A Publish/Subscribe Based Architecture of an Alert Server to Support Prioritized and Persistent Alerts

Sharma Chakravarthy and Nishant Vontella

Computer Science and Engineering Department and Information Technology Laboratory, The University of Texas at Arlington, Arlington, TX 76019-0015
sharma@cse.uta.edu

Abstract. This paper discusses the design and development of a publish/subscribe based distributed alert server whose requirements include: priority-based delivery, persistence, recovery, time-to-live and various other features. The approach described in this paper provides a lightweight implementation that is general-purpose and can be used for a number of applications. A new efficient sweeping algorithm is used to make sure alerts are delivered correctly and satisfy several requirements such as priority, sending existing alerts to new subscribers, and regular expression based subscription.

1 Introduction

Enterprise messaging products (or as they are sometimes called, Message Oriented Middleware products or MOM) [4] are becoming an essential component for integrating intra-company operations. They allow separate business components to be combined into a reliable, yet flexible, system. In addition to the traditional MOM vendors, several database vendors and a number of Internet related companies also provide enterprise-messaging products. Message-oriented middleware (MOM) is a client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms.

This paper discusses the design and implementation of one such messaging system called the Alert Server. Alert Server is a general-purpose alert and acknowledgement message queue and distribution mechanism. It maintains transaction logs for a comprehensive audit trail of alerts, acknowledgements and receipts. At the alerts server, the alerts are logged and queued and if necessary persisted. The Alerts Server determines if there are any subscribers for this alert and if so, forwards it to the destination. An alert producer could be a human operator who “fills in the blanks” of an alert message through a GUI or other means. Alert producers can also be software components that execute “under the hood”, invisible to human operators. An alert producer assembles the alert in reaction to some system condition and then sends for distribution. Alert consumers are those applications that are interested in receiving (a

---

1 This work was supported, in part, by the Office of Naval Research, the SPAWAR System Center-San Diego & by the Rome Laboratory (grant F30602-02-2-0134), and by NSF (grant IIS-0123730).
subset of) alerts. This is always accomplished via “registering” or “subscribing” for alerts that contain a particular pattern in the alert destination or topic data element by specifying a filter (in the form of a regular expression) during alert registration. This Server has been designed to handle C/C++ as well as Java clients.

**Alert Server Objectives:** If our Alert Server provided a union of all the existing features of messaging systems it would be much too complicated for its intended users. It is crucial that the Alert Server includes appropriate functionality needed to implement sophisticated enterprise applications. Our design and implementation of the alert server attempts to minimize the set of concepts a programmer must learn to use enterprise-messaging products. It strives to maximize portability. We start with the concept of alert producers, consumers and distributors that act as servers.

**Alert Clients:** A client can be either a producer or a consumer or both. The alert producer does not necessarily need to know who the receiver(s) of the message will be. The producer “publishes/sends” the messages to the Alert Server which is responsible for the distribution of messages. Alert Consumers are responsible for processing and responding to the alert (message) by subscribing/registering through the alert server.

**Alert Server:** Alert Server manages the alert and acknowledges messages, distribution of alerts, and crash recovery. Important goals of an alert server are:

1. Implement a publish/subscribe model. This model has been chosen over the point-to-point model because point-to-point (PTP) models are built around the concept of message queues. Each message is addressed to a specific queue; clients extract messages from the queue(s) established to hold their messages. Clients have to pull the message from the server rather than the server pushing it to the client after processing the messages. Publish and subscribe (Pub/Sub) clients send messages to the alert server. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the alert server. The alert server pushes the messages arriving from a node’s multiple publishers to its (multiple) subscribers.
2. Insure delivery of alerts before it expires (time-to-live) based on priority.
3. Dynamic delivery of alerts between multiple producers and consumers based on their registration/subscription topics.
4. Persistence of alerts and acknowledges to handle to crash recovery of clients.
5. Maintain the privacy and integrity of the messages.

### 2 Design of the Alert Server

First, we describe the functionality to be supported by the Alert Server and then describe how the Alert Server has been designed to achieve this functionality. The Alert Server should provide API to send messages from one application to another. The client applications should also be able to register and unregister topics of interest. They should be able to cancel messages and should also be able to send acknowledgements and/or receipts. Besides, the Alert Server should have a delivery