CptS 464/564 Project #3

Assigned: Tuesday, November 2, 2004
Due: **Tuesday, November 30, 2004 (start of class)**
Weight: 10% of Final Grade

Overview & Objectives

You have been asked to instrument a team of racing cars. The team manager has heard that CORBA is 'cool' and that the OMG Event Service is even 'cooler', so you must use events and event channels to complete your assignment. The team has several cars, Red, White and Blue. Each car periodically reports its average speed and fuel consumption since the last report. Each report is sent to an event channel. A RaceDataRepository (RaceDataRepo for short) program consumes these reports (and all other events) from the event channel. Statistics are kept on car speeds and fuel consumption. The RaceDataRepo offers a servant for the team manager to obtain the average speed of each car, both from the start of the race and from the most recent "split". Graduate students will implement a second servant allowing the pit manager to obtain the amount of fuel consumed since the most recent "fill-up".

Project #3's objectives include:

- Gain more CORBA experience
- Gain an understanding of and experience working with the OMG Event Service
- Learn how to co-locate multiple servants within a single program
- Learn how to use the CORBA ANY type.
- Learn (or refresh) your knowledge of how to perform events periodically in a program.

Resources that might be of use to you include:

- /net/niflab/orbacus/cs564/Project3/CarRace (Class supplied IDL files, etc.)
- /net/niflab/orbacus/OB-4.1.2/event/demo

The Race Data Repository

A diagram of the completed Race Data Repository and all of its parts is shown in Figure 1.

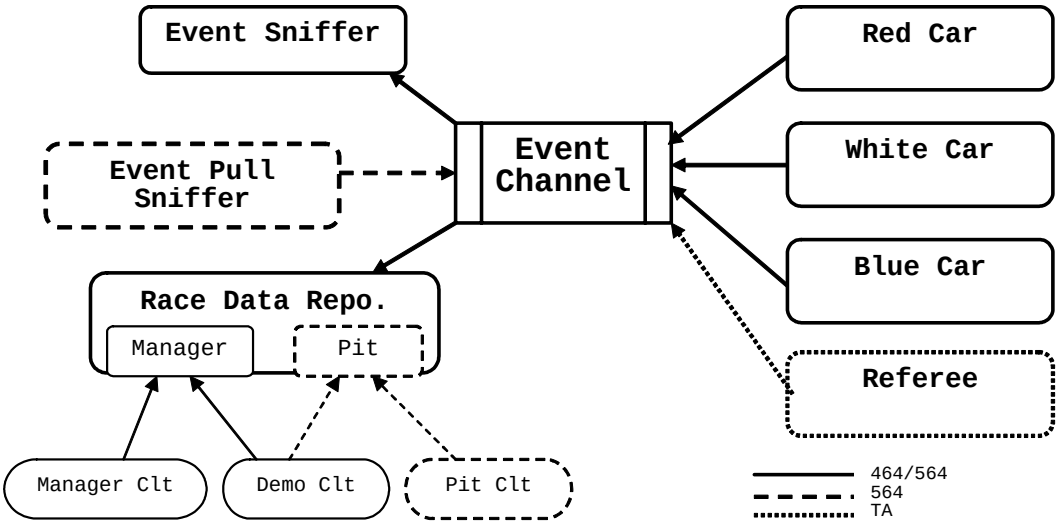


Figure 1: The Race Data Repository

On the right of Figure 1 are the ***event suppliers*** (aka the cars). The red, white, and blue cars are to be implemented by you. The TA will implement a fourth supplier, the referee. The different colors of cars provide their status updates at different rates: every 3 seconds for the red car, 5 for the white, and 4 for blue (i.e., the

blue car will send an event to the event channel every four seconds). The Red car travels a (different) random speed in [100.0, 200.0] mph and uses a random amount of fuel in [.05, .10] gal. during each interval. The Blue car travels a random speed in [120.0, 180.0] mph and uses a random amount of fuel in [.06,.09] gal. during each interval. The White car's speed is in [130.0, 170.0] mph and it uses fuel in [.065,.085] gal. during each interval.

The TA's referee sporadically sends flag events of type Flag.Flag. After the "checkered" flag is sent the race repository must stop updating statistics for all three cars. (The other colors of flag are just for fun and don't have any particular meaning.) The referee may also occasionally send a string value as a comment.

All of the event suppliers will operate in 'Push' mode**. Your car events must have the team name set to your EECS user name.**

On the left side of Figure 1 are the ***event consumers***. CptS 464 students are required to implement an event sniffer and a race data repository, both of which will operate in 'Push' mode. The event sniffer must capture and print out every event received regardless of its type (i.e., car status, flag, or string). Graduate students must also implement a 'Pull' mode event sniffer. The race data repository is introduces the new idea of two co-located servants: an event consumer servant and a team manager servant. CptS 564 students will also implement a third co-located servant for the pit boss.

At the bottom left-hand side of the Figure 1 are two ***client programs*** that must be implemented (three for graduate students). The manager and pit boss clients communicate with their respective servants. These client programs should have a (very) simple user interface. The demo client program will have no user interface because its purpose is to automatically test every IDL method call.

Before proceeding further, the IDL for cars, the repo, and the flags needs to be introduced.

Car.idl

```
// Car.idl
// CptS 464/564 -- Project #3 – Fall 2004
// This file defines a car structure used to inform about
// car speed and fuel consumption
#ifndef _CAR_IDL_
#define _CAR_IDL_
module CarMod {
  enum ColorType { C_RED, C_WHITE, C_BLUE };  // car colors
  struct Car  {
    string    team;
    ColorType color;
    long start;
    long end;
    double speed;
    double fuel;
  };
};
#endif
```

RaceRepo.idl

```
// RaceRepo.idl
// CptS 464/564 -- Project #3 Fall 2004
// This file defines a race data repository for a race team. The team has a
// manager in charge of monitoring the cars' speeds.
// Graduate students also need to implement the pit interface
// where the fuel consumption can be monitored
#ifndef _RACEREPO_CS564_IDL_
#define _RACEREPO_CS564_IDL_
module RaceRepo {
  // This will monitor cars so we'll include the car type
  //-----
  #include "Car.idl"
  struct SpeedStats {
    // Stats is designed to report statistics about a car
    //-----
    long  eventCnt;     // total number of events received for this car
    string team;       // name of the team for which stats are being reported
    double avgSpeed; // average speed since start
    double splitSpeed; // avg. speed since last split (or start if none)
  };
  exception NoStatsInfo {
    // If stats are requested for a car for which  no events have been
   // received then
    // this exception is thrown.
    string reason;
  };

  interface Manager {
    // Team  manager interface
    //-----
    void
    Split(in CarMod::ColorType color);
    // After a split operation, the starting time for subsequent
    // splitSpeed values is the end time of the last event received
    // prior to the split.
    //-----
    void
    GetStats(in CarMod::ColorType color, out Stats currStats)
    raises (NoStatsInfo);
    // Get the stats
  };
  //-----GRADUATE STUDENTS ONLY BEYOND THIS POINT-------
  interface Pit {
    // The Pit boss monitors fuel use via this interface
    //-----
    double
    GetFuelUsed (in CarMod::ColorType color);
      // The amount of fuel used since last fill-up (or start) is returned.
      //-----
    void
    FillUp (in CarMod::ColorType color);
      // Reset fuel used to 0.0
  };
};
#endif
```

Flag.idl

```
// Flag.idl
// CptS 464/564 -- Project #3 – Fall 2004
// This file defines a flag structure used to inform about
// referee's flags
#ifndef _FLAG_IDL_
#define _FLAG_IDL_
module FlagMod {
    enum ColorType { C_RED, C_YELLOW, C_GREEN, C_CHECKERED };
    // flag colors
    struct Flag  {
      ColorType color;
      long time;
      };
};
#endif
```

The ***manager client*** must be able to invoke the 'Split' and 'GetStats' methods. Your race repository must track statistics for all three cars.

(564)The ***pit client*** must be able to invoke the 'GetFuelUsed' and 'FillUp' methods.

The ***demo client*** is an automated (testing) tool that will demonstrate the key functionality of the RaceRepo code that you've written. It must execute the following steps:

1. Obtain object references for the Manager (& Pit) servants (via the Naming Service)
2. GetStats (Red, currStats)
3. Split (Red)
4. wait 60 seconds
5. GetStats(C_RED, currStats)
6. GetStats(C_WHITE, currStats)
7. GetStats(C_BLUE, currStats)

—Graduate Students Only—

8. GetFuelUsed(C_RED)
9. FillUp(C_BLUE)

You must submit a printout of this program's execution. Of course, after each call that acquires info (GetStats, GetFuelUsed) it should print the result. During the demo you will be asked to run this program. Once it has run, you must then justify / explain why you think its results are correct.

Additional Work for CptS 564 Students

Two of the additional tasks that CptS 564 students must do have been mentioned previously. These tasks are to implement a 'Pull' event sniffer and to implement a pit servant and client. The third task is to consider the OMG Notification service as an alternative to the event service: in class we briefly mentioned the concept of event filtering. You need to learn more about the OMG Notification Service and then explain how race car monitoring assignment could be modified to use the Notification Service. You must then also explain why using the Notification Service would or would not be an improvement (or could it be both?). This discussion should be limited to a 1/2 page (single-spaced) or less. (Find discussion of the Notification service on www.omg.org.)

Simplify, Simplify, and Simplify

The point of this programming assignment is to work with CORBA. To that end you may make as many simplifying assumptions as you would like as long as you can still meet all of the given design constraints. Fancy data structures are not needed for keeping track of events – a simple array will do. However, you will need to exercise some care in computing the average speeds. I do not want to see three copies of the car code – parameterize it by the car color passed as a startup argument.

Grading

Turn in hardcopy of all your created files. Note, the "created" files include only those files that you've created/edited yourself. You must also turn in printouts showing the run-time execution of your project's executables. These printouts must show/verify that your event suppliers and consumers work and that every method and exception specified in the RaceRepo.idl file is implemented. One method of obtaining these run-

time printouts is to embed print statements in your code. For example, every event supplier (or consumer) should print out a notification message every time an event is generated (or consumed).

You will be expected to demo your application to a TA. During the demo you will be asked to run your code and to explain it as well. 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 code that you've copied from the example programs (at least you should know what the copied example code is doing in a general sense). Demo times and arrangements will either be mentioned in class or via the class e-mail list. Please rehearse your demo before hand and come prepared to answer simple questions about your project.

Finally, do not modify or otherwise change the last accessed dates on your created files after the submission date, in case we need to check the files.

Grading Criteria (Penalty points listed inside parenthesis)

Project Writeup

10/15[g] pts	Cover Sheet	
	2	Your Name
	2	Code Location
	3	One (1) positive comment about the project (no more than a paragraph long!)
	3	One (1) suggestion about improving the project (no more than a paragraph long!)
	0/5[g]	OMG Notification Service comments (1/2 page or less!)
10/13[g] pts	Run-Time Printouts	
	5	Event Sniffer output
	5	Demo Client output
	3[g]	Pull Event Sniffer output
10 pts	Source Code (Files may be in any order as long as DemoClient.cpp is first)	
	5	Labeled
	5	Complete
	(-5)	DemoClient.cpp not included as the first source code printout

Demonstration / Working Code

20/22[g] pts	Makefile & Running Programs	
	7	[Red | White | Blue] Cars program
	3	EventSniffer
	1[g]	EventPullSniffer
	3	RaceDataRepo
	2	ManagerClient
	1[g]	PitClient
	5	DemoClient
12 pts	Event Suppliers	
	5	Connect to Event Channel
	3	Supply car updates (red, white, blue)
	2	Correct production rates (3, 5, 4)
	2	Timestamps are correct
	(-5)	Incorrect team name
15 pts	Event Sniffer	
	9	works
	2	Flag sniffed
	2	string sniffed
	2	Car sniffed
0/7[g] pts	EventPullSniffer	
	5[g]	works in 'Pull' mode
	2[g]	events sniffed
18/21[g]	RaceRepo	
	5	works
	5	EventConsumer servant
	2	Manager servant
	6	Correct statistics computed
	1[g]	Pit servant
	2[g]	Correct fuel levels kept
5 pts	ManagerClient	
	1	works
	2	calls Split
	2	calls GetStats
0/5[g] pts	PitClient	
	1[g]	works
	2[g]	calls GetFuelUsed
	2[g]	calls FillUp

Penalty Points

(-10) pts	Use of global variables
(-2..-20) pts	Event suppliers are left running after you log off. (You will be warned the first time, the second time will cost your 2 points, after that you will lose 5 pts per time that I 'kill' your leftover tasks.)

Note: [g] denotes points for a CptS 564 task

Note: You will receive only partial credit if the correct working of your code cannot be verified by watching screen printouts when your DemoClient program runs—you ***must*** embed print statements if you want full credit!

Implementation Notes

The event service we will use is part of the ORBacus system, and is located at /net/niflab/orbacus/bin/eventserv. You should each run your own instance of the event service. You will need to be able to find the event service from your programs. The easiest way I know to do this is to start the event server with the --ior flag, causing it to write its IOR on standard output. Redirect standard output to a file, thusly:

```
/net/niflab/orbacus/bin/eventserv --ior >Event.ref
```

Then, when you start your programs use the backquote technique to incorporate the contents of file Event.ref on the command line:

```
yourprogname -ORBInitRef EventService=`cat Event.ref`
```
Within your program you can use resolve_initial_references("EventService") to get the handle you need for the event service.

We will use the so-called untyped event service, the default, and we will not use the ORBacus proprietary Event Channel Factory feature.

The OMG Notification service

ORBacus does not provide a Notification service. To write your discussion about the Notification service you will need to refer to the specification at http://www.omg.org/docs/formal/02-08-04.pdf. The specification is over 200 pages – I don't expect you to read it in detail, but concentrate on the overview: try to understand how the Notification service goes beyond the features of the Event service and solves some problems that were discovered in the specification of the Event service.