Tutorial 4: Vehicle Speed Control and Service Announcement

Vehicle Speed Control and Service Announcement

Prof. Sangyoung Park

Module ”Vehicle-2-X: Communication and Control“

Let‘s make a new WaveApplFile (cc and h)

- New-> Class (OMNet++)
  - VehicleControlApp.cc and VehicleControlApp.h are generated
- Let‘s copy the contents from MyVeinsApp.cc/h
  - Veins/src/veins/modules/application/traci/
- But of course, you should change the file content to reflect the name change

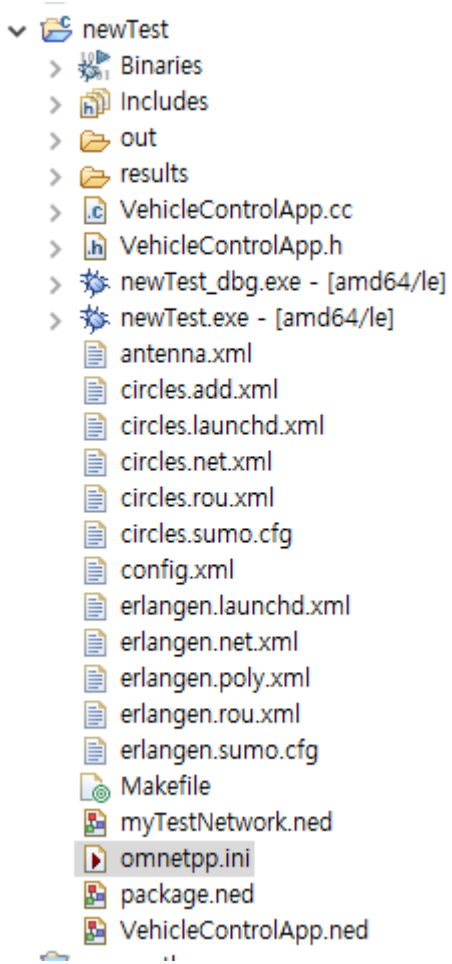
```
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see http://www.gnu.org/licenses/.
//

#include "VehicleControlApp.h"

Define_Module(VehicleControlApp);

const simsignalwrap_t VehicleControlApp::mobilityStateChangedSignal = simsig

void VehicleControlApp::initialize(int stage){
    BaseWaveAppLayer::initialize(stage);
    if (stage == 0) {
```



Let‘s make a new WaveApplFile (ned)

- New -> Network Description File (NED)
  - Again, copy the contents from MyVeinsApp.ned to VehicleControlApp.ned and fix the names accordingly
  - But, there will be errors!
  - We need to add a ned file called „package.ned“ in order to be able to use „package newTest“ in the MyVeinsApp.ned file
  - In the package.ned file, you also need „**package newTest;“**
  - The name you write in „package.ned“ should appear in the VehicleControlApp.ned in order to avoid error

```
//

package newTest;
import org.car2x.veins.modules.application.ieee80211p.BaseWaveApplLayer;

//
// network description file for your Veins Application. Add parameters here
//
simple VehicleControlApp extends BaseWaveApplLayer
{
    @class(VehicleControlApp);
    string appName = default("My first Veins App!");
}
```

Let‘s make Wave Service Announcements (WSA)

- Let‘s make the first car, which appears on the map, to make the service announcement (WSA)
- We can make use of startService() to start a WAVE service
- However, we don‘t want every car to start their own services

```
void VehicleControlApp::initialize(int stage){
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {
        //Initializing members and pointers of your application goes here
        EV << "Initializing " << par("appName").stringValue() << std::endl;
        mobility = TraCIMobilityAccess().get(getParentModule());
        //traci = mobility->getCommandInterface();
        traciVehicle = mobility->getVehicleCommandInterface();
        //findHost()->subscribe(mobilityStateChangedSignal, this);
        subscribedServiceId = -1;
        currentOfferedServiceId = 7;

        wsaInterval = 5;
        beaconInterval = 0.1;

    }
    else if (stage == 1) {
        //Initializing members that require initialized other modules goes here
        if (getId() == 14){
            // this is the head vehicle
            startService(Channels::SCH2, currentOfferedServiceId, "Platoon Lead Vehicle Service");
            //scheduleAt()
            scheduleAt(computeAsynchronousSendingTime(beaconInterval, type_CCH),sendBeaconEvt);
        }
    }
}
```

Let‘s make Wave Service Announcements (WSA)

- During the initialization of each car node, we check for the ID by using myId()
- If it‘s the first car, we call startService(), you can put any number for currentOfferedServiceId
- Types of WAVE messages available in Veins
  - Wave service message (WSM)
  - Wave service announcement (WSA)
  - Basic safety messages (BSM)
- If you go inside the function startService(), you will see that WSA will be scheduled using scheduleAt() for the next CCH period

```
void BaseWaveApplLayer::startService(Channels::ChannelNumber channel, int serviceId, std::string serviceDescription) {
    ....
    mac->changeServiceChannel(channel);
    currentOfferedServiceId = serviceId;
    currentServiceChannel = channel;
    currentServiceDescription = serviceDescription;

    simtime_t wsaTime = computeAsynchronousSendingTime(wsaInterval, type_CCH);
    scheduleAt(wsaTime, sendWSAEvt);
}
```

Let‘s make Wave Service Announcements (WSA)

- If you are wondering what scheduleAt() function would end up, it ends up in the following function
- It fills up the message (populateWSM), and sends the message to MAC layer (sendDown)
- It‘s going to schedule WSA periodically (period is wsaInterval) once a service is started

```
void BaseWaveApplLayer::handleSelfMsg(cMessage* msg) {
    switch (msg->getKind()) {
    case SEND_BEACON_EVT: {
        BasicSafetyMessage* bsm = new BasicSafetyMessage();
        populateWSM(bsm);
        sendDown(bsm);
        scheduleAt(simTime() + beaconInterval, sendBeaconEvt);
        break;
    }
    case SEND_WSA_EVT:   {
        WaveServiceAdvertisment* wsa = new WaveServiceAdvertisment();
        populateWSM(wsa);
        sendDown(wsa);
        scheduleAt(simTime() + wsaInterval, sendWSAEvt);
        break;
    }
```

Let‘s send Velocity Information

- Let‘s go back to initialize function
- We update the member variables curPosition and curSpeed defined in class BaseApplLayer using the traCI interface (mobility)
- We make a WAVE packet, in this case a BSM, populate the packet, and schedule to send it later
- It is likely that we would require this information periodically
- Fortunately, there is already a mechanism in the WaveApplLayer

```
else if (stage == 1) {
        //Initializing members that require initialized other modules goes here
        if (getId() == 14){
            // this is the head vehicle
            startService(Channels::SCH2, currentOfferedServiceId, "Platoon Lead Vehicle Service");
            //scheduleAt()
            scheduleAt(computeAsynchronousSendingTime(beaconInterval, type_CCH),sendBeaconEvt);
        }
    }
```

Periodic Transmission of WAVE Messages

- Remember „scheduleAt()“ function from the OMNet++ Tictoc tutorial?
  - scheduleAt() is used for self-messages
- In BaseWaveApplLayer.cc, there is function handleSelfMsg()
  - Once scheduleAt is used with either SEND_BEACON_EVT or SEND_WSA_EVT kind of cMessages, it‘s going to be re-scheduled periodically

```
void BaseWaveApplLayer::handleSelfMsg(cMessage* msg) {
    switch (msg->getKind()) {
    case SEND_BEACON_EVT: {
        BasicSafetyMessage* bsm = new BasicSafetyMessage();
        populateWSM(bsm);
        sendDown(bsm);
        scheduleAt(simTime() + beaconInterval, sendBeaconEvt);
        break;
    }
    case SEND_WSA_EVT:   {
        WaveServiceAdvertisment* wsa = new WaveServiceAdvertisment();
        populateWSM(wsa);
        sendDown(wsa);
        scheduleAt(simTime() + wsaInterval, sendWSAEvt);
        break;
    }
    default: {
        if (msg)
            DBG_APP << "APP: Error: Got Self Message of unknown kind! Name: " << msg->getName() << endl;
        break;
    }
    }
}
```

Periodic Transmission of WAVE Messages

- So, how do we initiate the periodic transmission of BSM?
  - We schedule the first sendBeaconEvt in intialize()

```
void VehicleControlApp::initialize(int stage){
    BaseWaveApplLayer::initialize(stage);
    if (stage == 0) {
        //Initializing members and pointers of your application goes here
        EV << "Initializing " << par("appName").stringValue() << std::endl;
        mobility = TraCIMobilityAccess().get(getParentModule());
        //traci = mobility->getCommandInterface();
        traciVehicle = mobility->getVehicleCommandInterface();
        //findHost()->subscribe(mobilityStateChangedSignal, this);
        subscribedServiceId = -1;
        currentOfferedServiceId = 7;

        wsaInterval = 5;
        beaconInterval = 0.1;

    }
    else if (stage == 1) {
        //Initializing members that require initialized other modules goes here
        if (getId() == 14){
            // this is the head vehicle
            startService(Channels::SCH2, currentOfferedServiceId, "Platoon Lead Vehicle Service");
            //scheduleAt()
            scheduleAt(computeAsynchronousSendingTime(beaconInterval, type_CCH),sendBeaconEvt);
        }
    }
}
```

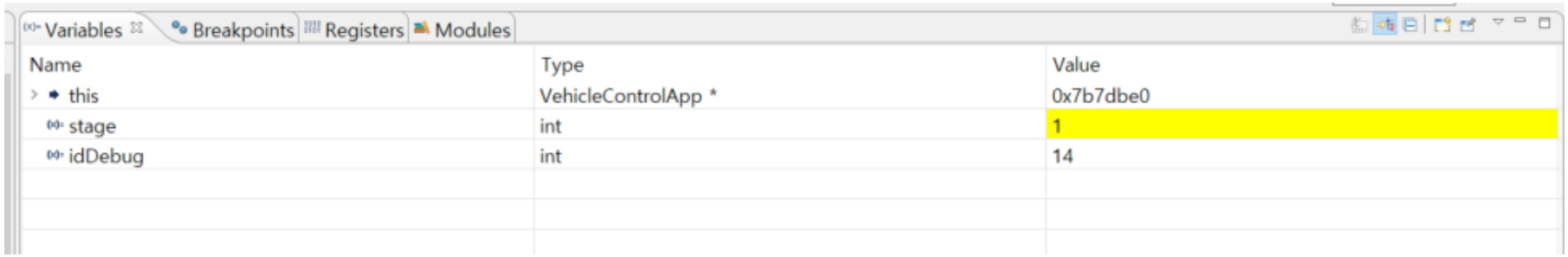
Using Debugger

- Wait, how did I know the ID of the first car would be 14?
- Let‘s use the debugger
- Add the line in the red rectangle to the source code
- Double click on the left to create a „breakpoint“
  - A small blue dot will appear
- Omnetpp.ini (right click) -> debug as -> omnet++ simulation

```
    }
    else if (stage == 1) {
        //Initializing members that require initialized other modules goes here
        int idDebug = getId();
        if (getId() == 14){
            // this is the head vehicle
            startService(Channels::SCH2, currentOfferedServiceId, "Platoon Lead Vehicle Service");
            //scheduleAt()
            scheduleAt(computeAsynchronousSendingTime(beaconInterval, type_CCH),sendBeaconEvt);
        }
    }
}
```

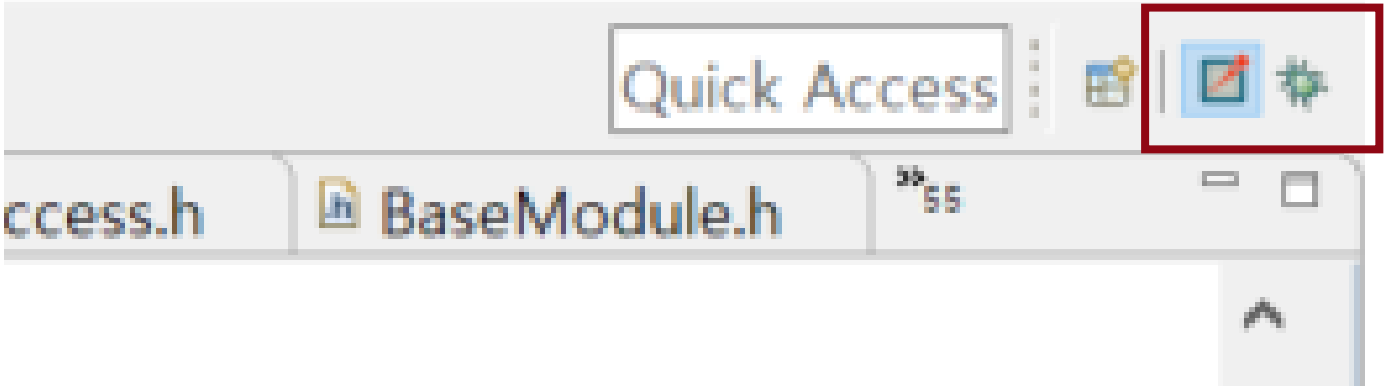
Using Debugger

- The perspective of the Omnet IDE changes to the „debug perspective“
- If you run, the program will stop at the breakpoint
- If you lay your mouse cursor on top of idDebug, you will be able to see the value of the variable
  - Or, you can look into the sub-window in the top-right corner to find „variables“ window to read the value of the variables or the member variables of the current object (this)
- In my case, the value was 14



Using Debugger

- Changing perspectives, if you want to exit the debugger perspective, you can click on the small buttons on the top-right corner to change perspectives



Oops, errors exist

- We haven‘t allowed the usage of SCH
  - We can configure such parameters in the omnetpp.ini file

```
##########################################################
#            11p specific parameters                     #
#                                                        #
#                    NIC-Settings                        #
##########################################################
*.connectionManager.sendDirect = true
*.connectionManager.maxInterfDist = 2600m
*.connectionManager.drawMaxIntfDist = false

*.**.nic.mac1609_4.useServiceChannel = true

*.**.nic.mac1609_4.txPower = 20mW
*.**.nic.mac1609_4.bitrate = 6Mbps
*.**.nic.phy80211p.sensitivity = -89dBm

*.**.nic.phy80211p.useThermalNoise = true
*.**.nic.phy80211p.thermalNoise = -110dBm

*.**.nic.phy80211p.decider = xmldoc("config.xml")
*.**.nic.phy80211p.analogueModels = xmldoc("config.xml")
*.**.nic.phy80211p.usePropagationDelay = true

*.**.nic.phy80211p.antenna = xmldoc("antenna.xml",
"/root/Antenna[@id='monopole']")
```

Changing the Speed of the Lead Vehicle

- The following function handles the position changes in SUMO and reflect the changes to the Veins simulation platform
  - The function is automatically invoked everytime there is a change of movement in the cars in SUMO simulator
- Hence, if we change the status of the lead vehicle (id == 14) according to time, we can control the movement of the vehicle

```
void VehicleControlApp::handlePositionUpdate(cObject* obj){
    BaseWaveApplLayer::handlePositionUpdate(obj);


    if ( this->getId() == 14 ){
        const simtime_t t = simTime();
        if ( t == 10 ) {
            traciVehicle->setSpeedMode(0x1f);
            traciVehicle->setSpeed(0);
        }
        else if ( t == 20 ) {
            traciVehicle->setSpeedMode(0x1f);
            traciVehicle->setSpeed(20);
        }
    }

}
```

Behavior of vehicles upon receiving a BSM

- Upon receiving the BSM from the leading vehicle, we can adjust the speed of the vehicle
- Why do I use .length() for speed?
  - Try to find out by exploring the „Coord“ class

```
void VehicleControlApp::onBSM(BasicSafetyMessage* bsm){
    Coord& leadVehicleSpeed = bsm->getSenderSpeed();

    traciVehicle->setSpeedMode(0x1f);
    traciVehicle->setSpeed(leadVehicleSpeed.length());
}
```

Simulation Speed

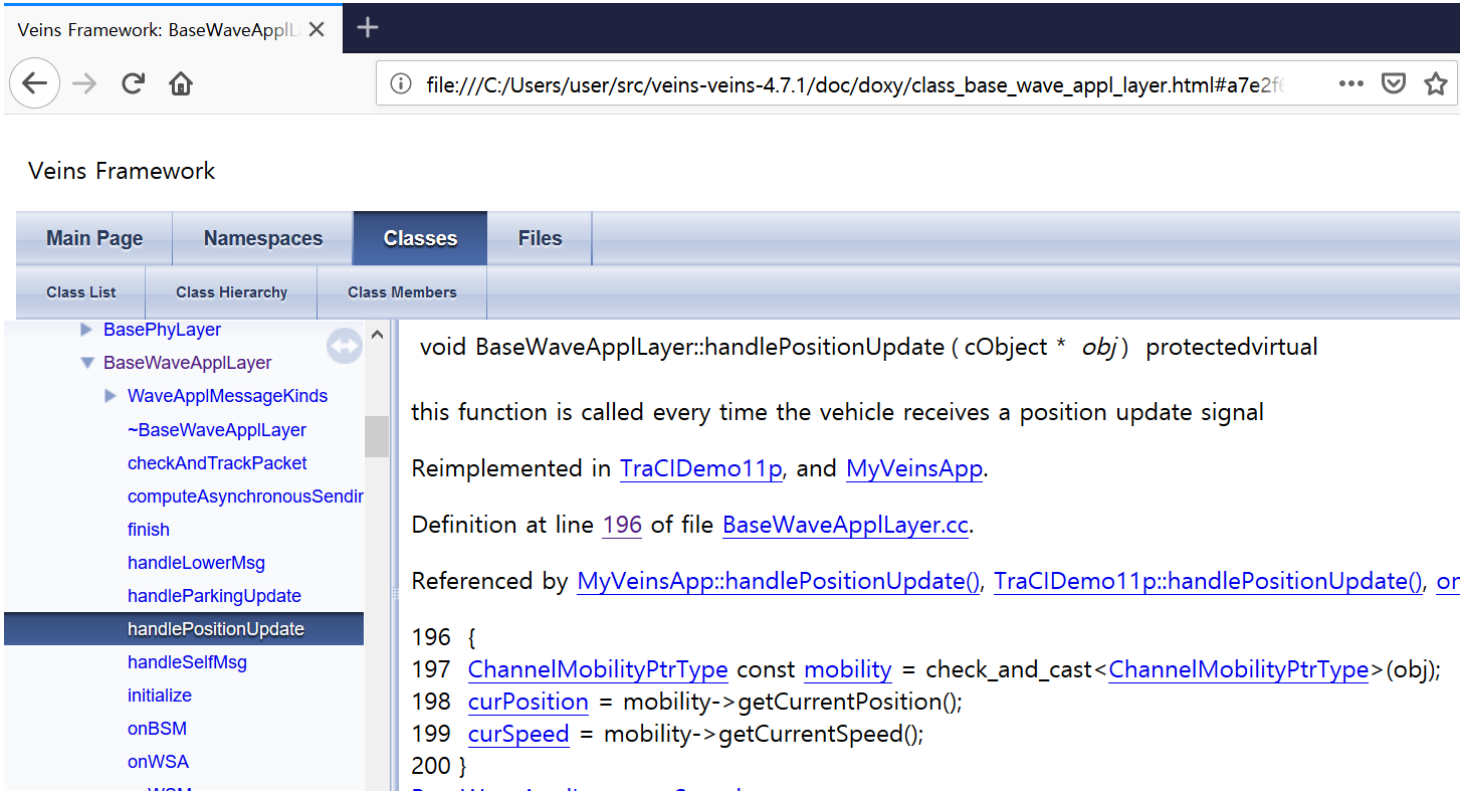
- Now, the speed of the lead vehicle is shared with other vehicles every 0.1 s
- There‘s too much animation going on to describe the packet movements
- We can speed up the simulation by adjusting the amount of animation we want to view
- In the omnet simulation environment
  - Simulate -> Fast run / Express run
  - Simulation will be performed with increased speed without executing the animations

Reading Out the Simulation Results

- https://sumo.dlr.de/wiki/Tools/Visualization
- Let‘s try to get some graphs!
  - (But I have to work on it... Sry)

Documentation of Veins Library

- In folder doc/doxy/
  - There is some documentation available on the veins library
  - (To be honest, it‘s not that useful)



Implementation of Simple Platooning Algorithm

- Let‘s start with something simple
- Let‘s read the distance to the preceding vehicle only and try to adjust the acceleration of the current vehicle
- Would you be able to implement this?
- $a = p \cdot (d - d_{desired})$