**Background and Description:**

OMNet++ is an extensible, modular, component-based C++ simulation library and framework, often used to build network simulators. We mainly accessed this library through the INET Framework, an open-source model suite for OMNeT++. It can be used for any of wired, wireless, or mobile networks; wireless networks ended up being the most useful for our work. We built upon research done in Dr. Xenofon Koutsoukos’ lab and by Himanshu Neema that is related to OMNet++ and INET. We have also tried to incorporate Ben’s research on cyber physical system security (attacks and attack detection in a CPS system, the Tennessee-Eastman plant) to complete this project and make a useful contribution to Himanshu’s work.

We used OMNet++ with the INET Framework to create a model of the sensors and actuators, along with the controller and physical process. Sensors in this case would measure values such as the temperature, pressure, level of the liquid in the tank, and various flow rates in and out. Meanwhile, actuators control how open (generally percentage-wise) a valve is, determining rates of flow for various chemicals. The job of the controller is to receive sensor measurements as inputs, run some number of calculations, and determine those percentage values to use to control the actuators. Finally, the physical process is literally what is happening within the plant, which is where the sensors determine their next values. This starts the cycle over again.

**The Problem:**

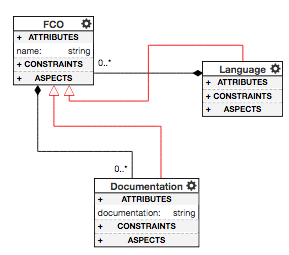
In general, with no outside interference, the model described above would work perfectly and constantly update all variables properly. The problem comes in when malicious attackers try to intercept and modify the values, known as a man-in-the-middle attack. Our main goal coming in was to represent these in the INET Framework, largely for the eventual purpose of studying defense strategies against the attacks. For example, one method has been to establish some kind of prediction for a sensor value based off of the other sensors as well as the actuators. This allows a defender to compare that value to the value being given by the sensor. Finding the difference between the values, we can keep track of these differences over time and trigger an alarm if the sum exceeds a certain threshold. The overall problem this model tries to solve, therefore, is the defense of a cyber physical system. While real-world datasets and physical testbeds are also needed to verify results, simulations - such as this one - are a useful tool to test defense strategies and observe how a system would react. A lot of work would still need to be done to truly achieve this goal. For example, the controller and physical process would need algorithms based off of real world examples to accurately interpret sensor and actuator values and deliver correct changes based off of the results. However, we still believe that the framework established here will be a solid cornerstone for that future work.

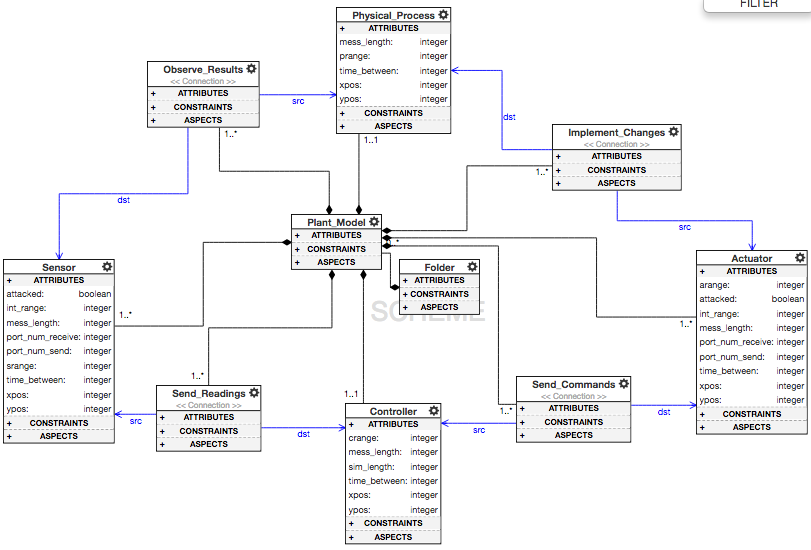
**Domain Model:**

The WebGME component of the process is fairly simple. The meta model largely consists of the four major components discussed above, plus unique connections between each component. The physical process and controller are constrained to only allow exactly one of each in any individual project, while there must be at least one sensor and actuator but the limit is theoretically boundless; as far as we are aware, there is no limit in OMNet++. Each node has a variety of attributes common to all of them, such as their positions on an (x,y)-plane, the range of their signals (in meters), length of the messages being sent (in bytes), and the expected time between each message being sent (in ms). Two unique ports are also established at each sensor or actuator node. One of these will become the port number for that particular node, while the other is the number that will be reserved for the node at the physical process or controller as appropriate. Many of these attributes have appropriate default values based on our observations, but they can be modified as necessary. There is also an attribute for the length of the resulting OMNet++ simulation. Finally, the sensors and actuators have boolean attributes for attacks, with a range of interference attribute that is only relevant if attacked is TRUE.

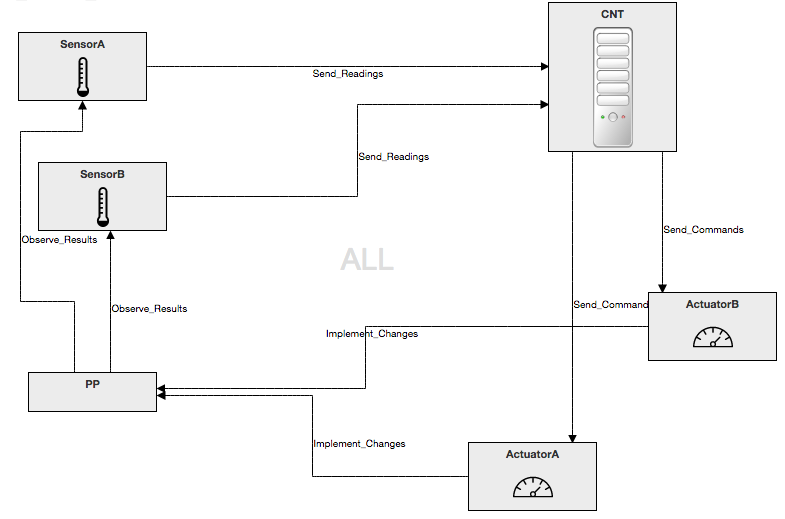
The more interesting piece of the WebGME model is the plugin. The plugin is part of the meta definition of the Plant\_Model component, within which the four main types of node along with their connections are contained. When run, the plugin first loads the node map starting at the current Plant\_Model node, and then gathers the attributes of each sensor, actuator, physical process, and controller. From this data and some general formatting, we are able to create a few (very long!) strings. The strings then go through a JSON.stringify conversion and are saved into the 2 files we need to run the OMNet++ simulation. The first is a .ini file that contains a variety of initialization data for the model, such as the number of ports each node will need and how the ports communicate with each other. The other is a .ned file, which stands for network definition. This file mainly just places the nodes to their proper locations in the (x,y)-plane.

The two pages of the meta-model:

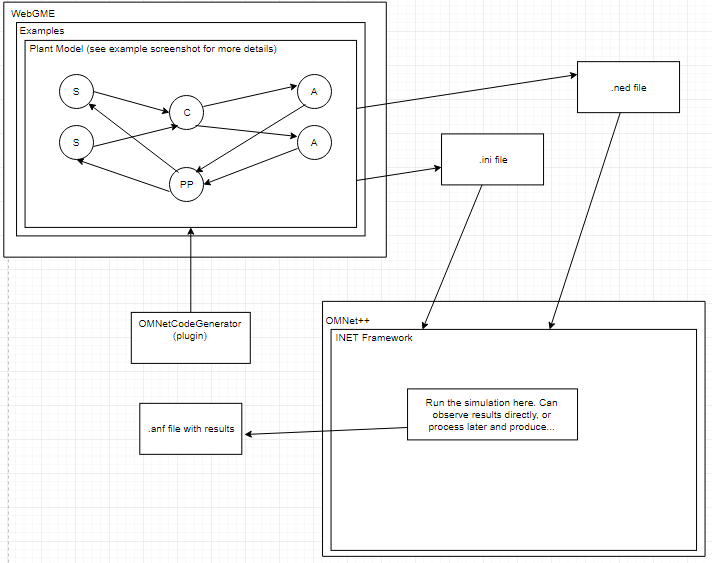


****

Small example similar to the one shown in class, now with physical process PP. Sensor B and Actuator B are attacked; we’ll see the results of that later.

****

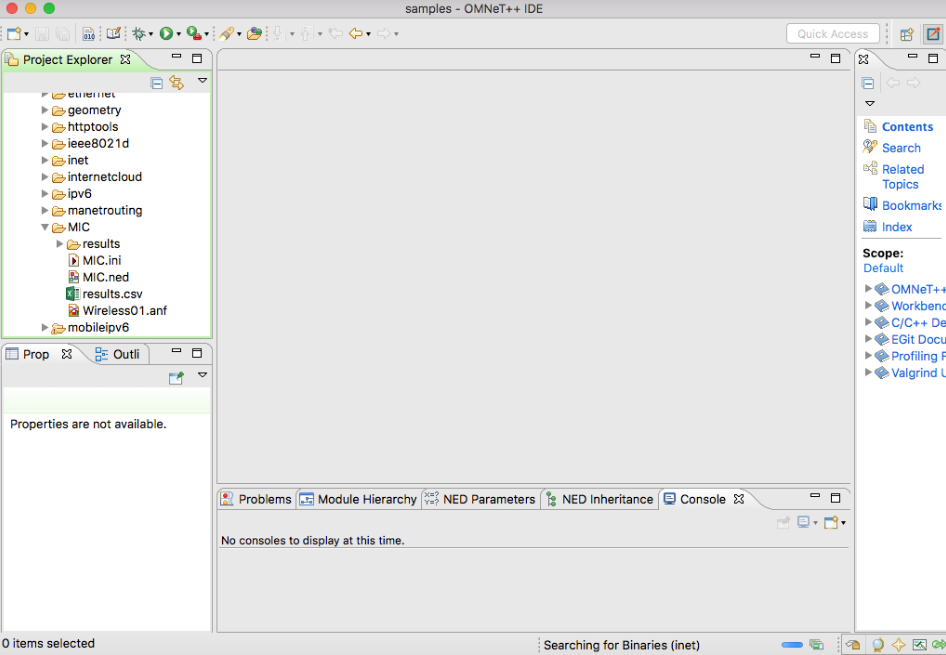
**High-level Description:**

****

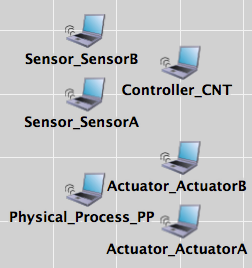
**Example (based on Plant\_Model\_1211 in our WebGME project):**

We built a simple example model with two sensors, two actuators, and the required physical process and controller, with appropriate connections between them. We changed the simulation length to be five seconds longer than usual and input appropriate port numbers for the sensors and actuators to send and receive signals. We also chose positions for the nodes on the (x,y)-plane; mainly just to get a good screenshot, but also so that nodes sending signals to each other are within the defined range and so that some interference will be present for our attack. Finally, we set the “attacked” values on SensorB and ActuatorB to TRUE. The WebGME model can be seen above.

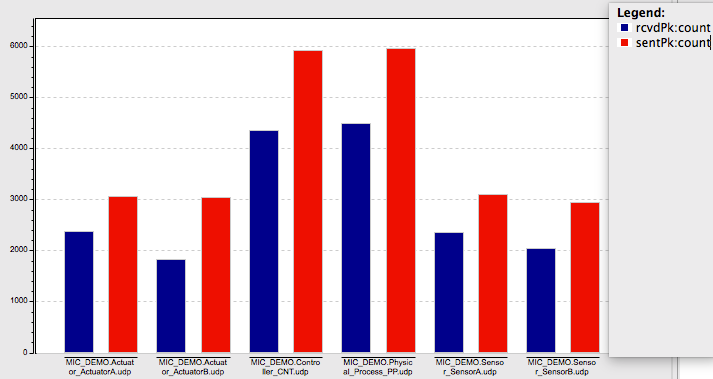
Once the model is created, we can simply navigate to Plant\_Model\_1211 and run the OMNetCodeGenerator plugin. This will generate the MIC.ini and MIC.ned files that we will need for a new OMNet++/INET project. We then download the two files and move them into an MIC directory set up within the omnetpp folder. The next step would be to initialize OMNet from the terminal, which opens up the IDE (based on Eclipse), as can be seen here:



We’ll next have the visualization of the OMNet model (just the important part):



Finally, we can run this model and generate the results after the 30 seconds of simulation have passed. The results from OMNet are pretty much a mess; we included the full spreadsheet in the repository, but couldn’t make much sense of it ourselves. Within the IDE, you can do a much better job of selecting individual attributes of interest and displaying them in some way. That is what we have tried to do here with this graph:



Here, we compare the number of sent and received data packets at each node. While some variation is understandable, with the same number of sensors and actuators, we expect the sent values values for the controller and process to be similar. We expect the same for the received values (received will always be somewhat less than sent due to nodes being busy when packets are sent out and other small effects that compound together). The main objects of interest here are the received packets at the sensors and actuators. There appears to be a moderate drop off in quantity when comparing the attacked sensor/actuator to those that were not attacked, as we would predict.

**Wrap Up:**

We were able to finish up the model with the physical process and sensor attacks as we hoped. Unfortunately, we were not able to get messages sent with actual appropriate values, nor were we able to launch attacks modifying those values. It appears to be possible based off of our research into the matter, but with the time remaining we were unable to really complete it, and especially could not make it open-ended enough to include in the WebGME model as attributes. We also ran out of time to incorporate some sort of automation to the process; a user of this would need to download and relocate the files to the correct location each time. It is a minor inconvenience overall, but can still be quite frustrating. We hope that this can at least be a good baseline for future work in the area of cyber physical systems security that makes use of both WebGME and OMNet++/INET.