

檢視 Network Centric Computing 的重要架構 Microsoft DCOM 和 OMG CORBA

吳上杰 林盈達

國立交通大學資訊科學系

新竹市大學路一零零一號

TEL:(03)5712121 EXT. 56667

EMAIL: gis85525@cis.nctu.edu.tw

摘要

隨著社會的進步，人們對電腦的需求日益提高，當單一的軟硬體已不能滿足需求時，唯一的辦法就是不斷的昇級。不是去買更好的硬體，就是去買更強大的軟體。而原有的設備不是棄而不用，就是讓售他人。當下一次的軟硬體瓶頸再發生時，事情又要再重複一遍。

分散式系統 (Distributed System) 就是要解決這類問題的產物，藉由各個子系統的同心協力，我們可以解決一些非常煩複的工作。從技術面來看，是我們可以運用電腦作更多的事，但從經濟面來看，這代表我們可以以一些比較便宜的設備來處理一些比往需要 昂貴設備才能解決的問題。

傳統上的分散式系統是以硬體為導向的 MPP(Massively Parallel Processor Network) 系統，最近打敗棋王的 IBM 深藍就是個鮮明的例子。至於以軟體為導向的系統，因為執行硬體 或是開發工具的限制，並沒有像令人滿意的成果。現在，隨著 Internet 的普及，將物件導向(Object Oriented) 和網路結合的 Network Centric Computing(NCC) 架構逐漸受到重視。

NCC 是由在其中的各個物件 (Objects) 和各個物件之間的關係 (Relation) 所組成，而以各個物件的交換訊息來進行運作。也就是說，整個系統的重心，已從經系統 的硬體架構轉移到藉著網路溝通的網路軟體物件。至於整體的 NCC 架構，也就是如何定義 物件和物件之間如何交換訊息，各家廠商各自有各自的看法。目前市場上的兩大架構是 Microsoft 的 DCOM 和 OMG 的 CORBA。這篇文章是對這二大系統進行探討和比較，並提出 NCC 未來應有的走向。

1. NCC 的由來

1.1 過去的網路

目前，幾乎重要的作業平台都具備有網路的功能，藉著 World Wide Web 的技術，我們已經可以做到若干程度的跨平台，儘管如此，我們仍然必須面對下面二個 嚴重的問題。

一是開發工具的依賴性。不可諱言的，除了目前當紅的 JAVA 之外，並沒有一個 跨平台的開發工具。

但 JAVA 的執行速度相當令人垢病，當我們只是用 JAVA 寫一隻在瀏覽器 (browser) 跑來跑去的貓時，大家只會在意這隻貓可不可愛，不會覺得慢，但當你發現你在 Pentium 166 下的 HotJava 上的工具列按鈕是一個一個慢慢畫出來的，你真的不會去改用 Internet Explore 或是 Netscape 嗎？用瀏覽器都已如此了，你能想像一個用 Java 寫的 compiler 嗎？也許會有人認為 microJava 或是 picoJava 晶片會是解決問題的方法，但這就意味著二件事。一是我們必須要生產所有主要硬體平台的 Java 硬體加速裝置，二是使用者願意花錢讓自己現在的電腦加上這個 Java 硬體加速裝置，如果真能作到這樣，那不如去說服使用者改用相同的軟硬體設備。

二是各個作業平台設定的多樣性。Unix 大概是親戚最多的作業系統，但是這些來自相同遠祖的作業平台，卻各有各的設定。凡是提到 Unix 系統管理的書籍，一定 要將各式各樣的 Unix 都提一提，也就是說，要讓 IBM AIX 和 SGI IRIX 和 HP HP-UX 和 Sun Solaris 一起合作無間、共用使用者帳號和電子郵件，並不是件容易的事。而 各 Unix 都已如此，若是再加上天生不同路的 Microsoft Windows 系列就更容易 天下大亂。

1.2 NCC 簡介

就如同前面所說的，在 NCC 中，是由各個物件在運作，藉由物件的 封裝特性(Encapsulation)，我們可以很容易同時作到跨平台和效率。因為 各個物件和外面溝通的管道是物件本身提供的介面，只要透過介面的一致化，就可以輕易作到跨平台的特性。另外，在介面之內的運作是黑箱子，所以，各個物件可以為自己的執行環境作最佳化考量。同時，當各個物件在面對所處的 NCC 時，因為執行環境的標準並不會依物件 所在的作業平台和硬體而改變，所以可以省卻各個平台的單獨設定，輕易具有跨平台的 特性。至於開發工具，則是各個 NCC 架構該注意和提供的地方，我們會在後段文字中發現，這個部分具有相當的難度，且目前重要的 NCC 架構都作的不夠好。至於各個物件該具備有多少的能力、該在怎麼的平台上運作和這樣的 平台該具備怎麼樣的特點，就是目前研究 NCC 所應注意的重點。

2. DCOM 和 CORBA

Microsoft 的 DCOM 和 OMG 的 CORBA 是目前兩大 NCC 架構，在附錄 1 和附錄 2 中有對它們兩者作一個簡要的介紹，詳細的資料 可以參考[1][3][4][5][6]。

雖然 DCOM 和 CORBA 都是嘗試將物件導向觀念和網路作一整合，但是它們存在著極大的差異，我們提出以下五點。

2.1 歷史起源

Microsoft 的 DCOM 實際上是從 COM 所衍生出來的，而 COM 則是由 OLE2 技術所精簡而來。至於有名的 OLE2，則是 Microsoft 當初為了要解決複合文件(Compound Documents)所發展出的技術 [2]。所謂的 複合文件，就是指一分文件可以由各個應用程式一起編輯，對使用者來說，他只有一個檔案，但對應用程式而言，其實是彼此的密切合作。因為當時的主流開發工具是 C 和 C++，所以，我們可以看見，在 COM 的上面，處處充滿了指標 (pointer) 的應用，而且，所有 Microsoft 關於 COM 的重要技術，都有複合文件的影子。也就是說，Microsoft 的 DCOM 實際在一開始並沒有考慮到網路的問題，他們是從 IPC (Interprocess Communication) 開始看問題。

而 CORBA 剛好相反，它一開始就是以網路和物件導向的觀點看事情，所以，使用者一開始就看到了網路，而且在 CORBA 在發展的過程中，是以分析和設計為基礎，並沒有和實際的實作混淆在一起，所以看不到 COM 中的指標。也就是如此，CORBA 甚至可以 Unix 的 Shell Script 來開發物件[4]。不過，當 CORBA 的系統是在單一機器上執行時，它不去利用 IPC 就減低了效率。

2.2 中央管理機置

OMG 的 CORBA 是以整個網路環境來設計，所有的 CORBA 物件只向 ORB 負責，是屬於比較集中式的管理。而 Microsoft 的 DCOM 則比較傾向各自為政，並沒有一個中央的管理機制。其實，是 DCOM 將管理的部分一部分交給物件所在的作業平台，一部分交給物件本身。而不論是中央管理機置的存在與否，他們同樣會面臨到可擴充性(Scalable) 的問題。我們會在後面文章 對這一部分作一詳細的討論。

2.3 開發工具的支援

在 DCOM 中，因為物件的溝通介面處處有指標的影子，所以目前的主要開發工具 多是 C++ 和 Java。但是在 CORBA 中，藉著所謂的 language mappings，將介面和一般的程式語言作一對應，就有了廣泛的開發工具。目前，有 C, C++, Smalltalk, Ada95, Unix Bourne Shell, 和 Cobol，之後還會有別的成員 陸續加入[4]。

2.4 動態介面的產生

在 CORBA 中，有特別為動態介面提出了一些解決方法[4]，雖然意義和傳統物件導向理論 的 Run-Time Type Identification(RTTI) 以及 Template 相去不遠，但至少是一個好的開始。相形之下，DCOM 這部分就 顯得不足。

2.5 開放程度

開放其實涉及二個部分，一是 NCC 架構對各式作業平台的支援，另一是 NCC 架構本身。在作業平台 的部分，CORBA 幾乎在所有重要的作業平台上都有影子，這使得以 CORBA 為發展平台的應用程式可以很輕易地提供 跨平台的支援，但是 DCOM 目前只有 Windows 系列和 Mac 的版本。再著，CORBA 中的 Common Facilities 為 協力廠商 (3rd party) 提供了一個明正言順的位子，使得 CORBA 本身可以容易擴充，但在 DCOM 中，目前所有重要的 技術都掛著 Microsoft 的商標，協力廠商可能不容易找到合適的著力點，就算找到了，可能還要考慮到球員兼 裁判的 Microsoft 的態度。

3. 一個新的 NCC 架構

人在變，時代在變，需求也在變。雖然藉由 DCOM 和 CORBA 的發展，我們解決了非常多以往不能解決的 問題，但是不可諱言的，我們眼前有了新的問題，需要新的 NCC 架構。

3.1 跨平台的物件

在 NCC 中，除了各個運作的物件外，還有各物件之間的關係。依照 OMT[7] 的分析方法，在某些情形之下，我們 必須要用物件來實作各物件之間的關係。但是在 NCC 之下，各個物件是分散在網

路各地，如此一來，代表物件之間 關係的物件就會位於不只一台機器上。例如，若是物件 C 是表示位於 VMS 的物件 A 和 OS2 的物件 B 的關係，那物件 C 就跨散二個不同的系統，那這物件 C 要如何實作呢？

3.2 開發時間和執行時間

在 NCC 中，我們的工具應該具有多樣的選擇。當開發時間急迫時，我們應可以用 RAD (Rapid Application Development) 來加速產品上市時間。但當以品質和效率考量為主時，我們就應以程式碼的品質作為考量。也就是說，我們應有足夠且多樣化的開發工具以應付各種需求。

3.3 和 NC 的整合

NCC 和 Network Computing(NC) 的整合是一條必經之路。在 NC 中，各個物件的大小就會是一個值得考慮的觀點，因為它非但會影響物件在網路上下載(Download)所要的時間，也決定一台 NC 要多大的記憶體。換言之，在 NCC 中，NCC 系統本身和 NCC 物件之間的權責分配會因為 NC 的出現而必須重新考量。

3.4 QoS 的支援

未來，不論是完全的 ATM 或是 Internet 2[8]，支援 QoS 是必然的趨勢。當網路層 (network layer) 具有 QoS 的特性時，上面的應用層(application layer) 沒有理由不支援，而且，像 Internet Phone 的應用程式在執行時，我們確實需要 QoS。所以，NCC 該加入對 QoS 的支援。

3.5 分散式保密的支援

在現在的環境下，保密是重要的機制。當加密趨向安全，解密就趨向複雜。如果整個系統是由各個 物件一起合作，我們就可以考慮以分散式運作來作加密和解密，也就是說，在加密和解密的過程，加入群組 (Group) 的觀念。群組內的信任度高，群組內的通訊可以用網路系統所提供的保密措施，當要和群組外的 物件 通訊時，就用很複雜的加密措施，但這加密是由群組內的各個 物件 一起平行來算，所以，每個物件 所要付出的額外負擔並不會很大。這麼一來，不但可以有安全的保障，每個 物件的解密負擔也可以降低。所以，NCC 的架構該在這部分作探討。

3.6 動態更新軟體

動態更新(Dynamic Update)在一個大型的系統中是非常重要。不論是因為修正錯誤或是功能加強，如果只是為了更新某一部份的 物件就要求所有的相關設施都停下來，是不合情理的。所以，NCC 的架構該有動態更新物件的能力。

3.7 可擴充性

在一作分散式的系統中，我們會以最少的資本來作最佳的運用，例如，當我們用了二萬元作到了百分之二十的 功能時，我們會希望用四萬元可以作到百分之四十的功能。換言之，我們會要一個具有可擴充性的系統。在 MPP 中，IBM 的深藍就是個鮮明的例子。如果，我們有了具可擴充性的 NCC，誰能保證下一次打敗棋王的不是一個以 Internet 2 平台的 NCC 系統？

4. 結論

這篇文章主要是要表達三個重點。第一，目前的分散式系統著重于硬體，但藉由新一代以軟體為導向的 NCC，我們可以作到更多的事情。第二，我們對現有的二大的 NCC 作了深層的技術性探討，並在應用面上 比較它們之間的差異。第三，在面對新的問題時，我們對於下一代的 NCC 提出了發展的方向。

5. 參 考 文 獻

- [1]Distributed Component Object Model,
<http://www.microsoft.com/germany/partner/sap/tech/proj/dcom.htm>
- [2]David Chappell, *Understanding ActiveX and OLE*, Microsoft Press, 1996
- [3]Dale Rogerson, *Inside COM*, Microsoft Press, 1997
- [4]Steve Vinoski, "COBRA: Integration Diverse Applications Within Distributed Heterogeneous Environments", *IEEE Communications Magazine*, Vol. 35 No. 2
- [5]Robert Orfali, Dan Harkey, Jeri Edwards, *Instant CORBA*, Wiley Computer Publishing, 1997
- [6]OMG Technical Library, <http://www.omg.org/library/library.htm>
- [7]James Rumbaugh etc., *Object-Oriented Modeling and Design*, Prentice Hall International, 1991

附錄 1. 簡介 Microsoft DCOM

1.1 COM 的基本觀念

Microsoft 的 Network Centric Computing 架構是由各式各樣的 Component Object Model (COM) Objects 所組成。每個 COM Object 皆是透過一組或多組的 Interfaces 對外界提供服務，如圖一所示。其中，每個 interface 是由一個或多個的 methods 構成，而 method 就是一般傳統的函數 (function) 或是程序 (procedure)。

每個 COM Object 的每個 interface 都有一個唯一的 Globally Unique Identifier (GUID)，使得客戶端 (client) 可以用 GUID 來表示所要用的 interface。若是 COM Object 因為某些原因必須版本更新，那新版的 COM Object 一定要提供舊版 COM Object 的全部 interface，而且這些 interface 的 binary 不可以改變，也就是說，使用某個固定的 GUID 的客戶端並不用因為提供這個 GUID 的 COM Object 的版本更新而一起更新。

1.2 COM 和傳統物件導向的比較

傳統物件導向的三大要素是封裝 (Encapsulation)、多形 (Polymorphism) 和繼承 (Inheritance)。在 COM 中，客戶端只能透過 interfaces 來得到 COM Objects 的服務，並不需要、也沒有辦法知道 COM Objects 內部的運作情形，這就是標準的封裝的觀念。另外，COM Objects 並不禁止不同的 interfaces 提供相同或類似的服務，也就是說，對於 Polymorphism 可以輕易作到。

COM 中和傳統物件導向觀念比較不同的是對繼承的解釋，傳統上，物件透過繼承是為了善用已發展

的物件，避免相同的工作一再地被重複作。至於善用的方法，有 implementation inheritance 和 interface inheritance 兩種。前者是子代 (child) 將親代 (parent) 的程式碼繼承，當子代要用親代的功能時，子代是用親代的程式碼。至於後者，則是子代繼承親代的介面，但是運作的方式則由子代自己處理。一般的傳統物件導向語言，如 C++，或是傳統物件導向分析和設計 (OOAD)，如 Booch Method，是二種都有支援，而在 COM 中，則只有支援 interface inheritance。

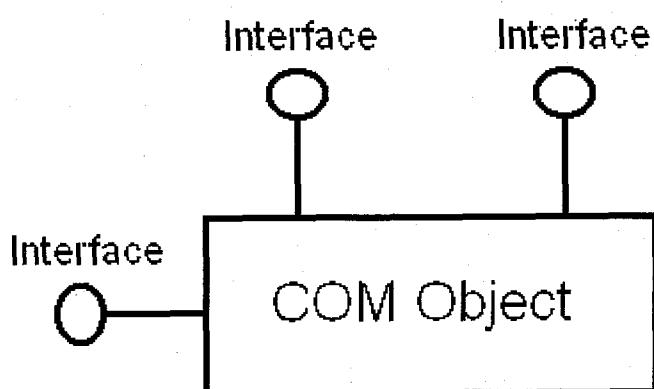
1.3 COM 的重要技術

在 COM 中，有許多重要的技術是由 Microsoft 所提出，基本上，這些是 Microsoft 在開發 Word、Excel 等應用程式時一起開發出來的，具有相當的代表性。

Automation 是將 COM Objects 的 interface 予以簡化成 dispinterface，使得一些比較簡單的語言也以 dispinterface 來用 COM Objects 提供的服務，Visual Basic 和 Microsoft Excel 合作就是一個有名的例子。若是一個 COM Object 要啓動並初始化另一個 COM Object，Monikers 就提供了是一個標準的方法。若是一個應用程式中的多個 COM Objects 要共用一個檔案，就可以用 Persistance，若是交換資料，就可以用 Uniform Data Transfer 和 Connectable Objects。而有名的 OLE2，則是運用了 Compound Documents 使得多個應用程式可以一起編輯同一份檔案。另外，對於某些有許多不同實作 (implementation) 但希望有相同介面 (interface) 的應用，如 Database 和 Directory Service，Microsoft 分別提供了 OLE-DB 和 OLE-DS 來處理。

1.4 DCOM

Distributed Component Object Model (DCOM) 是透過 RPC 將 COM 加上網路的功能。當透過網路，因為之前所提的 GUID 的編碼有利用每台機器的 MAC Address，所以在這裡不會有重複的問題，不過，有二件事情和單一台機器的 COM 是不一樣的。一是效率 (performance)，一是保密性 (Security)。在網路上運作的 COM Object，為了要降低網路上的傳輸量，會將一些片斷的工作一起作，也就是說，會有一些運作的機制，因為網路上效率的考量，和原本單一機器上的運作方式不同。另外，在保密性方面，DCOM 提供了 activation security 和 call security 二種。前者是考慮哪些 COM Object 被准許啓動這台機器上的 COM Object，後者是考慮到哪些 COM Object 可以使用這台機器上的正在執行的 COM Object 和它們的介面。



圖一 COM 物件

附錄 2. 簡介 OMG CORBA

2.1 CORBA 的基本觀念

OMG 的 CORBA 是另一種的 Network Object，它的基本架構如圖二所示。CORBA 是利用一個稱作 Object Request Broker (ORB) 來統籌一切的事項。其中，Common Object Services (CORBAservices) 是提供基本服務給各個客戶，是屬於系統的物件。Common Facilities (CORBAfacilities) 則是屬於客戶端的物件，和 CORBAservices 類似，不過，可以由各個協定廠商(3rd Party)自行發展，提供更廣的服務。而 Application Object (AF) 則是使用者直接操作的程式。

而在 CORBA 中，雖然是 CORBA Objects 提供客戶端服務，但所有的服務都透過 ORB 來運作，而且，對於客戶端，ORB 要隱藏 COBRA Objects 的位置、CORBA Object 的實作，和 CORBA Object 的執行狀態。也就是說，客戶端非且連 COBRA Objects 的內部運作不知道，連 COBRA Objects 是在網路上的哪一個位置和 COBRA Objects 是不是正在執行都不清楚。

2.2 CORBA 的 Object Adapters

COBRA Objects 和 DCOM 類似，也是透過 interface 提供服務，不過，COBRA 的 Object Adapters (OA) 將 interface 更一般化。當客戶端不知道詳細的 interface 資訊，便可以透過 OA 作中介者，和 COBRA Objects 作溝通。這樣作的最大好處是 COBRA Objects 不用為了每種程式語言建立每種介面，它只要有一組介面，然後再為各種程式語言建立各種 OA 即可。

2.3 Dynamic CORBA

在 CORBA 中，除了在編譯 (compile) 時確定的介面外，還支援動態介面的觀念。藉著 Interface Repository (IR)，COBRA Objects 的介面資料可以在執行時 (runtime) 動態產生，其中，負責提供服務的一端是 Dynamic Skeleton Interface (DSI) ，而作為客戶端的是 Dynamic Invocation Interface (DII)。如果和傳統的 RPC 對應，DSI 和 DII 可以視為動態 (dynamic) 或是一般型 (generic) 的 skeleton 和 stub ，不過，DSI 和 DII 是由 ORB 所掌控，不屬於任一個 interface 。

2.4 Common Object Services

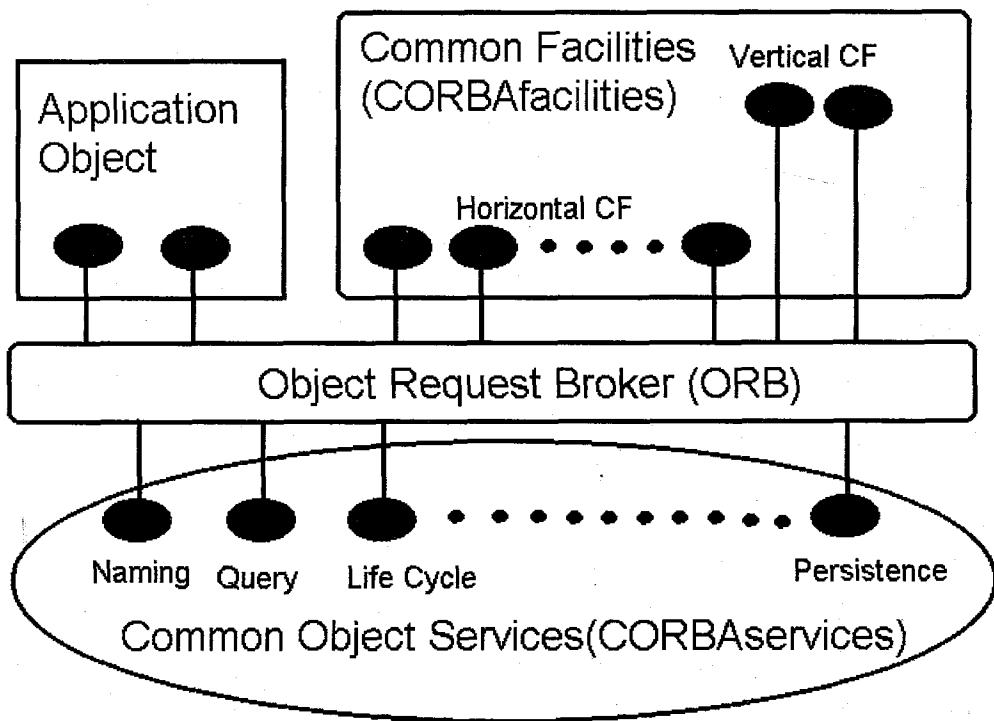
CORBA 的 CORBAservices 和 DCOM 比起來就可以輕易看出以 CORBA 網路為中心的架構。幾乎重要的網路功能，都在 CORBAservices 有支援。例如，和查詢 CORBA Objects 資料有關的有 Naming 和 Trading ，和平行處理有關的有 Concurrency 和 Time ，和保密有關的有 Security ，和 MIS 有關的有 Persistence 、 Transactions 、 Query 等等。

2.5 Inter-ORB CORBA

在原始的 CORBA 定義中，一切的系統都是由 ORBs 來控制，並沒有，同時也不需要考量各個 COBRA 系統之間的運作。但隨著 Internet 的普及，各個 COBRA 系統就有了互相溝通的必要。所以，CORBA 2.0 中就定義了 General Inter-ORB Protocol (GIOP) 作為各個 ORBs 在連接導向的網路 (connection-oriented transport) 上互相溝通時所用的訊息格式。

而 Internet Inter-ORB Protocol (IIOP) 則是 GIOP 在 TCP/IP 上的一個實作協定。透過 IIOP，各個不同的 ORBs 可以在 Internet 上一起運作。當某個 ORB 想要知道另一個 ORB 中的 CORBA Objects 的資料時，就以一種標準的 Interoperate Object Reference (IOR) 來查詢和記錄資料。

另外，CORBA 2.0 同時也提供了 Environment-Specific inter-ORB Protocol (ESIOP)。ESIOP 的主要目的是將 GIOP 和現存的網路環境作一整合，例如，第一個 ESIOP 就利用 Distributed Computing Environment (DCE)，稱作 DCE Common Inter-ORB (DCE-CIOP)。



圖二 CORBA 基本架構