CptS 464/564 Project #2
Simple Public Transport System

Given: Tuesday, March 27, 2019
Due: Friday, April 26, 2019 at 11:59pm
Weight: 25% of Final Grade

1. Overview

In this project, you will create a simplified public transport system using RTI-DDS middleware.

In a public transport system, there are buses following different routes with passengers who want to know the status of their bus and when it will arrive at their stop. There are also operators who need to know the status of the bus fleet from time to time.

More specifically, the passengers would probably be interested in the current status of their bus, and when it will arrive at their stop. Most of what they will be interested in will be short-lived information. That is, information that is transient and has little or no value to them once the moment has passed, since there will be more up-to-date information available or about to become available.

The operators, on the other hand, would possibly be interested in where all the buses are at a given time, whether they are on schedule or experiencing any delays, which buses have problems, and whether there are any breakdowns within the fleet. Some of this information is transient, but if it is retained it provides a historical picture of events that may be useful input to other processes.

2. Problem Setting

We will now describe the simplified public transport system in more detail.

2.1. Simulated Public Transport System

In this simplified public transport system, suppose we only have one kind of vehicles (bus) in one city (Pullman). This transport system is composed of two independent routes. Each route has the following attributes:

- ***Route name***: this is unique across the transport system.
- ***Number of stops***: this is the total number of stops along the route; stops are numbered starting from number 1. Every route is a closed path so the bus can move to the first stop after it reaches the last stop.
- ***Time between stops***: this is the number of seconds it takes for a vehicle to move from one stop to the next; we assume that all the stops on a given route are evenly spaced out along that route.

A route can accommodate a number of vehicles; each vehicle is identified by a name that is unique route-wide. All vehicles can contain up to 100 passengers.

Vehicles move along the route in a one-way fashion from the first to the last stop according to the "time between stops" attribute of the route. When a vehicle reaches the last stop, it then moves to the first stop and goes along the route again. It is removed from the system after it goes through the route 3 times. The only exceptions to this behavior are:

- The breakdown of a vehicle, which causes its temporary removal from the system, and a backup vehicle for the same route (if available) to be added to the route 15 seconds after the breakdown was detected, moving from the stop where breakdown happened. The broken-down vehicle is added to the backup vehicles 20 seconds after the breakdown was detected. (**Only for 564 students**)
- An accident found along the route, which delays the vehicle by a fixed amount of time (10 seconds).
- Light traffic conditions, which dynamically decrease the time between stops by a 25% factor.
- Heavy traffic conditions, which dynamically increase the time between stops by a 50% factor.

When a vehicle reaches a stop, the following information is collected:

- Timestamp (hh:mm:ss)
- Vehicle fill-in ratio (0 to 100)
- Traffic conditions (normal, light, heavy)
- Detection of any breakdowns (**for 564**)
- Detection of any accidents

Several attribute values in the simulated transport system are randomly generated according to the following distribution of probability:

- Breakdowns happen 5% of the time (**for 564**)
- Accidents happen 10% of the time
- Traffic is heavy 10% of the time, light 25% of the time, and normal for the remaining 65% of the time
- The fill-in ratio is a completely random non-negative integer less than or equal to 100.

For simplicity, breakdowns (**for 564**), accidents, and traffic conditions are only collected and published at stops. Breakdowns happen 5% of the time means that when a bus reaches a stop n times, it breaks down n*0.05 times when n approaches to infinity. It applies only to the time between current stop and the next stop that light traffic conditions decrease the time between stops by a 25% factor, and similar for heavy traffic conditions.

A vehicle publishes information about its position and state at each stop along the route, using the topic:

`P3464(or 564)_EECS_username: PT/POS`

In the event of a breakdown (**for 564**) or an accident, a message is published with topic:

`P3464(or 564)_EECS_username: PT/ALR/BRK`, or
`P3464(or 564)_EECS_username: PT/ALR/ACC`

2.2. Publishers

The publisher is governed by an application called *PubLauncher*. It starts a thread, *PubThread*, for each vehicle on each route.

- The controlling application: *PubLauncher*
- The thread, called *PubThread*, started for each vehicle

2.2.1. PubLauncher

It reads all of its initialization parameters from a properties file called *pub.properties* and starts a thread for each vehicle of every route. Each thread publishes all the messages, alerts (accidents or breakdowns) and positions for this vehicle.

pub.properties File

```
numRoutes=2
numVehicles=3
numInitialBackupVehicles=1 #for 564 students
route1=Express1 #route name
route2=Express2 #route name

#route1 parameters
route1numStops=4
route1TimeBetweenStops=2
route1Vehicle1=Bus11
route1Vehicle2=Bus12
route1Vehicle3=Bus13
route1Vehicle4=Bus14 #initial backup bus, for 564 students

#route2 parameters
route2numStops=6
route2TimeBetweenStops=3
route2Vehicle1=Bus21
```

```
route2Vehicle2=Bus22
route2Vehicle3=Bus23
route2Vehicle4=Bus24 #initial backup bus, for 564 students
```

2.2.2. PubLauncher coding logic

The *PubLauncher* is responsible for the three following tasks:

- Reading and parsing the properties file
- Starting the threads and providing them with the information from the properties file, e.g.:

```
pubThread = new PubThread(info);
pubThread.start();
```

- Waiting for each thread to complete and terminate:

```
pubThread.join();
```

2.2.3. PubThread

Each *PubThread* represents a vehicle on a route. It receives the following information before starting:

- The route and the vehicle it represents.
- The number of stops along the route.
- The time spent by the vehicle between two stops.

Once all *PubThreads* are created, they are started by the *PubLauncher*. At each stop, the bus publishes a position message and an accident or breakdown message depending on the situation.

2.2.4. The Publication Messages

In this transport system, each vehicle publishes two (**three for 564**) different types of messages: one type to indicate its position, another type when it has an accident, and a third one (**for 564**) when the vehicle has broken down.

The data is defined in following IDL files:

```
// position.idl
struct Position {
   string timestamp;
   string route;
   string vehicle;
   long stopNumber;
   long numStops;
   long timeBetweenStops;
   string trafficConditions;
   long fillInRatio;
};

// accident.idl
struct Accident {
    string timestamp;
    string route;
    string vehicle;
    long stopNumber;
};

// breakdown.idl #only for 564
struct Breakdown {
    string timestamp;
    string route;
    string vehicle;
    long stopNumber;
};
```

2.2.5. Subscribers

In this system, there are two kinds of subscribers: passengers and operators. For this project, you need to create two passenger subscribers and one operator subscriber.

Passengers can subscribe to all messages: position, accidents and breakdowns. The first passenger is waiting at stop #2 of route #1, and his destination is stop #4. He subscribes to all messages for route #1 and stop #2 before he gets on some bus, and after he gets on a bus, he subscribes only to the message sent from that bus (you can use content filtered topic). The second passenger is waiting at stop #3 of route #2, and her destination is stop #2. She subscribes to all messages for route #2 and stop #3 before she gets on some bus, and after she gets on the bus, she subscribes only to the messages sent from that bus.

Please print out some messages indicating when and by which bus the passenger is getting on board, when and by which bus he/she is arriving on each stop on the way and the destination.

For 564 students, please take care of breakdown events. That is:

1. Let the replacement bus substitute the broken one, from the same stop, after the specified amount of time.
2. Put the broken bus in the substitutes list.
3. If a passenger is in a broken bus, make him/her step down and wait for the next bus to continue the journey. This could be the substitute, or just the next bus passing by the stop.

The passenger subscriber program takes three parameters: route, starting stop, and destination stop.

The operator subscribes to all kinds of messages and prints them out with a table-like format.

2.2.6. Expected Output

Hint: You can use grep to filter out the license information from the program output:

```
% grep –i –v ‘rti’
```

This of course only works on Linux (or MacOS terminal/shell).
In Window’s Powershell you can alternatively filter using:

```
Select-String -Pattern 'rti' -NotMatch
```

Below is sample representative output from the passenger and operator windows, for Java; if you are using C++ or C# or another language you will need to figure out the equivalent first line output info. Below the username is `dalvarez`.

PubLauncher (Publishers):

```
[dalvarez@rti-dds-01 dds_project3]$ java -classpath .:$NDDSHOME/class/nddsjava
.jar PubLauncher | grep -i -v 'rti'
Launching publishers...
Added vehicleBus14  to backup for route Express1
Added vehicleBus24  to backup for route Express2
Thread Bus11 started
Thread Bus12 started
Thread Bus13 started
Thread Bus21 started
Thread Bus22 started
Thread Bus23 started
All buses have started.  Waiting for them to terminate...
Bus13 published a position message at stop #1 on route Express1 at 23:50:54
Bus21 published a position message at stop #1 on route Express2 at 23:50:54
Bus23 published a position message at stop #1 on route Express2 at 23:50:54
Bus22 published a position message at stop #1 on route Express2 at 23:50:54
Bus12 published a breakdown message at stop #1 on route Express1 at 23:50:54
Bus12 stopped. A backup should arrive in 15 seconds.
Bus11 published a position message at stop #1 on route Express1 at 23:50:54
Bus21 published a position message at stop #2 on route Express2 at 23:50:56
Bus13 published a position message at stop #2 on route Express1 at 23:50:57
Bus22 published a position message at stop #2 on route Express2 at 23:50:57
Bus11 published an accident message at stop #2 on route Express1 at 23:50:57
Bus11 published a position message at stop #2 on route Express1 at 23:50:57
Bus23 published a breakdown message at stop #2 on route Express2 at 23:50:57
Bus23 stopped. A backup should arrive in 15 seconds.
Bus21 published a position message at stop #3 on route Express2 at 23:50:58
Bus13 published an accident message at stop #3 on route Express1 at 23:51:00
```

Operator Subscriber:

```
rti-dds-02% java -classpath .:$NDDSHOME/class/nddsjava.jar OperatorSubscriber | grep -i -v 'rti'
MessageType     Route           Vehicle Traffic Stop#   #Stops  TimeBetweenStops        Fill%   Timestamp
Position        Express1        Bus12   Normal  1       4       3.00                    48      00:37:25
Position        Express2        Bus23   Normal  1       6       3.00                    1       00:37:26
Position        Express2        Bus22   Light   1       6       2.25                    61      00:37:25
Breakdown       Express2        Bus21           1                                               00:37:25
Position        Express1        Bus11   Normal  1       4       3.00                    2       00:37:26
Breakdown       Express2        Bus22           2                                               00:37:28
Breakdown       Express1        Bus12           2                                               00:37:28
Position        Express1        Bus11   Light   2       4       2.25                    80      00:37:29
Breakdown       Express2        Bus23           2                                               00:37:29
Position        Express1        Bus11   Heavy   3       4       3.75                    76      00:37:31
Position        Express1        Bus13   Normal  1       4       3.00                    14      00:37:26
Position        Express1        Bus13   Normal  2       4       3.00                    92      00:37:29
Position        Express1        Bus13   Light   3       4       2.25                    70      00:37:32
Accident        Express1        Bus13           4                                               00:37:34
Position        Express1        Bus13   Normal  4       4       13.00                   26      00:37:34
Accident        Express1        Bus11           4                                               00:37:35
Position        Express1        Bus11   Heavy   4       4       13.75                   63      00:37:35
Accident        Express2        Bus24           1                                               00:37:41
Position        Express2        Bus24   Normal  1       6       13.00                   99      00:37:41
Position        Express1        Bus14   Normal  2       4       3.00                    58      00:37:43
Position        Express2        Bus21   Normal  2       6       3.00                    2       00:37:46
Position        Express1        Bus14   Heavy   3       4       3.75                    20      00:37:46
Position        Express1        Bus13   Normal  1       4       3.00                    91      00:37:47
Position        Express2        Bus22   Heavy   2       6       3.75                    23      00:37:48
Position        Express1        Bus11   Heavy   1       4       3.75                    22      00:37:48
Position        Express2        Bus21   Normal  3       6       3.00                    57      00:37:49
```

Passenger Subscribers:

Passenger 1:

```
[dalvarez@rti-dds-02 dds_project3]$ java -classpath .:$NDDSHOME/class/nddsjava.jar
PassengerSubscriber Express1 1 4 | grep -i -v 'rti'
Waiting for the bus...
Getting on Bus12 at 00:46:08, Normal, 3 stops left
Arriving at stop #2, at 00:46:11, breakdown, 2 stops left
Getting on Bus11 at 00:46:23, Light, 2 stops left
Arriving at stop #3, at 00:46:26, Heavy, 1 stop left
Arriving at destination by Bus11 at 00:46:29
```

Passenger 2:

```
[dalvarez@rti-dds-02 dds_project3]$ java -classpath .:$NDDSHOME/class/nddsjava.jar
PassengerSubscriber Express2 3 2 | grep -i -v 'rti'
Waiting for the bus...
Getting on Bus24  at 00:47:06, Heavy, accident, 5 stops left
Arriving at stop #4, at 00:47:20, Normal, 4 stops left
Arriving at stop #5, at 00:47:23, breakdown, 3 stops left
Getting on Bus21 at 00:47:38, Heavy, 3 stops left
Arriving at stop #6, at 00:47:42, Heavy, 2 stops left
Arriving at stop #1, at 00:47:45, Normal, accident, 1 stop left
Arriving at destination by Bus21 at 00:47:58
```

3. Implementation language

You can use either Java or C++/C; with C# or other languages you won't have help available (but if you feel comfortable with these other languages, don't let that scare you away. Project 1 should have gotten you over the biggest language- and OS-specific configuration issues, anyway).

4. Implementation steps

1. Create the IDL files.
2. Generate the template source codes with *`rtiddsgen`*.
3. Modify or create files
    a. Create properties file
    b. Create *`PubLauncher`*, *`PubThread`*, etc.
    c. Modify publishers
    d. Create helper files for subscribers, e.g., *`SubThread`*.
    e. Modify subscribers
    f. Create any other files that you need

5. Grading

Submit via BLACKB OARD a zip file with all your created/modified files. You must also include screenshots showing the run-time execution of your project's executables. Those should all be in a single PDF, included in the zip file. If BLACKBOARD lets you submit more than one file for an assignment, also submit the PDF file so the TA does not have to open up the zip file to get at your PDF.

The course syllabus states "DO YOUR OWN WORK!" so naturally you should be able to explain all of the code you've written and most of the automatically generated code in publisher and subscriber (at least you should know what the generated code is doing in a general sense). There may well be a question on the final about this project, something that would trip up you if you either got code from someone else, or relied in suggestions on someone else more than you should have.

One note about accidents and breakdowns: since they happen with low probability, it's possible not to see them quickly during a screenshot of your running program. So please make your program configurable enough to change the probabilities easily (i.e. without requiring a recompilation).

Finally, do not modify the last modified dates on your created/modified files after the submission date, in case we need to check the files. And have them copied into your EECS

home directory before the deadline (this is in addition to submitting by BLACKBOARD), and don’t touch the files there so the modify date and time is before the submission deadline.

6. Grading criteria

To be provided later