part is model with a process. It has the benefit of easy to define the parts behavior and its problem is low performance.

Event based method

In this method, we cannot set a thread for each real component of system and as a result, we must simulate the behavior of system by saving the events.

The problem with simulation programs is the single thread that must simulate the total behavior of the system. This means that every time that an event happened, it runs the process of that event and than again return to its run. Regularly when an event is processed, it makes few events for next simulation time steps.

As an example suppose we want to simulate the behavior of a message generator. One solution is to assign a thread to it that generates messages, puts them on the network, and than waits for one second (one simulation step) and than generates the next message. Nevertheless, we cannot assign a thread for a generator. So we suppose that an event must be happen at time zero in order to declare a message must be create at time one. simultanesly with the process of this event another new event must be create at first

second that declare the generation of the next message at the beginning of the second second. Hence, at the event-based method, existence of the event at an special time for each component means that the simulator must call that component at that special time.

1-2 Primary concepts

Generally, we can suppose that we have some events, components, and one simulation engine. This engine contains the queue of events. The structure of the simulator engine is a while loop in a manner that the event at the beginning of the queue is processed with the engine and next events are processed in turn. Each event is the source of the some new events and as matter of fact, the queue of the events in the simulator engine never finished. An easier engine structure leads to a modular program. The engine does not need to know the events type and their work. It is sufficient to know which event belongs to which component.

The queue of events is sorted with the time of events. The data structure to save this queue is so important in performance evaluation. Arrays and linked lists have lower performance than AVL trees. In this simulator, we use the Red black tree, which is a special case of AVL trees. The events are saved in this tree. Each part of the tree contains a queue of events that have the same occurrence time. The events that must be run at time 50 run in parallel in real word. In fact, order of their run must have no effect on simulation result, and conceqently the events with same occurrence time must be saved in one queue. One type of Red Black tree is shown in Fig.1

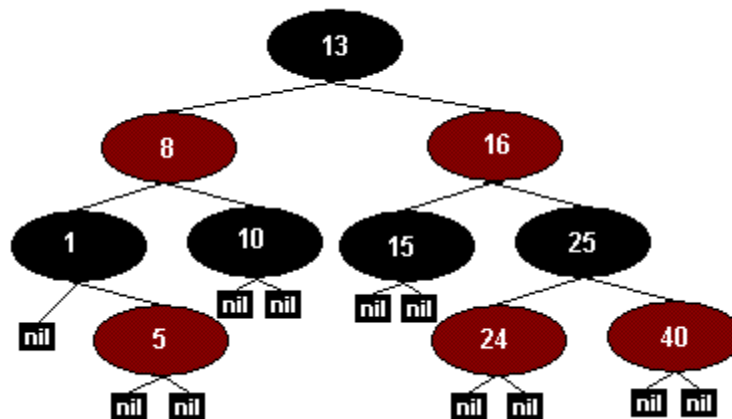


Fig.1: An example of Red Black tree

Before describing a simulation example, we must attend to some software engineering points.

1-3 Some Software Engineering Points

In software engineering, we must pay attention to some layering rules for a software design. Each layer consists of some packages that each of them contains of some classes. In C#, each package is determined with a namespace. The order of layering is very

important in software engineering, because each layer can know about the lower layers and use them when it is essential. A lower layer must never inform about the higher layers. In a single layer, the packages must have no horizontal two-sided communication. As an example if we have packages A and B in a layer in which A needs B classes and B needs A classes also, we do not follow design rules, and the first problem is accrued in project compile time.

2- Xmulator Architecture

Fig. 2 shows some layers and packages in the current simulator. As we can see in this figure, we follow the layering rules. As an example, the engine that is from the Base Component package must not be informed from the message generator in Queuing Network package, which means that the simulator engine code must be dependent from the message generator code.

2-1 Listener-based integration

An example:

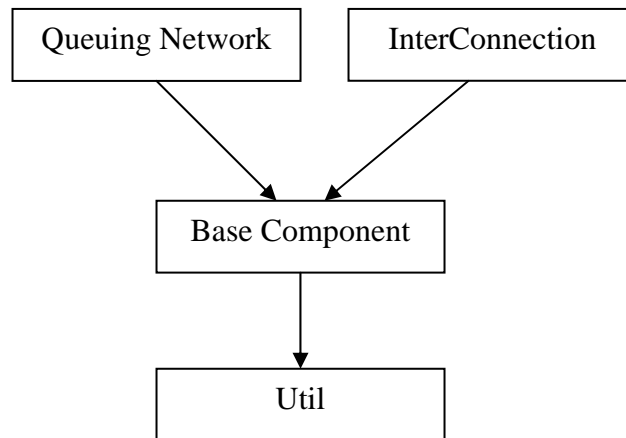


Fig. 2: some layers and packages in the current simulator

We have an XObject class in Xmulator which is the father of all base components. These components are model for elements that are in real world with their own behavior and all of them create their own events. XObject is in Base Component layer and have two base methods named START() and ProcessXEvent(). Each component which is inherited from the Xobject and created, register itself automatically with RegisterXObject() method in simulator engine. As matter of fact simulator engine have all the components list and before the start of simulation it call the Strat() method of all these components. The engine sees all these components as same XObject so it does not know about the type of different registered components.

Suppose a system consist of a message generator, a queue, a server and an engine. The message generator generates a message per second constantly and put it in the queue. The process time of a message is two seconds in server. At first, the engine calls the start

method of all components. The start method submits the first event from the message generator in first second in queue engine. This event is responsible to create a message in this second. Fig. 3 shows the system status in zero second.

Time = 0

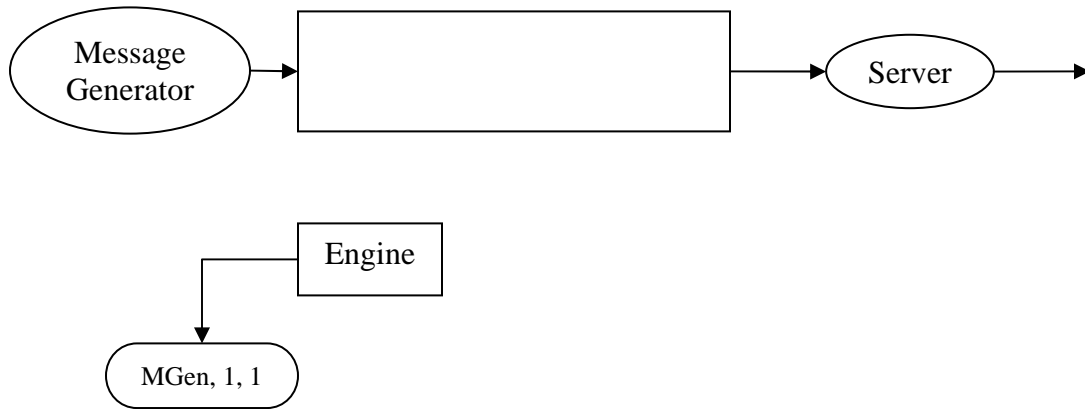


Fig. 3: system status in zero second

We do not have any work to do in zero second so the time is going forward. In this second, the second method of message generator (ProcessXEvent) which inherited from XObject is called with engine. With calling this method a message is generated with message generator and put in the queue. In this second the message generator register a new event to produce a new message in second second. With putting the message in queue a method named Enqueue() is called from the queue. The start method from the server is called because the queue is empty. The generated message is go to the server to run. An event registered at time tree because the run time for the server is 2 seconds. The system status in first second is like Fig. 4.

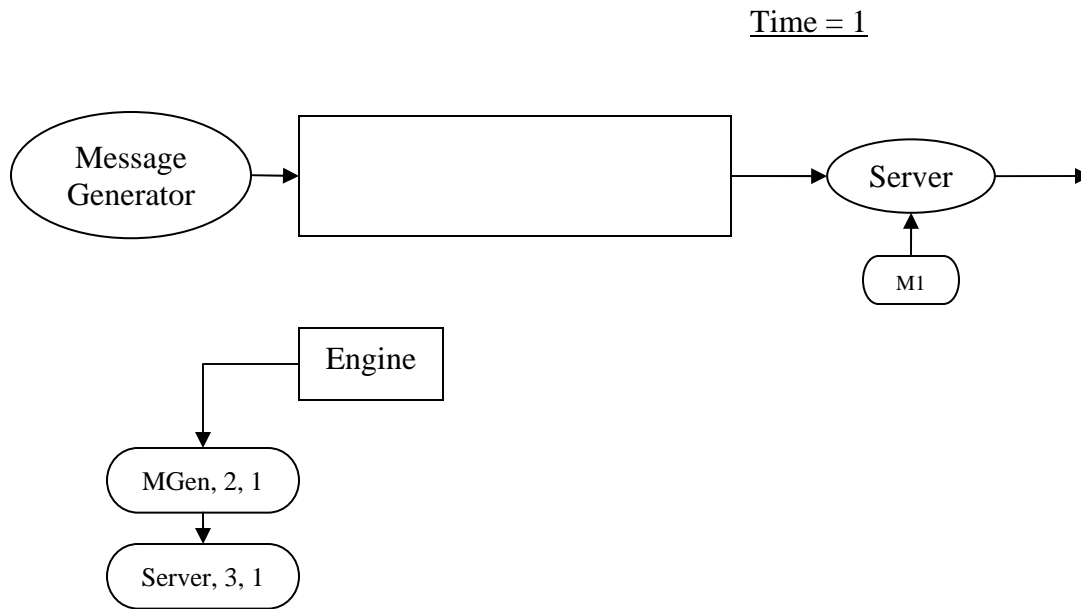


Fig. 4: system status in first second

Like this, system status is shown in times 2, 3, 4.

In third second, the event of service finished is called from the server and the message from the server is send out. The server must check the queue in order to run the next message and if there is a message in queue run it. The queue must inform the server when there is a message on it. This consistency between server and queue is not allowable in software engineering now a day, because we must now the queue code when we write the server code and vise versa. In another side if we want to write a new version of queue, it may inconsistent with server and we had to rewrite the server.

To solve this problem we use the delegate conception. In Xmulator simulator, we have two kinds of events. First types are events that are used to move forward the simulation time. This branch is XEvents that are declared in the next part.

Second type is events that used to make a connection between components. These are delegates in C# programming language. As an example suppose an empty buffer in which components connect to it like server are wait to receive the first message. In this case the buffer make an event of second type named OnFirstSlotFilled() and the components that

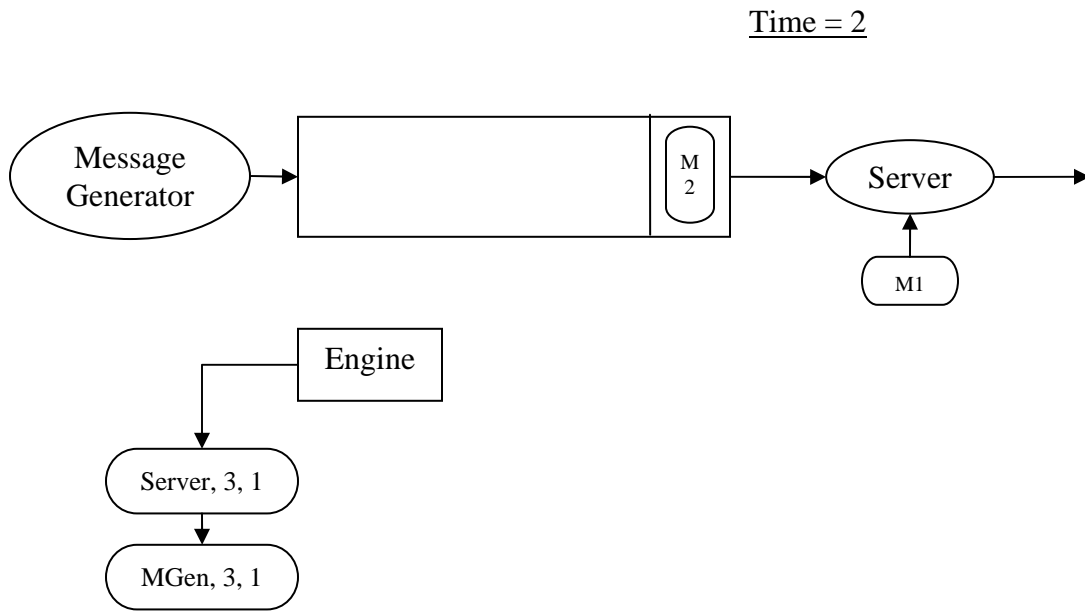


Fig. 5: system status in second second

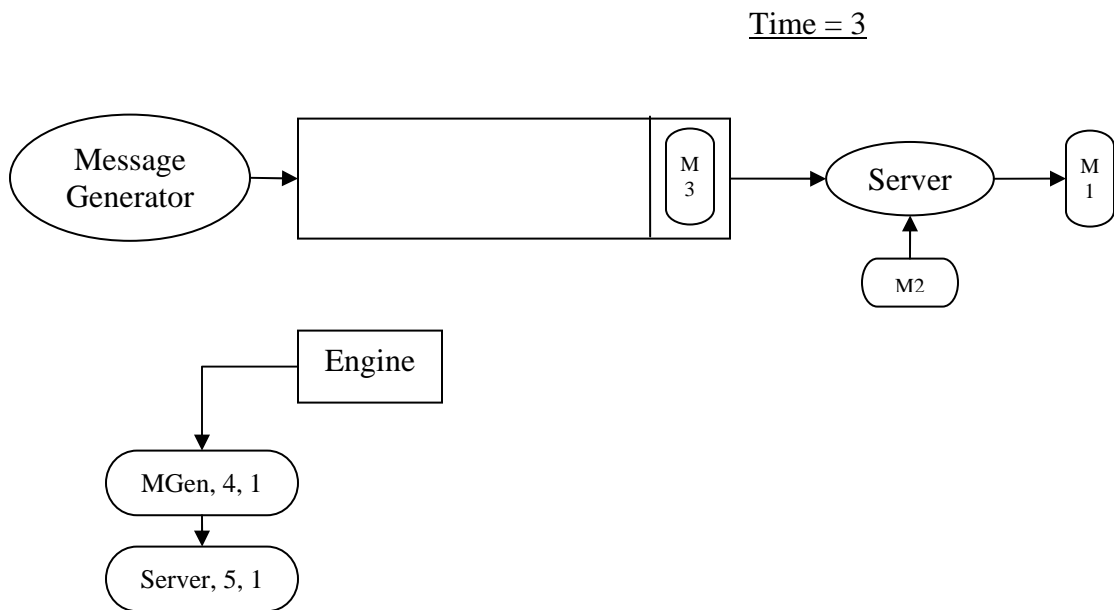


Fig. 6: system status in third second

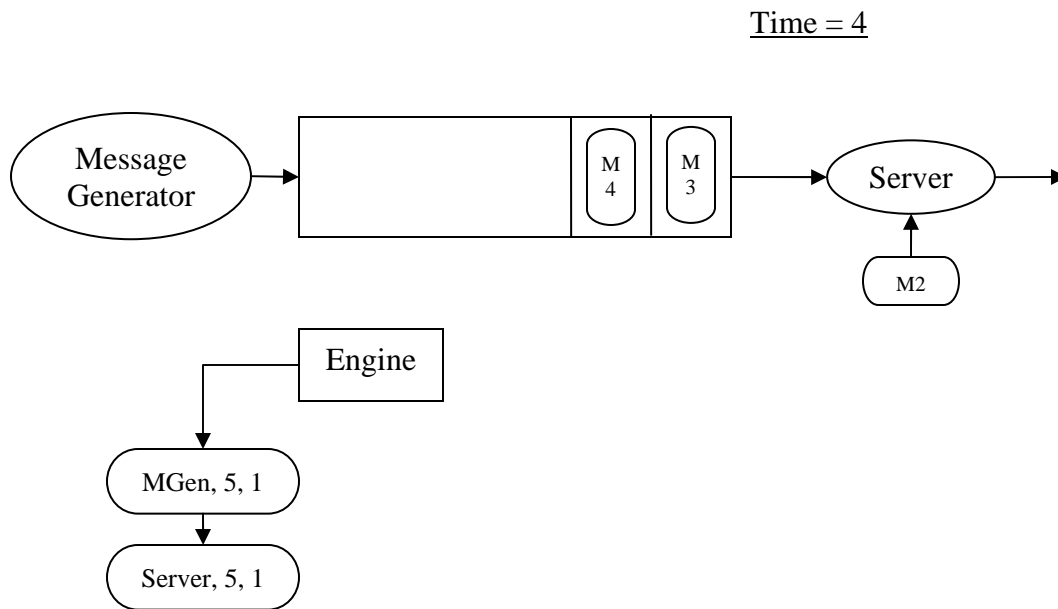


Fig. 7: system status in forth second

Connect to buffer listen on this event. When a message is put on the buffer the On FirstSlotFilled() method of that buffer is called and this method called the other components that wait for it.

3- Basic classes

3-1 XObject

This class is the father of all classes and has the following methods:

Start(): if we had any preparation we done it with this method. This method is called before simulation time.

RegisterXEvent(): every time we want to register an event in the engine we use this method.

ProcessXEvent(): when an event is registered in the engine with RegisterXEvent method, the engine call the processXEvent method to run the event on the happening time.

ResetState():with this method we can reset the network without any reconfiguration.

3-2 XEvent

As mentioned previously Xevents are events that used to move forward the simulation time. These events have the following attributes

Time: the time that the event must be run at that time.

TargetXObject: An XObject type object that the event must be run on it.

Parameters: A variable that points to a subject to the object on the event running time.
Type: every object can have many types of events. These types are shown with a variable named type, which is a real number. The important point is that the event type is known for object not for engine.

3-3 XEngine

RegisterXEvent(): The most important function of engine that used to register the events

Now(): It shows the virtual time of simulation.

RegisterXObject(): when an XObject is created its constructor is registered in engine with this method automatically.

StartSimulation(): It uses in order to start the simulation in a time period or known number of Xevents.

Point: To access a type of XEngine, which is common between all classes, we use XEngineFactory class.

3-4 Buffer

A basic component, which is derived from XObject, it has the XObject methods and also the following methods:

IsFull(): it set if the buffer is full.

IsEmpty(): it set if the buffer is empty.

OnLastSlotFilled(): it called when the last buffer place is empty.

Enqueue(): it uses to put an object in buffer

Dequeue() it uses to get an object from the buffer.

More than these methods, the buffer has two following attributes:

Count: it shows the number of objects inside the buffer.

MaxSize: it shows the max size of the buffer.

3-5 MetaComponent

Sometimes, it is necessary to put some objects together and make a new component. For example in a three dimensional mesh it may be necessary to suppose the nodes of each plane as a component, this is done with MetaComponent class.

This class is an XObject which consist of some other XObjects. Each MetaComponent has some links, which are used to make connection between different MetaComponents.

For example to make a five dimensional hypercube we can do it like this:

Each processing node is a metacomponent which consist of RE, Switch,... when we connect two Meta components we make a new metacomponent which is named one dimensional hypercube. Like this, we connect two metacomponents of type one-dimensional hypercube to make a new metacomponent, named two-dimensional hypercube. If we continue this, we can make a five dimensional hypercube with connection of two 4 dimensional hypercube metacomponents.

3-6 MiddlePort & EndPort

Each XObject consist of some EndPorts. These ports are used to make connections between components. Each MetaComponent has some MiddlePorts. Fig. 8 shows different type of connection between components.

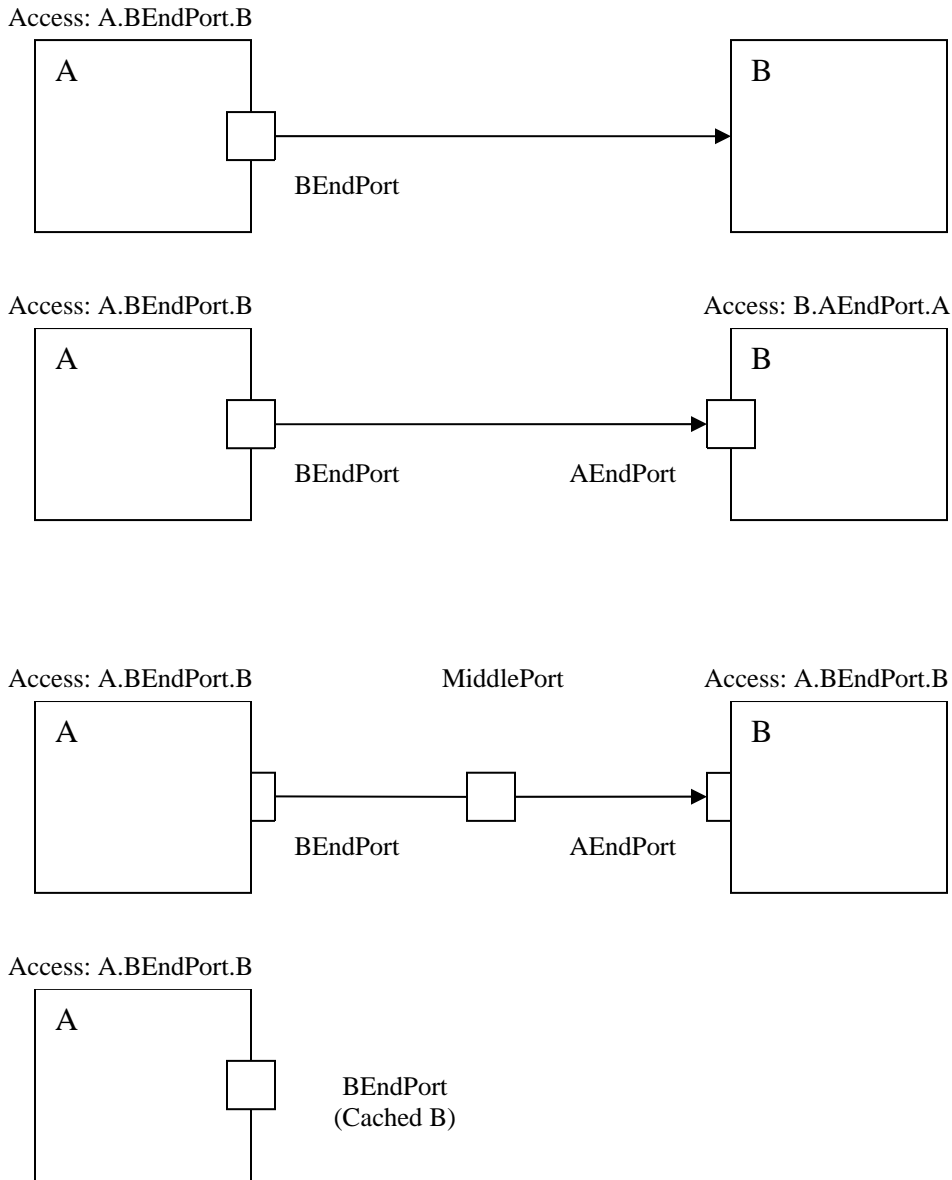


Fig. 8: different type of connection between components

3-7 BaseXEventGenerator

This component makes Xevents. It has the Xobject methods and one more method named OnXEvent() which is called when an event is created.

3-8 TimeIntervalXEventGenerator

It is a component, which is derived from BaseXEventGenerator component. This method is used to make Xevent on known periods. This time periods can be Exponential, fix, or any statistical distribution. The GenerateNextXEvent() method is used to make the next event.

3-9 BaseNode

It is a basic class, which each network node inherits from it.

4- InterconnectionNetwork Package

4-1 InterconnectionBuffer

It is a component, which is derived from the Buffer class.

Switched: It is an attribute which shows that the buffer exchange information or not.

SwitchingInfo: It shows that which buffers are connected to this buffer, the switching information is within the buffers because this is decrease the communication time.

OnTheFlyFlit: if we suppose the entire communication time one time quantum, we have one bubble between every two flits. Suppose a physical channel with both input and output buffers full. The channel must wait for its output channel being empty in order to accept the input buffer flits. This is due to send an empty packet between every two packets. To solve this problem, every time that we have data on input buffer of channel, on the next clock pulse the data will put into the OnTheFlyFlit and the input buffer of the channel is being empty.

4-2 InputVCBuffer & OutputVCBuffer

Both are special case of Buffer class. They feed the input and output of the virtual channels.

OnHeadInFirstSlot(): every time that the flit which is put in the first slot is a header flit, this method is called. The router is waiting for this method, and after this, it began to routing.

4-3BaseMessage

It is a basic class to make the message. It has an important property (ID) which is inherited from the XObject. A message is recognized with the ID during the reporting time and debugging.

GenerationTime: It shows the generation time of the message.

SingleMessageHeaderFlit: It is a message that contains a single header flit.

IsTail: when the last flit of a message is passed the switch. The switch must open the way for passing the next message. The IsTail Property shows the last flit of a message.

4-4 PhysicalChannelSender & PhysicalChannelReceiver

Each physical channel is split into two parts: a sender and a receiver. So in order to make a physical channel we must connect these two parts to each other. The connection of these two parts and making a physical channel is done with the properties physicalChannelReceiverEndPort and PhysicalChannelSenderEndPort.

ChannelDelay: It is the time that the message is transfer on the physical channel.

Deliver(): We send a message with this method on the physical channel.

OnDeliverd(): When a message reach the tail of the channel this method is called.

IsBusy(): It shows if the channel is busy or not.

PushFlit(): It get the data flit from the virtual channel and put it on the physical channel.

4-5 switch

Each switch has some sender and receiver physical channels that connect to it with an array of type two EndPorts. To access a special physical channel like channel 5 we can use the following code.

```
Sw.outputPhysicalChannelEndPort[5].PhysicalChannelSender
```

The ResetDelegate() method of this class listen on the status of these two channels InjectionChannel and EjectionChannel, and reset the delegates when the first slot of a channel is filled (OnFirstslotFilled) or the last slot of a channel is being empty (OnlastSlotFreed).

LockSwitch(): It hang the switch in order to transfer the message within it.

UnLockSwitch(): It open the switch in order to receive next message.

SetOutputChannel(), SetInputChannel(): With these two methods the input and output channels of the switch are set.

OnUnlockswitch(): when the switch is opened this method is called.

4-6 node

Making a node is using a special hierchal order. This is so important in logging time with XML. Each node composed of a set of components like EjectionChannel, InjectionChannel, RE, Switch The node class gathers the information of a node. Each node has a PeEndPort and an ReEndPort that can connect to PE and RE with them. As you can see in the main code (Node.cs) these two variables are sampled at the beginning

of the class code of node. Class node has three constructors that the constructor without any parameters is used to register the information in the XML.

Each class has a method named Validate() that its work is to analyse the condition before the simulation. For example in node class, if we have a node without Pe or Re the simulation must not be start and this is done by making an exception.

Node class has a property named switch that point to the switch variable of the Pe class, and used for easy to write. More than this there are two more properties named Pe and Re that are used to make a connection between PE and RE.

4-7 VCArbiter

It is a component that connects to the physical channel and put the messages of the virtual channels that connect to a physical channel on it periodically. This is done if the destination buffer of the message is empty otherwise, the message is blocked in the virtual channel. The arbitrate method of this class is used for this. This method called the deliver method of the PhysicalChannelSender if the condition for the communication is ready. The Arbitrate method is called in three situations: when a message delivers to destination, when the first free slot of physical channel is filled and when the last filled slot of a physical channel is freed.

4-8 InjectionChannel

This component has some input buffers of type InputVCBuffer and SwitchVCBuffer. These buffers are come from the switch side. We have another input buffers named PEVCbuffer which are supported with PE.

4-9 PE

The PE¹ component has a message consumer that can be available or unavailable. This message consumer is inherited from the baseMessageConsumer class which has a method named Consume(). This method is used for statiscal information gathering.

Each message has two times: generation time and network entrance time. With these two times and Consume() method we can calculate all the delays for a message.

Each PE has a messageGeneratorEndPort and a NodeEndPort that a messageGenerator and a node connect with them. Each of these two components has a PeEndPort, which is used to make a connection. This is a second type connection in EndPorts. The important point in Validate() method is that if PE has no generator or consumer an exception is accrued, but if the node do not have them the simulation is stopped.

4-10 Flit, HeaderFlit, HeaderFlitSingle

A class that is inherited from the TransferUnit class. The HeaderFlit class show an special type of flit named header flit, and HeaderFlitSingle show an special type of header flit which is a one flit.

¹ Processing Element

4-11 SwitchingInfo

Each buffer of InterconnectionBuffer type has some switching information, which is from the SwitchingInfo class. This class shows that buffer put in which physical channel and what is the virtual channel number of that physical channel, and also it is evident that the buffer is switched to which output channel.

4-12 MessageConsumer

It is a class which is derived from the class BaseMessageConsumer. This class is used for statistical operation. It has a variable named warmUp which shows the end of transient state in system. After warmUp time the system enter the steady state and it is evident that the different statistical parameters like average message latency, ... are important in system steady state time. The variables that end with the W show the number of that parameter after the warm up time. There is a method named consume() which is used for statistical operation. If the transfer unit entered the node is the last message flit, the message must be consumed, so the consumed property of this class get the true value.

4-13 PhysicalEjectionChannel, PhysicalInjectionChannel

During the connection of a switch and PE sometimes we use a real physical channel, in this case we use two above classes.

4-14 RE²

Re is one of the most important basic classes. Each RE is connected to three components of type Node, Switch and routingFunction. This connection is done with second type EndPort. The RoutingFunction class has a method named Route() that must be rewrite for different routing algorithms. RE has a queue of blocked messages that checked with Round Robin method and every time that a message can be routed, exit from the blocked messages queue, this is done with CheckBuffer() method.

4-15 RoutingFunction

It is a class that has only one method named Route() and used for routing. The needed information for Route() method are: current node address, header flit(in order to access to message), input buffer(in order to get the number of input and output virtual channel) and switch(in order to get the information of output physical channels).

² Routing Element

4-16 SimpleMessageGenerator

It is a class, which is derived from the BaseMessageGenerator class. This class generates the messages according to message length distribution, destination distribution and the time between to generation time distribution. As shown in the code, these three distributions are connected to this class with second type EndPort. The GenerateMessage() method is responsible for generating the message according to three above distributions.

4-17 NetworkGeneration Package

This package is a subset of main package, InterconnectionNetwork. It has a class named NodeGenerator that is due to connect a processing node parts. Each node is made with connecting different components. Fig. 8 shows how these components connect to each other.

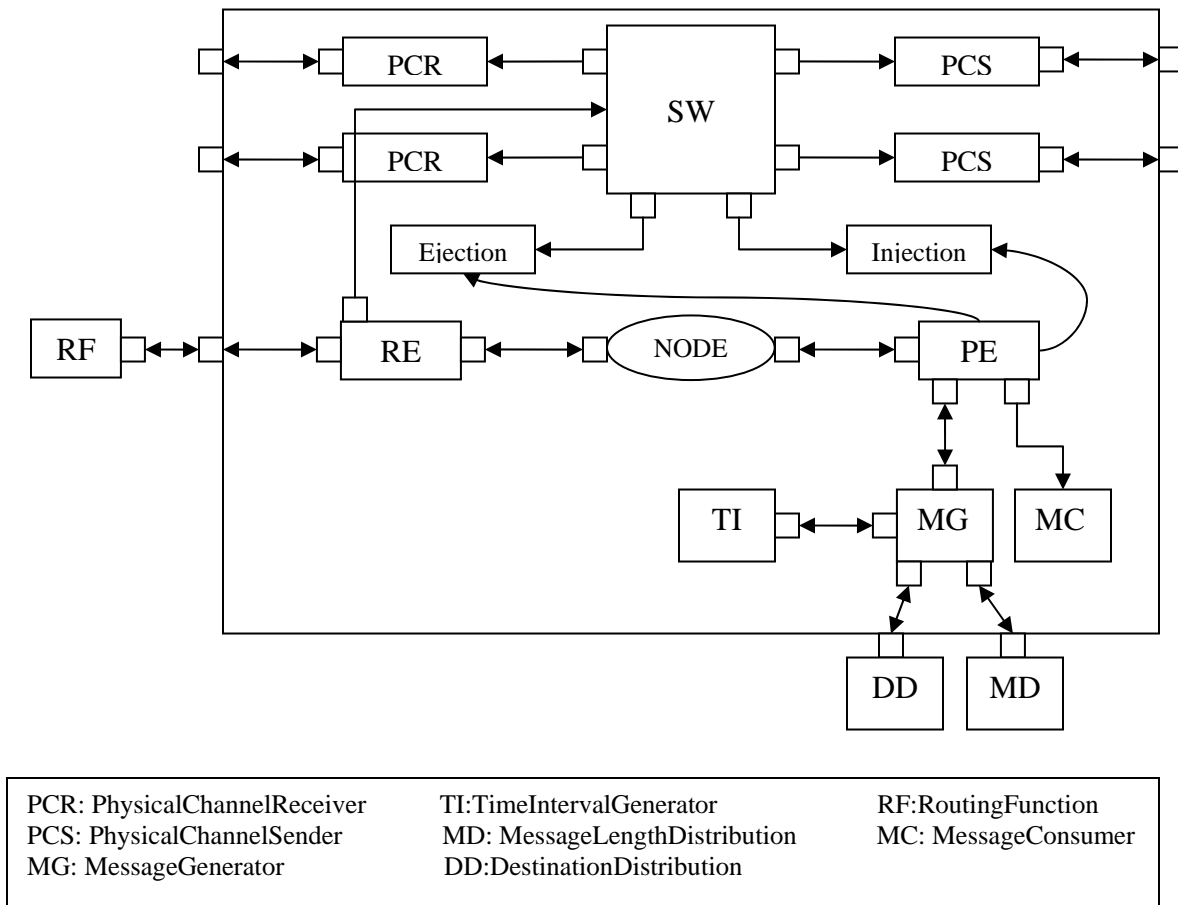


Fig.8: Connection of different parts of a node

Each node is a metacomponent. The BuildSimpleNode method is derived from the MetaComponent class and its work is to connect different parts of a node. As you can see

in the code of the program (NodeGenerator.cs) different components are generated and added to processing node with addXObject() method. When a new component is made the constructor function of that class with no parameter is called and in next lines its parameters are valued. These are done in order to write the events in XML. In order to make the connection between components we use Endports and Middleports and these connections are made with AddAndConnectLink() method.

5- Queuing Network Package

We present the queuing network package with a simple example

5-1 A Simple Example

In this part, we explain the implementation of an M/M/1 queue with XMulator. M/M/1 queue is a queue that the clients entrance and they time duration is according to the exponential distribution and there is only one server. The base components of this system are: Jobgenerator, Queue, Dispatcher, server and the main engine. The connection of these components is shown in Fig. 9.

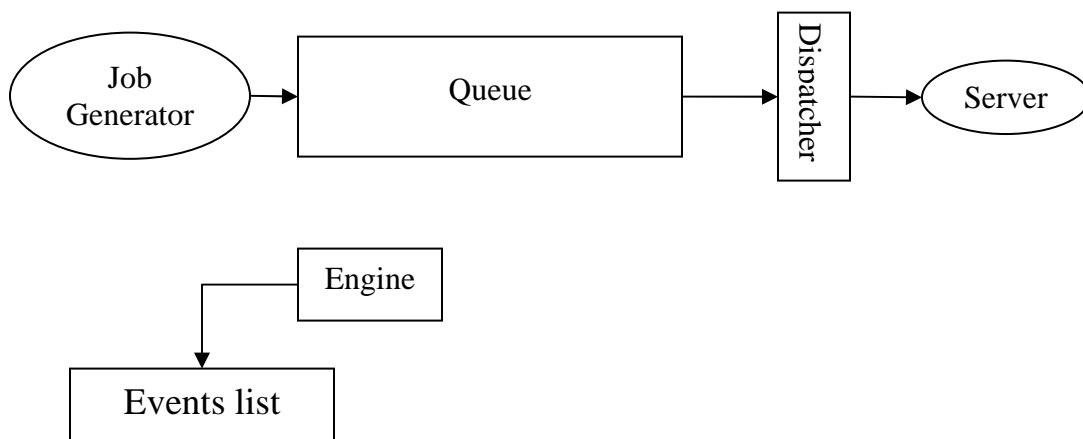


Fig. 9: Connection of an M/M/1 queue parts

At first, we make a new project named SimpleQueue of ConsoleApplication type. The components of this example use the BaseComponent and queuingNetwork classes so we add these two projects to following example with right click on the project name in SolotionExplorer part. We use the classes of these two projects in the program, so we add the name of these two classes at the beginning of the main project file (claas1.cs):

```
Using Xmulator.BaseComponents;  
Using Xmulator.queingNetwork;
```

Now we make all the components. At first, we make a message generator with the following instruction:

```
JobGenerator jobGen =new JobGenerator("JGen",evGen,q1);
```

JobGenerator is a class from the QueuingNetwork package. jobGen is a new object which is samppled from this class. JGen show the identifier of this object. Each message generator must has an EventGenerator, evGen do this in this example. The consumption

place of these events is shown with q1. evGen is an object from the BaseXEventGenerator class which is inherited from the ExpTimeIntervalGenerator class. This is done with the following instruction:

```
BaseXEventGenerator evGen = new ExpTimeIntervalGenerator("EGen", 1 / arrivalRate, 1234);
```

There is a base class named ExpTimeIntervalGenerator which three following class are derived from it: FixedTimeIntervalGenerator, TimeIntervalGenerator and ExpTimeIntervalGenerator.

We can add new statistical distributions as new classes of type BaseTimeIntervalGenerator to program code. In this example, we use the exponential distribution. In this distribution, the arrival rate is equal to λ and its reversal shows the average of exponential distribution. As a result, a variable named arrivalRate is used for exponential distribution.

The number 1234, which you see here, is used to generate random numbers with random generator. If this number is changed, the random generated numbers are changed also. This is evident that if this number is equal for message generators, all of them generate the same message at same times that is far from the reality. As matter of fact this number must be different for different message generators. The result of running the above line is generating a set of XEvents at time periods with exponential distribution, and we have a poison process.

During the sampling from the jobGen object, we use the q1 constructor. We have two interfaces named ObjectSource and Objectsink. ObjectSource is a class with GetObject() method and its rule is generating transfer units. ObjectSink is a class with PutObject() method and its rule is to get transfer units. We can connect the components with these two classes easily.

In part three of object jobGen we need an ObjectSink to receive the generated messages. This object is a queue that was shown with q1. Consequently we must built the q1 at first this is done with the following instruction.

```
Xmulator.QueuingNetwork.Queue q1 = new Queue("q1", 100, warmup);
```

C# has a class named Queue, in order to avoid mistake we show it with Xmulator.QueuingNetwork.Queue. The number 100 shows the quantity of the queue. The warmup variable shows the time that the components begin their work and reach the determined average. As an example, a queue is empty at the beginning and sometimes left to reach the desirable limit, this time is named as warmup time.

Now we want to create a message distributor that its work is to get the message from the input (ObjectSource) and distribute the message to outputs (ObjectSink), the following instruction show this:

```
Dispatcher disp = new Dispatcher("disp.", q1, new IObjectSink[] {server});
```

With this instruction an object named disp from the dispatcher class is sampled with disp identifier. As the source of this object is a queue, we put at ObjectSource part the queue name, q1. The ObjectSink part has an array of servers, that in this example we have only one server because of the nature of M/M/1 system. This server is sampled from the Server class with the following instruction:

```
Server server = new Server("Ser.", 1, dist, warmup);
```

The identifier of this object is Set and the number 1 shows the capacity of the server. The warmup variable is like previous. The only new point is the distribution function that was constructed with the following instruction:

```
RealDistribution dist = new ExpDistribution("dis", 1.0 / serviceRate, 234);
```

dist is an exponential distribution which is sampled from the ExpDistribution class. Its identifier is dist and its average is the reverse of service rate. The number 234 like previous determine the domain of random numbers.

In this way, all the components are made and connect to each other. Now we must write an instruction to start the simulation, the following instruction does it:

```
XEngineFactory.XEngine.StartSimulation(-1, 1000000);
```

The number 1000000 shows the number of steps that the simulator must be passed. Now if we run the program the simulator is run successfully. Before describing the output format, we explain the log4net package:

The log4net is a package for reporting of C# programs (like Log4j in java). With this package, we can show the layers of reporting, this is mean that we can determine which details are recorded in the report, and also determine reporting locations. In order to use this package we must declare it at the beginning of the program code with the instruction:

```
Using Log4net;
```

Then we must run the Configure() method of this package at the main() method of the program. The following lines must be added to the program:

```
StaticSettings.DisableLogs = true;
```

```
log4net.Config.DOMConfigurator.Configure();
```

Different outputs can get from this package. For example suppose we want to see average length of queue, this can be done with the following instruction:

```
Console.WriteLine("Average Queue Length = " + q1.probe.AvgQueueLengthW);
```

As you see the AvgQueueLengthW method calculate the average length of the queue (after warmup), this method is one of the probe class methods.

6- Conclusion

Here, we introduced a simulator based on listener-based integration mechanism which has a great impact on extensibility of the software. Mixed-mode event processing improves the performance of the simulator. By decoupling individual parts of the code, Xmulator enables independent code development and creates a flexible and extensible environment for different aspects of network design. This simulator extensively uses XML format for defining topologies, parameters, and outputs, which leads to more level of flexibility. To the best of the author's knowledge that is the first simulator that is able to simulate any arbitrary topology of interconnection network in presence of faults. Many features can be added on this simulator. The ability to simulate the sensor networks is the current project that the authors work on it. Bulding a GUI that helps the users to simulate their networks easily and make the Xmulator more user friendly is another aspect that will be addressed by authors.

7- References

- [1] Tutsch, D.; Brenner, M. 2003, .MINSimulate . A *Multistage Interconnection Network Simulator*. In 17th European Simulation Multiconference: Foundations for Successful Modelling & Simulation (ESM'03); ottingham, SCS, 211.216.

- [2] Ewing, G.; Pawlikowski, K.; McNickle, D. 1999,.Akaroa2: *Exploiting Network Computing by Distributing Stochastic Simulation*. In Proceedings of the European Simulation Multiconference (ESM'99); Warsaw, SCS,175.181.

- [3] A. Agarwal. *Limits on interconnection network performance*. IEEE Transactions on Parallel and Distributed Systems, 2(4), October 1991.

- [4] Sang Gue Oh, Cheol Min Hwang, *Object Oriented Parallel Architecture Simulator*, ACM Proceedings of the 8th international conference on Supercomputing, January 2000