CptS 464/564 Project #3
Simplified Public Transport System

Given: Friday, November 3, 2010
Due: 12:00 PM Friday, December 10, 2010
Demos: Afternoon, December 10, 2010
Weight: 25% of Final Grade

1 Overview

In this project, you will create a simplified public transport system using RTI (Real-Time Innovations) DDS (Data Distribution Service).

In a public transport system, there may be buses following different routes with passengers who want to know the status of their bus and when it will arrive at their stop. There may also be 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 an 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 (if available) for the same route will 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 25% factor.

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

- Time-stamp (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 25% of the time, light 25% of the time, and normal for the remaining 50% 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:
EECS username: PublicTransport/Positions.

In the event of a breakdown (**for 564**) or an accident, a message is published on topic:
EECS username: PublicTransport/Alerts/Breakdowns
or EECS username: PublicTransport/Alerts/Accidents.

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
- The categories of messages published for each vehicle

2.2.1 PubLauncher

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

2.2.1.1 The properties file - pub.properties

```
numRoutes=2
numVehicles=3
numInitialBackupVehicles=1 #for 564 students

route1=Express1
route2=Express2

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

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

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

2.2.1.2 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:
  ```
  pubThread = new PubThread(info);
  pubThread.start();
  ```
- Waiting for each thread to complete and terminate:
  ```
  pubThread.join();
  ```

2.2.2 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 thread publishes a position message and an accident or breakdown message depending on the situation.

2.2.3 The publication messages

In this transport system, each vehicle publishes two (**three for 564**) different kinds of messages: one kind of message to indicate its position, another kind of message when it has an accident, and a third kind of message **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;
   int stopNumber;
   int numStops;
   int timeBetweenStops;
   string trafficConditions;
```

```
    int fillInRatio;
};

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

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

2.3 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 only subscriber to the position messages. The first passenger is waiting at stop #2 of route #1, and his destination is stop #4. He subscribes to the message for route #1 and stop #2 before he gets on some bus, and after he gets on the 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 the message for route #2 and stop #3 before she gets on some bus, and after she gets on the bus, she subscribes only to the message sent from that bus. Please print out some messages indicating that 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**. The passenger subscriber 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.

2.4 Expected output

Hint: You can use grep to filter out the license information from the program output: `grep –i –v ‘rti’`

PubLauncher (Publishers):

```
[chuan@localhost project3]$ gmake -f makefile PubLauncher | grep -i -v 'rti'
Start-ing publishers...
Thread 0 started.
Bus11 has published a position message at stop #1 on route Express1 at 11:28:43
Thread 1 started.
Bus12 has published a position message at stop #1 on route Express1 at 11:28:43
Thread 2 started.
Bus13 has published a position message at stop #1 on route Express1 at 11:28:43
Thread 3 started.
Bus21 has published a position message at stop #1 on route Express2 at 11:28:43
Thread 4 started.
Thread 5 started.

All buses have started. Waiting for them to terminate...

Bus23 has published a position message at stop #1 on route Express2 at 11:28:43
Bus22 has published a position message at stop #1 on route Express2 at 11:28:43
Bus11 has published a position message at stop #2 on route Express1 at 11:28:44
Bus13 has published a position message at stop #2 on route Express1 at 11:28:44
Bus12 has published a position message at stop #2 on route Express1 at 11:28:45
Bus21 has published a position message at stop #2 on route Express2 at 11:28:45
Bus23 has published a position message at stop #2 on route Express2 at 11:28:45
Bus11 has published an accident message at stop #3 on route Express1 at 11:28:46
Bus11 has published a position message at stop #3 on route Express1 at 11:28:46
Bus12 has published a position message at stop #3 on route Express1 at 11:28:46
Bus13 has published a position message at stop #3 on route Express1 at 11:28:46
Bus22 has published a position message at stop #2 on route Express2 at 11:28:46
Bus12 has published a position message at stop #4 on route Express1 at 11:28:47
Bus13 has published an accident message at stop #4 on route Express1 at 11:28:47
Bus13 has published a position message at stop #4 on route Express1 at 11:28:47
Bus21 has published a position message at stop #3 on route Express2 at 11:28:49
Bus23 has published a breakdown message at stop #3 on route Express2 at 11:28:49
Bus23 stopped.
```

Operator Subscriber:

```
[chuan@localhost project3]$ java -classpath .:/opt/RTI/ndds.4.5c/class/nddsjava.jar OperatorSubscr
iber | grep -i -v 'rti'
MessageType     Route           Vehicle Traffic Stop#   #Stop   TimeBetweenStops Fill%  Timestamp
position        Express1        Bus11   light   1       4       1.50            64      11:28:43
position        Express1        Bus12   normal  1       4       2.00            47      11:28:43
position        Express1        Bus13   light   1       4       1.50            49      11:28:43
position        Express2        Bus21   heavy   1       6       2.25            38      11:28:43
position        Express2        Bus23   heavy   1       6       2.25            9       11:28:43
position        Express2        Bus22   normal  1       6       3.00            22      11:28:43
position        Express1        Bus11   normal  2       4       2.00            90      11:28:44
position        Express1        Bus13   normal  2       4       2.00            54      11:28:44
position        Express1        Bus12   light   2       4       1.50            82      11:28:45
position        Express2        Bus21   normal  2       6       3.00            70      11:28:45
position        Express2        Bus23   normal  2       6       3.00            82      11:28:45
accident        Express1        Bus11           3                                       11:28:46
position        Express1        Bus11   light   3       4       11.50           13      11:28:46
position        Express1        Bus12   light   3       4       1.50            92      11:28:46
position        Express1        Bus13   heavy   3       4       1.50            53      11:28:46
position        Express2        Bus22   normal  2       6       3.00            30      11:28:46
position        Express1        Bus12   normal  4       4       2.00            58      11:28:47
accident        Express1        Bus13           4                                       11:28:47
position        Express1        Bus13   normal  4       4       12.00           51      11:28:47
position        Express2        Bus21   light   3       6       2.25            37      11:28:49
breakdown       Express2        Bus23           3                                       11:28:49
position        Express1        Bus12   light   1       4       1.50            97      11:28:50
position        Express2        Bus22   light   3       6       2.25            38      11:28:50
position        Express2        Bus21   normal  4       6       3.00            17      11:28:51
position        Express1        Bus12   normal  2       4       2.00            25      11:28:51
position        Express2        Bus22   heavy   4       6       2.25            89      11:28:52
accident        Express1        Bus12           3                                       11:28:53
position        Express1        Bus12   normal  3       4       12.00           26      11:28:53
position        Express2        Bus21   heavy   5       6       2.25            99      11:28:55
position        Express2        Bus22   normal  5       6       3.00            48      11:28:55
```

Passenger Subscribers:

Passenger 1:

```
[chuan@localhost project3]$ java -classpath .:/opt/RTI/ndds.4.5c/class/nddsjava.
jar PassengerSubscriber 1 1 4 | grep -i -v 'rti'
Getting on Bus11 at 11:35:26, normal, 3 stops left
Arriving at stop #2 at 11:35:28, light, 2 stops left
Arriving at stop #3 at 11:35:30, light, 1 stops left
Arriving at destination by Bus11 at 11:35:31
[chuan@localhost project3]$
```

Passenger 2:

```
[chuan@localhost project3]$ java -classpath .:/opt/RTI/ndds.4.5c/class/nddsjava.
jar PassengerSubscriber 2 3 2 | grep -i -v 'rti'
Getting on Bus21 at 11:37:02, heavy, accident, 5 stops left
Arriving at stop #4 at 11:37:16, normal, 4 stops left
Arriving at stop #5 at 11:37:20, normal, accident, 3 stops left
Arriving at stop #6 at 11:37:35, light, 2 stops left
Arriving at stop #1 at 11:37:38, light, 1 stops left
Arriving at destination by Bus21 at 11:37:40
[chuan@localhost project3]$
```

3 Implementation language

You can use either Java or C++/C.

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

Details will come later.