Chapter VII
Event-Based and Publish/ Subscribe Communication

Erwin Aitenbichler
Technische Universität Darmstadt, Germany

ABSTRACT

Ubiquitous Computing assumes that users and their computing devices are highly mobile. Because it is unlikely that mobile networks will be equally available in the same quality everywhere, there may be varying levels of connectivity, ranging from full network availability through low-bandwidth connectivity, to no connection at all. As a consequence, software components in the system cannot assume that the connections between them are static and always available. The event-based style is essential for ubiquitous computing, since it offers a good decoupling of the communicating entities in terms of space, time, and program flow. This chapter starts with an introduction to the different interaction models found in distributed systems. Next, a classification of publish/subscribe-systems is presented. We then describe a formal data and filter model that allows us to precisely define the semantics of event filters. Based on this model, we discuss different routing algorithms for the efficient distribution of event notifications in a network. Finally, a number of examples for publish/subscribe systems are presented.

INTRODUCTION

The communication models found in distributed systems can be classified according to the following two attributes: who initiated the communication and how the communication partner is addressed. The resulting four models are shown in Table 1. Provider-initiated communication is also often called *push* communication and consumer-initiated communication is also often called *pull* communication.

Table 1. Taxonomy of communication models

	Consumer-Initiated (“pull”)	**Provider-Initiated (“push”)**
Direct Addressing	Request/Reply	Callback
Indirect Addressing	Anonymous Request/ Reply	Event-Based

Request/Reply: The most widely used model is request/reply. Any kind of client-server communication or remote procedure call belongs to this class. The initiator is a client, which requests the execution of an operation or the delivery of data. The provider is a server that processes the requests issued by clients. Servers are directly addressed by clients, resulting in a tight coupling of the cooperating entities.

Anonymous Request/Reply: builds on request/reply as basic action, but without specifying the provider directly. Instead, requests are delivered to an arbitrary provider or a set of providers automatically chosen by the communication system. For example, the IP Anycast mechanism routes a packet to the nearest member of a group of destinations without resolving the target IP address in advance.

Callback: In this model, consumers register their interest in certain events directly with specific, known providers. Providers maintain lists with the consumers' addresses and notify them when an event of interest has occurred. Thus, this model introduces a tight coupling between the communicating entities. The *Observer Design Pattern* is a well-known example for this interaction.

Event-based: In the event-based model, providers publish notifications and consumers subscribe to notifications by issuing subscriptions, which are stateless event filters. Communication between providers and consumers is decoupled by a broker. Consumers can have multiple active subscriptions, and after a client has issued a subscription, a message broker delivers all future matching notifications that are published by any provider until the client cancels the respective subscription.

The event-based communication style has been used for decades in the area of graphical user interfaces. It is now also increasingly being applied to widely distributed and critical systems such as real-time, automotive, traffic control (including air and rail), e-commerce, logistics, workflow, and mobile computing. The acceptance of this paradigm is also witnessed by its incorporation into such widespread standards as CORBA and Java Enterprise Edition.

Events, Notifications, and Messages

Any phenomenon in the real world or any kind of state change inside an information system can be an *event*. However, it must be observable and some component in the information system must observe it in order to notify parties interested in the event.

The reification of an event is called *notification*. Observers create these notifications. Notifications represent the data describing events. They contain precisely formatted data to notify interested parties in the distributed systems about the occurrence of events. The content of a notification is usually application-dependent. It may simply indicate the occurrence of an event, or may as well contain additional information about the object observed and its context.

For example, consider a room that is equipped with a badge-based location system (see the chapter "A Focus on Location Context" in this book). Now, if *a person enters the room*, we call this occurrence an event. The badge system is able to observe this event and publishes an event *notification* to inform interested parties about this occurrence. This notification is a data structure that would typically contain the event type, the identifier of the badge, the identifier of the receiver, and a timestamp. The most common data models for notifications are typed name-value pairs, semistructured data (i.e., XML), or objects of some programming language.

Messages are transport containers that carry data between communication endpoints. In the event-based model, the endpoints are *publishers* and *subscribers*. Beside request and reply messages, notifications are messages with a special purpose. They inform subscribers that an event has occurred.

Decoupling

Event-based communication decouples providers, or more specifically, producers and consumers in terms of the following three aspects: