Distributed Coordination-based Systems

A genration of distributed systems that assume that the various components of a system are inherently distributed and that the real problem in developing such systems lies in coordinating the activities of different components. In other words, instead of concentrating on the transparent distribution of components, emphasis lies on the coordincation of activities between those components.

Game servers are typical example of distributed coordination based systems since servers need to coordinate diverse activities. Examples are managing instances, users and group objects, persistent data and other game states.

Introduction to coordination models

Key to the approch in coordination-based systems is the clean separation between computation and coordination.

The coordination part of a distributed system handles the communication and coopration between processes. It forms the glue that binds the activities performed by processes into a whole.

Two dimensions:

- Temporal and referential
- Coupled and decoupled

mailbox : temprorally decoupled, referentially coupled, which means one needs to know the other, but each can work separately.

meetings: temporally coupled, referentially decoupled, which means processes need to work at the same time withoug the knowlege (direct communication) of others. publish / subscribe systems are examples of meetings.

generative communication: temporally and referentially decoupled. The key idea in generative communication is that a collection of independent processes make use of a shared persistent dataspace of tuples. Tuples are tagged data records consisting of a number (but possibly zero) type fields. Processes can put any type of record into the shared dataspace (i.e., they generate communiation records).

generative communication is an approach used in x2clr, a c# game server framework, to provide mesage field value based communication. It uses broadcasting to processes to provide shared dataspace.

generative communication is considered as a form of publish/subscribe system.

Architectures

An important aspect of coordination-based systems is that communication takes place by describing the characteristics of data items that are to be exchanged. As a consequence, naming plays a crucial role.

Overall Approach

data item: a series of attributes. data item is published when it is made available to other processes to read. A subscription needs to be passed to the middleware, containing a description of the data items that the subscriber is interested in. Such a description typically consists of some $(attribute, value)$ pairs, possibly combined with $(attribute, range)$ pairs. Descriptions can sometimes be given using all kinds of predicates formulated over the attributes, very similar in nature to SQL-like queries in the case of relational databases.

subscriptions need to be **matched** against data items, as shown in Fig 13-2. When matching succeeds, there are two possible scenarios. In the first case, the middleware may decide to forward the published data to its current set of subscribers, that is, processes with a matching subscription. As an alternative, the middleware can also forward a notification where subscribers can execute a read operation to retrieve the published data item.

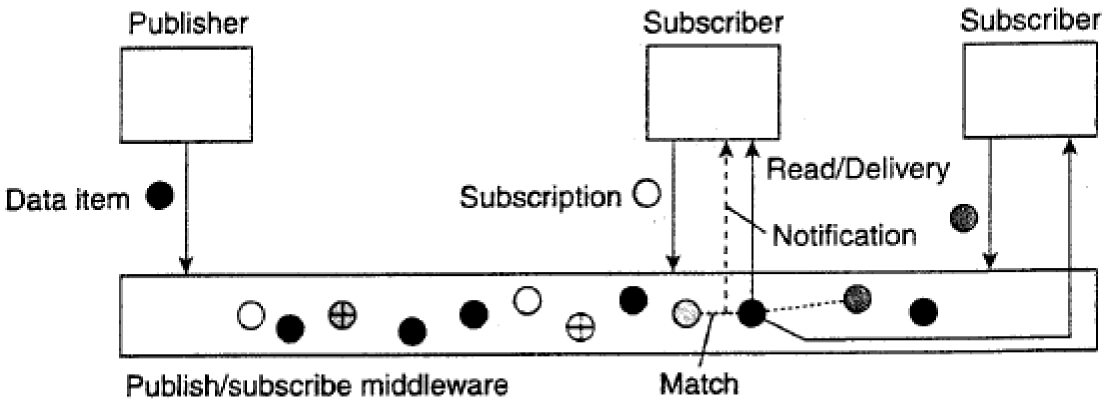

Figure 13-2. The principle of exchanging data items between publishers and subscribers.

In the model described so far, we have assumed that there is a fixed set of n attributes $a_1, a_2, \ldots, a_n$ that is used to describe data items. In particular, each published data item is assumed to have an associated vector $(a_1, v_1), (a_2, v_2), \ldots, (a_n, v_n)$ of $(attribute, value)$ pairs. In many coordination-based systems, this assumption is false. Instead, what happens is that events are published, which can be viewed as data items with only a single specified attribute.

대다수 게임들에서 domain / type 또는 key 형태의 정수 필드를 주로 사용하여 publish/subscribe를 처리한다. subscription 하드 코딩 되는 경우도 많이 사용하고 있다.

Events complicate the processing of subscriptions. To illustrate, consider a subscription such as "notify when room R4.20 is unoccupied and the door is unlocked". Typically, a distributed system supporting such subscriptions can be implemented by placing independent sensors for monitoring room occupancy (e.g., motions sensors) and those for registering the status of a door lock. Following the approach sketched so far, we would need to *compose* such primitive events into a publishable data item to which processes can then subscribe. Event composition turns out to be a difficult task, notably when the primitive events are generated from sources dispersed across the distributed systems.

Clearly, in coordination-based systems such as these, the crucial issue is the **efficient and scalable implementation of matching subscriptions to data items** along with the construction of relevant data items. From the outside, a coordination approach provides **lots of potential for building very large-scale distributed systems due to the strong decoupling of processes**. On the other hand, as we shall see next, devising scalable implementation without losing this independence is not a trivial exercise.

Traditional Architectures

The simplest solution for matching data items against subscriptions is to have a centralized client-server architecture. This is a typical solution currently adopted by many publish / subscribe systems, including WebSphere, JMS. A more elaborate generative communication models such as Jini and JavaSpaces are mostly based on central servers.

Jini의 JavaSpace와 Tuple을 사용한 generative communication에 대해 설명한다. 각 필드 값에 대해 매칭을 하는 시스템은 정교함이 분산되기 어렵기 때문에 (각 프로세스마다 타잎이 다를 수 있어) 중앙화 될 수 밖에 없다는 점을 설명한다.

TIBlRendezvous의 멀티캐스팅 모델이 있다. 로컬에 subscription하고 멀티캐스팅된 데이터 아이템을 포워딩한다. 멀티캐스팅 때문에 WAN (Wide Area Network)에 대해서는 필터링이 필요하다. 멀티캐스팅은 중앙 저장소를 만들어 주므로 정교한 필드 데이터 기반 subscription이 가능하게 한다.

P2P Architectures

DHT (Distributed Hash Table) based matching. Field 기반 매칭에 적용하려면 매우 어려워진다. 단순한 속성 값의 해시 값들에 대한 매칭은 보다 쉽다.

Example: A Gossip-Based Publish/Subscribe System

범위 기반 매칭을 지원하는 시스템의 한 예에 대한 설명을 진행. 복잡하다.

매칭이 빠르고 subscription이 편한 시스템의 구성. 상대방을 몰라도 되고 같은 시간에 동작하지 않아도 되는 구조. 단지, timeout 은 있는 구조. 전송과 수신을 구분하지 않는 구조.

Application to Game Servers

temporal decoupling : not possible. In game, distributed states are dependent on each other mostly. Some services can be designed and implemented with temporal decoupling, but not all of the services are possible.

referential decoupling: possible to some degree? field value based subscription is required to provide referential decoupling because field based filtering is required to resolve reference.