ELEC 477 Distributed Systems Assignment 4 - Data Distribution System

Due April 8, 2024, 23:99.

1. Purpose

In this assignment use DDS to distribute and analyze air traffic control data.

2. Lab Setup

In this lab you will use Cyclone DDS version 0.11.0. This version is installed on whisper.engineering.queensu.ca in */opt/cyclone* and */opt/cyclone_cplusapi*. This is a atypical installation, as the instructions at *https://cyclonedds.io/docs/cyclonedds/latest/installation/installation.html* suggest installing both the C and C++ libraries to the same location. On my Mac I chose to install both at */Users/dean/Teaching/477/DDS/install/*. Installation requires C++ on the Mac, g++ on Linux, cmake and git.

If you do not use homebrew on MacOS (I don't), you can install cmake as an application from the cmake website (cmake.org), The command *sudo "/Applications/CMake.app/Contents/bin/cmake-gui" --install=/usr/local/bin* will add the command line version to */usr/local/bin*. The default path on MacOs should include */usr/local/bin*, so no other change to use cmake is necessary.

If you want to use the debugger to look at any of the cyclone DDS components, then you will have to build your installation in debugging mode using the command line parameter -*DCMAKE_BUILD_TYPE=Debug*. Otherwise the debugging will only work on your code and not on any of the Cyclone DDS classes.

Once installed, you will have to add the bin directory in the install path to your path. This is /opt/cyclone/bin on whisper. Add PATH="/opt/cyclone/bin:$PATH" to your .profile on whisper, or the equivalent path to your .profile (or .bash_profile) on your local Linux VM or MacOS. (Or you can change the CMAKE_PROGRAM_PATH to include it)

Cyclone DDS is supposed to work on Windows, but I have not tested it.

You can test your installation with the example code from OnQ. Change the line in CMakeLists.txt that sets the CMAKE_PREFIX_PATH to be the location of your installation. Since whisper has the two installed in two different locations, both paths are provided in the file in the sample project (separated by a semicolon).

Use the cd command to enter the build directory, and use the command "cmake -DCMAKE_BUILD_TYPE=Debug .." to create the makefiles for the example project (the last pair of dots are significant). Then the command *make* will compile the example code resulting in two files, psr_publisher and psr_subscriber, in the build directory. You can run them in two separate windows and they will communicate with each other. If you are compiling on whisper, you should change the number 1 on the line "dds::domain::DomainParticipant participant(1);" to be your team number. Otherwise if you and another team happen to run your code at the same time, they will talk to each other. (You might test the sample code with another team at the same time to see it happen).

3. Starting point

The starting point for your assignment is the zip file assign4.zip on OnQ. This contains two programs, and idl file and a data file. The data file is historical data of flights in the airspace around Toronto provided by the Open Sky project. The file is a comma separated values (csv) file that contains 16292 samples. The first column is the timestamp in UTC. The second column is the icao24 flight identifier, which we will use as a key in the topics. Other fields include the latitude, longitude, speed, heading and other information. The details are in the state.idl file including comments.

The first program consists of the files flight.cpp, aircraft.hpp aircraft.cpp. It reads the csv file and broadcasts the topic "Flights" with the data type defined in the state.idl file. The second program, calls subscriber.cpp is an example subscriber that subscribes to the Flights topic.

The CMakeLists.txt file is used to configure the CMake command. There are 4 sets of lines that matter. The first is the one that invokes the idlcxx_generate macro. This creates a dependency target that can be used in other commands. You give it the name of the idl file. If you add another idl file for the assignment, it will have a different target since not all programs use the same topic data types. The line in the example is:

```
idlcxx_generate(TARGET psrdata FILES psr.idl WARNINGS no-implicit-extensibility)
```

The next group of lines define the executable programs that are part of the project. The first parameter of the add_executable macro gives the name of the resulting program. The remainder of the line is the list of the source files that are combined to be the program:

```
add_executable(psr_publisher psrpublisher.cpp)
```

Notice that the files generated from the idl file are not included. The next group of lines identify the libraries that are needed by the program. The macros provided by the eclipse cyclone project treat the code generated by the idl files as a library attached to the given target. In this case the libraries are the cyclone C++ libraries and the idl data type library. If you program uses more than one idl type (e.g. the radar fusion program in the lecture), then list all of the relevant idl targets on this line.

```
target_link_libraries(psr_publisher CycloneDDS-CXX::ddscxx psrdata)
```

The last set of lines add some come time properties such as which C++ standard to use.

```
set_property(TARGET psr_publisher PROPERTY CXX_STANDARD ${cyclonedds_cpp_std_to_use})
```

4. Flight Control Zones

In general there are three zones of flight control. The first is tower control. This zone is responsible for managing the aircraft on ground and getting them between the terminal and the runways. The second zone is approach/departure. This zone is responsible for the airspace directly around the airport to a given distance and altitude. In real life the handoffs and transfers between airspace are complex, but for the purpose of this assignment we will simply and use 8 nautical miles and 3000 feet. The rest of the airspace is enroute airspace (i.e. between airports. As aircraft transition the boundaries, they must be handed off from one controller to another. This involves sending a specific message from one air traffic control controller to another, and radio communication between ATC and the pilots. We will simulate this with an DDS message.

There is also a controlled C airspace around busy airports with a radius of 8 nm and a ceiling of 2500 feet. No aircraft are permitted in this airspace without permission.

5. Assignment

You will write three programs. Two of the programs represent the two airspace zones: the Toronto approach/departure (called Toronto AD), and the surrounding enroute airspace called Toronto Centre. The third process will be a general query process that will accumulate information and allow you to retrieve the last state sample for a given flight.

You will have to add a new message type that will represent the transfer of control. You can think of this message as the message that would be sent when the controller in one zone tells the system to hand the aircraft to the next zone. Since we don't have live controllers in the assignment, you will send the message automatically when the aircraft approaches the boundary in the space. For example, when an aircraft below 3000 feet and within 8 nm of the airport is heading away from the airport and will cross the 8nm boundary (or is climbing through the 3000 ft boundary). Similarly if an aircraft is heading towards the airport below 3000 feet and approaching the 8nm boundary or is within 8nm and is descending to 3000 feet. Distance between two latitude and longitude coordinates can be calculated using the Haversine formula (http://www.movable-type.co.uk/scripts/latlong.html),

Note that the condition for sending the message may occur for more than one update. Only send one message in this case.

Your idl file should use an enumerated type to list the two flight zones and the struct should contain the aircraft call sign, the source and destination zones, and the timestamp of the state message that triggered the message. Since both programs will publish and subscribe to this topic, use content filtering to filter based on the destination zone.

The third query program should ask for a flight number and provide the current state of the aircraft wth that flight number. You should not build any of your own data structures to store the data locally within the program. Instead, leave the data in the DataReader and query the data reader to retrieve the information (see the psr_ex_contentFilter example).

6. Testing

There is a file that contains real time Toronto data. You can also create your own state file to test the specific conditions in the assignment. These are:

1. Aircraft leaving Toronto AD by crossing the 8nm barrier
2. Aircraft entering Toronto AD by crossing the 8nm barrier
3. Aircraft leaving Toronto AD by ascending through 3000 feet
4. Aircraft entering Toronto AD by descending through 3000 feet

We will also provide another data file later in the assignment for testing.

5. What to hand in

As with previous assignments, you should hand in a design document that covers your three programs, a testing document, and the inputs and outputs of any test cases.