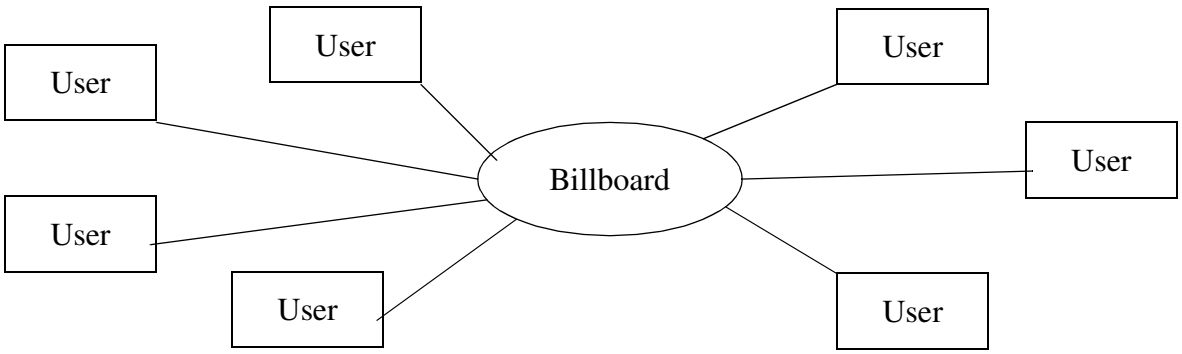
Exercise 1: CORBA

This exercise a billboard system for publishing messages will be implemented. Clients should be able to publish messages that then can be read by all Clients. The system should consist of a server process that handles all the messages and a random number of client processes that can post or read messages. A message should be able to be deleted by the user that posted it, but no others.

The system will then look like:



The system should be implemented using CORBA with the following interface (IDL)

```
module MessageBoard {
 struct Message_info
 {
   string message_id;
   string message_title;
   string message_originator;
   long long message_create_time;
 };

 exception does_not_exist
 {
    string reason;
 };


 exception not_allowed
 {
    string reason;
 };

 interface Message   // to read info from a message
 {
   string get_message_id();

   string get_text();
```

```
    string get_title();

    string get_message_originator();

    long long get_message_create_time();
  };

  interface Board // to post, read and remove messages on the board

  {
    void post_message(in string title , in string text);

    Message get_message(in string message_id) raises(does_not_exist);

    void remove_message(in string message_id) raises(not_allowed,does_not_exist);

    void set_init_message_info();
     // This operation creates a copy of all message titles on the server that
       // then can be retrieved by successive "get_next_message_info()" operations.

    Message_info get_next_message_info() raises(does_not_exist);
      // By successive calls to this operation all message titles on the server
      // can be retrieved. Before "set_init_message_info()" must be called.
      // When there are no more message titles exception "does_not_exist" will be
      // raised.
  };

  interface BoardCallback
       // this interface is used by the server to inform the clients about a change
      //  on the board.
      //  After receiving this the client should perform a new message title
      //   reading from the server as above)
  {
    void board_changed();
  };

  interface BoardConnect
       // Interface to the login object. A reference to this is given in the
       // ior-string.
  {
    Board login(in string user, in string passwd, in BoardCallback call_back)
                                            raises(does_not_exist, not_allowed);
       // Client login.
       // It is allowed to give null value for "call_back",
       //  then there will be no callback from the server.
  };
 };

 I
```

Note that the way to get all message information is to first use the operation `set_init_message_info()` which tells the server to create the information that will be sent to the client. Then the `get_next_message_info()` operation is used for getting that information for each message in turn. When there is no more message the operation will raise a `does_not_exist` exception.

There is an implementation that you can try out. The existing client can be run by the command: after you

```
java -cp BoardClient.jar CorbaClient.StartClient board.ior
```

have downloaded the necessary files (`BoardClient.jar` and `board.ior`) from the course homepage

The file `board.ior` should contain the text string ticket from the server. This can also be downloaded from the course home page. If the server has to be restarted this string can not be used. You must then get the new string from the course home page.
The client first require you to click on the Logon button where you give your login name and password. Then you can look at messages by clicking on their headers or post a new message by clicking on New

The first part of the exercise is to write your own client that uses the existing server. Don't implement the callback routine in this first implementation.

Your client does not need to have a graphical interface, especially not a complicated.

II

In the second part of the exercise you implement the server process. Wait with the call back part.

For the messages o be able to be read by other users, the messages must be put in a common data structure. This structure must be protected from concurrent changes by using the `synchronized` concept.

III

Now you should implement the callback parts. Try to do the client first.
Note that there is a danger of deadlock in the server due to the `synchronized` object. This can be solved by using multiple threads.

The IDL-file will be available at the course web page so you don't have to copy it manually.