Ryal O'Neil

11526252

4/28/2020

The first major hurdle I faced when working on this project was the website not working properly. Something had gone wrong with the formatting, preventing me from copying, but eventually I got that sorted by restarting my computer. Once I could pull the data off the website, I started working in C++ on Linux. After a few days of banging my head against the wall on that, the generated C++ code being unwieldy and hard to understand I shifted to working with Python on windows.

The first part of that I tackled was making the XML file. I used the python documentation and the RTI_ConnextDDS_CoreLibraries_XML documentation to piece that together, modifying the example code given as well as using the rtiddsgen -converttoxml to build an XML file from the IDL file. Most of that was fairly straightforward to understand. There was a lot of trial and error with how to properly format the file, once more the documentation on RTI connector for python was really useful. The hardest part of this was formatting the domain and participant libraries, but the python documentation had good examples I could tweak and play with. Once I had the XML in something resembling a functional state, at least what I thought was one at the time I moved onto the python code.

Once I got to the actual python code part of it, the python documentation and the examples on their github proved invaluable. After fiddling with it for a while I got a writer and a reader to talk to each other in python. From there I thought it will just be playing with the structure. For the most part I was right. Once I had proof of concept I worked on building the bus code. This was fairly straightforward once I figured out how to set a variable, send the message, get the message and then read the variable. The biggest struggle with the buses was the properties file and making them run in parallel.

I looked at a few approaches for making the buses run at the same time, however this is complicated by the fact that python does not actually allow multithreaded execution. There were two possible work arounds. One is to make them run in their own processes using a manager. The other and the one I went with was using asyncio. Asyncio isn't actually parallel, but when you are using code that gets stopped by waiting for network replies and sleeps, or anything else that is not constrained by the processor it might as well be. It works by putting two or more functions into a queue of sorts, and including awaits in said functions, when ever an awaited function happens while the program is waiting for it to return it allows other parts of the program to execute. In this case when the bus uses a sleep to simulate transit time, it switches to the next bus, as the actual computations and message sending for the bus is so fast it might as well be instant.

The other problem to overcome was the properties file. However as python is very flexible about setting variables, so by making it properties.py it can be included like any other code file.

Then all I had to do was put quotes around the strings to make them readable and referenceable. This brings up one of the reasons I rather like python, you can have the same var or function in two different files and it will let you call either. When I was trying to make this work with C++ and using the code generator both accident and position were generating with the same functions making it a mess all over.

The only major limitation I ran into with the python rti connector was trying to implement the topic filter. The short version is, you can't. So I made do and simulated it by having the subscriber filter the messages rather than having the publisher not send them to that subscriber. I wish I could have implemented this, but it just doesn't seem possible to my understanding reding the documentation. Beyond that I think I have it all working as it should be.

If I did this project again I would instead of having the buses going along a route I would make lists of stops, and then have the routes overlapping to allow the simulation to have passengers traveling from one route to the other using a shortest path algorithm on the part of the passenger or possibly bringing in the ideas we covered when we discussed how the internet works and the principles of routers knowing how to direct a packet from one network to another via a larger network. The later being my suggestion for how to make this project more interesting and bring in more concepts rather than just random chance on the part of the 564 students.

To elaborate and explain my idea fully, have three routes, A, B, and C. Where each route has some stops unique to it and some stops that overlap, however none of Cs stops overlap with A. This way some passengers are forced to use route B to go from A to C. I personally would probably solve this problem with having each stop have a list of possible stops getting on that bus could take you to. So that way the passenger can tell what possible places the different bus would take them.

Moving on from my imagining about how I would make this more interesting, my main feedback about this project is that RTI is…. Complex and while the class does a fine job teaching distributed systems conceptually, we don't spend much if any time actually working on the implementation side of it. I would have felt much more comfortable doing this in C++ if we had spent more time in class demonstrating how to actually build it in C++. Or Java. Or any language.

Overall feedback about doing the project in python. Python as to be expected was very easy to work with, and while it has a few limitations, if you understand how to write the XML file from their examples and the IDL translation that rtiddsgen makes, it is really very simple. There is no need for factories or anything of that sort in python. Literally only taking two lines of code to connect to the dds and then make a object that will push or pull messages, assuming your XML is written correctly. It did take me a few tries to get the XML right and I don't know if I did for sure, it works, but I feel like I could have done a better job and that I could have understood what I was doing in the XML file better.

In conclusion, while the project was pretty good, I feel like it focused far too much on learning how to use RTIDDS and no where close to enough learning about distributed systems. I think I would have learned more about actual distributed systems if we had to write it from a lower level, actually building the publish subscribe backend to support a simple application. RTI is far too specific to really learn much about how this works on a hands on level, instead mostly teaching me how to use it.

The first thing that comes to mind that would be a more beneficial project would be making a peer to peer file sharing system, as this would require each node to act as both a publisher and a subscriber. If you wanted to make this really interesting and actually see if we can practice the principles of a distributed system being both language and operating system independent the final test of the system could be everyone in the class logging into a VPN for this purpose and then having to participate in the peer to peer file sharing, using something like my bus approach to see if everyone can get all the files so that each user can only see a handful of other users. A final learning opportunity here would be trying to figure out what links in the chain failed and why, as well as seeing if anyone is missing any of the files.

Including as a reference the python documentation and their examples on github, incase anyone in the future wants to use python.

https://community.rti.com/static/documentation/connector/1.0.0/api/python/index.html

https://community.rti.com/static/documentation/connext-dds/6.0.0/doc/manuals/connext_dds/xml_application_creation/RTI_ConnextDDS_CoreLibraries_XML_AppCreation_GettingStarted.pdf

https://github.com/rticommunity/rticonnextdds-connector-py/tree/master/examples/python

Log files in the zip

Bus.txt

```
Bus11 is on route Express1 at stop 1 at 2020-04-30 02:35:30.528530
Bus11 reported an accident on route Express1 before stop 1 at 2020-04-30 02:35:30.528530
Bus12 is on route Express1 at stop 1 at 2020-04-30 02:35:30.685152
Bus13 is on route Express1 at stop 1 at 2020-04-30 02:35:30.842708
Bus21 is on route Express2 at stop 1 at 2020-04-30 02:35:31.251606
Bus11 is on route Express1 at stop 2 at 2020-04-30 02:35:32.017935
Bus22 is on route Express2 at stop 1 at 2020-04-30 02:35:32.143549
Bus12 is on route Express1 at stop 2 at 2020-04-30 02:35:32.691084
Bus13 is on route Express1 at stop 2 at 2020-04-30 02:35:32.848212
Bus23 is on route Express2 at stop 1 at 2020-04-30 02:35:33.802175
Bus11 is on route Express1 at stop 3 at 2020-04-30 02:35:34.023091
Bus11 reported an accident on route Express1 before stop 3 at 2020-04-30 02:35:34.023091
Bus12 is on route Express1 at stop 3 at 2020-04-30 02:35:34.195629
Bus21 is on route Express2 at stop 2 at 2020-04-30 02:35:34.260456
```

Operator.txt

```
Position: {'timestamp': '2020-04-30 02:35:30.528530', 'route': 'Express1', 'vehicle': 'Bus11', 'stopNumber': 1, 'numStops': 4, 'timeBetweenStops': 2, 'trafficConditions': 'Normal', 'fillInRatio': 73}
Accident: {'timestamp': '2020-04-30 02:35:30.528530', 'route': 'Express1', 'vehicle': 'Bus11', 'stopNumber': 1}
Position: {'timestamp': '2020-04-30 02:35:30.685152', 'route': 'Express1', 'vehicle': 'Bus12', 'stopNumber': 1, 'numStops': 4, 'timeBetweenStops': 2, 'trafficConditions': 'Normal', 'fillInRatio': 36}
Position: {'timestamp': '2020-04-30 02:35:30.842708', 'route': 'Express1', 'vehicle': 'Bus13', 'stopNumber': 1, 'numStops': 4, 'timeBetweenStops': 2, 'trafficConditions': 'Normal', 'fillInRatio': 85}
Position: {'timestamp': '2020-04-30 02:35:31.251606', 'route': 'Express2', 'vehicle': 'Bus21', 'stopNumber': 1, 'numStops': 6, 'timeBetweenStops': 3, 'trafficConditions': 'Light', 'fillInRatio': 95}
Position: {'timestamp': '2020-04-30 02:35:32.017935', 'route': 'Express1', 'vehicle': 'Bus11', 'stopNumber': 2, 'numStops': 4, 'timeBetweenStops': 2, 'trafficConditions': 'Light', 'fillInRatio': 81}
Position: {'timestamp': '2020-04-30 02:35:32.143549', 'route': 'Express2', 'vehicle': 'Bus22', 'stopNumber': 1, 'numStops': 6, 'timeBetweenStops': 3, 'trafficConditions': 'Normal', 'fillInRatio': 57}
Position: {'timestamp': '2020-04-30 02:35:32.691084', 'route': 'Express1', 'vehicle': 'Bus12', 'stopNumber': 2, 'numStops': 4, 'timeBetweenStops': 2, 'trafficConditions': 'Normal', 'fillInRatio': 33}
Position: {'timestamp': '2020-04-30 02:35:32.848212', 'route': 'Express1', 'vehicle': 'Bus13', 'stopNumber': 2, 'numStops': 4, 'timeBetweenStops': 2, 'trafficConditions': 'Normal', 'fillInRatio': 38}
Position: {'timestamp': '2020-04-30 02:35:33.802175', 'route': 'Express2', 'vehicle': 'Bus23', 'stopNumber': 1, 'numStops': 6, 'timeBetweenStops': 3, 'trafficConditions': 'Heavy', 'fillInRatio': 64}
Position: {'timestamp': '2020-04-30 02:35:34.023091', 'route': 'Express1', 'vehicle': 'Bus11', 'stopNumber': 3, 'numStops': 4, 'timeBetweenStops': 2, 'trafficConditions': 'Normal', 'fillInRatio': 62}
Accident: {'timestamp': '2020-04-30 02:35:34.023091', 'route': 'Express1', 'vehicle': 'Bus11', 'stopNumber': 3}
Position: {'timestamp': '2020-04-30 02:35:34.195629', 'route': 'Express1', 'vehicle': 'Bus12', 'stopNumber': 3, 'numStops': 4, 'timeBetweenStops': 2, 'trafficConditions': 'Light', 'fillInRatio': 79}
Position: {'timestamp': '2020-04-30 02:35:34.260456', 'route': 'Express2', 'vehicle': 'Bus21', 'stopNumber': 2, 'numStops': 6, 'timeBetweenStops': 3, 'trafficConditions': 'Normal', 'fillInRatio': 19}
```

Passanger.txt

```
Passenger 1 is waiting for the bus at stop 2 on route Express1
Passenger 2 is waiting for the bus at stop 3 on route Express2
Passenger 1 is on Bus11
Passenger 1 got off Bus11 at stop 4
Passenger 1 is at stop 4
Passenger 2 is on Bus21
Passenger 2 got off Bus21 at stop 2
Passenger 2 is at stop 2
All passengers at their stops

Process finished with exit code 0
```