DEVS/NS-2 Environment: An Integrated Tool for Efficient Networks Modeling and Simulation

Taekyu Kim
Moon Ho Hwang
Arizona Center for Integrative Modeling and Simulation (ACIMS)
Electrical and Computer Engineering Department
The University of Arizona, Tucson, AZ. 85721, USA
{taekyuk, mhhwang}@ece.arizona.edu

Doohwan Kim
RTSync Corp.
4731 E Southern Ave
Phoenix, AZ. 85042, USA
dhkim@rtsync.com

This paper presents a new DEVS/NS-2 modeling and simulation environment which supports both high and low levels of abstraction for network modeling and simulation. DEVS (Discrete Event System Specification) is a well-defined mathematical formalism specification for structure and behavior of dynamic systems. The NS-2 is a discrete event network simulator, whose primary use is intended to build and run various detailed network models and protocols such as TCP/IP, satellite links, and wireless networks. By combining the two powerful modeling and simulation systems, the significant benefits attained by the interoperable simulation of DEVS and NS-2 are reduction of the cost, increased high and low level modeling power, and enhanced reusability. To integrate the systems seamlessly, two major challenges are addressed. The first challenge is to synchronize the different ways of handling event schedules by the two simulation systems. This paper illustrates how the simulation time advances of DEVS and NS-2 are synchronized with each other. The latter problem is related to assigning the appropriate level of model structure and behavior within the combined system. The details of low level network with protocol and component description is modeled by NS-2 while DEVS serves as controller by modeling the high level behavior (e.g. use case scenario builder) of target network models and interaction of the associated actors. In this paper, we take two examples of wireless sensor networks. The first example is to describe our approach to the development process for modeling and simulation in DEVS/NS-2 environment, and the purpose of the second example is to show how DEVS/NS-2 environment is efficient for military system applications. This example is extended to demonstrate an effective way to make a decision on the appropriate level of sensor node's behavior. This leads to the discussion of tradeoffs between energy efficiency and effectiveness of decision making for the sensor network. The advantages and

the disadvantages are discussed in the last part of this paper by comparing DEVS/NS-2 environment with its related studies such as DEVS BUS and OPNET.

Keywords: DEVS, NS-2, interoperable simulation, wireless sensor network

1. Introduction

Recently, the huge growth of computer networks and communication systems has spurred the development of Modeling and Simulation (M&S) frameworks for both network protocols and applications. There exists a well known network simulator that is widely used by academia; it is the Network Simulator Version 2 (NS-2) [1]. NS-2 is a discrete event driven network simulator for various networking models. Considering only network behaviors, NS-2 supports packet transmitting related studies such as network protocol development, transmitting delay, and so on. Discrete Event System Specification (DEVS) [2] is a system specification modeling structural architecture which is based on a well-defined mathematical formalism. However, DEVS is weak at computer network simulations in that it doesn't have full ranges of detailed network protocol and component nodes compared to NS-2 or OPNET [3]. Expanding network protocols in DEVS requires high initial development cost so that the interoperable simulation of DEVS and NS-2 is expected to reduce modeling development cost. The combination of DEVS and NS-2 has two advantages in terms of modeling power and reusability.

These two simulators, DEVS and NS-2, have their own event scheduling methods. Because of this, time synchronization is the most challenging research issue in integrating the simulators. In this research, we first synchronize DEVS and NS-2. The interoperable simulation of DEVS and NS-2 is named as DEVS/NS-2. An example of wireless sensor networks is modeled and simulated. The behavior of a sensor node's application and its environmental behaviors such as battle fields are defined in DEVS modeling and the roles of networking protocol behaviors are assigned to NS-2 since NS-2 has well-designed network protocol libraries. In other words, DEVS models reside on the top of NS-2 layered network protocol models so that the roles of DEVS models and NS-2 models are distinguished and adjusted. Consequently, modeling power is increased and modeling development cost is reduced.

In addition, we model and simulate two examples. One is to prove the complete integration of DEVS/NS-2 environment, and the other is to show DEVS/NS-2's significant advantage of applicability to military system simulations. The second example, whose scenario is attacking enemy tanks with missiles in a battle field in which sensor nodes are scattered to detect enemies, is modeled and simulated in the section 5. This example introduces how DEVS/NS-2 is efficient for military applications. Finally, a feasible sensor node's behavior for effective decision making is introduced.

2. Background : Network Simulation Version 2 (NS-2)

Network Simulator Version 2 (NS-2) is a discrete event driven simulator. It has been developed at the University of California, Berkeley. It is object-oriented and is designed

primarily for local and wide area network simulations. Although it provides a lot of well organized documents, it is not easy to use because it has been extended by many developers so that the architecture of NS-2 is very complicated. In this section, some basic ideas of how the NS-2 simulation engine works, how to initialize simulation setup, and how to analyze simulation results are introduced.

2.1 NS-2 Architecture

NS-2 is an object-oriented simulator which is written in C++ and OTcl. The advantages of an object-oriented system are reusability and easy maintenance while there exists the drawbacks of performance (speed and memory) inefficiency and careful planning of modularity. The reason why NS-2 is written with C++ and OTcl separately is to compromise between modularity and speed. C++ is used for data while OTcl is employed for control. Because C++ is much faster than OTcl, C++ is used for run-time speed critical tasks such as detailed protocols. But, once network components which are written in C++ codes are compiled, no change can be made. OTcl is used for control such as an interpreter since it runs slow but changes quickly. Figure 1 shows the duality of C++ and OTcl.

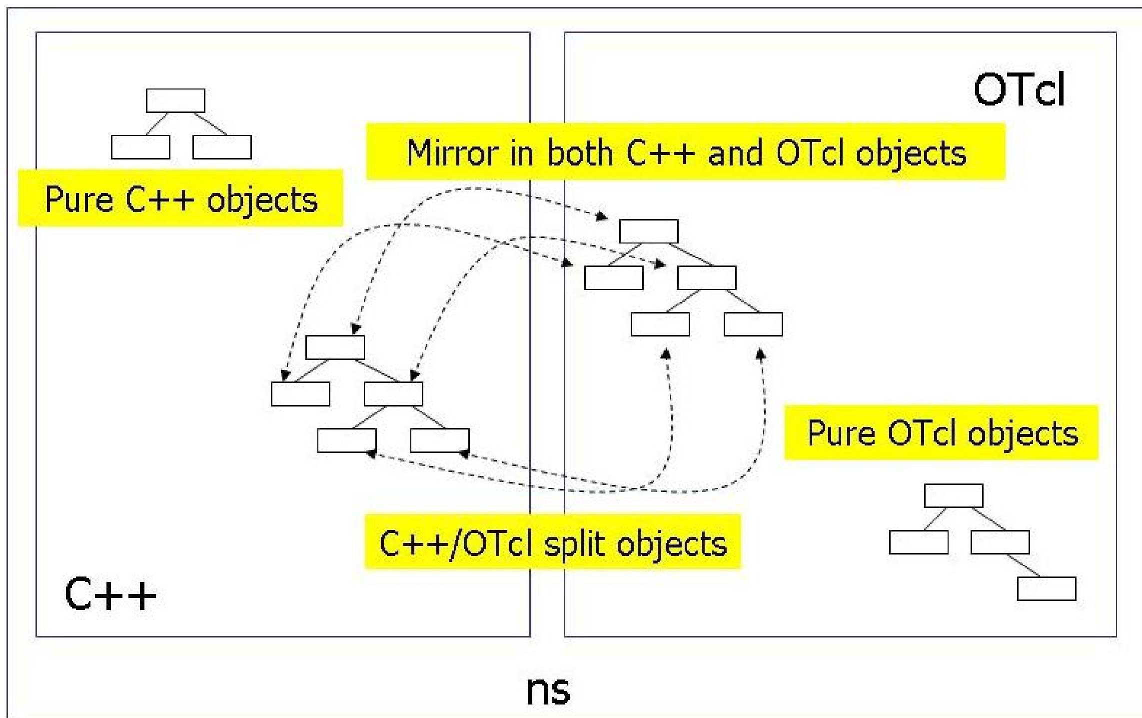


Figure 1. Duality of C++ and OTcl

Figure 2 depicts the architecture of NS-2. In Figure 2, users make simulation scripts using Tcl language in order to design and simulate. NS-2 is composed with C++ objects which are the network components and the event scheduler, OTcl linkage that is implemented in TclCL, and OTcl.

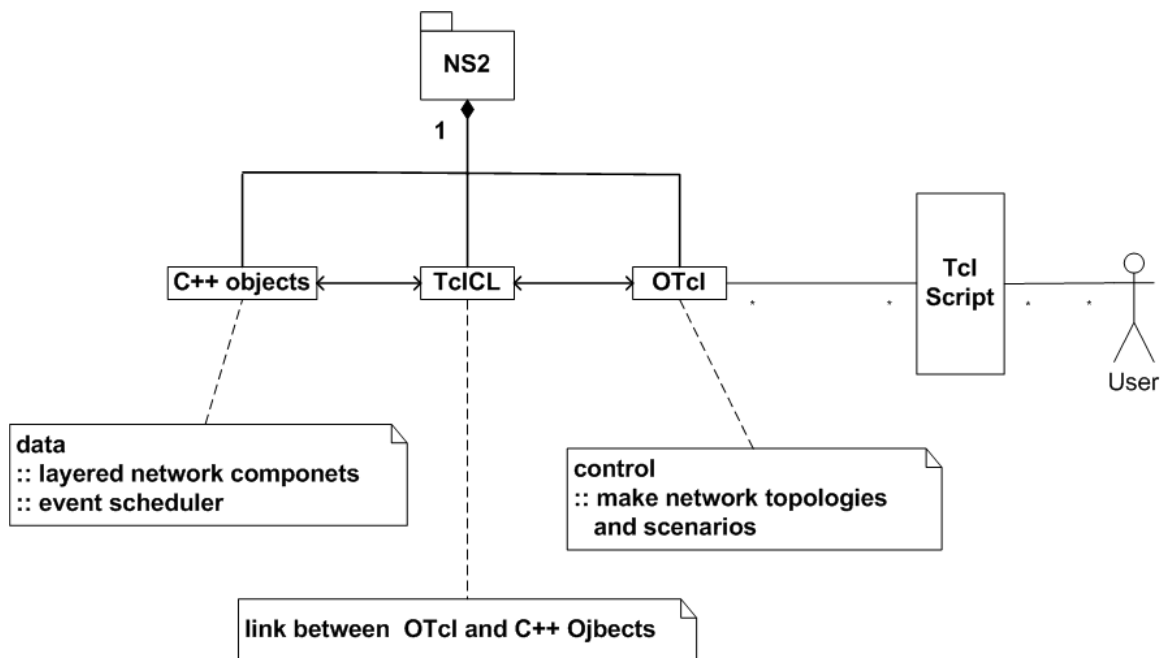


Figure 2. NS-2 architecture

2.2 NS-2 Event Scheduler

The event scheduler is a discrete event scheduler and one of the independent C++ objects in NS-2. It is based on a logical time scheduling. Figure 3 depicts a simple overview of the NS-2 event scheduler.

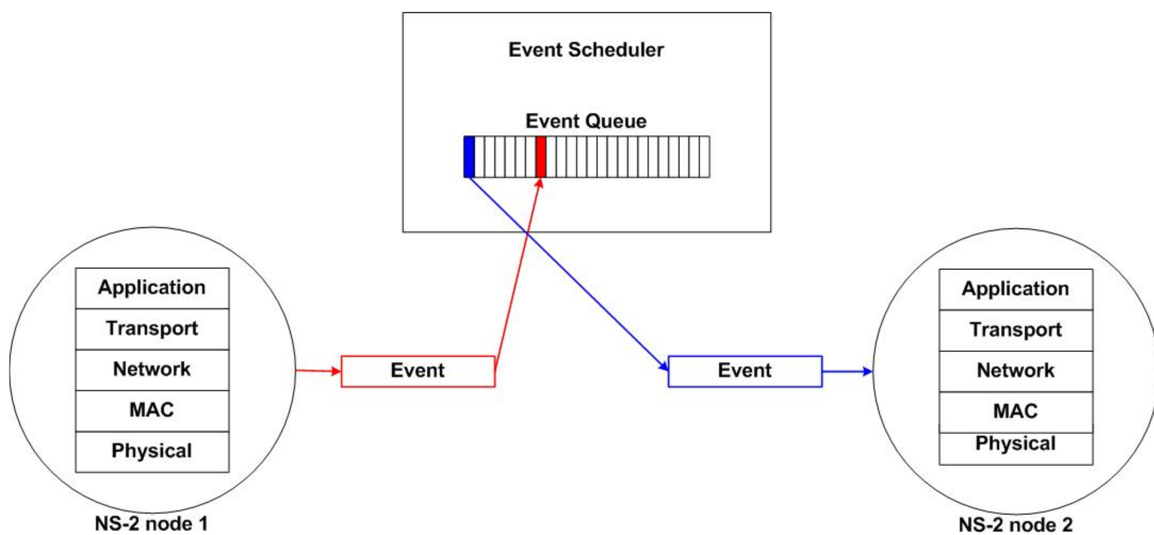


Figure 3. NS-2 event scheduler

The event scheduler has its own event queue and there is only one event queue during a simulation. Every network component such as network nodes has a link to the event scheduler. Once a network component creates an event, the network component puts the event into the event queue. As a simulation goes on, the event scheduler looks at the very first event in the event queue and assigns the first event to appropriate network components at a time which is included in the event.

3. Integration of DEVS and NS-2

3.1 Synchronization

The synchronization is the high priority issue to integrate DEVS with NS-2. Both DEVS and NS-2 have their own simulation time management systems. Thus, the synchronization of event scheduling is a high priority issue when integrating DEVS and NS-2. Whenever an event occurs in either DEVS or NS-2, the one which gets an event has to inform the other that an event has happened in it, and the other needs to synchronize with the simulator which caused the event. Initially, we have to make sure that they use the same time unit and time mechanism. Because NS-2 uses the logical simulation time and the wall clock time unit is a second, we use the logical time and the second for a time unit in DEVS. Figure 4 shows basic conceptual architecture of a heterogeneous simulation between DEVS models and NS-2 models.

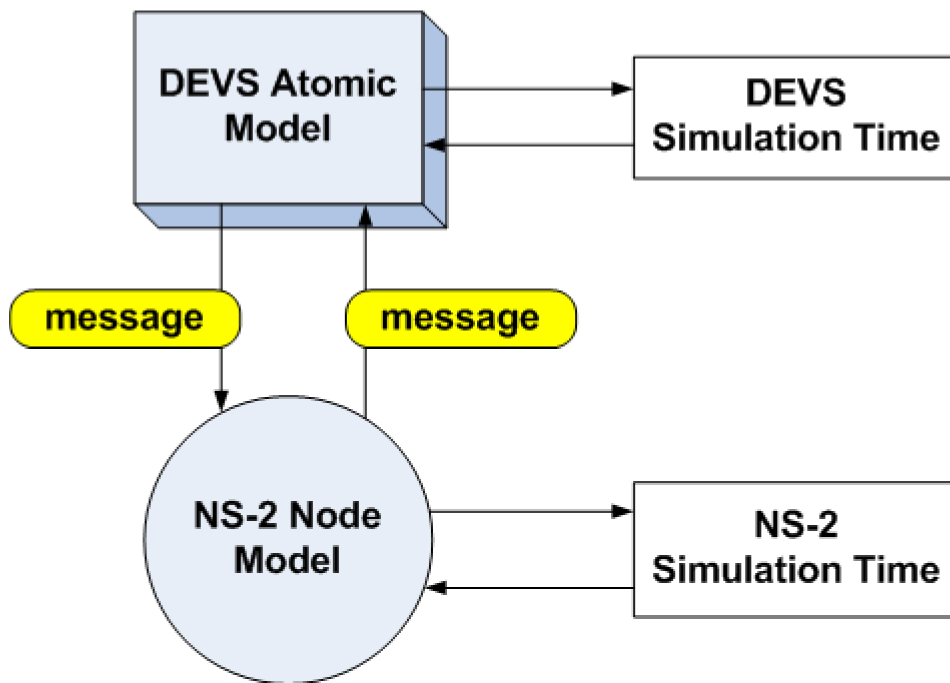


Figure 4. The conceptual architecture of DEVS and NS-2

Figure 4 represents the conceptual architecture of DEVS and NS-2 from the point of view of time synchronization. The DEVS simulation time is not a problem when a DEVS

atomic model, which is a network node model corresponding to an NS-2 network node model, has an event because the DEVS simulation time and a DEVS atomic model work together in the same coordinator. However, the NS-2 simulation time should be updated as soon as an event occurs in a DEVS atomic model. If a DEVS atomic model that has an event doesn't let a corresponding NS-2 node model know that it has an event, there is no way for the NS-2 node model to know that the DEVS simulation time had been advanced. As a result, whenever a DEVS atomic model gets an event, it should send a message regarding an event occurrence to update the NS-2 simulation time. Similarly as DEVS atomic models do, NS-2 node models must inform their matching DEVS atomic model that the simulation time is updated whenever they get events. There exists a DEVS atomic model which deals with the time synchronization between DEVS and NS-2. It is named as "NS-2 Event Queue Agent". The NS-2 Event Queue Agent model connects to the event queue of NS-2 through a DEVS NS-2 interface. As a result, this model triggers NS-2 to put events into the event queue whenever DEVS models get events. This is discussed in the next section.

3.2 NS-2 Event Queue Agent

The NS-2 Event Queue Agent model is the most important one for DEVS-NS-2 environment. This atomic model connects to the event queue of NS-2. So, this model controls the NS-2's schedule. Once the DEVS coordinator calls this model through the DEVS process, this model dispatches the first event and processes it. Figure 5 shows the relation with the DEVS modeling and NS-2 modeling.

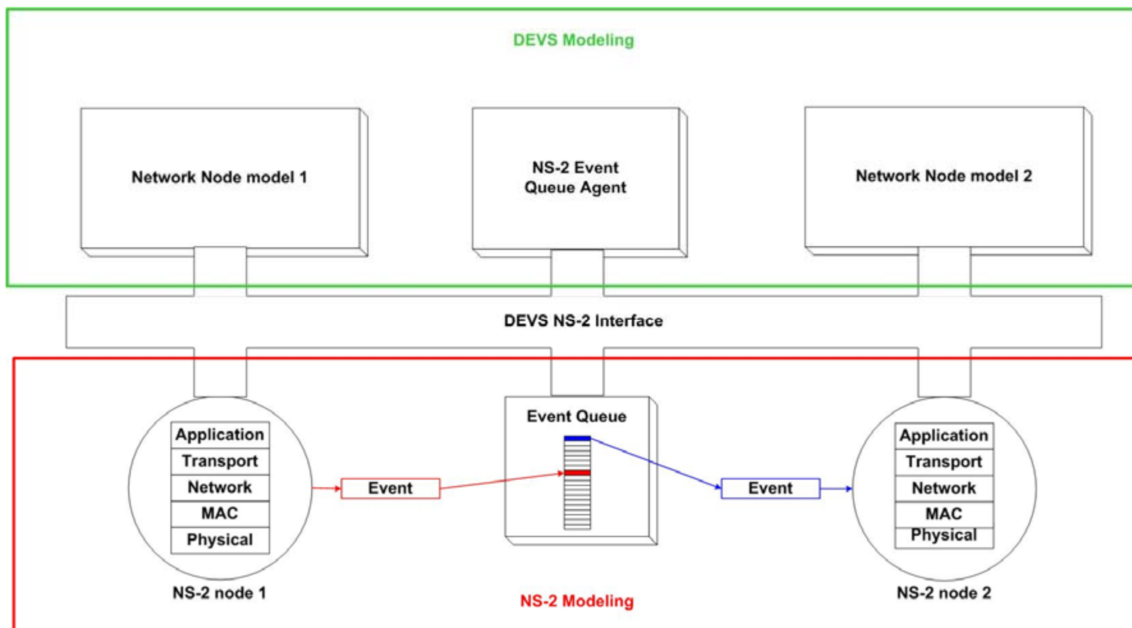


Figure 5. NS-2 Event Queue Agent Model

The time advance (ta) of the NS-2 Event Queue Agent model is measured using the information of the first event's scheduled time in the event queue and the current time.

ta = first event's scheduled time in NS-2 queue – NS-2's current time

If the NS-2 Event Queue Agent model is imminent (its tN = global tN), then it triggers NS-2 event queue by processing the first event. When δ_{int} is run, this model calls a function of NS-2 in order to dispatch the first event from the event queue and process the event. For example, if the current time is 10 seconds and the first event of the queue is scheduled at 10.3 seconds, then the time advance is set as 0.3 seconds. 0.3 seconds later, an internal transition function is called. In turn, the first event is dispatched from the event queue and processed. The synchronization algorithms are shown in Figure 6. In addition to the synchronization algorithm, the Figure 7 shows how DEVS and NS-2 get synchronized through NS-2 Event Queue Agent Model.

< Define the time advance of the NS-2 Event Queue Agent Model >

if an event exists in the first entry of the NS-2's event queue

then, ta = the event's scheduled time – NS-2's current time /* ta is time advance */

if ta < 0

then abort /* error */

else, ta = infinity

< Define the δ_{int} of the NS-2 Event Queue Agent Model >

δ_{int} triggers the event

1. dequeue the first event from the NS-2's event queue
   by calling NS-2's deque function (NS2::Scheduler::instance::deque())
2. run the event
   by calling NS-2's dispatch function (NS2::Scheduler::instance::dispatch())

After defining the time advance and the internal transition function

if the ta is the smallest ta

then, tn = tl + ta(s) /* tn is a time of next event */

s = δ_{int}(s) /* for NS-2 to process its scheduled event at this time */

tl = t /* tl is a time of last event */

Figure 6. Synchronization Algorithm

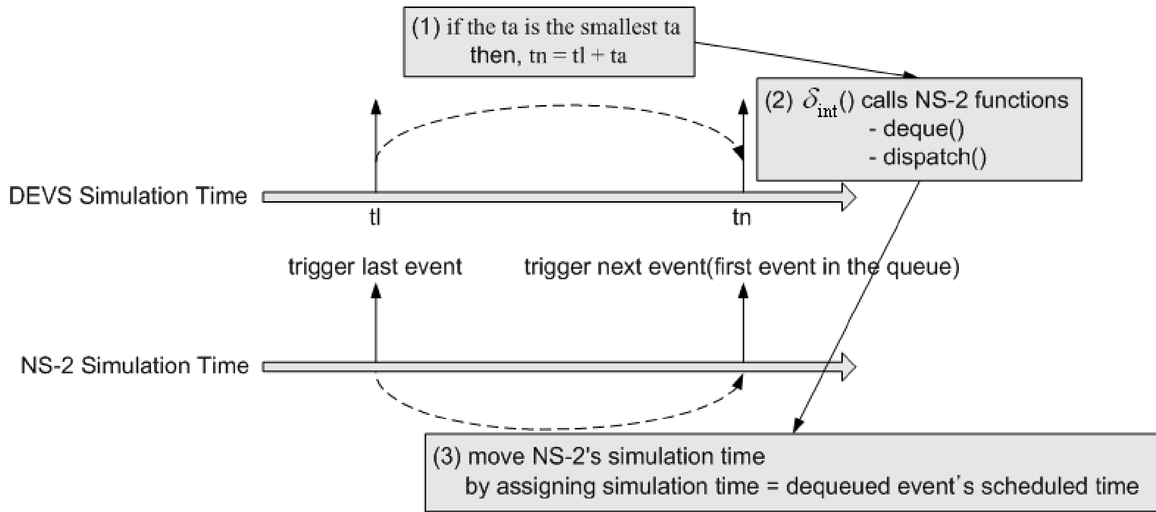


Figure 7. Synchronization

3.3 Node Configuration

To simulate the scenario for the integration of DEVS and NS-2, a node configuration needs to be defined. An NS-2 node configuration is defined in Tcl script code. Node configuration consists of defining the different node protocol characteristics before creating them. They may consist of the type of addressing structure used in the simulation, defining the network components for models, selecting the type of a routing protocol, and defining the energy model [4, 5, 6]. The node configuration setting is shown in Table 1. Since we model and simulate an example of wireless sensor networks in this paper, a wireless sensor node is defined for a network topology.

Node Configuration Attribute	Value
Channel type	Channel/WirelessChannel
Radio propagation model	Propagation/TwoRayGround
Network interface type	Phy/WirelessPhy
MAC type	Mac/802_11
Interface queue type	Queue/DropTail/PriQueue
Link layer type	LL
Antenna model	Antenna/OmniAntenna
Max packet in IFQ	50
Routing protocol	DumbAgent
Energy model	EnergyModel
Initial energy in Joules	100.0
Agent trace	ON
Router trace	ON
Mac trace	OFF
Movement trace	OFF

Table 1. Node Configuration

In this example, node configuration for a wireless sensor node assigns the Dumb Agent routing algorithm as its ad hoc routing protocol. The link layer type as LL and Medium Access Control (MAC) protocol as IEEE 802.11 are configured. The queue between the MAC layer and the link layer is used. At last, the wireless physical layer is defined. The network stack is configured by assigning the protocols for the link layer, the MAC layer, and the physical layer. Because the sensor nodes communicate with each other using a wireless network, a channel topology and a propagation model are needed. Consequently, the Omni Antenna, the wireless channel, and the two-lay ground propagation are assigned in the node configuration. The initial energy model is set as 100 joules in order to measure the energy consumption. The sensor nodes are assumed as they don't move. The router trace and the agent trace are turned on to investigate the simulation results. This procedure creates the sensor node object, creates an ad hoc-routing agent as specified, and creates the network stack consisting of the link layer, interface queue between the link layer and the MAC layer, the MAC layer, and the network interface with the antenna. This procedure also uses the defined propagation model and interconnects these components.

A Constant Bit Rate (CBR) traffic generator is assigned for the application agent. In the DEVS/NS-2, a DEVS model conceptually resides on the top of an NS-2 model. As a result, the DEVS sensor node model and the NS-2 sensor node model can communicate with each other by passing messages. The DEVS sensor node model doesn't have to know the details of the NS-2 sensor node model's configuration and how it works. What the DEVS sensor node model needs to do is send event messages to the NS-2 sensor node model when the DEVS sensor node model gets events. In this paper, the DEVS sensor node model is connected with its own NS-2 sensor node model's CBR traffic generator agent that is used for the application of the sensor node. The DEVS sensor node model considers only the CBR traffic generator as it own NS-2 node model, but the lower layers of the NS-2 sensor node are abstracted. Figure 8 shows the relation of the DEVS sensor node model and the NS-2 sensor node's layered network protocol model [1].

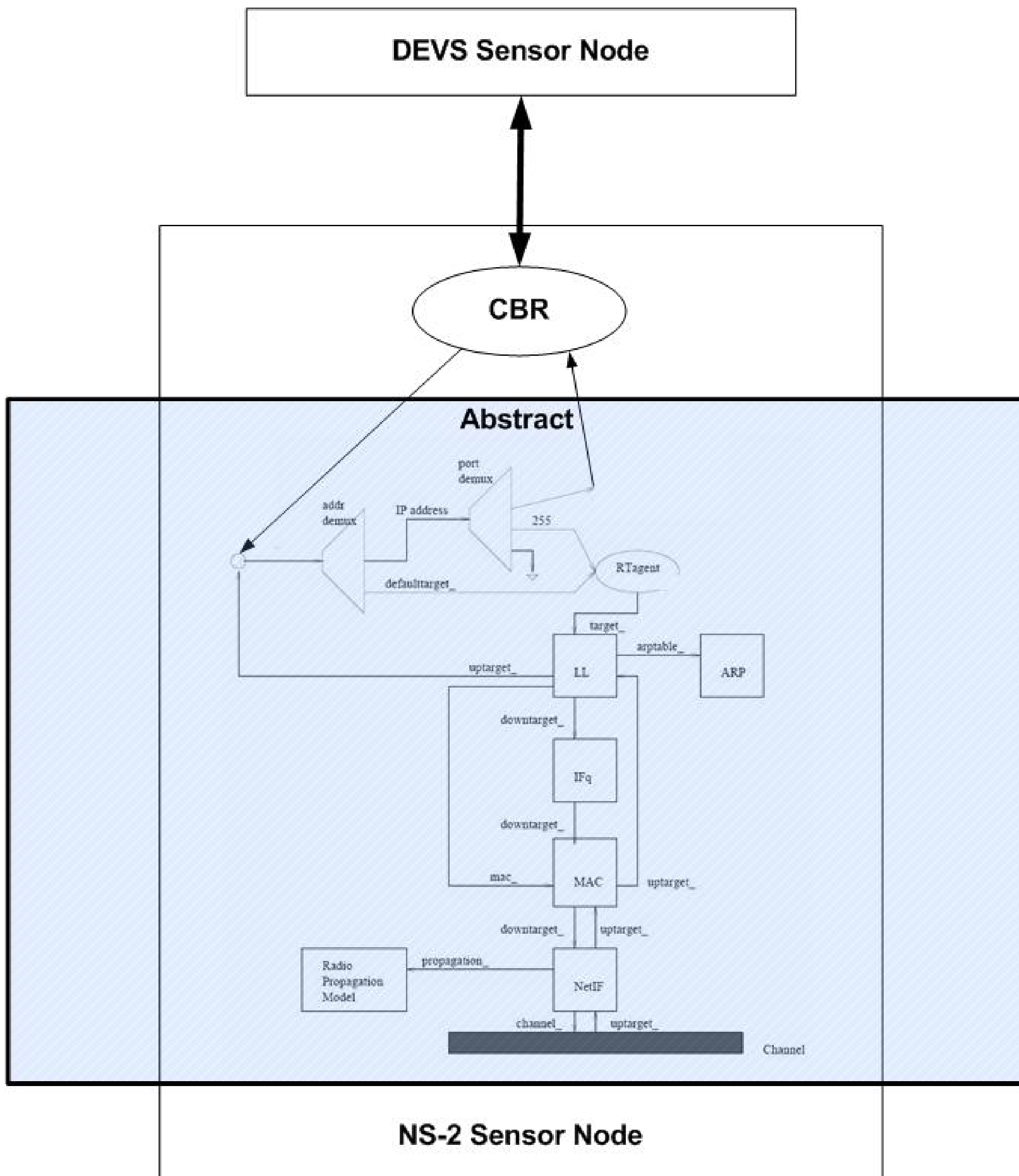


Figure 8. Schematic of the relation with DEVS and NS-2 Sensor Node

If a DEVS sensor node model detects objects and needs to send messages to a destination node model, this DEVS sensor node model passes a message such as “start sending packets” to a connected CBR traffic generator agent of a corresponding NS-2 sensor node model. Otherwise, if this DEVS sensor node model loses objects from its detecting ranges and, therefore, this DEVS sensor node model needs to stop sending messages to a destination node model, this DEVS sensor node model sends a message such as “stop sending packets” to the connected NS-2’s CBR traffic generator agent. This is how the message passing is performed in this integrated simulation environment.

4. Example

4.1 Network Topology

In this section, the simple network topology is presented to get the simulation result of energy consumptions and packet deliveries for proving the integration with DEVS and NS-2. Seven sensor nodes, one base node, and one tank (moving object) are initialized in a 100*100 space. Figure 9 shows the network topology and the route for tank.

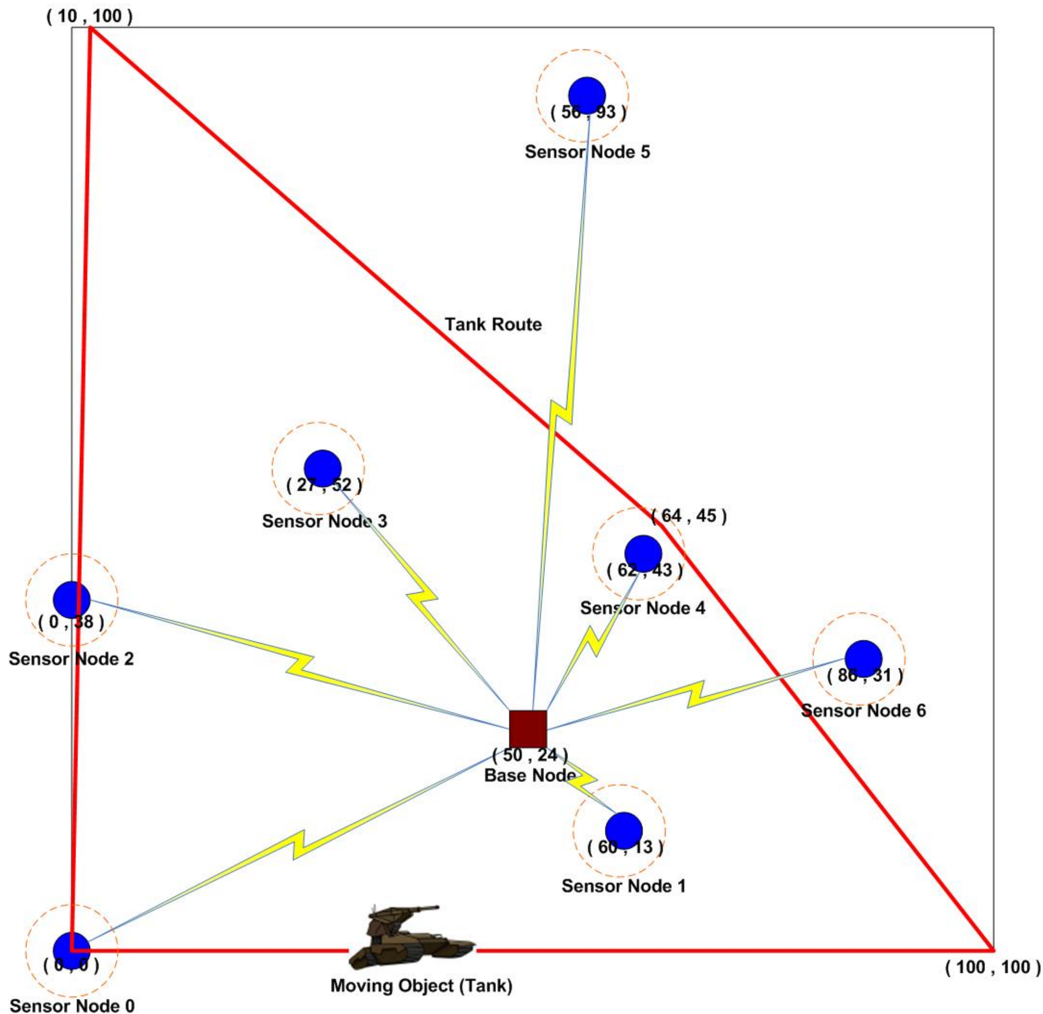


Figure 9. Network Topology and pre-defined tank route

In Figure 9, the filled circles represent sensor nodes and the dot circle lines mean sensing areas of each sensor node. A base station node is represented as the filled square. There is one moving object (tank) and a pre-defined tank route. All the sensor nodes and the base station node communicate through a radio based wireless network. The data packets are delivered from the sensor nodes to the base node using the NS-2 simulator and the NS-2 layered network objects. Because the DEVS node model resides on the top

of the NS-2 node model, the DEVS models don't have to consider the packet delivery but instead deal with the detection of moving objects. The sensor node zero, one, two, three, four, five, and six are able to detect the tank's movement. They send packets as soon as they sense the tank. The base station node, which is the destination node of all the sensor nodes, receives packets from the sensor nodes. When the base node receives packets, the base node discards the packets. In turn, it is ready to receive other packets.

4.2 Modeling

In this section, the simple DEVS modeling is introduced in order to show that the integration of DEVS and NS-2 works without any risk. There are four atomic models which are a sensor node model, a NS-2 event queue agent model, a moving object model (tank), and a collision checker model. In addition, there exits a conceptual interface named as a DEVS NS-2 interface, whose role is to connect DEVS models and NS-2 models. Figure 10 shows the architecture of DEVS models.

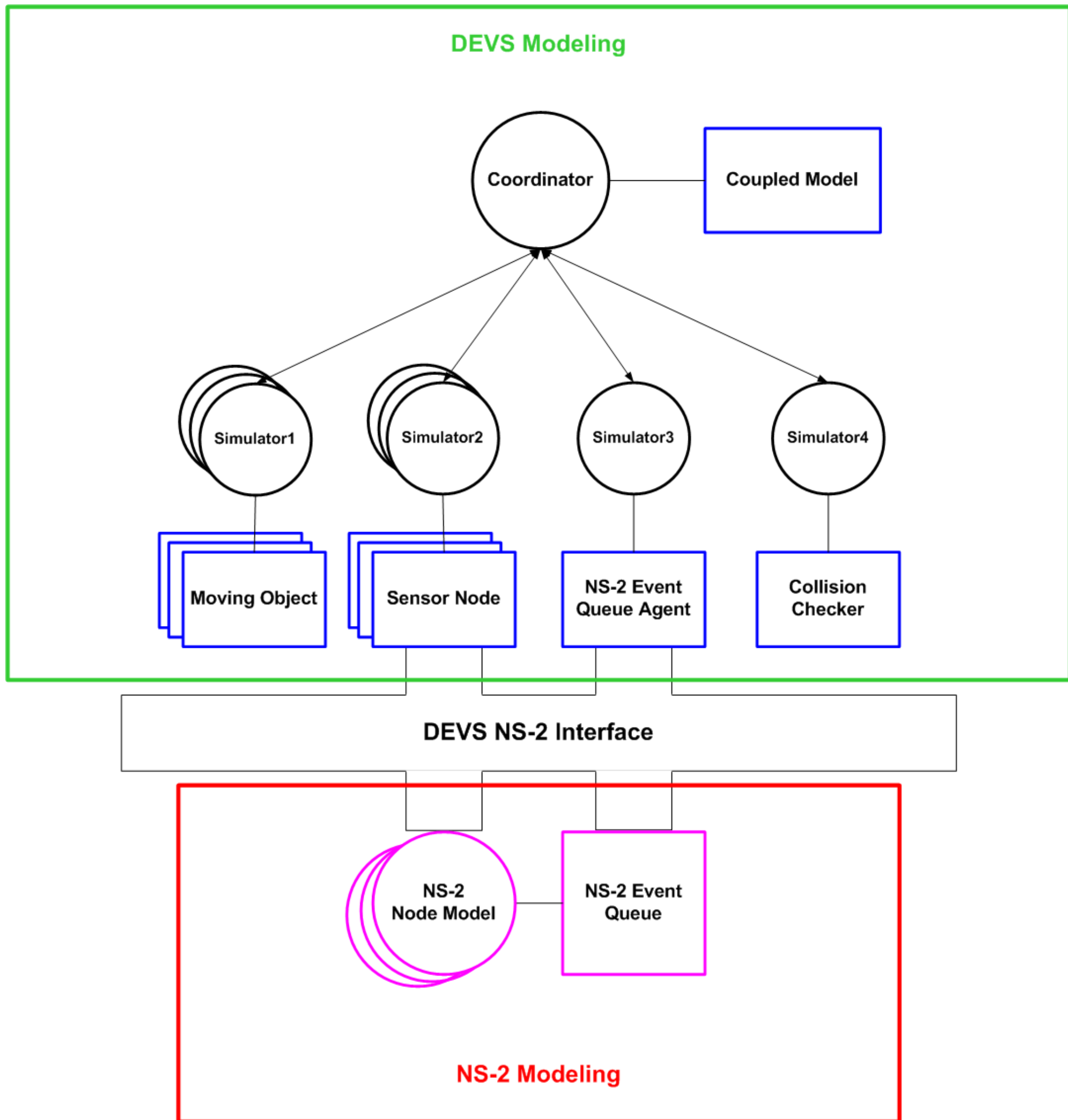


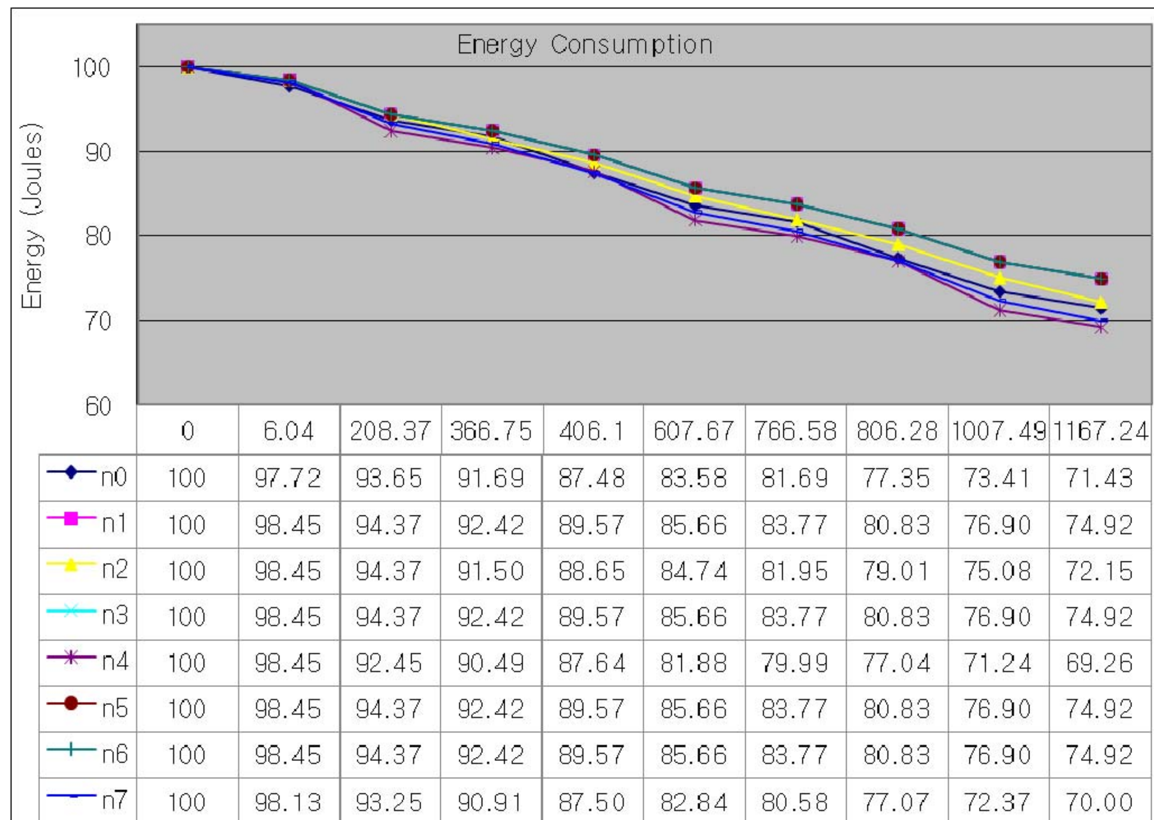
Figure 10. DEVS models in the example of wireless sensor network

The sensor node model has a connection with an NS-2 node model by one-to-one mapping through the DEVS NS-2 Interface. As soon as the DEVS sensor node model detects an event, it triggers its matching NS-2 sensor node to start generating and sending packets. The moving object model such as a tank moves following a pre-defined route, and the purpose of this model is to give events to the sensor node model. The moving object model is a continuous event model because of its moving behavior. The third atomic model is the NS-2 event queue agent model that is shown in the previous section. Because the NS-2 event queue agent model connects to the NS-2's event queue, this model triggers NS-2 to put events into the event queue whenever a DEVS sensor node model gets events. The last model is the collision checker model. It is a continuous event model and decides whether there are collisions among sensor node models and

moving object models. We assume that the word “collision” means sensor node models detect moving objects. Recall that the DEVS NS-2 interface is conceptual. In the last of this section, we will discuss in more detail about the DEVS atomic model’s behavior.

4.3 Simulation Results

The consumed energy and the total number of generated packets are measured through this simulation example. Figure 11 and Figure 12 represent the simulation results in terms of the energy consumption for each node and the total number of packets that are generated in the whole network topology. In this simulation, the initial energy is set to 100 joules for each node. Nodes spend energy when they send, receive, or transmit packets. Figure 11 is the chart that shows the remaining energy in each node. Although there are no packets generated in sensor nodes 1, 3, 5, and 6, they consume energy because they receive packets anyway and discard them because they are not the destination for the packets.



	0	6.04	208.37	366.75	406.1	607.67	766.58	806.28	1007.49	1167.24
n0	100	97.72	93.65	91.69	87.48	83.58	81.69	77.35	73.41	71.43
n1	100	98.45	94.37	92.42	89.57	85.66	83.77	80.83	76.90	74.92
n2	100	98.45	94.37	91.50	88.65	84.74	81.95	79.01	75.08	72.15
n3	100	98.45	94.37	92.42	89.57	85.66	83.77	80.83	76.90	74.92
n4	100	98.45	92.45	90.49	87.64	81.88	79.99	77.04	71.24	69.26
n5	100	98.45	94.37	92.42	89.57	85.66	83.77	80.83	76.90	74.92
n6	100	98.45	94.37	92.42	89.57	85.66	83.77	80.83	76.90	74.92
n7	100	98.13	93.25	90.91	87.50	82.84	80.58	77.07	72.37	70.00

Figure 11. The remaining energy for each node

The Figure 12 represents the total number of packets that are generated at every node during the simulation. The total number of the generated packets is measured at the end of the third cycle of the tank’s movement. 829 packets have been generated during the simulation.

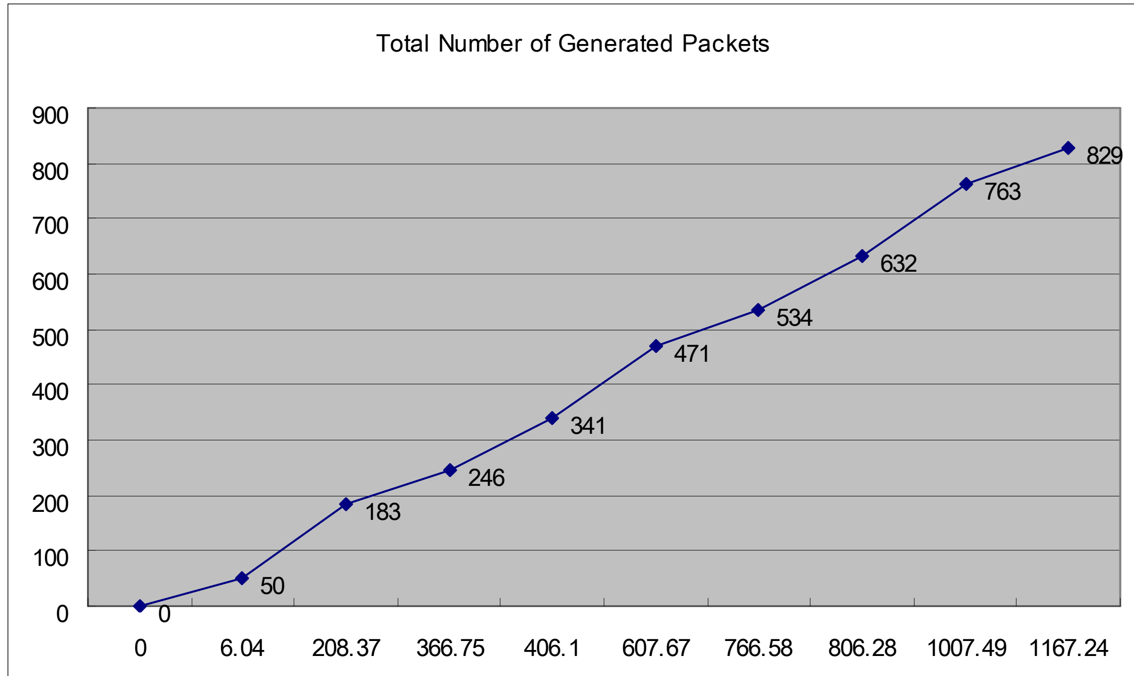


Figure 12. The total number of generated packets in the simulation.

The Figure 13 shows the number of packets generated and the energy consumption of the sensor nodes 0, 2, and 4 which are the nodes that detect the tank and generate packets. The results are reasonable since generating more packets leads to more energy consumption.

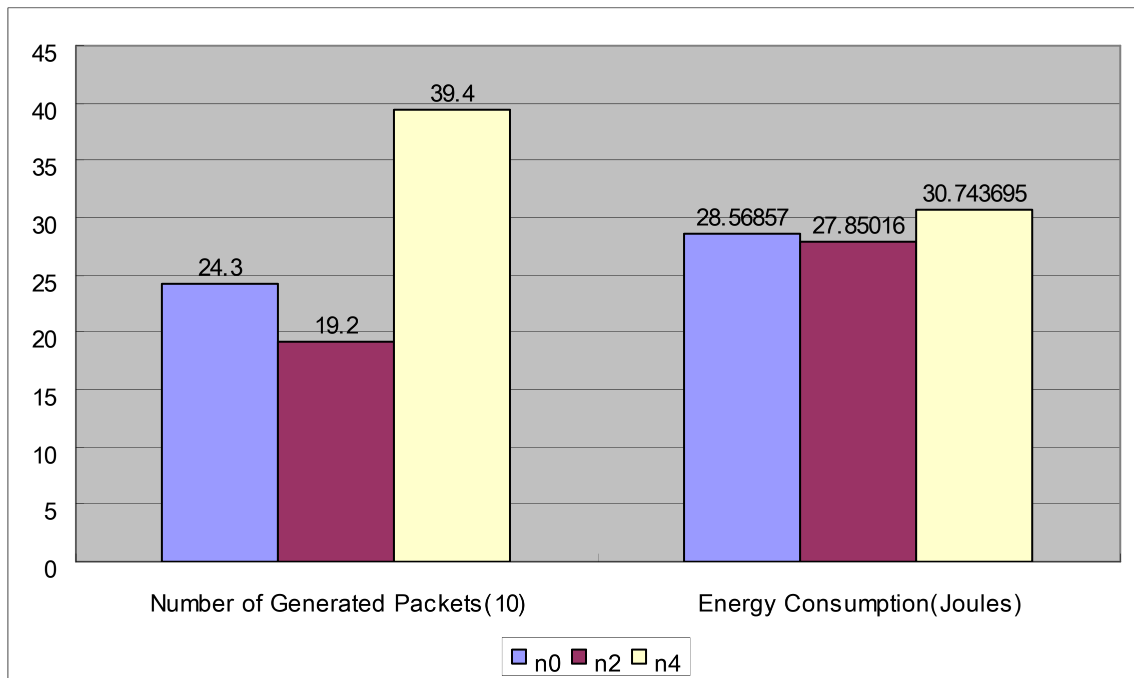


Figure 13. The simulation result at the sensor node 0, 2, 4

Figures 11, 12, and 13 depict the results of the DEVS/NS-2 simulation. Now, the simulation with NS-2 alone and the simulation results are needed in order to compare with the results which are obtained by the DEVS/NS-2 simulation. To make the same environment, the pre-defined scenario is batched in Tcl script code. Instead of the behavior of a tank and the behavior of a sensor node, generating packet times and stopping packet time, which are measured by the DEVS/NS-2 simulation, are assigned to the sensor nodes 0, 2, and 4. The pre-defined scenario in Tcl script codes are shown in Table 2.

Cycle	Time	Event
1	0.1	Node 0 CBR start
	5.1	Node 0 CBR stop
	192.3	Node 4 CBR start
	205.4	Node 4 CBR stop
	359.2	Node 2 CBR start
	365.7	Node 2 CBR stop
2	395.1	Node 0 CBR start
	405.0	Node 0 CBR stop
	592.3	Node 4 CBR start
	605.4	Node 4 CBR stop
	759.2	Node 2 CBR start
	765.7	Node 2 CBR stop
3	795.1	Node 0 CBR start
	805.0	Node 0 CBR stop
	992.3	Node 4 CBR start
	1005.4	Node 4 CBR stop
	1159.2	Node 2 CBR start
	1165.7	Node 2 CBR stop

Table 2. Pre-defined scenario

With this pre-defined scenario and the same network topology which is represented in Figure 9, the NS-2 alone simulation is processed. The results regarding the energy consumption and the number of packets generated are exactly the same as the DEVS/NS-2 simulation results. So, it is concluded that the integration of DEVS and NS-2 is accomplished risk free.

5. Example of a Military Application

In section 4, the integration of DEVS and NS-2 was shown through a simple wireless sensor network example. This section shows an example that only the DEVS/NS-2 can simulate, but an NS-2 simulation alone cannot do. The primary advantage of the DEVS/NS-2 simulation compared with an NS-2 simulation is that not only a network

simulation but also a real environmental simulation is possible. On one hand, because the NS-2 simulation considers not information included in packets but packet transmission events for developing network protocols, improving network system performance, and other purposes, a receiving node just discards received packets in the simulation. On the other hand, in a DEVS/NS-2 simulation, a node doesn't discard packets that may include very important data. Once a node receives packets, it may use information for secondary reactions. Due to this reason, examples which are modeled and simulated by DEVS/NS-2 have huge potential to be applied to military application. The example, which is presented in this section, could be a kind of military application even though the scenario of the example is not real but virtual. The simulation scenario and its network topology are presented, and the DEVS/NS-2 modeling is introduced consequently. Lastly, the simulation results are analyzed in the last part of this section.

5.1 Network Topology

The purpose of this scenario is to propose the behavior of a sensor node in the wireless sensor networks. The basic idea is that sensor nodes send packets while they don't detect anything. The packets are named as "alive packets". Sending alive packets requires energy but, it may help for effective decision making. Through the simulation, the effectiveness of alive packets is proven in regard to decision making. The Figure 14 shows the schematic of this scenario.

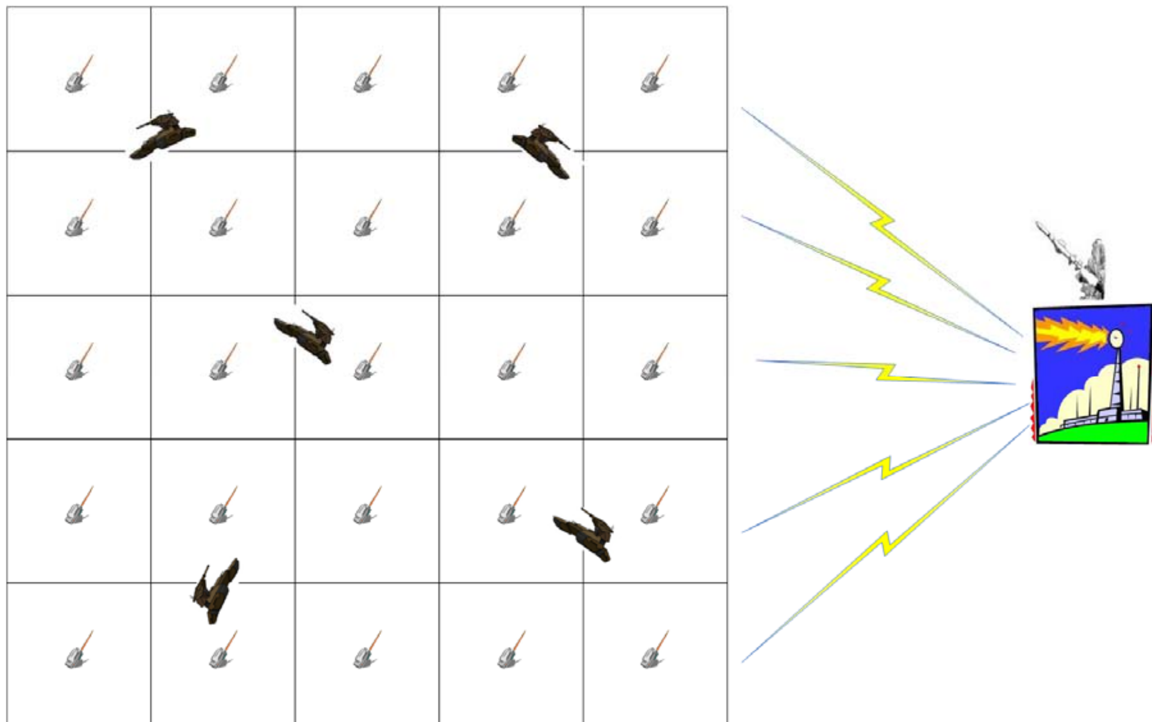


Figure 14. Schematic of the scenario

There are 25 sensor nodes in a battle field and one base station. All the sensor nodes have their own sensing radius and the sensor nodes communicate with the base station

through a radio based wireless channel. The base station has missiles and is ready to launch to attack enemy tanks. Five tanks also exist in the battle field and are moving. Once sensor nodes detect tanks, they send detecting packets to the base station constantly while the tanks are in the sensor nodes' sensing area. At the same time, sensor nodes which detect nothing in their sensing area send alive packets to the base station with an alive packet interval time. If the base station node receives packets, then it distinguishes packets as detecting packets or alive packets. If the base station decides the packet is a detecting packet, then the base station launches missiles to the point of a sensor node that is the source node of the detecting packet. Sometime later after the base station launches the missile, the missile arrives at the destination and explodes objects that are within the missile's explosive area. The tanks are to keep moving until they are destroyed by a missiles attack, and the sensor nodes are also to keep working on detecting tanks and sending packets that are either detecting packets or alive packets until either they are destroyed by missiles or they run out of energy. In this simulation, the time which all the sensor nodes are destroyed or all the tanks are destroyed is measured. If the base station doesn't receive any packets during an alive packet interval time, the base station considers that all the sensor nodes are destroyed and there is no way to detect enemy tanks. As soon as all the sensor nodes are destroyed, the base station gets to know that, and some action is needed to defend from an attack. The behavior of sensor node's sending alive packets is expected to be very effective for a defending system.

5.2 Modeling

The schematic architecture of the modeling is shown in Figure 15. We may divide the schematic architecture into three parts, the DEVS modeling, the NS-2 modeling, and the DEVS and NS-2 interface.

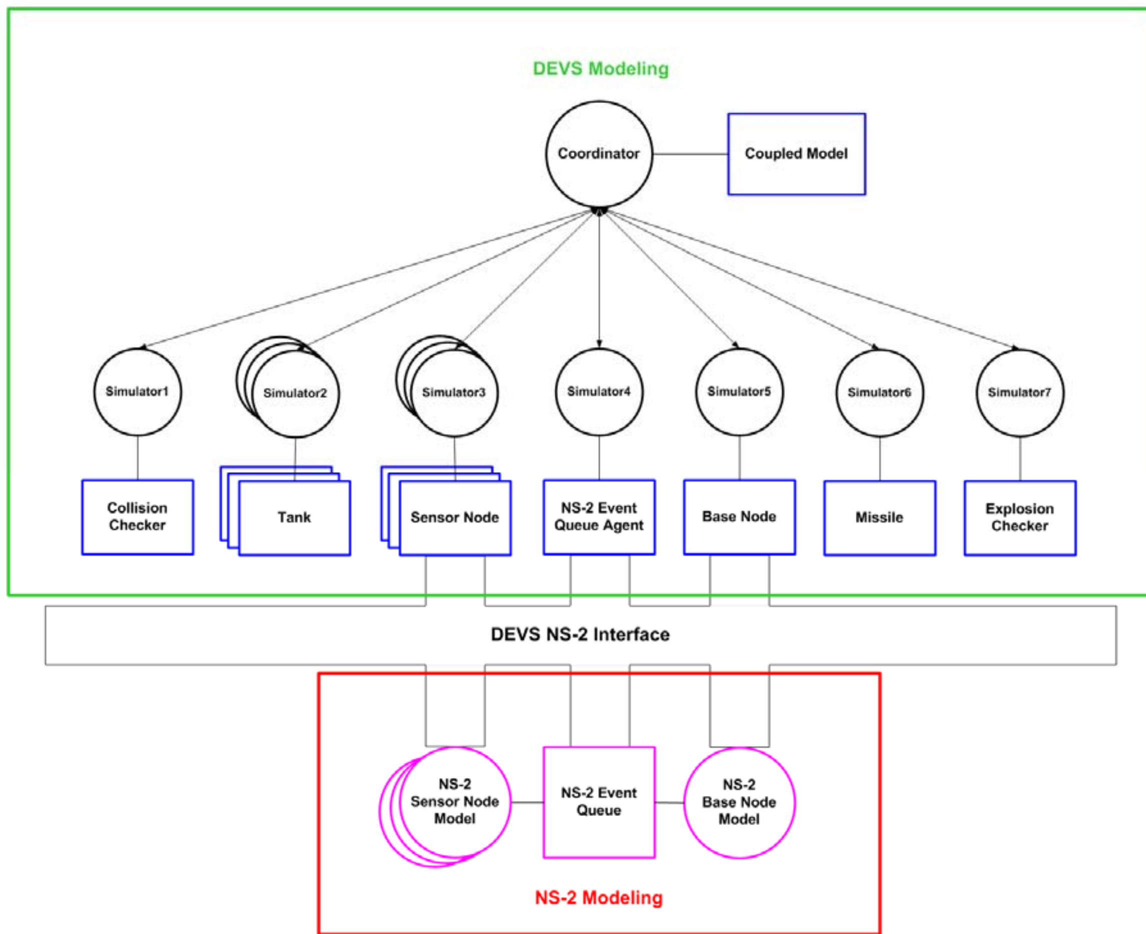


Figure 15. Schematic Architecture of the Modeling

There are seven DEVS atomic models which are the sensor node model, the base node model, the tank model, the missile model, the collision checker model, the explosion checker model, and the NS-2 Event Queue Agent model. The sensor node model and the base node model have connections with their matching NS-2 node models. The sensor node model, the tank model, the collision checker model, and the NS-2 Event Queue Agent model are based on the previously explained models in the section 4, but the previously explained models are so simple that we upgrade these models and models adjust to this example. The other three models that are the base node model, the missile model, and the explosion checker model are newly developed.

The sensor node model is based on the model which is introduced in section 4.2. But, we modify the model by adding and removing certain behaviors of the sensor nodes. The biggest change is adding the “send alive packet” state to it. This is our new approach to define the sensor node’s behavior in wireless sensor networks. We expect that sending alive packets would increase the decision making rate.

The base node model has not been introduced in the previous example. The previous example doesn’t react according to received packets because that concentrates on only integration issues. However, the purpose of this example is to show that the DEVS/NS-2 is capable of doing what the NS-2 alone cannot do. That’s why the base node model is

newly introduced in this section. Prior to explaining the behavior of the base node model, it needs to be mentioned that the DEVS base node model connects directly to the NS-2 base node model the same way the sensor node model does.

The tank model inherits and reuses the moving object model in the section 4.2. The difference is that the tank model moves following randomly generated routes during a simulation rather than pre-determined routes. The initial positions are also determined randomly as soon as the simulation starts.

Once the base node receives packets which are generated because of detecting tanks, the base node needs to launch missiles for the purpose of destroying tanks. We model the missile model in this example.

Like the tank model, the collision checker model is reused in this scenario. But, we modify the checking method according to this example. The original model checks for collision by measuring distances between each moving object and all the sensor nodes. However, the original collision checking method may decrease the computational power as the number of the sensor nodes increases since the model checks every sensor node with each moving object. In order to prevent the computational speed down, the collision checker model examines several sensor nodes which are close to each tank. In order to pick sensor nodes that are checked with one tank, a virtual position of a tank is decided by finding the closest position at which four sensor nodes meet from a tank's current position. Once four nodes are picked, the distances between a tank and a sensor node are measured. The sensor node which has the shortest distance to the tank is the node that is able to detect the tank. Ideally, we may expect $O\left(\frac{number\ of\ sensor\ nodes}{4}\right)$ speed up comparing to the original method that checks all sensor nodes.

The explosion checker model works like the collision checker model, but it is triggered when it receives messages through the input. The main objective of this model is to find sensor nodes and tanks destroyed by a missile attack.

5.3 Simulation Results

Under the circumstances of twenty five sensor nodes, five tanks, and one base station, two different sensor node's behaviors are simulated. The first simulation is done with conventional sensor node's behavior, and the second simulation is run with our proposed sensor node's behavior including sending alive packets. Excluding the difference of sensor node's behaviors, all the simulation variables are in the same condition. The energy consumption, the number of tanks destroyed, and the number of sensor nodes destroyed are measured during 1500 simulation time. If all the tanks are destroyed, the simulation needs to be stopped. However, unfortunately, our simulation can't destroy all the tanks for 1500 seconds of simulation time. The assumptions for the simulation are follows:

- All the sensor nodes and tanks are in the 100*100 simulation space.
- The sensor node sends alive packets with a 1 second simulation time interval.
- The sensor node covers one square size (20*20) of sensing area.
- The initial energy is set to 100 joules in all sensor nodes.
- It takes 5 seconds of simulation time for the missile to arrive at the destination position from the base station node.

- The missile's destructive radius is 7.
- The probability of destruction of both the sensor nodes and the tanks is 50% if the objects are in the missile's destructive area.
- The tanks are initially positioned arbitrarily and move on a randomly generated route before they get destroyed.

The numbers of destroyed sensor nodes in both simulations are the same and the destroyed tanks are also the same in both simulations. The same results in both simulations prove that alive packets don't affect the base station's behavior in regard to launching missiles. During 1,500 seconds of simulation time, eight sensor nodes and four tanks were destroyed. In addition to determining the number of sensor nodes and tanks destroyed during the simulation, the energy consumption required for networking activities is also measured. Since the assumption shows that alive packets are generated every second during 1,500 seconds, the sensor nodes, which have sending alive packet behavior, generate and send about 1,500 packets more than the conventional sensor nodes do. As a result, the sensor nodes in the second simulation consume more energy compared to the sensor nodes in the first simulation according to the number of alive packet sent. Table 3 is the chart of the total number of packets generated in both the first and second simulation.

Number of packets	**Simulation 1 (no alive packets)**	**Simulation 2 (with alive packets)**
Total	97,299	114,807

Table 3. Total number of packets

The energy consumed in each sensor node for both simulations is demonstrated in Table 4. All the sensor nodes which send alive packets consume more energy (about 18 percent) than the others. The energy consumption of the eight destroyed sensor nodes 2, 3, 7, 9, 12, 16, 18, and 21 are not measured.

Energy consumption (Joule)	**Simulation 1 (no alive packets)**	**Simulation 2 (with alive packets)**
Node 0	68.325	80.998
Node 1	68.325	80.98
Node 2	destroyed	destroyed
Node 3	destroyed	destroyed
Node 4	68.325	80.98
Node 5	68.325	81
Node 6	68.325	81.03
Node 7	destroyed	destroyed
Node 8	72.28	84.74
Node 9	destroyed	destroyed
Node 10	68.325	81

Node 11	72.44	84.732
Node 12	destroyed	destroyed
Node 13	72.51	84.795
Node 14	68.325	80.992
Node 15	72.98	85.22
Node 16	destroyed	destroyed
Node 17	72.92	85.161
Node 18	destroyed	destroyed
Node 19	325	81
Node 20	destroyed	destroyed
Node 21	68.325	85.03
Node 22	72.74	85.02
Node 23	72.1	84.44
Node 24	68.325	81.02
Total	1191.319	1408.089

Table 4. The energy consumption

Unfortunately, the simulation is stopped before any sensor node uses up its energy. But, the base station knows whether sensor nodes are deactivated because of the lack of energy after the alive packet interval time if the base station doesn't receive alive packets from those nodes. It helps for the base station to make a secondary reaction for deactivated sensor nodes. This simulation shows an advantage of DEVS/NS-2 compared with NS-2 by using dynamically simulated information. Through this simulation, it is recognized that the behavior of sending alive packets is disadvantageous to energy consumption, but the resulting behavior is advantageous when making decisions.

6. Comparison with Related Study

A previous related study presented a heterogeneous simulation framework using DEVS BUS [7]. The purpose of the DEVS BUS framework is to interoperate a collection of simulators which are developed in different environments such as DEVS, NS, and C++Sim [8]. The DEVS BUS framework is recapitulated first and, in turn, the DEVS/NS-2 is compared with the DEVS BUS framework. Also, advantages and disadvantages of OPNET are compared with those of DEVS/NS-2. The purpose of this section is to show the strengths of DEVS/NS-2. This section does not provide any sources and practical testing data. However, comparison between DEVS/NS-2 and DEVS BUS is based on our study and investigation in DEVS BUS, and our experience in OPNET supports reliable comparison between DEVS/NS-2 and OPNET.

6.1 Comparison between DEVS/NS-2 and DEVS BUS framework

The DEVS/NS-2 is implemented for integrating DEVS models into NS-2 models. However, the DEVS BUS framework is more flexible such that it extends the interoperability to not only NS-2 but also C++Sim and the other sequential simulator.

The lack of generality is the most disadvantageous aspect of the DEVS/NS-2 comparing to the DEVS BUS framework. DEVS/NS-2 has connections among DEVS models and NS-2 models through a one-to-one mapping method which means that one DEVS object is mapping with one NS-2 object. A DEVS model triggers its matching NS-2 model directly. As a result, the DEVS/NS-2 doesn't cause a bottleneck of message passing among DEVS models and NS-2 models. However, this simulation environment is still a centralized simulation scheme because there is one coordinator. On the contrary, every message is processed in one component, a centralized deliverer, to be delivered to destinations in DEVS BUS framework. This message passing mechanism results in a bottleneck. Consequently, we find the similarities between the DEVS/NS-2 and the DEVS BUS framework. First of all, both the DEVS/NS-2 and the DEVS BUS framework are centralized coupling. Scheduling approaches are very similar, too. As we presented earlier in this section, the DEVS/NS-2 has the NS-2 Event Queue Agent model that connects to the NS-2's event queue through the DEVS NS-2 Interface. The NS-2 Event Queue model controls the NS-2's event queue by triggering NS-2's functions to handle the event queue. The objective of this model is to synchronize DEVS and NS-2. Similar to the NS-2 Event Queue Agent model, the DEVS BUS framework contains one scheduler that handles every message among models for time synchronization. Table 5 shows the comparison between the DEVS/NS-2 and the DEVS BUS framework.

	DEVS/NS-2	**DEVS BUS framework**
Bottleneck	low	Higher than DEVS/NS-2
Generality	Not good(only with NS-2)	Good
Synchronization	Good	Good
Distribution simulation	No	No
Reusability	Good	Good
Developing cost	Good	Good

Table 5. Comparison between DEVS/NS-2 and DEVS BUS framework

The DEVS/NS-2 has one advantage and one disadvantage. Increasing the generality in order to integrate DEVS with not only NS-2 but also several different simulators is worthy of further research. Although the approaches and implementation issues are different between the DEVS/NS-2 and the DEVS BUS framework, the goals of increasing modeling power and reducing modeling development cost are the same.

6.2 Comparison between DEVS/NS-2 and OPNET

OPNET is a widely used commercial product and its availability is strictly limited. The simulator is available with full source code for all network component modules, but the code for the simulation engine is not supplied. Although there exist examples and extensive documentation, additional support, available through OPNET mailing list, requires additional maintenance license. Also, there are a number of additional OPNET packages that require separate purchase. The main programming language in OPNET is C (recent releases support C++ development). The initial configuration such as network

topology setup and parameter setting is usually achieved using a Graphical User Interface (GUI), a set of XML files or through C library calls. Simulation scenarios (e.g., parameter change after some time, topology update, etc.) usually require writing C or C++ code; although in simpler cases one can use special "scenario" parameters (e.g., link fail/restore time). OPNET modules implementations is very complex in contrast to NS-2's relatively well designed C++ objects. OPNET's implementations are monolithic and use a number of global variables. We had developed a network gateway model using OPNET, and we recognize that modifications of OPNET behavior are difficult and require significant effort and amount of time. Since OPNET's network equipment models are very detailed, and in fact the simulation process closely reflects the processing that happens in real-world equipment, OPNET's simulation is relatively slow and requires powerful workstations. The GUI conveniently models network elements and protocols; this is really helpful for a user in understanding how the things work, but the lack of scripting language is limiting from a developer's point of view.

NS-2 seems to be completely free for both educational and commercial purposes although some older code explicitly grants rights to educational type of use only. The simulator is available with a full source code, validation tests, a rich set of examples and a good manual. Moreover, additional support may be provided by the NS-2 user's mailing list. NS-2 has been used in a great number of research projects in academia and is the most often used simulator in research projects related to IP networks. The main reasons behind this popularity are the fact that it is usually free and easy to use and that many well-recognized scientists have contributed significant amount of work to this project. Thus, probably most modern features related to IP networks are implemented in NS-2 and even if not merged with the distribution, they could be found somewhere in the Internet. Many researchers contribute to NS-2 thus continually updating the simulator with reliable implementations for new protocols, queuing disciplines, etc.

In DEVS/NS-2 environment, the details of a low level network with protocol and component description are modeled by NS-2 while DEVS serves as controller by modeling the high level behavior of target network models and interaction of the associated actors. The primary advantage of the DEVS/NS-2 simulation comparing to OPNET simulation is that not only a network simulation but also a real environmental simulation is possible. On one hand, because OPNET simulation considers only packet transmitting related simulation such as network protocol development, transmitting delay, and so on, a receiving node just discards received packet in the simulation. On the other hand, in DEVS/NS-2 simulation, a node doesn't discard packets that may include very important data. Once a node receives packets, it may use information for secondary reactions. As a result, the DEVS/NS-2 is capable of doing what OPNET alone cannot do.

DEVS/NS-2 environment reuses the network protocol libraries of NS-2's. The remarkable advantage of DEVS/NS-2 compared to OPNET is usually free for the NS-2's network libraries and an additional support. The NS-2's libraries appear to be as good as the libraries of OPNET since, as mentioned earlier, they are updated with a reliable implementation for new protocols by many researchers in academia. Were the source codes of OPNET simulation engine to be available, an integration with OPNET would be a promising extension. (Another approach to integrate with OPNET might be to use middleware such as HLA. DEVS may be possible to be integrated with OPNET if OPNET opens channels to be connected to HLA.) The above reasons show the

advantages of DEVS/NS-2 environment. Table 6 shows the comparison of DEVS/NS-2 and OPNET.

	OPNET	**DEVS/NS-2**
Cost	highly expensive commercial software	completely free, open-source software
Speed	quite slow	Fast
Availability	available with source code for simulation modules (except for restricted protocols).	available with full source code, validation tests and examples.
Support	- excellent manual - mailing list (maintenance license required) - source code and examples	- good manual - publicly available mailing list - source code and examples
Scenario	- GUI, XML, imports - scenario parameters - Static (pre-defined)	- OTcl scripts (or C++) - DEVS modeling - Dynamic
Components	- C/C++	- DEVS (higher level) - C++ NS-2 libraries (lower level)
Summary	- widely used in military projects - run on powerful workstations	- popular in research projects in academia - well-defined mathematical formalism specification

Table 6. Comparison of DEVS/NS-2 and OPNET

As well as the comparison factors in the table 6, one of the most important categories to compare between two simulation environments is target users. The ideal simulation tool means different things to different users. If users are network managers or network designers, the users may operate the networks, troubleshoot and solve problems and want to specify and build new networks or improve existing ones. And users want their networks to meet performance requirements and reduce design time and enhance accuracy. In order for network designers or network managers to achieve their goal, an extensive network library and accurate network simulation tools are needed. However, the programmers' view should be different from the views of network designers or managers. To develop or modify protocols, devices, and traffic models with reducing development costs and risks, the full control of the simulation engine at the programming language level is necessary. OPNET includes easily usable hierarchical graphic user interfaces to manage and diagnose networks [9]. But, OPNET doesn't open its simulation engine's source code so that programmers feel it is difficult to develop new protocols, devices, and so on. On the other hand, DEVS/NS-2 opens source codes of the simulation engine. As a result, programmers don't have any limitation to develop network components by accessing the engine. The open source nature makes DEVS/NS-2 maximize composibility. The flexible composibility of DEVS/NS-2 strongly appeals more to programmers' demand for developing applications and protocols than either network managers or network designers. The freeware nature of DEVS/NS-2 is attractive

to researchers compared to the need to enter into an OPNET license agreement and associated costs. In addition, DEVS/NS-2 overcomes the NS-2's drawback, which is weakness of the graphic user interface.

Also, balancing simulation power and ease of use is a critical criterion of simulators. Generally, the learning curves for each of the simulators are different, and powerful simulators have steeper learning curves than simple simulation tools. OPNET is a very heavy simulation tool so that it provides well organized graphic user interfaces and results in high accuracy. But we need to trade off accuracy and execution speed because execution speed comes from the complexity of a system architecture. A bit higher learning curve leads to difficulty in implementing one's own algorithms. In contrast, DEVS/NS-2 is much simpler than OPNET. Consequently, the learning curve of DEVS/NS-2 is trivial comparatively. The small learning curve makes it easy to develop network protocols or network devices. Sometimes, OTcl and DEVS are unfamiliar with users, but the target users of DEVS/NS-2 are researchers and programmers, and they are assumed to know or are eager to learn how to customize scenarios with OTcl and DEVS.

In this section, DEVS/NS-2 and OPNET are compared. The advantages and the disadvantages for each of the simulators are investigated, and the target users of DEVS/NS-2 are presented in turn.

7. Conclusion and Future Works

The main objective of this research is to design and implement an interoperable simulation with DEVS and NS-2. DEVS and NS-2 models can participate in a simulation, and their appropriate role portions are decided in order to increase model reusability and reduce the model development cost. To accomplish these goals, mainly the time synchronization between two simulators and the aggregating method to match pairs of DEVS and NS-2 models are required. Our approach to synchronize both DEVS and NS-2 scheduling methods is to form one DEVS model which includes the event queue of NS-2 and handles inserting and removing activities. As a result, the NS-2 process looks like the DEVS simulator's behavior. In addition, DEVS models and their matching NS-2 models are connected with each other by applying a one-to-one mapping system.

To prove the concrete integration of the DEVS/NS-2, a very simple scenario in a wireless sensor network topology is built. The number of generated packets and the energy consumption for each sensor node in both the DEVS/NS-2 simulation and the NS-2 alone simulation are measured. The exact same results in two simulations conclude the success of the integration.

There are a few issues which are worthy of further research. The issues include refining models and generalizing the interface with DEVS and other simulators, and expanding to a distributed simulation. Refining models allows simulations to be more reasonable. The behaviors of models which are designed in this paper are not realistic but virtually assumed. If the real battle field data such as behaviors of tanks and sensor nodes can be applied to the models, a simulation increases its realism.

The second issue is the generalization of the interface with DEVS and other simulations. The DEVS/NS-2 is integrated with DEVS and NS-2 only. The integration with other network simulators such as OPNET and C++Sim increases modeling power and reduces model development cost. The third issue is to validate the concreteness of the

DEVS/NS-2 environment. In the previous section, we measured the energy consumption and the total number of packets generated in both the DEVS/NS-2 simulation and the stand-alone NS-2 simulation, and concluded that the DEVS is integrated with NS-2 is valid because the simulation results are exactly the same. However, we need to model and simulate the several examples more to gain confidence in the validity of the DEVS/NS-2 environment. For future work, we intend to provide a proof of correctness of the implementation using the DEVS Abstract Simulator concepts. An example of this approach appears in [10].

Lastly, expanding to a distributed simulation is a promising area for further research. The sequentially processed interoperable simulation may cause severe speed degradation as the size of the network increases. To get faster results in a large scale network topology simulation environment, a distributed simulation may provide a simulation such as in, [11, 12].

Through further research on those issues, it is expected that the DEVS/NS-2 will become a reliable simulator for general network related simulations.

8. References

[1] K. Fall and K. Varadhan, "*The ns Manual*," http://www.isi.edu/nsnam/ns , 2005

[2] B.P. Zeigler, T.G. Kim, and H. Praehofer, "*Theory of Modeling and Simulation,*" 2nd ed., New York: Academic Press, 2000.

[3] OPNET Technologies, OPNET Modeler, Commercial, Information, http://www.opnet.com/products/modeler/home.html

[4] E. Cayirci, R. Govindan, T. Znati, and M. Srivastava, "Wireless Sensor Networks," *Computer Networks*, Volume 43, Issue 4, pp. 417-419, November 2003

[5] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," *IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pp. 1567-1576, June 2002

[6] T.V. Dam and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," *ACM SenSys'03*, Los Angeles, California, November 2003

[7] Y.G. Kim, J.H. Kim, and T.G. Kim, "Heterogeneous Simulation Framework Using DEVS BUS," *SIMULATION,* Vol. 79, Issue 1, January 2003

[8] M. Little, C++SIM "User's Guide Public Release 1.5," http://cxxsim.ncl.ac.uk

[9] Gilberto Flores Lucio, Marcos Paredes-Farrera, Emmanuel Jammeh, Martin Fleury, and Martin J. Reed, "OPNET Modeler and Ns-2: Comparing the Accuracy Of Network Simulators for Packet-Level Analysis using a Network Testbed." *WSEAS Transactions on Computers*, No.3, Vol. 2, July 2003, ISSN 1109-2750, pp. 700-707.

[10] J. Nutaro, "Time Management and Interoperability in Distributed Discrete Event Simulation," in *The Department of Electrical and Computer Engineering*: M.S. Thesis, the University of Arizona, 2003

[11] S. Park, "Cost-Based Partitioning For Distributed Simulation of Hierarchical Modular DEVS Models," in *The Department of Electrical and Computer Engineering*: Ph.D Dissertation, the University of Arizona, 2003

[12] B.P.Zeigler, Yoonkeon Moon, Doohwan Kim, and Jeonggeun Kim, "DEVS-C++: A High Performance Modeling and Simulation Environment," *29th Hawaii International Conference on System Sciences*, pp. 350, 1996

Authors Biography

Taekyu Kim is currently a research engineer at SK C&C CO., LTD., Seoul, Korea. He received Ph.D. in Electrical & Computer Engineering (ECE) at the University of Arizona. He holds an M.S. (2006) in ECE from the University of Arizona and B.S. (2000) in Computer Science and Engineering from the Chung-Ang University, Seoul, Korea. His research interests include DEVS (discrete event system specification) based hybrid system modeling, model based system design, ontology methodology, data engineering, the Semantic Web, web services, and ubiquitous computing.

Moon Ho Hwang is currently a research assistant professor of Electrical and Computer Engineering at the University of Arizona, Tucson, AZ. He received BS (1990) from Hong-Ik University, Seoul, Korea, and MS (1992) and PhD (1999) from Korea Advanced Institute of Science and Engineering (KAIST), Taejon, Korea, all in departments of Industrial Engineering. Since 1998 to 2003, he has developed several commercial simulators for automated manufacturing systems as the director of a Simulation and Control Group at the research center of CubicTek Ltd. Co., Seoul, Korea. His research interests include system analysis of DEVS (discrete event system specification) and Timed Automata, multi-model formalism with timed Petri-net and DEVS, and scheduling and optimal control of manufacturing systems, workflow systems, and business process management systems.

Doohwan Kim is the founder and CEO of RTSync Corp, a commercial spinoff of Arizona Center for Integrative Modeling and Simulation at the University of Arizona.
He holds research professorship at the University of Arizona.