國防科技學術合作協調小組研究計畫成果報告

大型軟體即時作業系統之規劃研究

The Research on Large-Scale, Real-Time Software and Operating Systems

計畫編號：NSC 90 - 2623 - 7 - 009 - 004

執行期間： 90 年 1 月 1 日至 90 年 12 月 31 日

計畫主持人：胡 毓 志 助理教授

共同主持人：袁 賢 銘 教 授

執行單位：國立交通大學資訊科學學系

中華民國 91 年 2 月 20 日

國防科技學術合作協調小組研究計畫摘要表

<table>
<tr><td rowspan="2">計畫名稱</td><td colspan="5">中文　大型軟體即時作業系統之規劃研究</td></tr>
<tr><td colspan="5">英文　The Research on Large-Scale, Real-Time Software and Operating Systems</td></tr>
<tr><td rowspan="2">學術執行單位
主持人</td><td>姓名</td><td>胡毓志</td><td rowspan="2">軍方對應單位
主持人</td><td>姓名</td><td>劉培楠</td></tr>
<tr><td>單位級職</td><td>交大資科系
助理教授</td><td>單位級職</td><td>中山科學研究院
簡任技正</td></tr>
<tr><td rowspan="2">計畫時程</td><td colspan="2" rowspan="2">本期計劃：　自　90 年 1 月 1 日
至 90 年 12 月 31 日</td><td>計畫編號</td><td colspan="2">NSC90-2623-7-009-004</td></tr>
<tr><td>金額</td><td colspan="2">544500</td></tr>
<tr><td>計畫歸屬</td><td colspan="5">（ ）材料與應用化學　（ x ）電子與資訊系統　（ ）系統管理　（ ）遙　測
（ ）航空技術　（ ）機械製作與應力　（ ）兵器系統　（ ）其　他</td></tr>
<tr><td colspan="6">研究內容摘要：

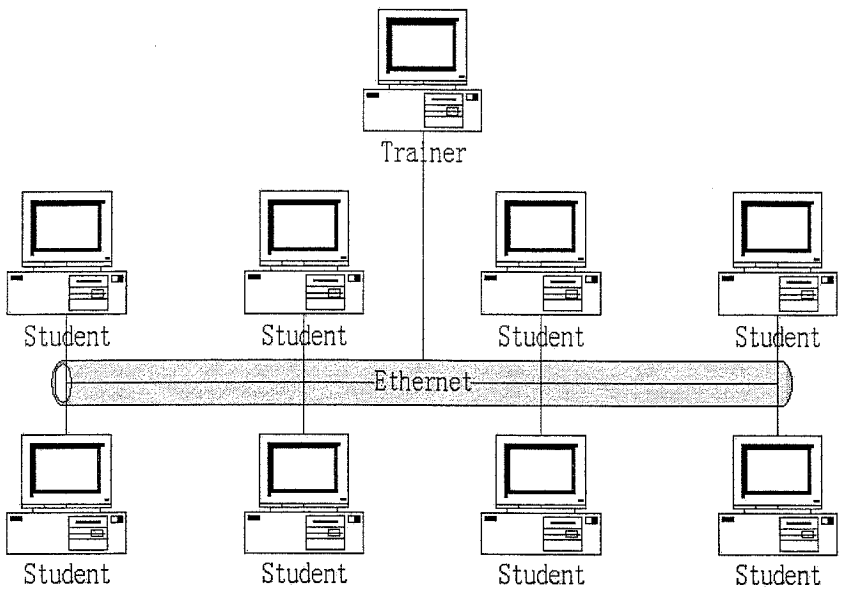
由於 Java 本身所具有的下面幾項優點，如平台中立性、簡潔的物件導向模型、執行時期的動態類別載入、整合了多執行緒、網路功能及系統安全的維護機制…等多項特質，Java 漸漸地將會取代 C，成為未來主流的程式語言。也因此，如何擴充 Java 的功能，使它能用來開發 real-time system，也已經變成目前 real-time community 裡，一個非常重要的主題，也有超過一個以上的 real-time Java 的標準正在制定中。除了 Java 之外，將 open source 的 Linux 作業系統應用在 real-time system，也是最近相當熱門的一個潮流。為了掌握未來開發 real-time system 的主流技術，本計畫將以 real-time Linux 作業系統為基礎，參考 Sun 的 real-time Java 標準草案，以及未來 Sun Microsystems 的 distributed real-time Java 所使用的 distributed thread 技術，來開發一個以 Java 為基礎的及時分散式計算平台。</td></tr>
<tr><td colspan="6"></td></tr>
</table>

註：本表格如不敷使用，請另紙繕附。

1 計劃目標

本計劃的目標是評估一個分散式的即時訓練系統的雛形（如下圖）。系統分別提供訓練者及受訓者的操作平台，每次訓練時，訓練者一位，受訓者可多達 8 位。整個訓練系統藉由模擬場景達到訓練的目的，受訓者操作使用介面，並判斷當時情況以攻擊場景中的飛行目標。在系統的每次模擬場景中，最多可同時處理約 100 個目標，每個目標每隔 1/4 秒更新自己的狀態，受訓者會在操作平台上接收到這些目標鎖更新的資訊。

整體而言，這是一個通訊量非常高的系統，並且要同時兼顧效能及網路的容忍度。因此，我們要完成底層通訊技術的評估，設計系統架構，蒐集資料，以完成本計劃的目標。



這份成果報告書將說明我們完成以上這些工作的成果，並將這些成果依序說明。

2 分散式物件環境

2.1 說明項目

2.1.1 基本原理

說明 Java/RMI、CORBA、DCOM 三種分散式物件環境的基本運作原理，包含 client 與 server 間的要求(request)及回應(response)，以 client 角度來看的遠端物件參考(remote reference)，介面描述語言 (IDL, Interface Definition Language)，以及要求與傳送時的包裝 (Marshalling)等特性。

2.1.2 提供與使用 service

這部分會介紹如何提供及使用 service。包含註冊 server，命名 (naming)，尋找特定 service，靜態呼叫與傳送，及動態呼叫與傳送等方法。

2.1.3 其他特性

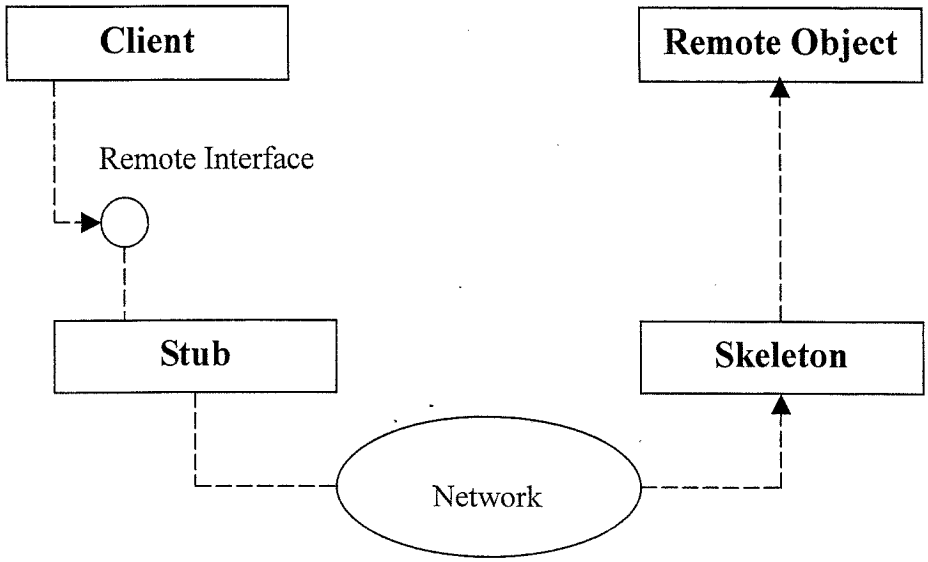
討論分散式物件環境的其他特性，如垃圾收集 (garbage collection)。

以下，就分別針對 Java/RMI 、 CORBA 、 DCOM 三者作架構上的分析。

2.2 Java/RMI

在 Java 環境中，目前有兩種方法可以處理分散式物件。(1)利用 Java RMI (Remote Method Invocation) 與 Java object Serialization 服務；(2) 使用 CORBA IIOP 協定。這裡只討論 Java/RMI 分散式環境。

2.2.1 要求與回應 (request and response)



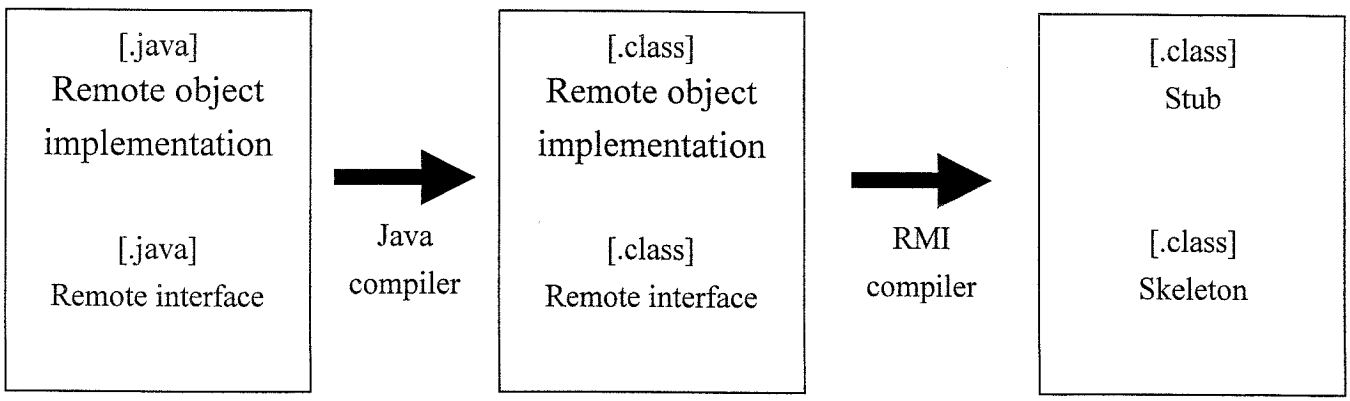
Client 發出 request 後的流程圖如上。Client 呼叫 Remote Interface，經由 Stub -> Network -> Skeleton 後傳送到 Remote Object。Remote object 傳回的 response 則依逆向順序送回。

2.2.2 遠端參考 (remote reference)

RMI 中的遠端參考，就是遠端物件的參考，不過它不能被直接存取，而是封裝在 stub 與 skeleton 中。在 client 的角度來看。

2.2.3 IDL 介面

Java/RMI 不像 CORBA 和 DCOM，需要特別弄一套 IDL 規格。rmic 會由 remote object 的 bytecode 編譯出 client stub 與 server skeleton。



2.2.4 Marshalling

Java/RMI 會自動處理 Marshalling 及 Unmarshalling 。而參數傳遞時，RMI

利用 java object serialization 的方法來取得物件拷貝，傳送到遠端後利用 unserialize 將物件還原。

2.2.5 Register a server

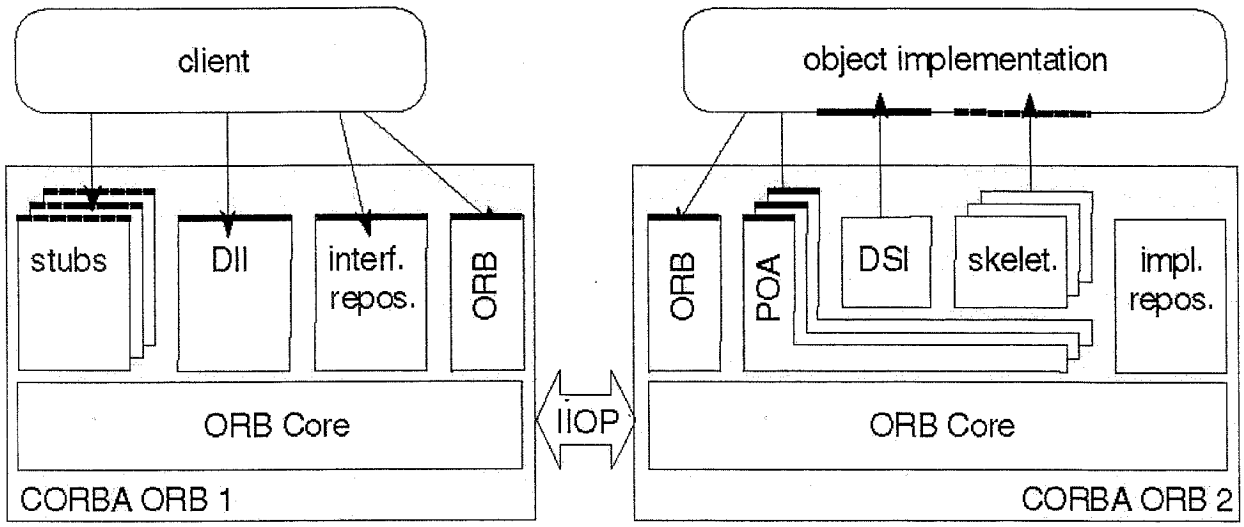
當 server 想要讓一物件可以遠端存取時，必須要向 RMI registry 註冊。當註冊完成後，RMI registry 會將所有送到此物件的要求全部轉過來讓物件處理。

2.3 CORBA

OMG (Object Management Group) 是一個由會員贊助而成立的非營利組織, 其目的在推廣物件導向 (Object-Oriented) 的觀念及使用並致力於加強軟體的可攜性 (portability), 再利用性 (reusability), 以及互通性 (interoperability)。該組織會員包括了廠商、學術單位及用戶。

CORBA(Common Object Request Broker Architecture) 爲 OMG 在一九九一年十二月提出之物件導向分散式工作環境規格, 一九九三年十二月、一九九四年 九月此規格多次被修訂, 目前最新之版本爲 一九九五年七月之 CORBA 2.0 版 , 互通性及 C++ 對應 (mapping) 之標準, 均於新版之標準中更明確地被製訂。 CORBA 規格對一個物件請求仲介者 (Object Request Broker) 之標準介面作了詳細的規定。根據 OMG 的定義, 一個物件請求仲介者爲一可提供分散式環境上各個物件透明化 (transparent) 的請求服務與回應接收功能的應用程式建構工具。

2.3.1 要求與回應 (request and response)



client 發出 request 的方法有三種：經由 OMG IDL stub(Client stubs)、Dynamic Invocation Interface 或特殊的 interface 來與 ORB 溝通。

當 client 有 request 的時候，ORB 首先要去找出適當的 object implementation，並讓這個 object implementation 能接受 request，同時 ORB 還要與其交換和這個 request 有關的資料。

Object Implementation 接收 request 經由 OMG IDL generated skeleton(IDL skeleton)或由 dynamic skeleton 來取得。當 Object Implementation 在處理 request 的時候，可能會呼叫 Object Adapter 或 ORB。

一個 server 可以包括一個或一個以上的 object 實作，甚至只提供特定 method 的實作。但是，通常在 multithread 的環境下，一個 server 會封裝許多 object 實作。

2.3.2 物件參考 (object reference)

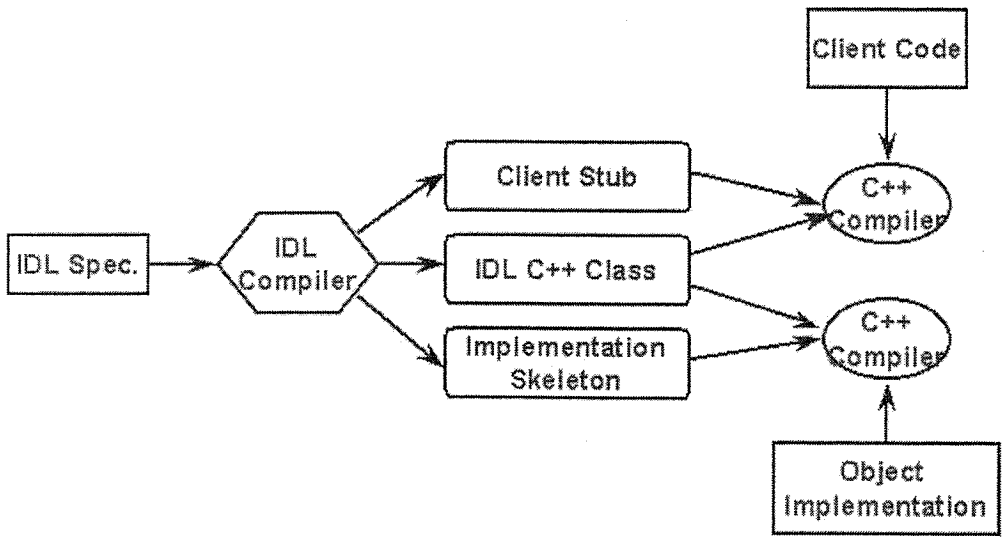
Client 當使用到其他的 object 時，便會產生一個 object reference。所有對 object 的處理均要透過 object reference 來處理，像是 invoke 或傳遞參數等。取得

object reference 通長都在於 invoke 這個 object 的時候，通常都會回傳一個參數作爲將來 reference 物件時使用。

對 Client 來說，使用遠方的 object 就如同使用一般的 object 一樣，不用特別去指定使用的 object 位於何處，只需要與 ORB 來溝通就可以完整的去控制這些處理。

2.3.3 IDL 介面

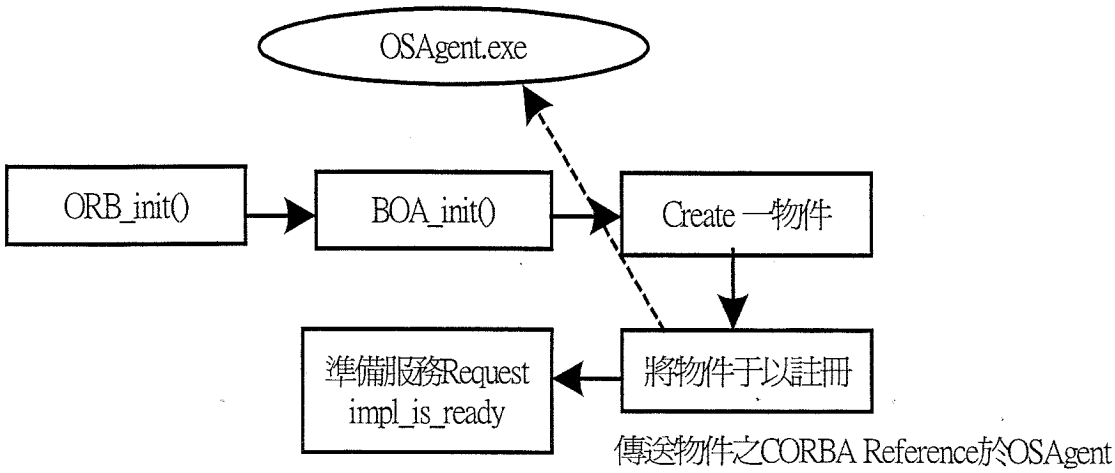
CORBA 提供了一個界面定義語言 (IDL)來描述一個物件的界面。透過一致的界面定義，使物件的界面 (Interface) 與物件的實體 (Implementation) 可以獨立開來。程式開發者在選定使用 CORBA 的語言後，可以經由適當的 IDL 編譯器將 IDL 轉換成該語言，如 C、C++、Java、Ada、SmallTalk 等。



2.3.4 Marshalling:

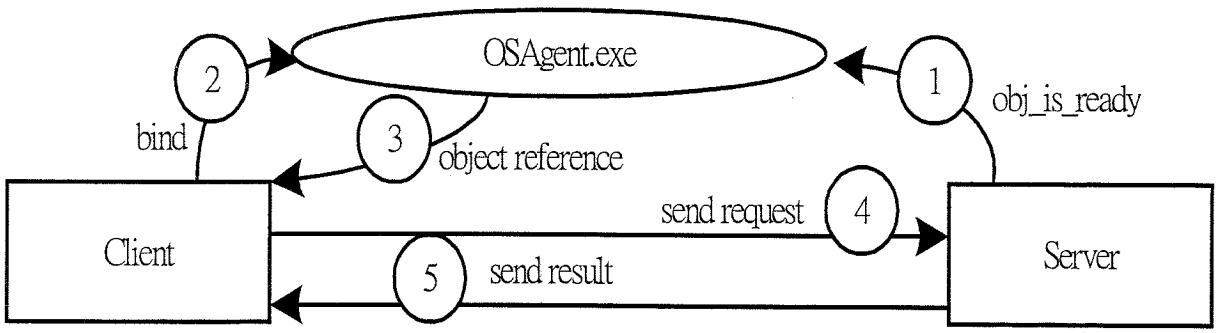
Client request 與 server response 皆透過一個固定的資料格式，經由 IIOP 傳送給對方。Client 方面，marshalling 動作在 IDL stub 或 DII (Dynamic Invocation Interface) 時完成。在 server 方面，unmarshalling 於 IDL skeletons 或 DSI (Dynamic Skeleton Interface) 完成。

2.3.5 Registering a Server



爲了提供遠端存取服務，server 必須先在 broker 註冊。首先 server 建立一物件，將物件的 CORBA reference 傳給 broker 註冊，之後呼叫 Requestimpl_is_ready ，通知此物件已經可以接受 request。

下圖爲註冊後，整個 client 與 server 互動的流程圖。



2.3.6 Naming

正式上，CORBA 提供了 Naming Services，提供了 name space 和物件參考與命名之間的對應。但以實務上來說，大多數的 CORBA 實作廠商提供了各自的方案，解決直接註冊 services 與 client "bind" services 的操作。

2.3.7 Finding a service

在 client 的角度來看，有兩種方法可以取得物件參考：1. 向 naming service 詢問，解析某特定名字的物件參考，2. 詢問 trading service 以取得 services 列表(Object reference)，並利用關鍵字檢索。

有些 CORBA 實作廠商另外提供了 location service，混和了 naming service 與 trading service 的特性，client 可以提供部分的資訊(如不完整的名稱，或 service 的 interface)，讓 location service 找出適當的 service。不過 location service 並沒有包含 naming service 及 trading service 全部功能。

2.3.8 Static invocation and delivery

透過 Stub 及 Skeleton 的分派(Dispatch)通常叫做 *Static Invocation*。

Stub 又稱代理者(proxy)。Stub 直接在 client 端的 ORB 做包裝(marshal)要求的工作，那就是說 stub 幫助要求從程式語言表示法轉成適合在網路上轉輸的型式。

Skeleton 做 unmarshal 要求，從轉輸型式轉成程式語言型式且分派它到物件。一但物件完成要求，若有回應則順著原來的路線送回。

2.3.9 Dynamic invocation and delivery

除了藉由 static invocation，CORBA 還支援兩種動態 invocation 介面：*Dynamic Invocation Interface* (DII)—支援動態的 client 要求呼叫，以及 *Dynamic Skeleton Interface* (DSI)—支援動態分派物件。

使用 DII，client 應用程式可以呼叫沒有編譯時期的物件介面庫資訊的物件，DII 也可用來做程式間的互動。它透過由 CORBA::Object 介面所提供 create_request operation 建立 Request pesudo-objects，藉由在目標物件的物件參考上呼叫這個 operation，應用程式可以動態的在這物件上建立要求。

DII 類比到 server 端就是 DSI，只是 DII 允許 client 不用存取靜態 stub 便可呼叫要求，而 DSI 允許 server 不用將呼叫靜態地編譯進程式裏便可被寫成。

2.4 DCOM

微軟的分散式系統環境是建立在它的 DCOM (Distributed Component Object Model) 上。COM (Component Object Model) 是在 windows 環境下 component-base 的發展環境，而 DCOM 是它的延伸。COM 支援同一台機器中，同一行程(process)或跨行程的物件交流，DCOM 提供類似功能，但能夠跨出同一台機器的限制，如下簡圖所示。

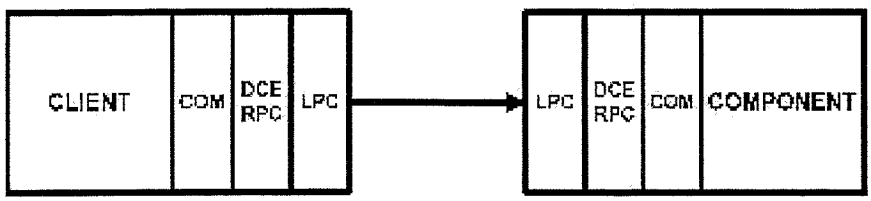


Fig: COM component (1 machine)

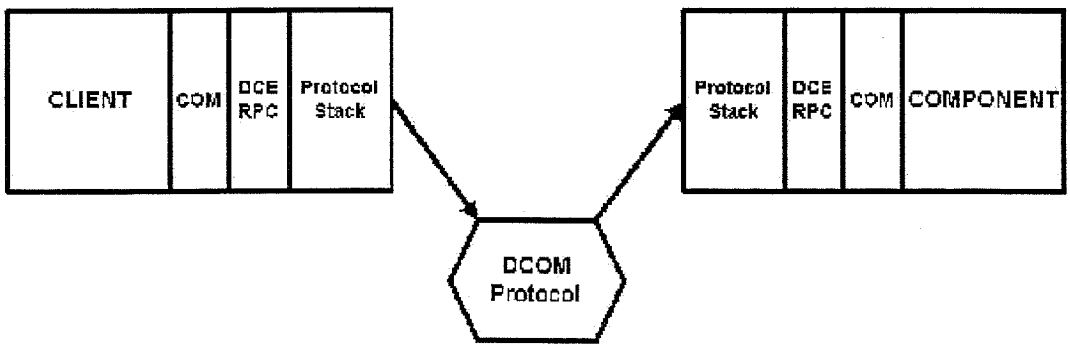


Fig: DCOM Component (across machines)

2.4.1 要求與回應

在 COM 中，client 對 server 物件的特定介面發出要求，server 可以與 client 處在同一個行程 (inprocess server)，或是同一機器不同行程 (local server)，DCOM 支援 client 與 server 不在同一機器上 (remote server)。COM/DCOM 的遠端呼叫都是同步呼叫。COM 的 request 及 response 傳遞採用的是 LRPC (Lightweight Remote Procedure Call)，DCOM 採用的是架構在 OSF/DCE 上的 ORPCs (Object-Orient RPCs)。

2.4.2 遠端參考 (Remote reference)

COM 中，遠端參考不像 CORBA 是一個 object reference，而是 server

objects' interface 。server 端每一組介面都有自己的遠端參考，當進行遠端呼叫時，server 端傳回一組 tuple，包含了被呼叫介面的資訊。

2.4.3 IDL 介面

COM/DCOM 使用的 IDL 語言稱爲 MIDL (Microsoft IDL)，介面規格經由 MIDL 編譯後產生 server stubs 與 client proxies。MIDL 亦可產生 type library。

2.4.4 Marshalling

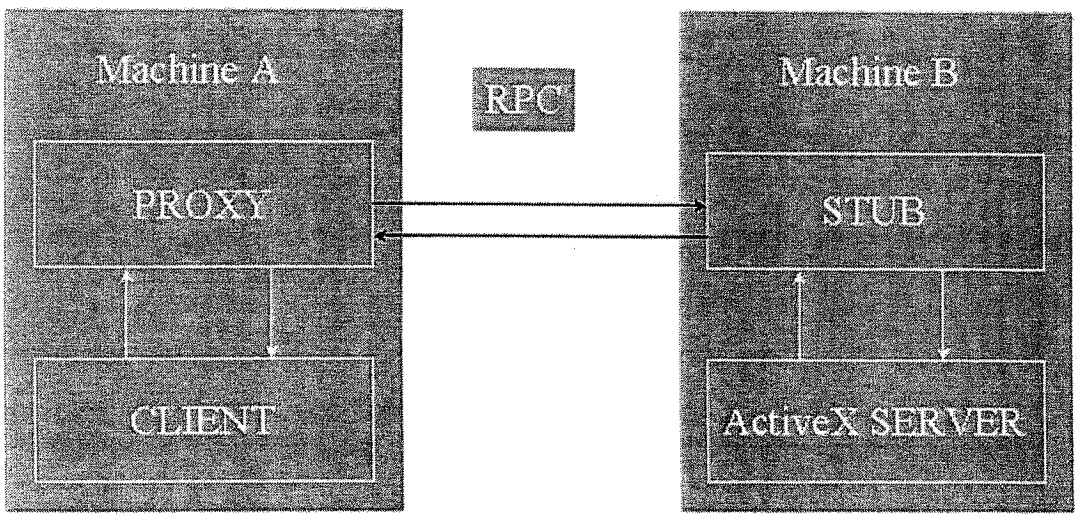
COM 使用專有的格式來封裝 request 與 response，當遠端呼叫要傳遞參數時，會封裝成網路資料表示(*network data representation*)的資料。當 server 物件介面的 reference 傳遞回來時，會封裝成一組 tuple。

2.4.5 Naming

COM/DCOM 有兩種不同的全域 naming spaces，分別爲了解決特定需求。第一個 naming space 包含 GUIDs (Global Unique identifies) ，用來指定 COM object 的 class (CLSIDs) 與 COM object 的 interface (IIDs)。第二個 naming space 包含了 *names of monikers* ，用來解決永續物件 (persistent objects) 的問題。

2.4.6 Static invocation and delivery

當 client 將 request 寫死在程式碼中，client 很容易的利用 proxy object 來達成遠端呼叫。同樣的，server 利用 stub 將結果傳回 client。



2.4.7 Dynamic invocation and delivery

DCOM 與 CORBA 一樣都提供動態 invocation 以及 metadata，這是 OLE automation 的基礎，我們現在簡稱 automation，經由 IDispatch Interface 它允許 client 動態的呼叫 method。

IDispatch 提供了跟 invoke method 一樣的機制，可以詢問某個 object 到底有哪些 methods 與 properties 可使用。

2.5 討論

因爲本計畫的目的，在於評估以 Java 建構大規模、高資料量 (large scale、high data traffic) 的分散式計算應用程式時，下層該使用怎樣的基礎架構，所以，在本節裡，我們將會討論一下 invocation-based middleware 是否適合用來架構這類型的應用程式。

在本計畫裡，我們所要評估的系統，是一個分散式的即時訓練系統。我們希望這個系統，同時能夠處理 100 個目標，每個目標每隔 1/4 秒，會對外更新一次自己的狀態，諸如目前位置、速度、加速度...等。所以，這是一個 traffic 量很大的一個分散式系統。基於以下的兩點理由，我們認爲 invocation-based middleware 並不適合用來處理這類的問題:

第一點: 我們認爲，分解像分散式即時訓練系統的最好方式，是把系統內元件的互動，拆解成元件之間流動的 event flow。要用 invocation-based middleware 實作出這樣的 event flow，invocation-based middleware 下層的 protocol，會產生許多額外的 overhead，包括多餘的 acknowledgement 及 reply message...等。相較之下，使用 message-oriented middleware 來實作元件間的 event flow，一方面

比較直覺，另一方面，也省去了用不到的 reply message。此外，message-oriented middleware 還可以使用類似 negtive message acknowledgement 的技術，更進一步地減少 acknowledgement 的訊息量。

第二點:模擬系統內的目標，在對外更新自己的狀態時，通常都是一對多的通訊方式。但除了少數學術界的 prototype 之外，目前廣爲使用的 invocation-based middleware，所採用的幾乎都是點對點 (uni-cast) 的通訊方式。也就是說，若使用 invocation-based middleware，則目標在對外更新自己的狀態時，會需要做多次的 remote invocation。相較之下，message-oriented middleware 通常會利用底層網路的 multicast 或 broadcast 機制，來傳遞訊息。因此，message-oriented middleware 會比 invoaction-based middleware，更適合用來處理一對多的通訊情況。

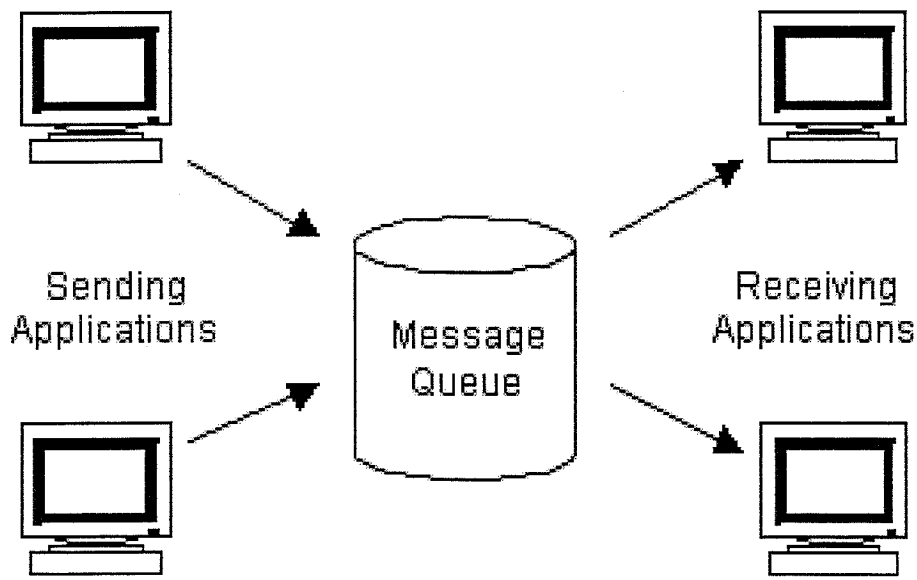
2.6 結論

我們首先分析了幾種 invocation-based middleware: 包括 Java/RMI、CORBA、以及 DCOM 的架構，並針對這三種分散式系統的基本原理、提供及使用 service、以及其他特性，做了一番討論。接著，我們也討論了 invocation-based middleware，適不適合用來建構大規模、高資料量 (large scale、high data traffic) 的分散式計算應用程式。我們認爲，相較於 message-oriented middleware，invocation-based middleware 因爲通訊的 overhead 較大，而且不適合用來處理一對多的通訊情況，所以，並不適合用來架構前述的分散式計算應用程式。接下來會進一步地評估，哪一種 message-oriented middleware 才適合用來建構大規模、高資料量的分散式計算應用程式。

3 傳統通訊軟體

3.1 Microsoft Message Queue (MSMQ)

MSMQ 是以下圖的基本觀念來製作，這個圖其實也跟大多數 Queue-Based Middleware 的觀念相同。

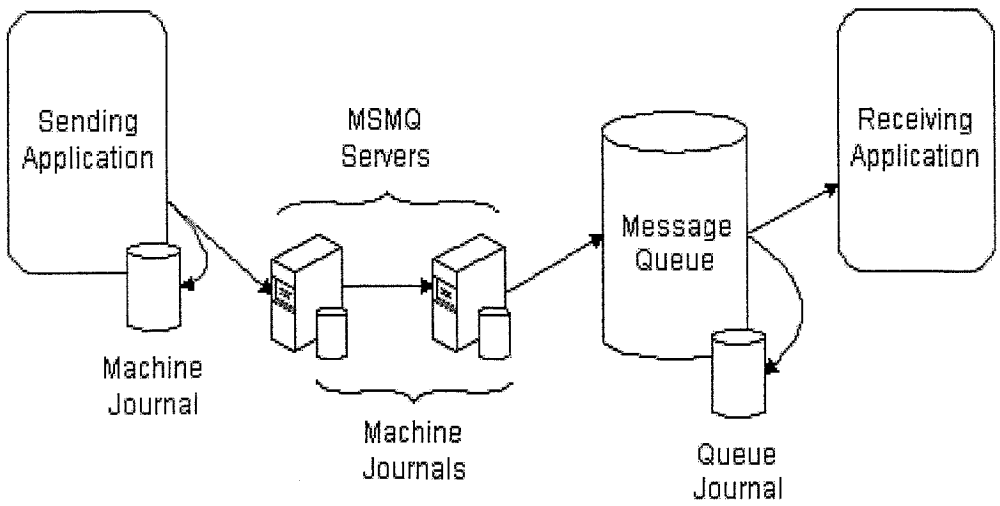


在 MSMQ 中分成兩大部分，再細分爲六種 Queue 來實作整個 Queue 的系統，這些 Queue 的分類如下：

- Application Queues :
  - Message Queues
  - Administration Queues
  - Response Queues
  - Report Queues
- System Queues :

- Journal Queues
- Dead-Letter Queues

Application Queues 的功能大致和名稱一樣，在系統中所扮演的角色較容易瞭解。System Queues 的運作部分，我們以下圖說明 Journal Queues 在系統中所佔的角色。

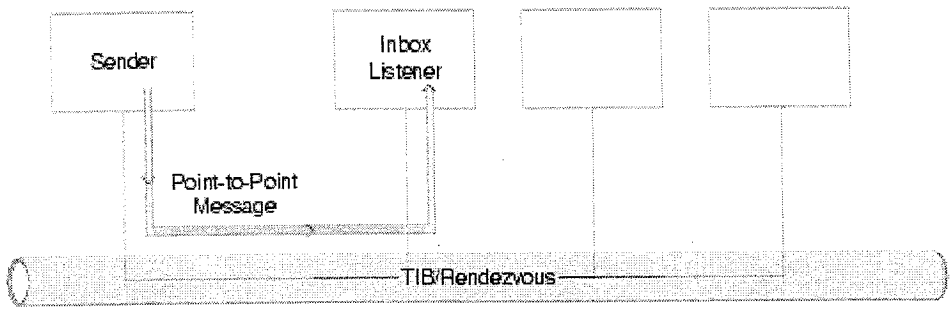


以上說明就是由 Microsoft 所提出的通訊軟體 MSMQ 的大致介紹

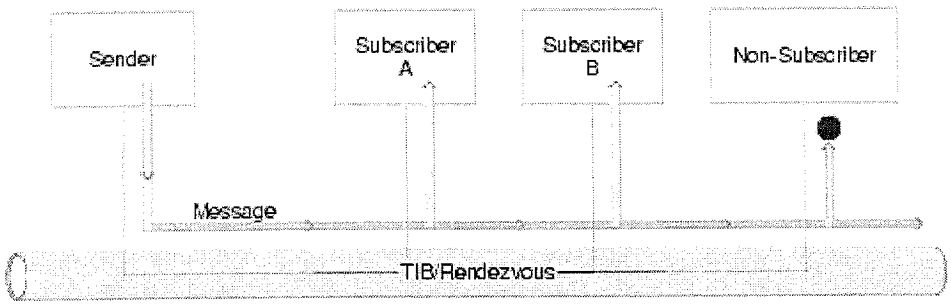
3.2 TIBCO Rendezvous (RV)

這是由 TIBCO 公司所提供的 middleware，內部運作的 message 主要分爲 point-to-point 以及 reliable multicast message 兩種，運作方式分別由以下兩張圖來說明：

- Point-to-point message



■ Reliable multicast message



RV 在使用上，主要的 middleware 運作皆由 RV 的程式負責，使用者根據 RV 所提供的 API 跟 RV 互動，便可讓程式藉由此 middleware 達到分散式運作的目的。

3.3 討論

上述所見到的兩種 middleware 皆爲市面上可以購買到的產品，但是我們認爲並不適合在本系統中使用，有幾點原因，首先，這兩個系統所提供的 API 相當受限制，MSMQ 必須運作在 Microsoft 爲基礎的環境中，RV 雖然提供多種的 API 模式，但是在使用時必須根據不同的需要來呼叫函式，並無提供單一的標準。第二，兩者皆由公司所開發保護的產品，因此在效能上無從比較，這裡所說的比較是指在相同的 API 設計之下，只有一套實作，因此無法比較效能。第三，

程式開發完成之後，要使用這些程式時，在不同的系統上，這些程式需要重新編譯。

基於以上幾點原因，所以我們認爲這些現有的 middleware 並不完全適合本系統所使用，所以我們計劃採用 Java Message Service (JMS) 來作爲底層的通訊軟體，主要的考量其實跟上述的幾點原因雷同，首先，因爲 Java 的執行環境較不受限制，每個平台上都可執行相同的程式，而且所使用的 API 函式都不需要更改。第二，JMS API 是公開的，所有有許多的收費或免費的實作可以參考使用，不僅在效能上可以互相比較，也可以在價格上做採購的考量。第參，以 JMS API 爲底層的程式開發完成後，這個程式便可以在不同的系統上執行，並不需要將程式重新編譯。所以，我們認爲以 JMS 作爲本計劃系統架構的底層通訊軟體是相當符合需求的。

在選擇 JMS 作爲系統的通訊機制後，下一步便是開始尋找要用的 JMS Implementation。我們尋找許多的 JMS Implementation 作候選，因爲 JMS API 雖然由 Sun Microsystems 所訂定，但是實際上有許多符合 JMS API 規格的實作出現，不論是 open-source 或是商業軟體，我們都找來作爲參考。

我們在網路上廣爲搜尋，查看許多團體對於 JMS API 的實作情形，當其宣佈提供符合 JMS 規格的功能後，便設法取得該功能的執行檔，我們將找到的 JMS Implementation 蒐集，並根據不同的 Implementation 情況取得執行權。其中，有些 Implementation 是公開的，甚至連原始碼都有，這個部分的 Implementation 可以直接使用，而且對之後的分析而言相當方便。另外，有些 Implementation 屬於商業軟體，因爲我們尚未購買該軟體的正式使用權，要先取得試用版的 license 才能試用。針對不同的情形，我們皆設法取得 JMS Implementation 的執行檔和執行權。

找到的 JMS Implementation 包括有 Fiorano 的 FioranoMQ〔12〕、ObjectWeb 的 JORAM〔10〕、OpenJMS〔11〕、Softwired 的 iBus//MessageBus 及 iBus//MessageServer、Sonic software 的 SonicMQ〔13〕、Talarian 的 SmartSocket for JMS〔14〕（依字母順序排序）等等許多的實作，這其中有的是免費的 open-source，也有的是屬於公司的產品。

取得這些 JMS Implementation 的程式碼或網路上的試用版 license 後，接下來我們分析這些 Implementation 中所採用的機制。因爲 JMS API 是制定一套通訊模式，每個要實作 JMS API 的人在遵循這套模式的大原則下，可以透過不同的機制來達成 JMS 的功能。所以，我們一開始便是針對底層機制作分析，也就是分析該 JMS Implementation 所採用的機制，以選擇出效能較好、較適合本系統使用的機制，之後再從所有的 JMS Implementation 中篩選出符合的來進行效能測試，經過測試後，便依據測試結果選出本計劃用作通訊底層的 JMS Implementation。

我們將分析焦點放在 publish/subscribe 的機制上，因爲本計劃的系統並不適用 peer-to-peer 的通訊架構：訓練者和受訓者都同樣要接收來自 target 所傳出的資訊，如果使用 peer-to-peer 時，會造成將同樣的資料一直重複地送出給訓練者和受訓者的情形，尤其 target 的更新資訊在系統中佔了相當重的比例，這種重複送出的情況會更加嚴重，因此本系統並不選用 JMS 中 peer-to-peer 的溝通方式，而主要使用 publish/subscribe 來通訊，所以分析的焦點也放在 publish/subscribe 機制上。將 publish/subscribe 機制選擇好的話，便可以避免將資訊重複送出的情形。

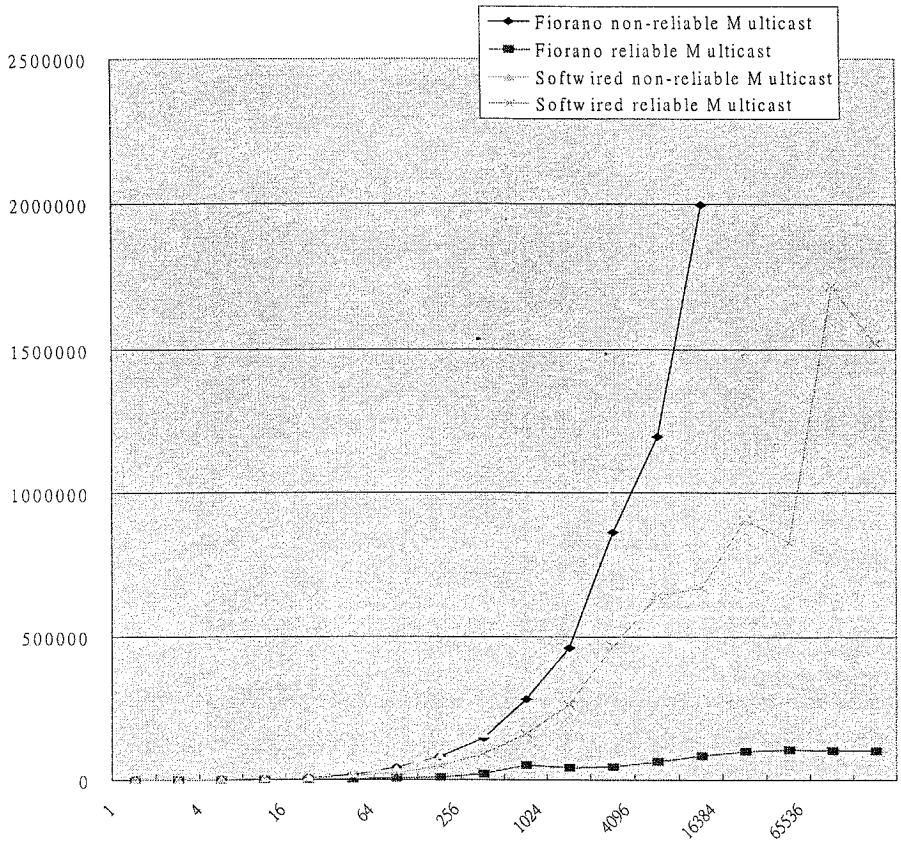
經過分析後，網路上的 JMS Implementation 主要使用的 publish/subscribe

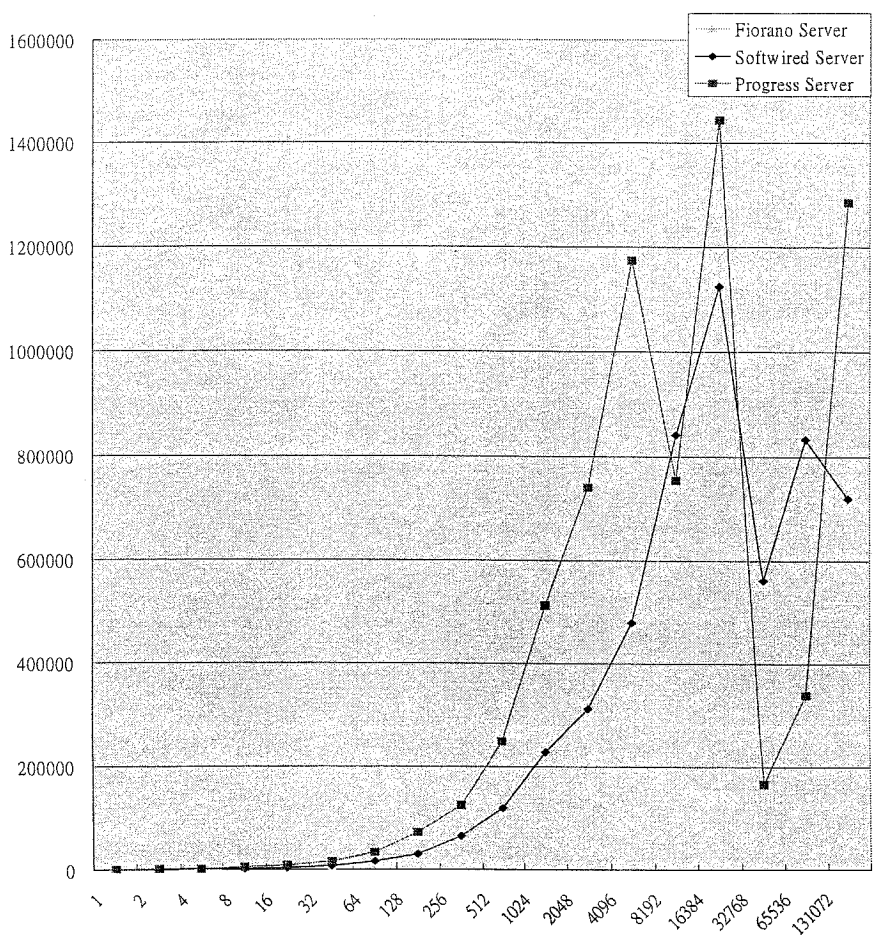
機制主要爲 server-based 和 IP-multicast based 兩種：

- server-based JMS Implementation 主要的部分是一個 JMS 的 message server，所有管理便都由該 server 負責，包括對 topic 收送的註冊和取消、儲存並發送 message、message 重送機制的運作等等，因爲所有的管理機制都由 server 負責，所以這一類的 JMS Implementataion 大多附有一個 administrator tool 來方便使用者管理這個介面。
- IP multicast-based JMS Implementation 並不存在於一個中央的 server，在非主從式架構下，每個 JMS Implementation 啓動之後都獨自成爲個體，message 透過 IP multicast protocol 傳送，topic 收送的註冊和取消藉由 multicast group 來管理，message 重送的機制則由 JMS Implementation 增加在 IP multicast 上來完成。

根據上面對於兩種機制的描述，我們可以發現較適合本系統的其實是 IP multicast-based JMS Implementation，原因如下：

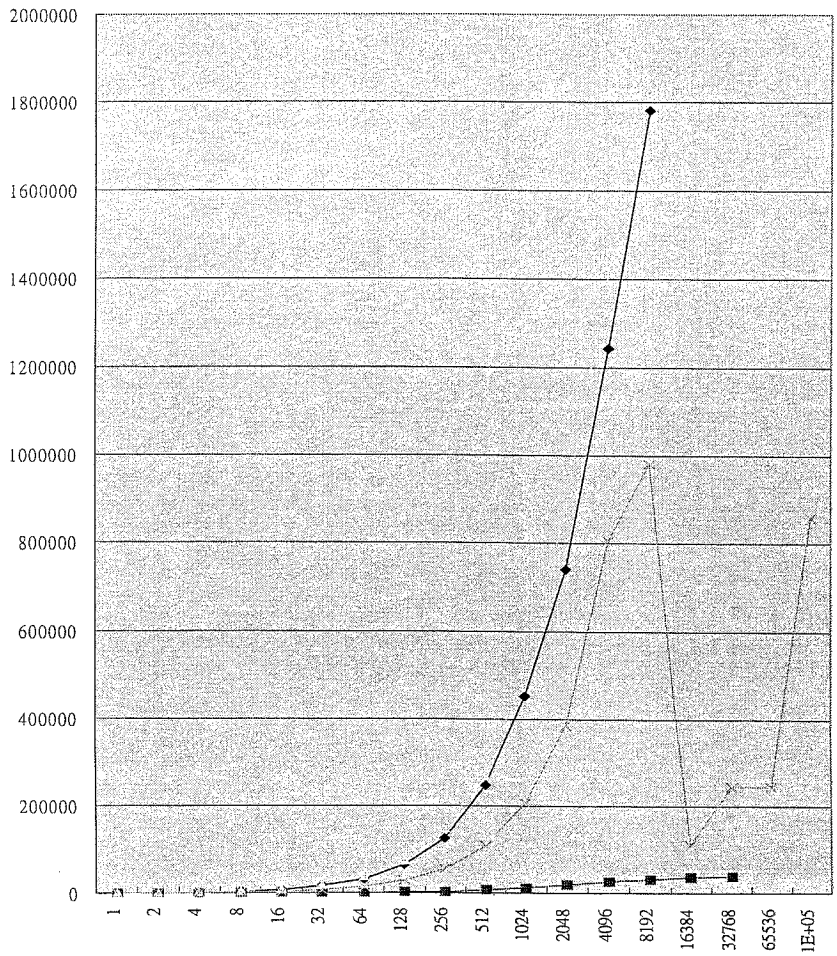
- IP multicast-based 相較於 server-based 會比較節省資源。因爲不需要一個 server 專門負責儲存資料，我們便節省下 server 所用的系統資源，也省下管理方面的資源。
- IP multicast-based 會比較有效率，因爲 IP-multicast 在傳送訊息時，本身就只送出一個封包，不論 subscriber 有多少，所以對同一個 topic 的 subscriber 超出一個以上時，server-based 就要送出比 IP multicast-based 更多的網路封包。下面我們列出 IP multicast-based 和 server-based 兩種 Implementation 的 throughput 比較圖，上圖爲 IP multicast-based，下圖爲 server-based。


Fiorano non-reliable Multicast
Fiorano reliable Multicast
Softwired non-reliable Multicast
Softwired reliable Multicast
2500000
2000000
1500000
1000000
500000
0
1
4
16
64
256
1024
4096
16384
65536


Fiorano Server
Softwired Server
Progress Server
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072

- 本計劃的系統，對於大部分過時的封包，也就是 target 的更新資訊，並不需要額外的機制來處理，更不需要保留起來。這一點和 IP multicast 原本的運作方式非常接近，和 server-based 中的 queue 機制大不相同。在本系統中 server-based JMS Implementation 的 queue 使用的機會並不多，反而因爲要先經過 queue 會造成額外的負擔，這還不包括要產生和保存 queue 本身所使用的系統資源。

以上幾點，所以我們選擇 IP multicast-based JMS Implementation 最爲本計劃的通訊底層。根據這個原則篩選過後，再加上針對這些 JMS Implementation 所做的效能測試（如下圖）。



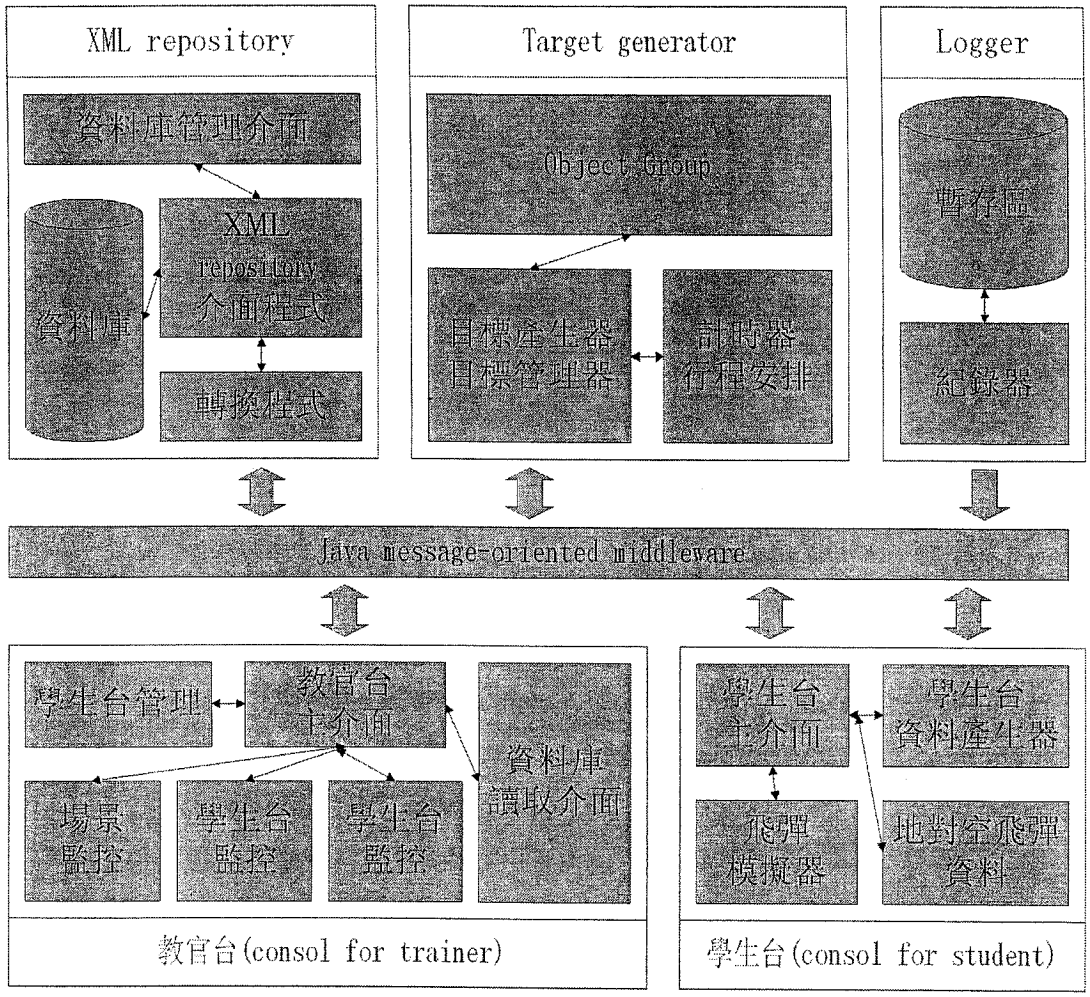
最後，我們選擇採用 Softwired 的 iBus//MessageBus 作爲本計劃系統底層的通訊機制。

Softwired 的 iBus//MessageBus 是根據 JMS API specification 所製作的軟體，底層採用 IP multicast protocol。因此，用在本計劃的系統上，可以減少網路上所傳送的封包數量，以讓系統的網路設備可以容忍更多的封包流量，這對本計劃要求的高 traffic 系統是相當有幫助。

4 架構介紹

在 Java message-oriented middleware 設計中，因爲是以 message 作通訊，所以要先將獨立的 message source 分離開來，讓每個 message source 發出的 message 有系統性和獨立性，這樣 message source 才不會雜亂無章。還有一點值得注意的是，因爲採用 publish/subscribe model，所以只要注意 message 的流向即可，而不需考慮傳送的對象。上述是設計架構時的參考方向。

在本系統中可以被分爲以下的五個部份，分別是教官台、學生台、XML repository、logger，以及 target generator。這些部分都可以視爲是功能獨立的個體，因此，在設計系統時我們將這些部分分別用獨立的元件代表。在這些元件的底層以 message-oriented middleware 作爲通訊媒介，也就是負責幫元件收送 message，協助他們完成互動。以下是整個架構的概觀。



在說明個別功能之前，我們先約略描述一下整個架構的運作情形。首先，在使用模擬訓練之前，系統管理人員用資料庫管理介面將模擬場景及相關的資料放入資料庫中，讓教官們可以用這些資料來作訓練。接著，教官在執行模擬訓練時，從資料庫的眾場景中選取適當的場景，將場景內容讀取到負責模擬訓練的目標產生器上，然後教官將場景執行，正式開始模擬訓練。在場景執行的過程中，教官可以隨時指派目標給學生。在訓練過程中，教官可觀看到場景中的物件的情形；而學生則根據被指派的任務在模擬場景中操作和學習。訓練過程中紀錄器會記錄各物件所發出的指令，並將這些指令儲存到資料庫中，作爲本次訓練的紀錄。訓練完成後，教官可根據學生的訓練過程表現評分，或者事後觀看紀錄來講解。上述約略說明訓練時的運作情形。

在各元件功能描述的部分，我們將元件依功能分成幾大類。所以，介紹主要元件時，可能包括相關元件一起介紹。

4.1 通訊媒介（communication middleware）

選擇採用 Java message-oriented middleware，並支援 publish/subscribe model。

Publish/subscribe model 的運作模式類似 multicast，要發送 message 的元件向 middleware 註冊一個頻道來使用，然後將想送出的 message 放入該頻道中，這稱爲 publish；而對該 message 有興趣的元件跟 middleware 訂閱所屬的頻道，稱爲 subscribe，訂閱後就可以收到相關的 message 了。Message-orienred middleware 所提供的 message delivery 的方式，可能是 reliable 的，也可能是 unreliable 的。Unreliable 的 message delivery 方式通常 overhead 會比較低，對重視 performance 的應用程式來說比較有利，但應用程式本身則必須有能力處理由 site failure 所帶來的問題。

Communication middleware 主要協助系統中的元件收送 message，讓所有元件透過 message 傳送來互動。

4.2 教官台（consol for trainer)

給訓練者所使用的平台，希望給訓練者一個方便執行模擬訓練的平台；這也是作模擬訓練時的主控台。

其功能包括挑選想要作模擬訓練的場景，管理參與訓練的學生台，操作模擬場景的執行，指派任務給學生，以及在模擬過程監控場景中發生的事件。在訓練

過程中，教官台不僅可以監控場景中的物件，也可以監控學生台；也就是從教官台可得知學生台的動態。

4.3 學生台（consol for student）

給受訓者所使用的平台，希望幫助受訓者完成整套的模擬訓練過程。啓動後，學生台會先向教官台登記參加模擬訓練。在訓練過程中，受訓者收到被指派的目標後，便操作學生台介面去完成任務。另外，因爲學生台會負責操縱一個或多個地對空飛彈的發射系統，所以，在模擬的過程中，如果有地對空飛彈發射，它也會產生模擬飛彈的物件，負責追蹤及擊發目標。

每次模擬訓練時，可有不同數量的學生台參與。

4.4 資料庫（XML repository）

整個系統儲存資料的部分。儲存的內容有場景資料，據點資料，武器資料，和儲存訓練紀錄。武器資料描述該武器的各種屬性，如極速，航程….等等。據點資料描述一個武裝單位的資料，如該據點所裝載的武器，防禦能力….等等。場景資料描述一場模擬訓練的場景，包括整個場景的武力配置，各物件所採取的行動….等等，就像是一本劇本。

我們希望能用 XML 來描述儲存在資料庫中的文件規格和內容，以增加擴充性和閱讀性[9]。不過，我們目前尚未決定，資料在 middleware 上傳送的 network representation 的格式。因此，我們提供了一個可以抽換的轉換程式，以便將 XML 資料轉換成傳送 message 時所使用的 network representation。

另外，我們尚未決定 XML repository 的實作方式。如果，本系統的資料用檔案系統就可以管理，就使用作業系統的檔案系統；不然，就需要裝設 Oracle 或 SQL server 這類資料庫管理系統來管理本系統的資料。XML reposity 界面程式也是一個可抽換的元件，則負責將檔案系統或 relational database 的 schema，轉換成 XML 的格式，並實作所需的 XML 檔案的 naming、load，以及 store…等機制。

4.5 目標產生器（Target Generator）

負責產生場景中的目標物件，以及執行每個目標物件所發生的事件。目標產生器會根據所選擇的場景去讀取資料庫中的資料，用這些資料產生場景及其中的物件，並且根據場景的描述，協助每個目標物件產生事件。在場景執行時，目標產生器還要計時，每隔一段固定的時間去跟物件要求動態資料，彙整後傳送出去，讓系統其他的元件，知道現在場景內，它所關心的目標物件的狀態。在發送資料前，要根據當時網路情況，物件數量，或其他外在因素，物件產生器需要動態安排資料發送的時間，以讓系統的效能最佳化。

另外，因爲目標產生器內執行場景和物件，所以也負責維護或摧毀這些物件，換句話說，就是要操作產生器麾下的物件時，是透過產生器來操作，而不是直接操作該物件。這樣可以減少效能和安全上的顧慮。

4.6 紀錄器（Logger）

紀錄模擬時所發生的事件。紀錄器和目標產生器及兩種控制台接收相同的資料，不同的地方在於，紀錄器將收到的資料蒐集起來，彙整後紀錄到資料庫中。

根據紀錄的內容，可以讓教官作爲事後評分或分析的文件。

這部分還希望可以依場景資料和紀錄內容將整個模擬的過程重現，把紀錄從文字提升到圖形層次。

4.7 資料庫管理介面

管理資料庫的程式。許多資料儲存在資料庫中，所以這項元件的功能主要是用來管理或觀看所儲存的資料，希望能方便處理系統內的資料。根據所選的資料庫系統的不同，這部分會配合設計。

5 內部機制設計

針對本系統的運作需要，我們設計了一些運作機制來有效率的達成系統功能。這些機制可以解決一些系統間互動的問題，另一方面，這些機制也是詳細的系統設計。後面就將介紹這些機制、運作的方式及使用的地方:

5.1 Target 動作命令的運作方式

在場景中有大量的動作設定是針對 target 所描述的，相對的，在系統模擬場景時，便需要處理大量的動作設定，如果將每個 target 的指令都放在 target generator 中的話，會是一筆相當龐大的資料量，在資料處理時會耗費大量的時間和資源，而且當要指定動作命令給 target 時效率也會比較差；就好比說每個動作指令是一本書的話，每個 target 的動作指令集就像一堆書，如果將所有的

動作指令集中在 target generator 時，就像把所有的書都放在同一個書櫃，不論是要尋找、修改、減少書籍，都是很耗費精神的。

所以，設計本系統採用另一種方式，將所有的動作指令在 target 物件初始化時便指派給每個 target，在模擬過程中 target 便根據資料中的命令一個接一個的執行。就像飛行員在出任務時，根據要完成的任務一項一項的去完成，而不需要一直和基地要求指令。換句話說，本系統並不將所有的書籍放在 target generator 這個大書櫃中，而讓每一個 target 擁有一個自己的小書櫃，target generator 只要定時告訴每一個 target 要執行動作指令即可。

Target 根據擁有的動作指令，依序地執行，直到將所有的指令執行完，然後這個 target 物件便會結束；另一種結束的情形是，如果 target generator 收到摧毀通知時，此時便由 target generator 根據該通知的內容，通知虧毀 target。在 target 摧毀的同時，存在於 target 中的動作命令也就一起被摧毀。在這個摧毀 target 這方面，我們可以發現將動作命令放在每個 target 中的另一個好處，也就是 target generator 不需要從一大堆動作指令中刪除屬於被摧毀的 target 的部分，也不需要因爲要刪除很麻煩，而將已經確定不用的動作指令留在系統中，這樣就減輕了系統的負擔。

5.2 地對空飛彈運作方式

模擬飛彈的情形和目標不同，因爲武器的下一個動作和所設定的目標息息相關，所以運作模式是不同的。

首先，飛彈在發射之前要先指定一個 target 物件，讓該飛彈作爲攻擊目標，

然後才將飛彈發射出去，發射之後，飛彈就要不斷的追蹤該目標所發出的資料，也就是目標每隔 1/4 秒所更新的資料，收到這些資料再經過武器內部的計算後，來決定武器的下一步，一直到該目標摧毀或命中爲止。

根據前面所描述的，我們將武器設計成每次決定下一個動作時，要一直接收從網路上傳來、每 1/4 秒更新一次的目標資訊，然後根據手上的已經接收到的資料和巡標模式的設定來計算下一個動作。重複不斷地接收目標的資訊然後再計算動作，雖然會造成系統的負擔，但是這樣可以讓 target 物件所發出的狀態資訊被多次利用，透過連續不斷的計算，還可以確保武器更準確的追蹤目標的位置。

飛彈完成攻擊的指令後，學生台便會將該飛彈物件摧毀。

5.3 學生頻道機制

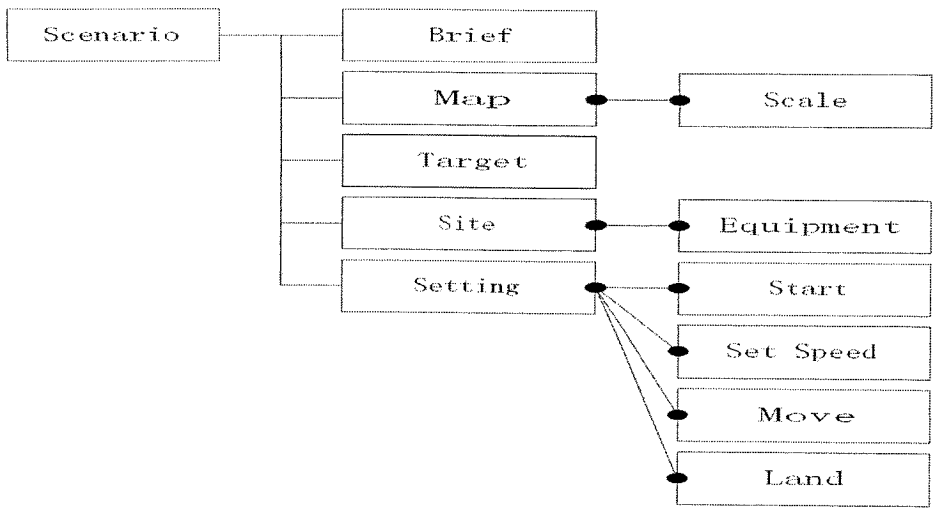
之前提到過，在模擬訓練進行時，學生台之間無法看到彼此的情況，也就是每個學生只知道自己被指派的任務內容，而且只會接收到指定任務內容的資訊，因此無法得知其他受訓者的情況，只有教官才擁有全部的資訊。因此我們需要將學生間的資訊分開，讓每個學生收到屬於自己的部分，彼此不會互相干擾，但是教官要知道每個學生的情形，所以本系統設計了一套機制來達到這種功能。

當學生向教官申請註冊以參加訓練時，教官台會自動的根據准許學生註冊的先後，分別給予每個學生一個頻道代號，同時教官並將該頻道代號通知 target generator 和學生，學生收到這個頻道代號後，便將自己的資訊接收切和到該頻道上。這樣知道這個頻道代號的就有 target generator、該學生、教官三者，其他

受訓者並不知道不屬於自己的頻道代號。在教官台要將 target 指派給該學生時，target generator 便將該 target 的資料輸出轉換到該頻道，學生台便可以購過這個頻道接收到屬於自己的 target 的資訊，透過這樣的設計，每個學生便可以只接收到自己被指派目標的資訊。

5.4 場景描述

爲了將場景清楚的描述出來，我們設計一套描述場景的文件架構。如下圖。



這套文件架構最上層就是一個場景，場景之下分成五個子部分，包括了說明、地圖、目標、基地、目標的動作等等。以下分別說明五個部分的設定內容和功能：

- 說明：儲存場景的說明文件，可以說明這個場景的特色，也可以說明這個場景的功能，其內容可能是幾句話，也可能是一小篇文章。每次模擬訓練執行時，教官都要選擇所要執行的場景，教官除了結由記憶和觀看場景的內容之外，便可藉由這段簡短的說明文件來判斷。
- 地圖：儲存場景所使用的地圖，以及這個地圖所使用的比例尺。每個場

景可以設定一張地圖作爲模擬時的背景，訓練時所有的物件都會顯示在這個地圖上。地圖的比例尺是在模擬時的計算依據，因爲目標及飛彈都在地圖上顯示，所以設定正確的地圖比例尺是相當重要的，才不會造成意想不到的效果。

- 目標：設定場景中所有目標的種類及代號。每一個目標，在準備場景時都會被 target generator 初始化，target generator 便根據這裡設定的種類來選擇要產生的物件種類。在模擬場景中，目標物件是以代號來辨別，因此這裡設定的代號，在之後的動作指令設定中也會用到，使用者可以根據代號來指定要設定的目標。在本計劃中，我們實作了一個目標物件以作爲模擬分析之用。
- 基地：設定場景中基地的代號、位置和基地的武裝。學生操作的地對空飛彈儲存於基地中，因此教官指定給學生武器時，便是以基地爲單位。這個部分的設定，便是設定在地圖上的基地，每一個基地基本都有一個代號和基地的位置，然後再加上一些基地內飛彈資料的設定。在模擬場景使用時，當學生收到被指派基地的通知時，也會收到該基地的這些設定資料，學生操作介面便根據這些資料，將擁有的飛彈物件初始化，之後學生只要操作這些飛彈物件就可以攻擊目標。在本計劃中，我們實作了一個飛彈物件以作爲模擬分析之用。
- 目標的動作：儲存場景中目標的動作指令。所有目標的動作都要在模擬之前事先在場景中設定好，就像是設定目標的預定軌道。這個部分可以使用的指令有起飛、設定速度、設定移動目的地、著路等四個指令。透過這四個指令將目標在場景中的運動情形描述出來。在準備場景時，target generator 便將這些動作指令設定給相對的目標物件，模擬場景進行時，目標就根據指令移動，就像目標在執行任務一樣。

透過上面這五類場景設定，便可方便的閱讀和模擬每個場景。這些場景設

定都可以從編輯器中方便地完成，詳細的設定請參考場景編輯器的部分。

5.5 物件管理機制

這個機制是根據程式的需求所設計出來的。如果沒有這個機制會發生以下的情況，在每次要更新物件的資訊時，管理物件的程式便需要將物件串成 list 的形式，然後根據這個 list 一個接一個的呼叫物件，然後把更新的資訊放上網路，這時候如果 list 中的某一個物件毀滅了，那 list 就會斷掉，當程式在依序呼叫物件時，便會產生錯誤，造成該次更新資訊產生問題。像這種情形會發生當 target generator 收到毀滅物件的通知時。

根據前面的說明可以知道，物件的資料更新和管理是要分開執行的。所以，我們設計了一套機制，這套機制是這樣運作的：當收到物件摧毀的通知時，本系統並不直接摧毀該物件，而先將這個摧毀指令放在一個稱爲命令列的 queue 中，在物件每隔 1/4 秒的更新資訊之前，便先執行這個命令列中的命令，待所有的命令執行完後，才處理物件的資料更新動作。這樣每隔 1/4 秒，物件都會先將有關管理的指令完成，然後才執行更新資訊的部分，這兩個有關物件的動作就不會發生衝突。

這套機制在需要管理物件的地方都會用到，本系統中主要用在 target generator 和學生台中，target generator 管理目標物件群，而學生台要管理飛彈物件群。所以，我們便在本系統中的這兩個元件中使用這套物件管理機制。

6 結論

本計劃的目標是評估一個分散式的即時訓練系統。系統分別提供訓練者及受訓者的操作平台，每次訓練時，訓練者一位，受訓者可多達 8 位。整個訓練系統藉由模擬場景達到訓練的目的，受訓者操作使用介面，並判斷當時情況以攻擊場景中的飛行目標。在系統的每次模擬場景中，最多可同時處理約 100 個目標，每個目標每隔 1/4 秒更新自己的狀態，受訓者會在操作平台上接收到這些目標鎖更新的資訊。

經過不斷的研究過程後，我們決定了本系統比較適合採用的底層通訊機制，然後再一些通訊機制的實作中，根據測試數據，提出一個叫適當的實作版本。另外，在這個通訊機制上，我們根據本系統的需求設計了一套模擬系統的架構，並詳細構思在此架構下所運作的系統機制，以期讓整個系統能夠達到需求。

相信在這些有關系統底層的先期研究，以及設計系統的參考架構之下，將有助於往後的開發者參考，可以藉由本計劃研究成果將本系統時作出來，或作爲設計類似系統時的參考文件。

7 參考文件

7.1 Book and Paper

- 〔1〕Michael Leventhal. David Lewis. Matthew Fuchs. (1998). Designing XML internet applications.
- 〔2〕Alexander Nakhimovsky. Tom Myers. (1999). Professional Java XML programming.
- 〔3〕Ma K. Jiao L. Shi Z. (1997). WaveJava-wavelets based network

computing. SPIE-Int. Soc. Opt. Eng. Proceedings of Spie - the International Society for Optical Engineering, vol.3078, 1997, pp.254-65. USA.

- 〔4〕Bitti MG.. Matta G.. Tuveri M. Paddeu G. Pescosolido M. Pili P. Scheinine A. Zanetti G. (1996). A WWW-based distributed system for medical data analysis and 3D reconstruction. CAR '96 Computer Assisted Radiology. Proceedings of the International Symposium on Computer and Communication Systems for Image Guided Diagnosis and Therapy. Elsevier. 1996, pp.345-50. Amsterdam, Netherlands.
- 〔5〕Singh G. Yiwen Gu. (1997). An object oriented system for developing distributed applications. Fourth International Conference on High-Performance Computing (Cat. No.97TB100185). IEEE Comput. Soc. 1997, pp.192-7. Los Alamitos, CA, USA.
- 〔6〕Burge LL III. George KM. (1999). JMAS: a Java-based mobile actor system for distributed parallel computation. Proceedings of the Fifth USENIX Conference on Object-Oriented Technologies and Systems (COOTS'99). USENIX Assoc. 1999, pp.115-29. Berkeley, CA, USA.
- 〔7〕Konuru R. Srinivasan H. Jong-Deok Choi. (2000). Deterministic replay of distributed Java applications. Proceedings 14th International Parallel and Distributed Processing Symposium. IPDPS 2000. IEEE Comput. Soc. 2000, pp.219-27. Los Alamitos, CA, USA.
- 〔8〕Fu Wang Thio. Hinny Pe Hin Kong. (2000). Distributed multimedia database: a design and application study. Proceedings Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region. IEEE Comput. Soc. Part vol.2, 2000, pp.842-7 vol.2. Los Alamitos, CA, USA.
- Orfali, Robert and Harkey, Dan and Edwards, Jeri and J. Wiley (1997) :

Instant CORBA

- Orfali, Robert. (1998) : Client and server programming with Java and CORBA
- Rogerson, Dale E. (1997) : Inside COM
- Sessions, Roger. (1998) : COM and DCOM: Microsoft's vision for distributed objects
- Thai, Thuan L. (1999) : Learning DCOM
- Troy Bryan Downing : Java RMI: Remote Method Invocation
- Voge, Andreas, and Duddy, Keith, and J. Wiley (1997) : Java programming with CORBA
- Plasil,F., Stal,M. (1998): An architectural view of distributed objects / components in CORBA, Java RMI/COM/DCOM. Software Concepts & Tools (vol. 19, no. 1)

7.2 Internet Resource

- 〔9〕JMS website：http://java.sun.com/products/jms/index.html
- 〔10〕Objectweb website：http://www.objectweb.org
- 〔11〕Openjms website：http://www.openjms.com
- 〔12〕Fiorano website：http://www.fiorano.com
- 〔13〕Softwired website：http://www.softwired-inc.com
- 〔14〕Sonic software website：http://www.sonicsoftware.com
- 〔15〕Talarian website：http://www.talarian.com
- CORBA website : http://www.corba.com/
- Java RMI website : http://java.sun.com/products/jdk/rmi/
- Microsoft website : http://www.microsoft.com/

- OMG website : http://www.omg.org/
- RMI guide website : http://java.sun.com/j2se/1.3/docs/guide/rmi/
- A Detailed Comparison of CORBA, DCOM/Java/RMI :

  http://www.execpc.com/~gopalan/misc/compare.html
- CORBA versus Java/RMI :

  http://www.csd.uu.se/~e99re44/projects/pva/reports/comparison/
- Distributed systems with CORBA/RMI :

  http://www.ciol.com/content/technology/techbytes/00012502.asp