Publish-Subscribe for Mobile Environments

Mihail Ionescu and Ivan Marsic

Center for Advanced Information Processing (CAIP), Rutgers University
{mihaii, marsic}@caip.rutgers.edu

Abstract. The Publish-Subscribe paradigm has become an important architectural style for designing distributed systems. In the recent years, we are witnessing an increasing demand for supporting publish-subscribe for mobile computing devices, where conditions used for filtering the data can depend on the particular state of the subscriber (quality of the connection, space and time locality, device capabilities). In this paper we present a stateful model for publish-subscribe systems, suitable for mobile environments. In our system, the server maintains a *state* for each client, which contains variables that describe the properties of the particular client, such as the quality of the connection or the display size. The interest of each client can be expressed in terms of these variables. Based on the client interests, an associated agent is created on the server. The agent filters the data that reach the client based on the current client state. Experimental results show good performance and scalability of our approach.

1 Introduction

In a common scenario of a publish-subscribe system, publishers connect to a centralized server to publish events and subscribers connect to the server to establish connections (or subscriptions) in which they specify the set of messages they are interested in receiving. The job of the server is to match published messages with the subscribers conditions and deliver only the relevant messages to each subscriber. For example, a client can express interest about the companies DELL or IBM when the stock quote dropped under $100 with a query as this:

$$company \in \{IBM,DELL\} \text{ AND } price \leq 100$$

A common characteristic of these systems is that the server does not maintain any state about the clients. The server keeps only the address of the client (usually an IP address) and the associated conditions. The condition is usually specified in a language based on first-order predicates and decides whether or not the message is to be sent to the particular client only based on the content of the message. The condition cannot effect any changes on the actual data that is transmitted.

The development of the mobile devices made an increasing demand to support publish-subscribe paradigm. However, the above publish-subscribe solution cannot be directly applied in mobile environments. The client should be allowed to specify how the conditions change depending on information like the quality of the connection or the space locality. We believe that the main differences between a classic publish-

J.M. Cueva Lovelle et al. (Eds.): ICWE 2003, LNCS 2722, pp. 547–550, 2003.

subscribe system and a publish-subscribe system suitable for mobile clients are the following:

- *Stateful subscribers*

  The server must maintain a state associated with each subscriber. The state of the subscriber can change due to the subscriber's mobility and activities or due to external reasons, e.g., changes in the quality of the connection.

- *Rich language for expressing the conditions*

  The language in which the conditions are written should be able to allow the interaction with the content of the message and with the state of the subscriber and it should be able to modify the message content, according to the conditions.

In this paper we present a novel architecture to support the publish-subscribe mechanism for mobile devices that meets the above requirements. An important class of applications that might benefit from our approach is also described, and a motivational example is considered.

The paper is organized as follows. We first review related work in this area. Next, we describe our approach for supporting general publish-subscribe systems in mobile environments. We then present an example in detail, elaborating the process of creation and deployment of the conditions and finally conclude the paper.

2 Related Work

Content-based publish-subscribe systems are intended for content distribution over a communication network. The most important content-based publish-subscribe systems that we are aware of are GRYPHON [1], SIENA [2], ELVIN [4] and KERYX [3].

The subscription languages of the existing systems are similar to each other and are usually based on the first-order predicates. The filtering algorithm in these situations can be made very efficient using techniques such as parallel search trees [1] or other techniques. As a general observation, a trade-off between expressive subscription languages and highly efficient filtering engines is characteristic to these systems.

3 Stateful Publish-Subscribe Systems

In our system, the server maintains a state (profile) associated with each client. The state is an abstract concept and can be composed of any number of *variables***,** which describe the properties of a particular client. The variables can be *static* (like the client's device capabilities, for example), or *dynamic* (the quality of the connection in a wireless network). In the latter case, the server has to monitor the values for the variables in order to comply with the client's conditions.

The architecture of the proposed publish-subscribe system is presented in Figure 1. The server publishes an interface, which indicates what variables from the subscriber state are available to be used as parameters in the conditions.
A subscriber uses the interface to construct one or more *conditions*, in a high-level language, and send these conditions (as a text file) to the server using a `setCondition()` command. Based on this information, the server generates an *agent*, which will be associated with this subscriber. Each time a message arrives at the server from a publisher, the associated agent for each of the subscribers is invoked to decide whether or not the message should be sent to the respective subscriber. If the message will be transmitted to the subscriber, the agent may also perform certain modifications of the data according to the user conditions. The reason for this may be to meet the network bandwidth constraints or the reduced display size. In our architecture, all of the messages are contained in a particular data type, called *UForm*.

4 Case Study: A Collaborative Graphical Editor

Here we present an example application that can benefit from the concept of stateful publish-subscribe systems. It is a multiuser graphical editor application, targeted for use in collaboration on a situation map, running on diverse devices, such as laptops or pocket PCs. In this case, each publisher is at the same time a subscriber.

We assume for simplicity that there are three types of objects that can be created: rectangles, circles and images. The user-defined policy (i.e., conditions) for distributing the messages is as follows. First the connection is considered to be of *good* quality if the available bandwidth is more than 100Kb/s, *average* quality if the bandwidth is between 10Kb/s and 100Kb/s and of *poor* quality when the bandwidth drops under 10Kb/s. If the client has a good connection it is interested in receiving notifications about all types of objects. If the connection quality is average, it is interested only in rectangles and images. Lastly, if the connection is poor, it is interested in rectangles only. In all cases, if the size of an object is greater than the size of the display available at the client site, the client is not interested in that particular object. The implementation of such a policy is shown in Table 1.

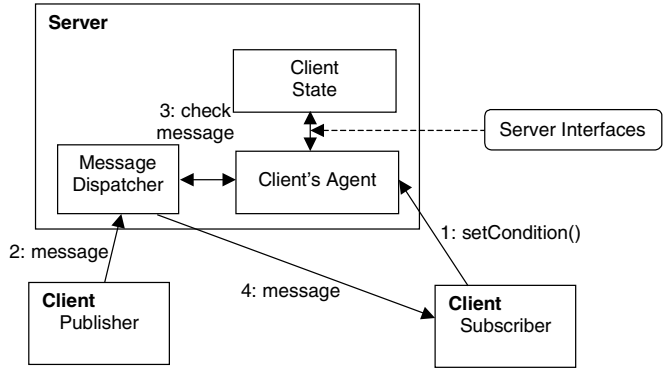

Fig. 1. The architecture of the proposed publish-subscribe system.

Table 1. The implementation of the user policy

```
public UForm testUForm(UForm uform)
{
if (getClientSize()<uform.getSize())
    return null;
if (getAvailableBandwidth()>=100.0)
{
            return uform;
}
if (getAvailableBandwidth()>=10.0)
{
if (uform.getProperty("type")="rectangle")
       return uform;
```

```
if (uform.getProperty("type")="image")
       return uform;
}

if (uform.getProperty("type")="rectangle")
       return uform;

return null

}
```

We do not expect each user to be able to generate the conditions, as presented in Table 1. Using a wizard with an appropriate graphical user interface, the application can generate automatically the conditions, based on the input from the user.

Because of the space limitation, we cannot present the detailed performance evaluation of our system. However, our tests show that the proposed architecture is scalable and perform very well under medium and high traffic.

5 Conclusions

In this paper we introduce the notion of stateful publish-subscribe systems and argue that the current publish-subscribe systems are not suitable for mobile environments. Our approach allows the conditions to interact with the current state of each subscriber and decide whether or not to send it to the subscriber and how to modify the message if necessary. Moreover, our approach can be incorporated in the current publish-subscribe systems in an incremental manner, by allowing for some of the subscribers to maintain a state at the server. Experimental results demonstrate good performance and scalability of our approach.

References

1. M. K. Aguilera, R. E. Storm, D. C. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *18th ACM Symposium on Principles of Distributing Computing (PODC),* 1999.
2. A. Carzaniga, D. S. Rosemblum and A. L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In *19th PODC,* 2000.
3. Keryx homepage, http://keryxsoft.hpl.hp.com
4. B. Segall and D. Arnold. Elvin has left the building: A publish-subscribe notification service with quenching. In *Proceedings of the Australian UNIX and Open Systems User Group Conference,* 1997.