Document Information

Title:	Implementation of a Distributed and Simulation Modeling Project

Authors:	(i) Syed S. Rizvi
Email:	srizv004@odu.edu
	(ii) Adam Livingston
Email:	alivi001@odu.edu

Date:	April 09, 2004
Status:	Tentative
Report:	Distributed Simulation ECE 748/848

Electrical & Computer Engineering Department
Old Dominion University
Norfolk-VA

1. Test Plan

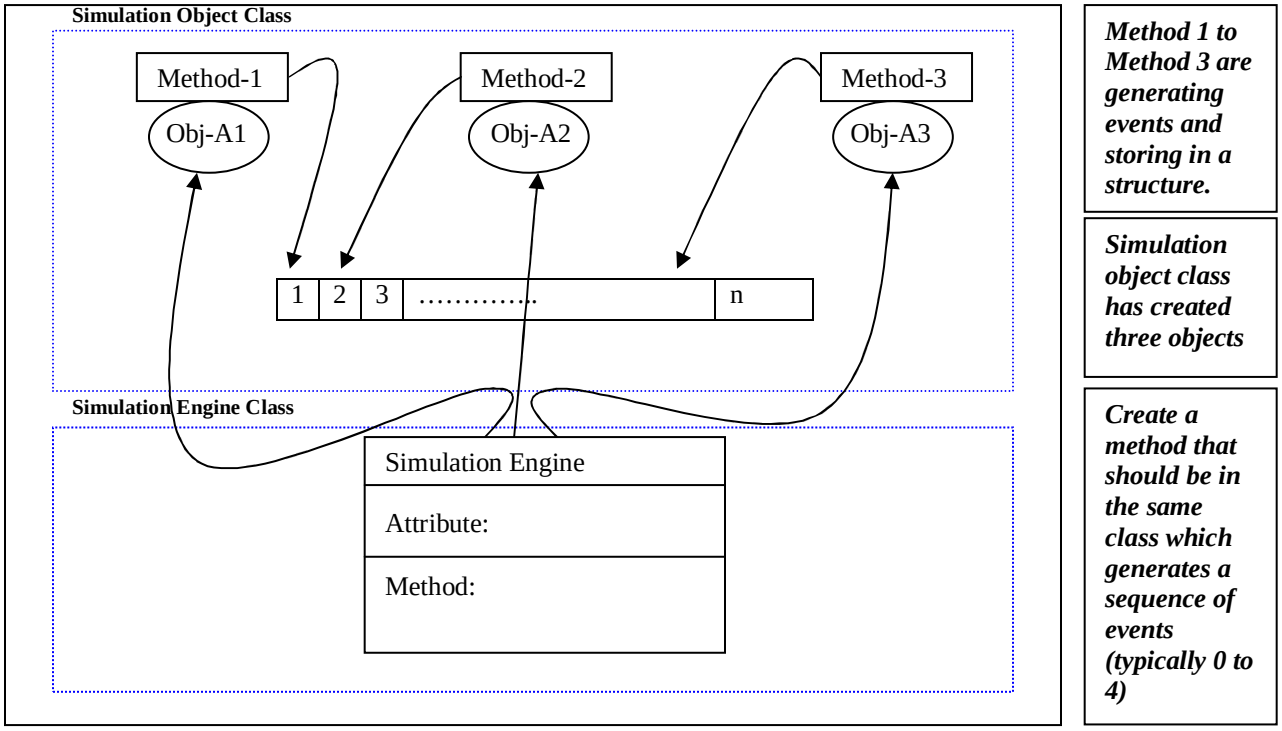
The test plan is divided into the following two sections: *non-integrated simulation model* and an *integrated simulation model*. The non-integrated simulation model deals with an individual simulation model that generates, manage, schedule, and execute events for a single simulation engine. On the other hand, the integrated simulation model deals with multiple simulation engines (typically three for our project) to generate, manage, schedule, and execute events.

1.1. Phase I: Non-integrated simulation model

We divide this phase in two major parts. One describes object creation with respect to simulation object and simulation engine classes.

1.1.1. Objects Creation

Figure 1 provides an overall functionality of a non-integrated simulation model.

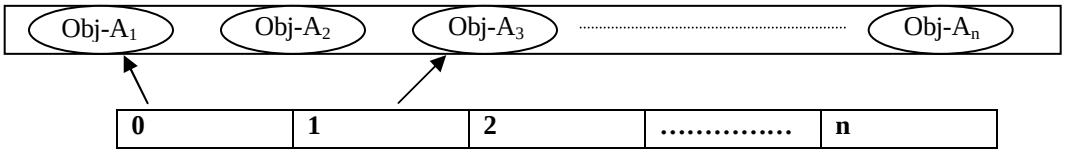


1.1.2. Event Management, Distribution, and Generation

This section briefly describes the management, distribution, and generation of events.

1.1.2.1. Event Management

We create a simple array so that we can easily pull out which objects we want to schedule for an event. Following figure shows the simple structure.



1.1.2.2. Event Distribution

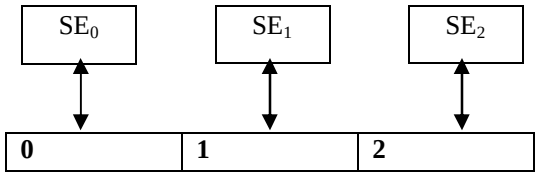
Events are uniformly distributed between numbers of objects (typically three for our project)

1.1.2.3. Event Generation

- Initially we start generating one and a half events per clock cycle
- As the testing goes on, we can vary the event generation rate. For instance, we may test the system for more than 2 events per clock

1.2. Phase II: Integrated Simulation Model

The testing is based on three simulation engines as shown in the following figure:



- For multiple simulation engines, we follow the same uniform distribution for the other two simulation engines.
- We decide 50% of the total generated events should schedule for our own simulation engines where as the other 50% will be equally divided between the rest of the simulation engines (i.e., 25% each in this example).
- In addition, in order to perform comprehensive testing that can use different choices of parameters; the distribution for the other two simulation engines can be varied.
- Furthermore, in order to enable VALID communication among three simulation engines, there should be a mechanism that can verify that the messages are sent and received in time stamp order. Thus, this feature of our simulation project determines the effectiveness of a valid communication.

1.3. One Test Example

Input parameters:
These parameters can be set, for example, as follows:
L= 3, Events = 0 to 9, start simulation

Output verification:
This can be verified as follows:

- Events send and receive in time stamp order.
- Average queue waiting time for outgoing events when event distribution is uniform.
- Average queue waiting time for an incoming link

- Etc…

Public Interface for the Simulation Engine Project

SimObj
+ SimObj(); + ~SimObj(); + void EventHandlr(Delegate* del);

SimEng
+ SimEng(); + ~SimEng(); + int GetTime(); + void StartSim(int stopschedtime); + bool SchedEvent(Delegate* del, int time);

Delegate
+ SimObj* objref; + int methodred; + int param1; + int param2;

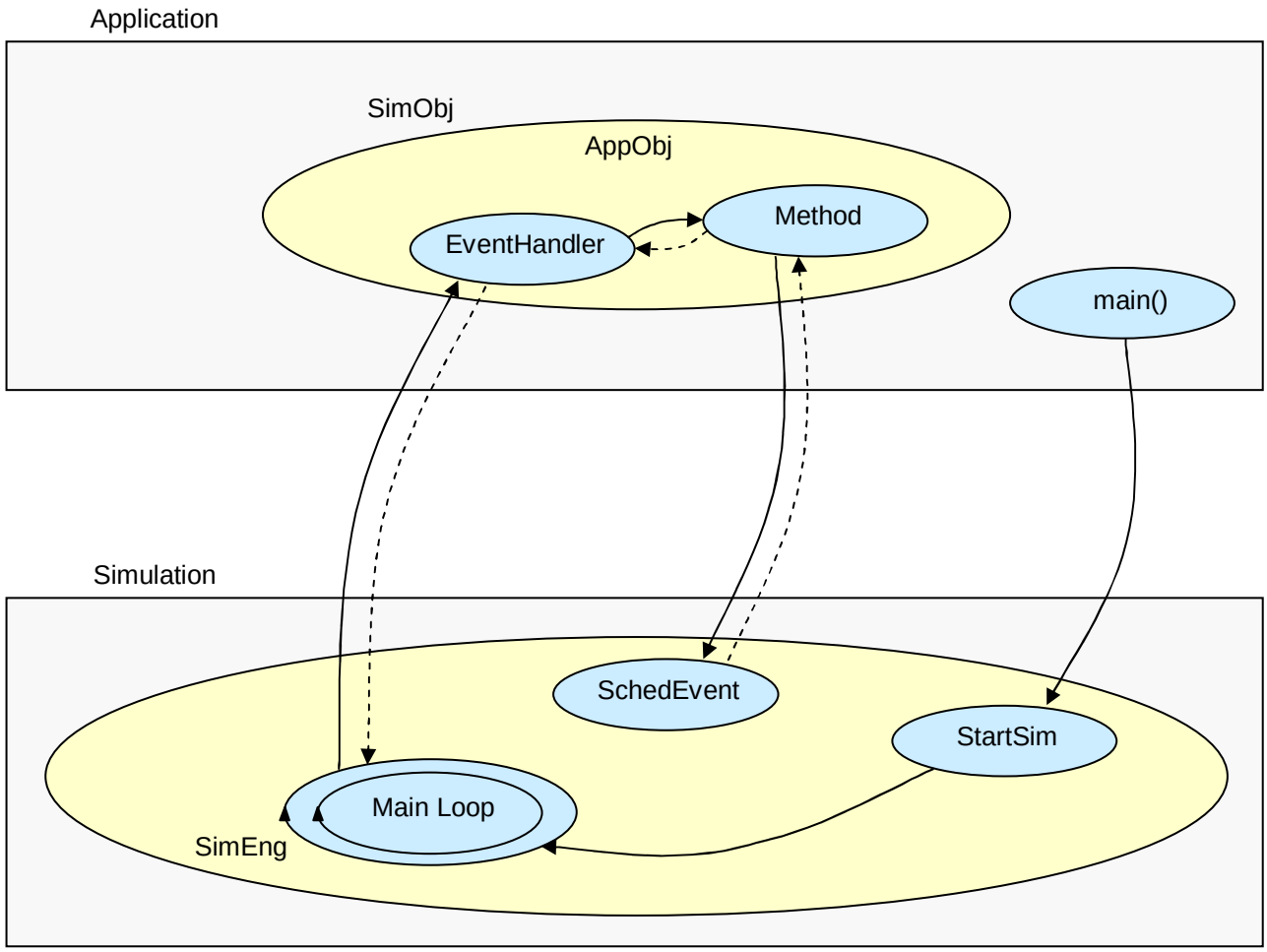
Classes not part of the Public Interface for the Simulation Engine Project

Pending Further Development

Event
+ Delegate* del; + int time;

EventListManager
+ EventListManager(); + ~EventListManager(); + bool Insert(Delegate* del, int time); + Delegate* GetNext(); + int GetTime();

Graphic representation of the structure of the Simulation Engine Project



SimObj	
string name;	// We will need a descriptive name for all objects in the simulation
SimObj(string name);	// Constructor. Requires name as a parameter (maybe an empty string)
~SimObj();	// Destructor
Event * EventHandlr(Delegate* del);	**// Events can be spawned by handing events**

SimEng	
int time;	// If SimEng is going to keep track of the time, it needs this variable
EventListManager * list;	**//Gives engine direct access to the event list manager**
SimEng();	
~SimEng();	// Destructor
int GetTime();	
void StartSim(int stopschedtime);	// Pass -1 as a parameter if there is no predefined end time
bool SchedEvent(Delegate* del, int time);	// It is better to pass a pointer to Delegate, instead of a copy
bool ExecEvent(Delegate* del);	// same

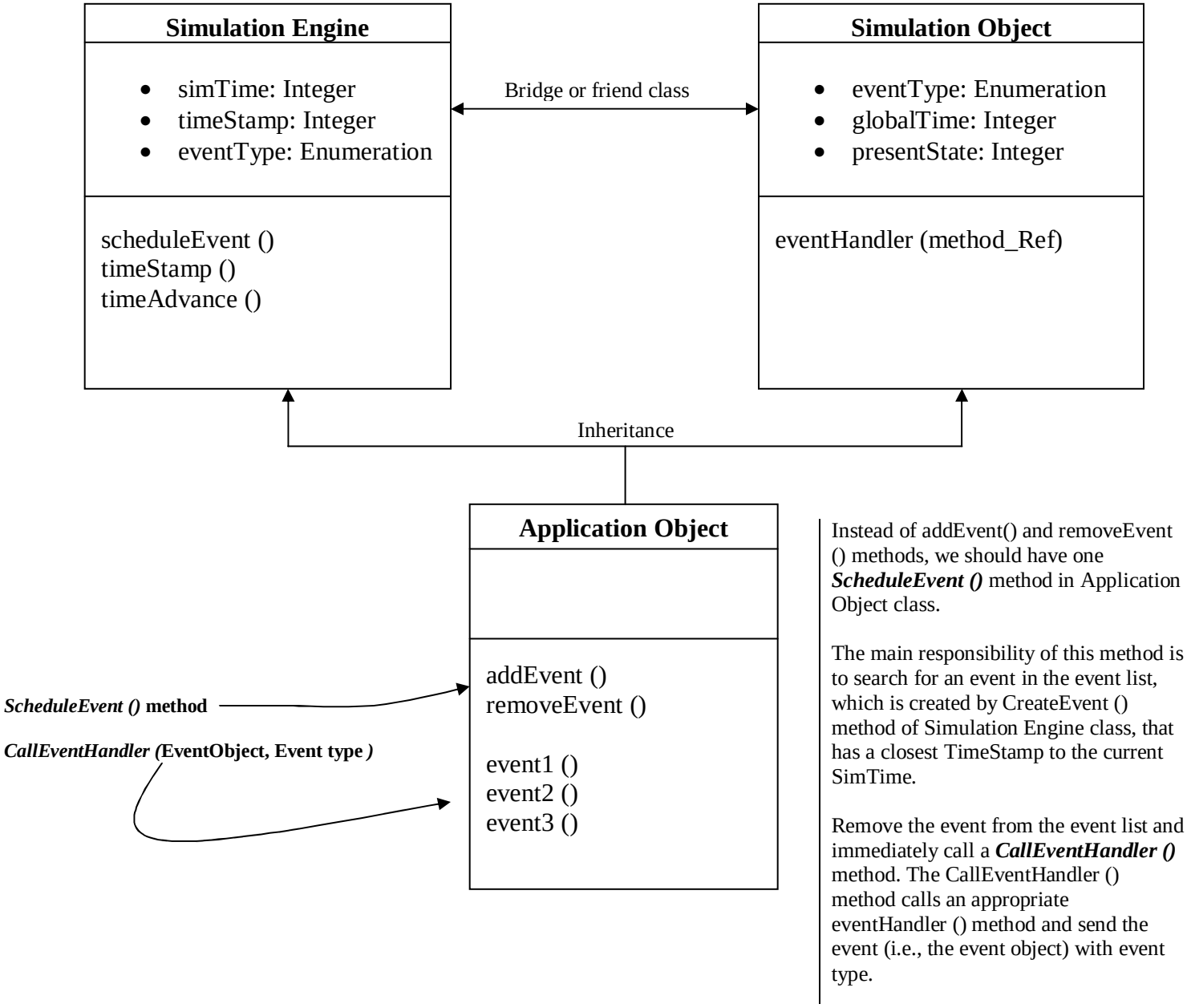
EventListManager	
int size;	// If we do not need to implement rollbacks, we only need size; otherwise, we will need all three
Event * currentpos;	**//currentPos points to the current event which is also the current sim time**
int remaining;	**//are there any more events to execute**
Event * lstHead;	**//starting point of event list**
EventListManager();	
~EventListManager();	// Destructor
bool Insert(Delegate* del, int time);	// It is better to pass a pointer to Delegate, instead of a copy
Delegate* GetNext();	// Read the next Delegate and delete it from the list
Delegate* PeekNext();	// Read the next Delegate without deleting it
int GetTime();	// What time will this method return?
int GetSize();	// May be useful
int GetCurrentPos();	// May be useful if we implement rollbacks
int GetRemaining();	// May be useful if we implement rollbacks

Delegate	
SimObj* objref;	// cosmetic changes
int methodred;	
int param1;	
int param2;	

Event	
Delegate* del;	// Who is using this structure? Is it really needed?
int time;	// It is better to store a pointer to a Delegate
Event * prev	**//allows rollback**
Event * next	**//to next event**

Comments by Kevin

By Syed Rizvi



Instead of addEvent() and removeEvent () methods, we should have one ***ScheduleEvent ()*** method in Application Object class.

The main responsibility of this method is to search for an event in the event list, which is created by CreateEvent () method of Simulation Engine class, that has a closest TimeStamp to the current SimTime.

Remove the event from the event list and immediately call a ***CallEventHandler ()*** method. The CallEventHandler () method calls an appropriate eventHandler () method and send the event (i.e., the event object) with event type.

Simulation Engine class schedules all events by creating and assigning Time Stamp to each event as well as managing overall simulation time.
SimTime is representing overall simulation time where as TimeStamp is associated with each newly created event. In addition, timeStamp should be equal or greater than to the SimTime.
ScheduleEvent **()** method creates new events and assigns appropriate time stamp through ***TimeStamp*** ()method . ***TimeAdvance ()*** is a method responsible to update overall simulation time.

Simulation Object class maintains event handlers (different types of events have their own events handler) and can be called by ***CallEventHandler ()*** method (i.e., event1, event2 etc..).
When we create an object of *Simulation Object* class, we will use event type attribute to create the appropriate ***eventHandler ()*** depending on the type of event.