以 Orbix 製作的 CORBA 相容性出版－訂閱機制
CORBA-Compliant Publish-Subscribe Mechanism in Orbix

陳鵬宇
Chen Pen-Yu

國立台灣科技大學資管學程
Department of Information Management
National Taiwan University of Science and Technology
pychen@event.cs.ntust.edu.tw

陸承志
Luh Cheng-Jye

國立台灣科技大學資訊管理系
Department of Information Management
National Taiwan University of Science and Technology
luh@event.cs.ntust.edu.tw

摘要

本研究利用 Orbix 實現一個符合 CORBA 標準的 Publish-Subscribe 機制。此機制具有重用性高、移植性高、應用彈性高， 及跨平台的特性， 可做爲建立一個完整的 CORBA-based 訊息系統之基礎。
關鍵字: 分散式物件、 CORBA 、分散式運算

Abstract

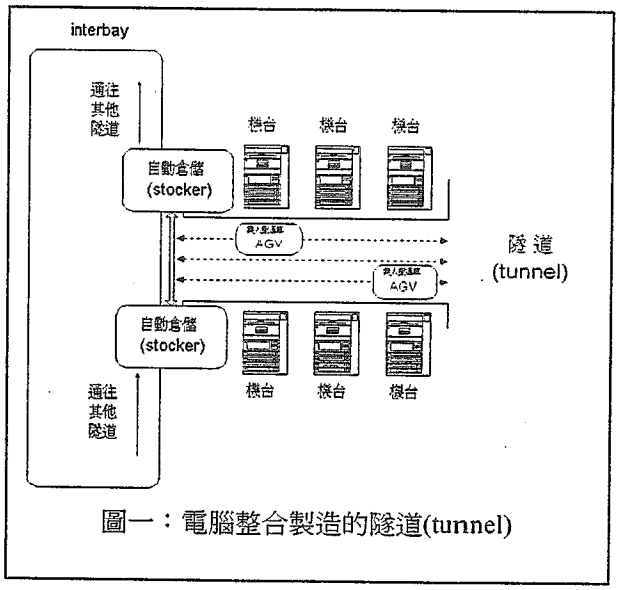
This research implemented a CORBA-Compliant Publish-Subscribe Mechanism in Orbix. This mechanism has features of reusability 、 portability 、 flexibility ， and interoperability. A Complete CORBA-based message system can be built on the base of this mechanism.
Keywords: CORBA 、 Publish 、 Subscribe 、 Distributed Object 、 Distributed Computing

1. 前言

無論在那個領域中，常可見到相同或不同主機上的程式間，需要達到"資料同步化"的功能，或者是因爲系統整合的需要，必須發展一套訊息系統。例如在以隧道(tunnel)爲基礎的電腦整合製造系統(CIM)的架構中(如圖一)，每一個隧道都有一個物料控制系統 (MCS) 負責記錄目前隧道內的材料在那個機台做什麼動作，以及負責材料的運送管理。若隧道內需要某種材料時，此隧道的 MCS 就會去向有此材料的隧道的 MCS 索取， 所以每一個 MCS 也都要有別隧道的 MCS 的資料，因此便需要讓所有 MCS 的資料同步化。

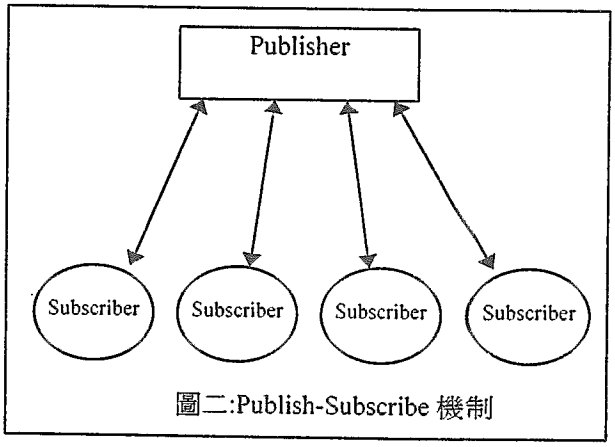
若要解決這樣的問題，可以把其中一個 MCS 設爲 master ， 其他 MCS 修改自己的資料後，必須告知 master MCS，再由 master MCS 去通知所有 MCS 改變資料，這樣就形成了一個 Publish-Subscribe 的機制，其中 master MCS 就是 Publisher ，其他的 MCS 都是 Subscribers 。

我們的目標就是以 CORBA 爲基礎，建立一個 Publish-Subscribe 的物件機制，並且讓別人可以很方便地將它套用在自己的程式裏; 或者，也可以利用它來建立一個完整的 CORBA Based 訊息系統。



圖一：電腦整合製造的隧道(tunnel)

2. Publish-Subscribe Mechanism



圖二:Publish-Subscribe 機制

所謂的 Publish-Subscribe 機制，指的是有一或多個 Subscribers 向同一個 Publisher 註冊，當此 Publisher 有特殊訊息或資料時 (視應用的情況而定)

就會把此訊息分送給所有的 Subscribers 或者是某個特定的 Subscriber 。

這樣的 Publish-Subscribe 機制可以應用在各種需要訊息處理的系統中，也可以應用在需要資料同步更新的情況裏。但是每個系統大都必須自行發展此機制，以及自行定義其資料格式、傳送方式等，因此本文希望能藉由 CORBA 的分散式物件環境來建立一個來建置一個重用性高，移植性高，應用彈性高，而且可跨各種不同平台的 Publish-Subscribe 機制。

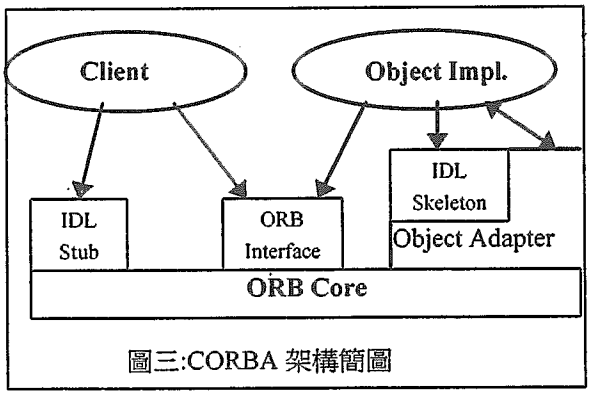
理想的 Publish-Subscribe 機制必須達到以下要求:

1. Publisher 必須能夠辨認並記錄所有向它登記的 Subscriber 。
2. Subscriber 在訊息來到之前必須能夠處理其他事情，不能爲了等待訊息而使整個程式被 Block 住。
3. 爲了提昇效率， Subscriber 的訊息處理函式最好是由訊息驅動的，也就是由 Publisher 將訊息主動傳給 Subcriber ，並且由此驅動 Subscriber 的訊息處理函式，而不要由 Subscriber 定期地詢問 Publisher 有沒有訊息要給它。
4. 此機制的使用者必須能定義自己的訊息處理函式及訊息格式，而且定義時可以不必修改原始程式，只要使用物件導向的繼承和 Override 即可。
5. Publish-Subscribe 機制必須保留最大的空間讓使用者可以自己決定它要的訊息系統的結構。

3. CORBA

CORBA (Common Object Request Broker Architecture) 是 OMG 制定的分散式物件環境規格。[1][2]

圖三是 CORBA 架構的簡圖，爲了方便說明，與本文無關的一些細節，如:Dynamic Invocationg Interface 省略不談。下面就各別說明圖三中各元件的用途，以及在此架構中如何呼叫一個分散式物件的成員函式。



圖三:CORBA 架構簡圖

IDL Stub / IDL Skeleton：在製作各種分散式物件之前，需用 IDL (Interface Definition Language) 語言定義此分散式物件的 Interface ，然後用 IDL Compiler 將此 Interface Definition 編譯成程式語言(如 C++)。 IDL Compiler 會產生兩種檔案，一個是給 Client 端使用的 Stub Class ，此類別爲此分散式物件在 Client 端的代理人， Client 端對此類別的各種操作都會被傳到 Object Implement (物件的實作部份)去，結果也會經由 Stub 傳回給 Client 。與 Stub 相對應的是 Skeleton 類別，對物件的呼叫傳來時，會交給 Skeleton ，讓它去實際操作 Object Implement 做 Client 端要求的動作。

ORB Core: 負責基本的尋找各種 Object Service 的功能，並且傳送 Client 的要求訊息及 Object Implement 傳回的執行結果。

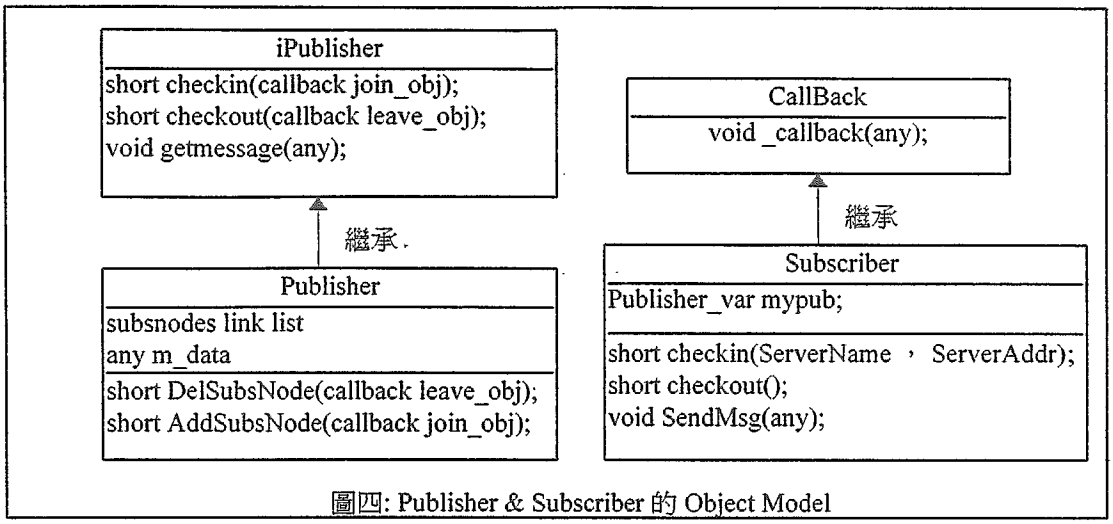
Object Adaptor: 負責提供一些較爲複雜的服務給 Object Implement ，例如:產生 Object Reference 、對 Skeleton 呼叫成員函式、安全檢查、 Object Implementation 的啓動及關閉、以及 Object Implementation 的註冊等等。

當一個 Client 要使用分散式物件時，必須先取得此分散式物件的 Object Reference ，所以 Client 要向 ORB 要求某 IDL Stub 類別所對應的 Object Impl. 之 Object Reference ， ORB 找到此 Object Impl. 所在的位置後，就交給 Object Adapter 去啓動它(Object Impl. 通常被寫在可執行的程式檔案內，稱爲 Object Server ，被啓動之後會自行產生 Object Impl. 的 instance)，並產生它的 Object Reference ，然後交給 ORB 去傳回給 Client 。此 Object reference 會包含在 IDL Stub 中，因此 Client 對 IDL Stub 的所有操作都會被傳送到 Object Impl. 去。

最後還有一個重要的觀念， Object Impl 其實是一個用程式語言寫的 Object Class ，所以它是被包含在 Object Server 程式中，換句話說，一個程式可以同時包含相同或不同物件的 Stub 、 Skeleton 和 Object Impl. ，它可以同時是某種物件的 Object Server ，又是別的物件的 Client 程式， 這點對我們的 Publish-Subscribe 機制十分重要。

利用 CORBA 來實現 Publish-Subscribe 機制的好處有:

1. **重用性高**: 這個好處主要來自於 OOPL ，本文的 Publisher 和 Subscriber 都是以 C++ 實做而成的物件，而且其主要的成員函式多以虛擬函式寫成，故在應用時只要定義一個新的物件繼承 Publisher 或 Subscriber ，並 Override 某些成員函式即可。
2. **移植性高**: 因爲是建構在 CORBA 標準之上，並且有固定的 Object Interface ，所以就算在不同的 CORBA Implementation (如 Orbix ， ILU 或是不同平台上的 Impl.版本) 只要讓它的 IDL Compile Publish-Subscribe 的 Interface Definition 以產生 Skeleton 及 Stub ，剩下的工作就是把 Publish-Subscribe 的程式碼放進 Object Implementation 的地方就好了。
3. **應用彈性高**: 有兩個原因會使得 Publish-Subscribe 機制的應用彈性增強，一是上述第 1.



圖四: Publisher & Subscriber 的 Object Model

並且有固定的 Object Interface，所以就算在不同的 CORBA Implementation (如 Orbix，ILU 或是不同平台上的 Impl.版本) 只要讓它的 IDL Compile Publish-Subscribe 的 Interface Definition 以產生 Skeleton 及 Stub，剩下的工作就是把 Publish-Subscribe 的程式碼放進 Object Implementation 的地方就好了。

3. **應用彈性高**：有兩個原因會使得 Publish-Subscribe 機制的應用彈性增強，一是上述第 1. 點中所提到的 OO 的特性，使得使用者可以針對自己的需要定義自己的函式，另一個原因則是 CORBA 標準中的 Any 類別。Any 類別可以包含任何型式的資料，它並包含了 TypeCode 類別以供人判斷它正包含了什麼型式的資料。以 Any 類別當做訊息在 Publisher 與 Subscriber 間傳送，就等於爲使用者解決了訊息資料的格式問題，使用者可以依其應用上的需要來傳送任何型態的資料。在程式中也可以用 TypeCode 類別很輕鬆地判斷訊息的種類，並以適當的函式處理之。
4. **跨平台**：Publisher 及 Subscribers 可以各自散佈在各種不同的平台上，只要 Publisher 所在的機器上有 CORBA Deamon 存在，可以爲 Publisher 及 Subscriber 建立連線即可。Subscriber 的機器上則不需任何額外的程式，就可以和 publisher 互通訊息，這是因爲 CORBA 本身用 ORB 來溝通各個物件，而 ORB 又是建構在 TCP/IP 上，故只要是有 TCP/IP 功能的平台皆可使用。

4. Implementation

Overview

Publish-Subscribe 機制的精神在於，許多 Subscribers 可向同一個 Publisher 註冊，而 Subscribers 向 Publisher 註冊後，Publisher 若有需要傳送訊息時，會把訊息傳給每一個有向它註冊的 Subscribers。因此 Subscriber 必須能對 Publisher 做 Check In (註冊) 和 Check Out (取消註冊) 的動作；而 Publisher 則要傳送訊息給 Subscriber，在這裡我用了類似 Microsoft Windows 作業系統的訊息處理方法[3]。也就是:Subscriber 在 Check In 向 Publisher 註冊時，也一併告訴 Publisher 它的訊息處理函式的位址。當 Publisher 要傳訊息給 Subscriber 時，Publisher 會主動呼叫 Subscriber 的訊息處理函式，並以此處理它要傳給 Subscriber 的訊息(Subscriber 的訊息處理函式是由 Publisher 呼叫，但在 Subscriber 所在的機器上執行)。Publisher 與 Subscriber 的 Object Model 可以表示如圖四。

圖四中的 *iPublisher* 及 *CallBack* 爲 CORBA 的 Remote Object Interface(爲 abstract class)，*Publisher* 及 *Subscriber* 則分別是前兩者的 Implementation(繼承 Interface 並 Override 所有虛擬函式)。

首先說明 Publisher 各個成員函式的用途，iPublisher::checkin() 需要一個 callback remote object 的 Object Reference，它會把得來的 Object Reference 加到一個 link list 之中(經由呼叫 Publisher::AddSubsNode() 函式)。反之，iPublisher::checkout() 則是把某個 Object Reference 由 link list 中去掉經由呼叫 Publisher::DelSubsNode() 函式)。iPublisher::getmessage() 的參數是一個 any object，Publisher 以此函式取得訊息資料，在目前的版本中，iPublisher::getmessage() 拿到訊息後馬上就呼叫 Link List 中所有 Object Reference 的 CallBack::_callback() 函式，把此訊息送出去，Publisher::getmessage() 是一個虛擬函式，使用者可以依其需要 override 此函式以做一些不同的處理。

Class *Publisher* 的 instance 會放在一個 Object Server 的程式裏，等待 Subscribers 的連接，*Subscriber* 的成員函式 *checkin()* 接受 ServerName 和 ServerAddr 分別 Publisher 的 Mark Name，及其所在的位址。所以在 Subscriber 的 checkin() 中，會找到並 bind 其參數所指定之 Publisher，即取得此 Publisher 的 Object Reference。然後呼叫它的 iPublisher::checkin() 並把自己的 Object Reference

傳給 Publisher 正式向它註冊，所以 Subscriber::checkin()的程式碼會像這樣:

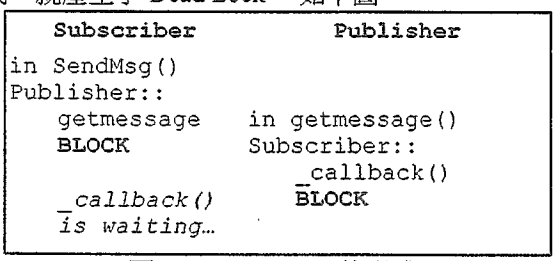
```
short Subscriber::checkin
     (char* SrvName , char* SrvAdd)
{
   mypub=Publisher::bind(SrvName,SrvAdd);
   mypub->checkin(this);
}
```

其他的函式還有 Subscriber::SendMsg() 呼叫 iPublisher::getmessage() 以便傳送訊息給 Publisher ， Subscriber::checkout() 則會呼叫 iPublisher::checkout(this) 向 Publisher 取消註冊。

CallBack::_callback() 是一個很重要的函式，Publisher 呼叫它以便傳送訊息給 Subscriber ，所以它需要一個 any object 當參數，而在 _callback() 程式中，則要判斷這個傳進來的 any object 是那種 type ，並呼叫其對應的函式去處理這項訊息。對 Subscriber 而言最重要，最需要使用者改變的就是這個函式，所以它最好是一個虛擬函式；另一方面，它也必須是一個虛擬函式，這是因爲當 Subscriber 向 Publisher checkin 時，是傳 callback 這個基底類別給 Publisher ，若 Subscriber::_callback() 不是虛擬函式， Publisher 只會呼叫到 Subscriber::_callback() ，就算使用者由 Subscriber 所衍生的子類別有 override _callback() 也不會被呼叫到。這也是使用者必須注意的，如果使用者所衍生的 Subscriber 子類別有可能會再衍生別的類別，則它所 override 的_callback() 函式最好也要宣告成虛擬函式，才能確保以後的子類別的 _callback()函式也能正確地被呼叫到。

Oneway Function

在 CORBA 環境中，一般呼叫 Remote Object 的成員函式都是 Synchronous 模式的，也就是說，呼叫的一方會 Block 住，等到 Remote 端執行完被呼叫的函式後，呼叫方才繼續執行。這樣的模式會使得 Publish-Subscribe 產生 Dead Lock ，亦即當 Subscriber 在 SendMsg() 裏呼叫 Publisher 的 getmessage()後， Subscriber 即 Block 住，等待 Publisher 執行完 getmessage()， 但若 Publisher 的 getmessage 裏要傳某項資料回同一個 Subscriber 而呼叫了 Subscriber 的 _callback()，並 Block 住，則因爲 Subscriber 本身也在等待 getmessage()的完成，就產生了 Dead Lock ，如下圖。



圖五： Dead Lock 的形成

要解決 Dead Lock 的問題，就必須把 iPublisher::getmessage() 宣告爲 oneway function ，即呼叫後就直接執行後面的指令，不必等待它結束。 Oneway Function 的宣告在 IDL 裏:

```
// Publisher.idl
interface iPublisher
{ ...
  oneway void getMessage(in any SubsMsg);
}
```

另外，爲了節省 Publisher 的工作時間，_callback() 也宣告成 Oneway Function ，以免 Publisher 還要等 Subscriber 做完 _callback()才能再去呼叫下一個 Subscriber 的 _callback()。

```
interface callback
{
   oneway void callback(in any CBMsg);
};
```

Message Type

在這裡我要把焦點放在 Publisher - Subscriber 之間的通訊所使用的資料結構，也就是 CORBA 中的 Class Any 。使用這個類別我們可以很輕鬆地把資料包裝起來，以及判斷訊息的類別。

CORBA 的 Class Any 可以包含任何型態的資料，包含基本的 short int 、 long 、 char 、 boolean 、 float 等，甚至 struct 、 enum 也都可以，而 Any 所包含的 TypeCode object 可以用來表示這些型態。在 CORBA 中，判別 TypeCode 最簡單，也是最適合 Publish-Subscribe 機制使用的方法，就是直接判斷它的 Constant 爲何。 TypeCode 的 Constant 包括了一些基本的型態，如： _tc_null 表示 NULL ， _tc_short 表示 short ，_tc_char 表示 char 等等。判斷的方法是:

```
if(mytypecode == _tc_short) {...}
     或
if(mytypecode.equal(_tc_short)) {...}
```

然而，一般在應用訊息系統時，大都會有自訂的 Message Type ，以及其對應的資料結構，例如一個訊息名爲 EVENT_ERR ，其資料結構爲一個 struct，包括了一個 int errno，及一個 char* reason 。這時使用 Class Any 就更顯得出其方便性了，我們只要在 IDL 檔案裏多加這個 strcut 的宣告:

```
// EVENTS.idl
struct EVENT_ERR
{
  short errno;
  string reason;
};
```

並在用 IDL Compiler 編譯時使用 -A 參數(這會依 IDL Compiler 而有所不同)，IDL Compiler 會自動爲此 struct 產生一個 TypeCode 的 Constant ，_tc_EVENT_ERR 。於是我們在 Subscriber::_callback() 裏就可以用

```
if(typecode_of_Msg==_tc_EVENT_ERR) {…}
```

來判斷傳進來的訊息是那一種訊息了，而不必在資料裏加其他的訊息判斷的欄位。再於 Class Any 可以直接包含任何型別的資料，而且直接宣告資料結構，就可以在程式中判斷，這些特性使得 Publish-Subscribe 的設計和使用都十分方便。

5. Applications

以上所寫的都是 Publish-Subscribe 最基礎的部份，也就是它的運作方式。然而，在實際應用時，爲了能應付各種情況，或者爲了增加此機制的效率，可以做一些特別的改變，特別是加入 Multithread 的功能，在這裡我分別說明 Multithread 在 Client (Subscriber 所在的程式)及 Server 端 (Publisher 所在的程式) 的貢獻。

MultiThread in Client

雖然 Subscriber 接受 Message 的方法是讓 Publisher 主動呼叫 Subscriber 的 _callback() 函式，但不表示 Subscriber 就可以不必爲接受 _callback() 做一些準備。其實 CORBA 的 Remote Call 是經由 TCP/IP 送一個 Request 資料到 Remote 端而已，所以如果 Subscriber 一直不去處理 Publisher 呼叫 _callback() 的 Request，這個 Request 就一直留在 Subscriber 的 Queue 裏面。

在 Orbix (CORBA 的實作之一)環境之中，Subscriber 要執行 *CORBA::Orbix.processEvents()* 或 *CORBA::Orbix.processNextEvent()* 來處理留在 Queue 裏的 Request[5][6]，或等待 Request 的來臨並處理之。然而，一般來說，Subsriber 平時都有其他的工作要做，這時就會需要用到 Multithread 或 Timer 了。

一個 Chat Room 程式是一個 Publish-Subscribe 的好例子，每個 user 進入一間 Chat Room 就好比是一個 Subscriber 向 Publisher 註冊了一樣，不管那一個 user (Subscriber) 送出聊天的訊息給 Chat Room (Publisher)，它就必須傳送給 Chat Room 中的所有 user(Subscriber)。在這種情形裏， user (Subscriber) 就要一邊等待 Chat Room 送來的訊息，一邊又要處理使用者 key in 的資料。所以 Subscriber 這端就要用 Multithread 以建立兩個 Thread ，一個只執行 processEvents()以等待並處理所有的 Request，另一個就去處理使用者 Keyin 的資料。在 CORBA 中，Client 端若使用 CallBack 常需要用到這種方法。

MultiThread in Server

此外，Multithread 也可以用在 Publisher，我們可以安裝一個 Thread Filter 在 outRequestPreMarshal 中[6]，如此一來，每當 Publisher 要呼叫 Subscriber 的 _callback() 時，就會先建立一個 Thread 再去呼叫，這樣的好處是，若 Link List 中的某一個 Subscriber 與 Publisher 之間的連線太慢或甚至已經中斷了，都不會影響到對其他 Subscriber 的 _callback()呼叫，而且也可以提高系統的效能。

6. 結論

本研究在 Orbix 環境製作一個符合 CORBA 標準的 Publish-Subscribe 機制，透過此機制 Subscriber 可以不必經由自己主動詢問來取得 Publisher 傳給它的訊息，而且使用者可做各種結構的訊息系統的應用，只要繼承 Publisher 或 Subscriber 並 Override 它們的成員函式即可。

爲了使此機制更加完整，未來必須在兩個方面進一步研究:

1. **Scheduling**: Publisher 應該要以符合公平或高效率，或是其他的原則的順序來決定傳送訊息給 Subscriber 的次序。
2. **可轉移的 Publisher**: Publisher 因爲種種原因會需要將所有 Subscribers 轉移給另一個 Publisher ，例如某個 Publisher 所在的主機即將 shutdown ， Publisher 需將所有 Subscribers 轉移給別台主機的 Publisher ，甚至，若 Publisher 已經當掉了， Subscribers 可以自行轉移到別的 Publisher 去。

參考文獻

[1] "The Common Object Request Broker:Architecture and Specification" , Revision 2.0 July/1995

[2] "http://www.omg.org/library/specindx.htm",specification library on OMG Web Site.

[3] "Windows 程式設計實務",李元泰、施威銘著,旗標出版,12/1993

[4] Steve Vinoski , "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments" , IEEE Communications Magazine , Vol. 14 , No. 2 , Feburary , 1997

[5] "Orbix Reference Guide",Release 2.0 Nov. 1995,IONA Technologies Ltd.

[6] "Orbix Programming Guide", Release 2.0 Nov. 1995,IONA Technologies Ltd.