Laboratory Goals / Objectives

The purpose of this lab is to investigate the event notification service by designing and implementing a Publisher/Subscriber solution. At the end of the lab, an understanding of the event notification service should be demonstrated, by providing a working simulation of the implementation.

Overview

This laboratory work is dedicated to extending the student knowledge and skills on event notification services in general and the Publisher/Subscriber model in particular. To achieve this goal, the student is required to take decisions regarding a set of event types, their attributes and when an event becomes an alert. Then, create the set of protocols for the publishers to advertise their event types, clients to subscribe to publisher(s) for event types, and the notification of the subscribers when the events occurred. In addition, the publisher can notify subscribers of its departure ("bye-bye"), or the subscriber can unsubscribe to an event type notification.

Concepts

If an object generates events and want them to be received by subscribers, it will have to publish the type of events that it will make available for observation. Objects that want to receive notifications of events published by an object subscribe to the types of events that are of interest to them. Objects that represent events are called notifications. Notifications may be stored, sent in messages, queried etc. An event particularly relevant becomes an alert.

When a publisher experiences an event, subscribers that expressed an interest in that type of event (subscribed to it) will receive notifications.

To implement an event-notification service, several decisions, as the following, have to be considered.

1. The convention used for the event type format (fields for type, publisher id, number of attributes, etc.) and its description by the set of attributes. Point out when an event becomes an alert.

2. The next important decision if about the client learning about publishers and their event types. One possibility is for clients to have already this information (wired). However, the best option is to have a notification manager where publishers can advertise their services (you can use the previously created directory service) and therefore clients can learn how to subscribe to their events' types.

3. Consider each component's functionality and define their interfaces accordingly.

Lab Work

There are several assumptions that need to be considered. The most important is that the publishers, the notification manager and the clients/subscribers will run on the same computer.

Tasks to Do

1. **Analysis**. First, draw the general, distributed event notification model, explaining the role played by each component. Choose and present the convention for event type definition. Then, describe all the protocols used by the system, such as
    a. Publishers registering their event types records and reference with the notification manager, if this is you solution.
    b. Clients learn about event types and receive publishers' references for subscription.
    c. Clients subscribe/unsubscribe for event notifications to one or more publishers.
    d. The publisher leaves the system – "bye-bye"

2. **Design**. Outline the protocols in pseudo-code. This will be your design.

3. **Programming**. Adapt the protocols above for one computer environment and then provide an implementation of the publisher, notification manager and client-subscriber (in Java or Python).

4. **Testing**. Test your implementation by simulating the execution of the protocols: the publishers register their event types with the notification manager – print message; the client looks-up event types and, if successful, subscribes to the publisher(s); the publishers send event notifications to subscribers. Subscribers print the notifications they receive, time-stamped.

Hints

1. When creating the set of notification types, it can be helpful if you'll consider a domain of interest to you, e.g., music, sports, weather, etc.
2. The publisher interface will include subscribe and unsubscribe methods. The publisher will store the call-back references for its subscribers. The publisher will use a random generator to create and issue event notifications.
3. The client interface should include the call-back method for notification, including the "bye-bye" one.

Return your results to Tasks 1-4 described above, including the pseudo-code, the **code** and **screenshots**, in a **pdf** file on moodle, by the deadline.

Add comments to your code. Place your name and student number at the top of each class file. Place comments at the method level; use one-line comments inside method bodies to describe more complex statements if you feel they are not obvious.

Questions

The following questions are to be filled in individually by each student and the pdf file retuned through Moodle before the deadline.

1. What is the event-notification service?

2. Provide the convention used for event type definition and explain your choice.

3. Provide the code for the notification manager.

4. Provide the code for the publisher.

5.Provide the code for the client/subscriber.

7. Your additional comments on implementing this service.