UML Modeling of A Component-Based Software Architecture
元件式軟體架構之 UML 模式化

楊鎮華，黃鈺晴
國立中央大學 資訊工程系
jhyang@se01.csie.ncu.edu.tw

張誠，王台中
中山科學研究院 三所十三組 信號處理實驗室

摘要

本篇論文的目的是報告我們對雷達數位信號處理系統之軟體架構以統一化模式語言 (Unified Modeling Language, UML) 進行模式化這個計劃現在的研究結果。在此計劃中雷達數位信號處理系統的軟體架構我們採用由 OMG 所制訂的 CORBA 規格的軟體架構，以便讓雷達數位信號處理系統能真正實現軟體重用的理想以及減少系統發展者的工作。因統一化模式語言 (Unifie Modeling Language, UML) 已通過 OMG 的標準認證，成為現在最普遍被採用的標準模式語言。所以我們將對雷達數位信號處理系統之軟體架構中的ORB 以 UML 進行模式化，以瞭解在雷達數位信號處理系統 中 ORB 的架構。

關鍵字(Key words)：軟體架構 (Software architecture)，軟體元件 (software compon ent)， 統一化模式語言 (UML)， 數位雷達數位系統 (digital radar digital systems)。

1. 簡介

由於分散式應用程式被廣泛的採用而引導應用程式的發展程序由“從零開始”轉變為“系統整合”。在“元件式 (component-based)” 的發展方法下系統發展者可以從不同的機器上去選擇很多種不同的硬體及軟體元件，然後將這些元件以系統組態的方式整合以達到計算效能及費用的需求。因為在分散式環境中相同的軟體架構下我們只要將各個軟體元件用 IDL（Interface Definiti Language）包裝起來，則各個軟體元件便可互相的存取彼此間各軟體元件所提供的服務，所以軟體元件 (Software Component) 能夠徹底的發揮軟體重用 (Software Reuse) 的概念。因此 “元件式 (Component-Based)” 的軟體發展程序便成為現今發展軟體應用程式的主流了。以 “元件式 (Component-Based)” 的技術的範例包含有 design patterns、 software architecture、 software reuse、 COTS (commercial off the shelf)、 JavaBean、 Active 以及 middleware。

傳統上一個雷達數位信號處理器 (digital signal processor) 包含了數以百計的硬體模組。在設計、實作及生產硬體模組上是一個很費時的過程。此外，對於這樣的一個精密複雜的雷達數位信號處理器的後勤支援系統也是複雜昂貴的。每一年撰寫雷達數位信號處理器的程式的動機是來自於電腦硬體效率極大改進和電腦系統的價格下落。此外，因為軟體比硬體更有彈性，透過採用平行多重處理器的架構和藉由使用軟體來實現多數雷達數位訊號處理器的功能來發展新數位信號處理機的紀元已經開始。在簡化硬體架構的複雜性以後，在發展上及數位信號處理機的後勤支援系統上的費用皆可被降低。藉由這個方法，設計雷達數位信號處理器的挑戰便是要如何有效地設計電腦程式及管理這些處理器。

設計數位信號處理器的新傾向是不僅要透過採用平行多處理機的架構來改進他們的處理速度而且要使他們的基礎架構賦予彈性，以便於一般性的數位信號處理機的構造組織可以應用於許多不同的系統。負有彈性和可再利用性的需求激發了使用”以元件為導向 (Component-oriented)” 的 middleware 的想法。Middleware是一個在分散式應用程式下允許每個應用程式可利用仲介者(Broker)彼此互相溝通但卻不用去關心應用程式是位於何處、是誰去發展這個應用程式的、它們是用那種程式語言所撰寫而成的以及他們是在那種作業平台下執行的一種軟體經紀人的概念。Middleware促使能夠在一個 “插入式執行(plug-and-play)”的”元件式 (component-based)” 的軟體環境下建構軟體系統。Middleware 的工作就像將軟體轉換成以監控交易處理和透過一個發表公佈和訂閱(publication-and-subscription 機制管理服務要求。這些伺服器端(server)將它提供的服務的種類對 middleware做發表公佈，並且這些客戶端(Client)藉由訂閱(subscription)服務來提出要求服務，此時 middleware就分發服務要求到相對應的伺服器端(server)並將結果傳回給客戶端(Client)。

本篇 paper 的目的是報告我們對雷達數位信號處理系統之軟體架構以統一化模式語言 (Unified Modeling Language, UML) 進行模式化的研究。任何系統在於分析與設計之時，最初亦是最重要的是對整個系統在於問題聲明（使用者需求）的描述表達，而問題聲明（使用者需求）的描述部分須要清楚及正確。因為對於往後整個軟體生命週期各個階段而言，所依循的方針即是最初的問題聲明（使用者需求）的描述部分。所以，欲降低風險及成本，一份清楚及正確的系統問題聲明（使用者需求）之描述是很重要的。UML 是一種用來描述系統藍圖的模式語言，其中所定義的符號不但具有豐富的語意並且讓系統分析與設計者便於溝通瞭解。UML 承襲物件導向分析與設計 (OOA&D) 方法，所以擁有著重複使用 (reuse) 及方便維護的特點。是一種用來描述系統的藍圖的標準模式語言。UML 所使用的符號不但容易了解且具有豐富的語意，並已通過物件管理組織 (Object Management Group, OMG) 的標準認證，成為現在最普遍被採用的標準模式語言。其應用領域遍及軍事電子、企業資訊系統、通訊、醫療電子及網際網路。利用 UML 的模式語言文法，搭配 Rational Rose 這套工具軟體的優點如下[9, 11, 12]：

一、 程式視覺化 (program visualization)：利用 UML 所定義的圖 (diagram)，描述出整個系統架構以及所有重要的相互關係。

二、 產生程式碼樣板 (program language code template generation): 利用 Rational Rose 中所提供的功能，可將所繪製的類別圖自動轉成部分的程式碼（以 C++, Java, 或 Visual Basic 等語言，以提供在未來實作階段時，撰寫程式參考之用。

三、 反向操作 (reverse engineering): 除了將圖自動轉成部分的程式碼以外 (forward engineering)，Rose 也提供了將程式碼自動轉回成類別圖。因此，UML 除了可在整個系統開發初期先做為分析及設計的工具；亦可對於一個已實作的系統，對其做反向分析。

四、 文件提供的自動化 (automatic documentation): UML 對於專案管理及文件記錄方面提供了模式語言及應用工具，其中記錄了使用者需求、整個系統架構、部分的程式碼。

對系統發展人員而言，任何可行性的分析都需要透過與系統使用者之間的互動才能夠達成，繼而才能降低成本與風險的程度。利用物件導向分析及設計的方法及 UML 簡易豐富的模式語言，其中的優點便可增進系統發展人員與系統使用者之間的溝通說明與系統發展的效率。而使用 Rational Rose 這套工具與之搭配可詳細且清楚地記錄系統模式化所應用的圖及文件說明。基於 UML 具有上列的優點所以我們採用 Rational Rose 這套工具軟體來對雷達數位信號處理系統之軟體架構來進行模式化(Model)，以便能清楚的來描述雷達數位信號處理系統之軟體架構的系統藍圖。

而 CORB 是由 OMG 在一九九一年所制訂的一個物件導向 (object-oriented) 的分散式環境標準。在 OMG 發表 CORB 這套物件導向分散式工作環境規格前分散式環境並無一定的標準可遵循。CORB 是一個 "軟體代理商" 的概念，在此概念下它允許軟體應用程式可以透過一個代理商彼此之間互相做溝通，而不用去注意應用程式是位於那裡，誰去發展應用程式，用那種語言撰寫應用程式，及應用程式是在那種平台下執行的等這些事情。CORB 也使得在一個 "插入式執行" 的元件軟體環境下開發一個軟體系統變得容易。

在本篇 paper 中，我們於第一節中說明為何要將 CORBA 規格引入雷達數位信號處理系統之軟體架構中。於第二節中說明"元件式(Component-Based)"軟體架構的興起原因。在 2.1 節中就 Figure 1. 物件請求仲介者 (Object Request Broker, ORB) 的架構來簡略的說明 OMG 所制訂的 CORBA 規格。在 2.2 節中針對 Figure 2. 來說明如何在 ORB 的軟體架構中開發應用程式的步驟。於第三節中說明了我們在雷達數位信號處理系統之軟體架構中 ORB 的 UML 模式化的結果。在 3.1 節中我們說明了雷達數位信號處理系統(Radar DSP)，並於 3.1.1 中說明雷達數位信號處理系統元件的功能。而於 3.1.2 中說明雷達數位信號處理系統的功能需求，並以 Figure3 來說明雷達數位信號處理系統。在 3.2 節中我們簡略的說明了雷達信號處理器(Radar DSP)的軟體架構為何，並以 Figure 4 來加以表示說明。在 3.3 節中則是說明了如何運用 UML 中的使用按例圖(Use Case Diagrams)來模式化物件請求仲介者 (Object Request Broker, ORB) 的軟體架構，並使用 Rose 這個工具來實作模式化 ORB 的工作，而將模式化 ORB 的使用案例圖(Use Case Diagram)的結果於 Figure 5 中表現出來，並針對 Figure 5 中的三個使用者(Actor)及九個使用案例(use case)來作一個簡略的說明。在 3.4 節中我們根據 3.3 節所模式化 ORB 的使用案例圖(Use Case Diagram)的結果來描述各個使用案例(Use case)的情節，及根據所描述的情節畫出相對應的順序圖(Sequence Diagrams)，而我們以 Bind Object 此使用案例(Use Case)為例，將其順序圖(Sequence Diagram)於 Figure 6 作說明。於第四節中我們說明了如何運用在第 2.2 節中所提出的開發 ORB 應用程式的步驟來發展我們的雷達數位信號處理器上的應用程式。於第五節中我們則是對本篇 paper 作了一個簡單的結論及討論在雷達數位信號處理系統上未來的工作該著眼於何處繼續的發展。

2. 元件式(Component-Based)軟體架構

軟體重用 (Software Reuse) 的想法是物件導向程式語言崛起的主要因素，而傳統的物件導向語言雖然解決了以往的軟體危機（例如大型軟體專案的維護不易、因程式碼龐大而不易閱讀進而造成維護不易及難以加入新的功能），但卻也造成了下列新型態的軟體危機: [6, 7]

一、 各種的物件導向語言無法直接溝通. 因為各種物件導向程式語言的物件模型(Object Model)皆不一樣，所以造成現今市面上的物件導向語言無法直接溝通，以致於無法在程式中直接使用由別種物件導向語言所寫的物件，如此便造成軟體重用 (Software reuse) 的另一種危機。

二、 物件導向語言無法連結舊有系統的程式庫: 在某些情況下（例如舊有系統的原始程式碼遺失）造成物件導向語言無法連結使用舊有系統的程式庫，如此便無法重用以前的程式碼，所以便無法達成軟體重用的想法。

三、 物件導像程式語言之程式庫的原始碼一定要公開: 因為在寫程式時常會使用一些標準程式庫的函數或聚集處理，所以必須將程式庫的原始碼公開以便程式設計師可較快瞭解程式的內容及功能，又加上有時必須要做除錯的工作，則此時連軟體程式的原始碼都必須公開以便於除錯，但在軟體業上若將軟體的程式碼公開的話，那將會致使軟體公司無法生存。

四、 各式各樣軟體間的溝通呼叫所使用的存取服務機制皆不相同: 雖然各類的軟體可以彼此互相存取別的軟體所提供的服務，但因著所欲存取的服務型態不同而致使所使用的存取服務機制也跟著不相同(例如呼叫函式庫中的函式以取得服務或呼叫其他行程(Process)中的所提供的服務時，此種狀況就必須使用各行程間的訊息通訊機制來做溝通，而另外一種狀況則是我們可能須要使用作業系統所提供的服務，則此時就必須使用系統呼叫的的機制來做溝通) --- 共有函式的呼叫、各行程(Process)間的訊息傳遞、系統呼叫、及網路通訊等溝通機制，進而造成軟體間的溝通呼叫變的非常的煩瑣。

因為基於以上四點的新型態的軟體危機，所以我們需要一個能夠跨各個程式語言的溝通機制，而非僅只是做某些語言之間的溝通即可，也就是我們所需要的是一個更高層次的軟體間的溝通機制，亦即定義一個各種軟

體間彼此存取服務的共同方式，也就是說不管須要何種型態的存取服務都使用同一種的溝通機制[7]。而現今 OMG 所定義的 CORBA[2]架構便提供了各應用程式間的溝通機制，以規範各種情況下各種軟體元件溝通的機制，以便能真正達到軟體重用的理想。根據在 OMG 的 CORB 架構下，我們只要將各個軟體 IDL (Interface definition Language) 包裝成軟體元件 (Software Component)，而每個軟體元件皆遵循 CORB 的標準所製作，那便使得在 CORB 的架構下，每個軟體元件皆可互相溝通，彼此利用各軟體元件所提供的各種型態的服務，則在這個 CORB 的架構下，你只需說明需要何種服務，至於有那些軟體元件提供這種服務，那些軟體元件位於何處，使用何種語言去實做這些軟體元件，這些軟體元件在那種作業平台下執行等等的事情，你都可以不須要知道，這些事情都可以由 CORB 來幫你處理即可，如此一來便可簡化系統發展者的工作。所以“以軟體原件為基礎”(“Component-Based”) 的軟體發展程序便由此因應而生，而軟體 IC[6]的概念更因此而興盛，進而使得現今在發展軟體時就不必像以前一樣都必須一切“從零開始”，而僅要去組裝市面上所具有的軟體 IC 便可形成另一個我們所需要的軟體應用程式，如此一來便使得現今撰寫軟體應用程式蛻變成好像只要做組裝的工作即可，所以“元件式”的軟體發展程序便成為現今發展軟體應用程式的主流了。

2.1 CORBA

Figure 1 為 CORB 規格中 ORB 的 Architecture[2]。根據 OMG 的定義，一個物件請求仲介者（Object Request Broker）為一個可提供在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。亦即客戶端(Client)僅需對 ORB 提出 request，此時 ORB 就會去尋找有提供 request 所要求的 service 的 server component，而 server component 是提供一個或數個 Object Implementatio 的程式。當 ORB 找到適合的 Server Component 後便會啟動 Client 端所呼叫相對應的 Object Implementation，進而執行 Client 端的 request。而在這整個過程中，有Client 端 request 所要求的 service 之相對應的 server Component 位於何處，是由誰去發展，及是用那種語言所撰寫而成的，以及是在那種平台下執行的等這些事情。皆由 ORB 來處理，所以 Client 與 object implementati o 可透過 ORB 彼此之間互相做溝通。

Server 端的程式所欲提供的方法需先由 IDL（Interface Definition Language）包裝起來，所包裝而成的 IDL file 的相關資訊會儲存在 Interface Repository 中以供查詢 Object 所提供的方法。而有 Server 端提供一個或數個 Object Implementatio 的程式實作方面的資訊亦會儲存於 Implementation Repositor y以提供 ORB 尋找(Locate)、啟動(Activate 、及執行(Execute)Object Implementation 方面的資訊。當 server 端的程式由 IDL 包裝成 IDL file 後，其 IDL fil 需經由 IDL Compile 去作編譯(Compile)以便產生 Client 端的 Stub 與 server 端的 Skeleton 以及一些可供 Client 端與 Server 端 include 的 Head file 以供 Clien t端與 Server 端使用。然後當 Client 端發出一個 request 後便經由 Client Stu 包裝(marshal)所傳入的呼叫參數再傳給 ORB，由 ORB 藉由 Object Adapter 經由查詢 Implementation Repositor 中的資訊進而去尋找一個適合此 request 的 Server component，更藉由 Object Adapter 的輔助傳送 Client 端的 request 到 Server Component 中相對應的 Object Implementatio n並啟動此 Object Implementation，進而 Object Adapter 經由 Server Skeleton 去作接收到參數的解包裝(unmarshal)的動作才去呼叫相對應的方法(method)，最後當 Object Implementation 中的方法執行完後便將執行結果傳回給 ORB，再由 ORB 將所得結果傳給 Client 端。ORB 不僅讓 Client 端可靜態的呼叫亦提供動態的呼叫，ORB 提供 Dynamic Invocation 來處理 Client 端動態呼叫物件的動作及 Dynamic Skeleton Invocatio 來處理 Server 端動態的分派物件。而 ORB Interfac 則是 CORB 規格對 ORB 所定義的一個介面，此介面有提供數各方法可供 Client 端與 Server 端來呼叫使用。

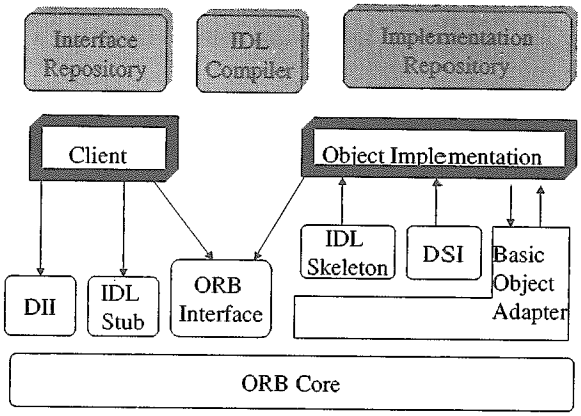

Figure 1. 物件請求仲介者（Object Request Broker, ORB）的架構

2.2 開發 ORB 應用程式的步驟

在 ORB 上發展應用程式的程序上有四個步驟。此四個步驟如 Figure 2 所示[8]，現在我們針對這四個步驟加以詳加說明如下[4]：

Step1、先將 Server implementation (Component) 用 IDL 包裝起來

Step2、用 IDL Compile 去 Compil 包裝好的 Server implementatio 的 IDL file.

Step3、實做 Server implementation (component)
- 一、Creating Impl class ： Server 端的所有 method 皆寫在此 class 中
- 二、Implement Server main routine
  1) 先對 ORB 及 BOA 做 initialization
  2) 宣告 Impl class 的 object
  3) 輸出所宣告新的 object，並告訴 BOA 所宣告的這個 object 已準備就緒可以讓 clien 端去提出要求（request）了。
  4) 通知 BOA 在 server 端已準備好 Object(一個或數個)來接收 request，並等待從 Clien 端所提出的 request。

Step4、實做 Clien 端的程式
- 一、 對 ORB 做 initialization 的動作
- 二、 Clien 端必須 bind 到 Server

Implementatio 中的一個 object，以建立一個到 Server implementation 的 connection。而由 Client 端 Bind 到 Server 中 Object 此方法會傳回一個 Object Reference，則 client 端就可利用此 Object Reference 去呼叫 Server 中的各個方法。

三、 利用 Bind 後所得的 Object Reference 去呼叫相對應的方法。

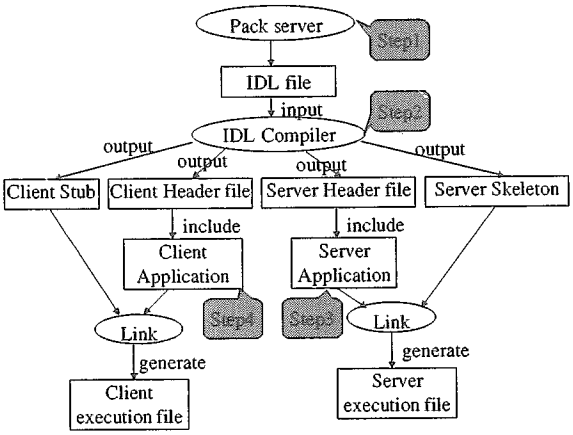

Figure 2. 開發 ORB 應用程式的步驟

3. ORB 軟體架構之 UML 模式化

3.1 雷達數位信號處理系統

3.1.1 本系統元件的功能說明

我們將對雷達數位信號處理系統以元件的方式表示之(請參考圖一)。對於每個元件所負責與擔當的工作及角色的說明如下： RCC (Radar Control Computer)負責產生 STIM 以及 Signal Processor Imbedded Test Equipment (SPITE) 這兩種命令。CCI (Control Computer Interface) 負責在 Digital Signal Processor (DSP) 與 Radar Control Computer (RCC) 之間遞送命令及資料。SPC (Signal Processor Control) 可以對 Digital Signal Processor (DSP) 中任一控制處理器做執行 Executive (EXEC) 的操作。其中 SPC 接收來自 RCC 所傳送過來的 STIM 命令，而後對於 STIM 命令做解譯 (interpret) 的處理工作，再將所解譯過的命令發佈至其他各個控制處理器。在 EXEC 中最重要的任務即是負責在於 DSP 與 RC 之間命令及資料報告的管理。WFG (Waveform Generation) 負責產生所被要求的波形，亦即是類比信號，繼而傳送至 TX (Transmitter) 發射出去。IFP / REC 負責將來自 ANT / BSC (Antenna / Beam Steering Control) 所傳送過來的類比信號轉換成數位信號。TX (Transmitter)負責發射雷達波的工作。ANT / BSC (Antenna / Beam Steerin Control)負責接收所回傳的雷達波的工作。EEC (External Equipment Control)負責 TX 及 ANT / BSC 設備操控的管理工作。RTC (Real Time Control)負責 TX 及 ANT / BSC 時序 (timing) 操控的管理工作。TOD (Time-of-Day)負責提供 RTC 在於時間上的記錄，其中時間的計量是由一 external clock 來產生之。IBU (Input Buffer Unit) 除了做為來自 Analog Signal Processor (ASP) 所傳送過來的數位信號資料之儲存空間，並會對於所暫存的信號資料做前段處理 (Pre-Processing) 的工作。VSP (Vector Signal) 負責將信號資料做偵測與後段處理 (Detection & Post Processing) 的工作，並將所處理完成的資料報告回傳至 SPC。

3.1.2 本系統的功能需求

對於雷達數位信號處理系統的功能需求，在以下內容說明之 (請參考圖一)：RCC (Radar C ontrol Computer) 透過 RCI (Radar Control Interface) channel，將命令以批次處理 (batch processing) 的方式傳送至 IOP (Input / Output Processor) 中的 CCI (Control Computer Interface)。其中 CCI (Control Computer Interface) 所扮演的是遞送 (deliverer) 的角色，接收來自 RCC (Radar Control Computer) 的命令遞送至 SPC (Signal Processor Control)。SPC (Signal Processor Control) 會將所接收到的命令做解譯 (interpret) 的工作，目的是將每個命令分配成若干個待被完成的程序 (program)。SPC (Signal Processor Control) 將這些程序透過 control bus 發佈出去。在於 ASP (Analog Signal Processor) 其中的 WFG 會先接收命令而產生所要的類比信號，並透過 TX (Transmitter) 發射雷達波出去。由 ANT (Antenna) / BSC (Beam Steering Control) 接收回傳的雷達波，將所接收到的信號傳送至 ASP (Analog Signal Processor)，由其中的 IFP / REC 做信號的轉換工作。將所轉 好的數位信號，傳送至 IBU (Input Buffer Unit) 儲存。TX (Transmitter) 及 ANT (Antenna) / BSC (Beam Steerin Control) 操控管理部分，由 EEC (E ternal Equipment Control) 負責設備操控的管理工作，由 RTC (Real Time Control) 負責時序 (timing) 操控的管理工作。其中由 RTC (Real Time Control) 所接收到的 TOD (Time-of-Day) 是由一 external clock 所產生之。IBU (Input Buffer Unit) 除了對於所接收到的數位信號做資料暫存的工作，並且會對於所暫存的信號資料做前段處理 (Pre-Processing) 的工作。由 IBU (Input Buffer Unit) 所處理過的信號資料，會再傳送至 VSP (Vector Signal Processor) 做處理的工作。VSP (Vector Signal Processor) 會將信號資料做偵測及後段處理 (Detection & Post Processing) 的工作，並將所處理好的信號資料，回傳至 SPC (Signal Processor Control 。最後，SPC (Signal Processor Control) 會再透過 CCI (Control Computer Interface)，將所處理好的結果報告回傳至 RCC (Radar Control Computer)。在於整個雷達數位信號處理系統中，由 Control Bus 負責命令部分的傳送工作，由 Data Bus 負責資料部分的傳送工作。

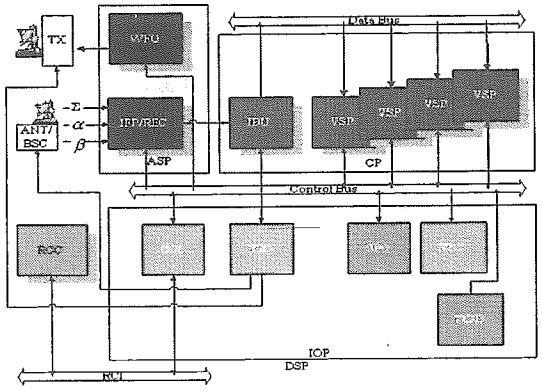

Figure3. 雷達數位信號處理系統

3.2 雷達信號處理器(Radar DSP)的軟體架構

Figure 4 is our radar DSP software architecture. 在本計劃 Radar DSP 中的軟體架構之所以採用 ORB 概念，其目的乃是增加系統的彈性，因為在 RCC (Radar Control Computer) 向 IOP(Input Output Processor 下一個 command (即 request)時，可能同時會有很多的軟體原件 (Software Component) 滿足此 command (因為在此系統中相同作用的 component 會有很多個，例如：此系統會有十餘塊的 VSP(Vector Signal Processor)板子等)，所以我們運用 ORB 的概念，讓 ORB 根據當時系統的狀況來決定要使用那個軟體元件 (Software Component)中的 method 來執行由 IOP 所下的 command，因此可增加其彈性。而另一方面若某個軟體元件 (Software Component) 壞掉了，那因為 ORB 還有別的軟體元件(Software Component)可以選擇讓其執行 comman ， 所以此 comman 仍然可以被完成， 如此便可確保 comman 能夠確實的被執行。而因為 ORB 的架構下，RC 只需下 command，至於有那些軟體元件提供這種服務、那些軟體元件位於何處、使用何種語言去實做這些軟體元件、這些軟體元件在那種作業平台下執行等等的事情，RC 都可以不須要處理，這些事情都可以由 ORB 來幫 RC 處理即可，如此一來便可簡化發展 radar DSP 系統應用程式的工作。[1]

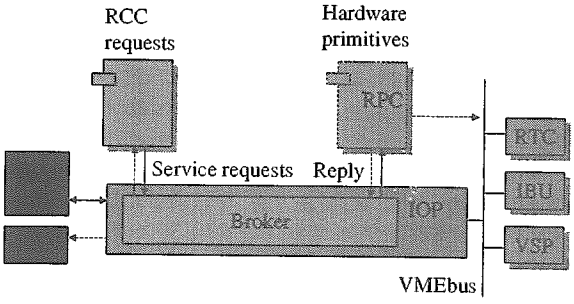

Figure 4. 雷達數位信號處理器(Radar DSP)的軟體架構

我們可以將做在每個硬體板子上的 Hardware Primitive 用 IDL (Interface Definition Language)來包裝成各個 component, 則我們就可以呼叫各個包裝好的 component 中的 method(即 Hardware Primitive) 。又因 ORB 為一軟體架構(Software Architecture)，所以此 ORB 的軟體架構不只可以應用在 radar DSP 的系統上，同樣的在別的系統上也可使用此 ORB 的軟體架構，只要更改其 Server 端的 Hardware Primitive 即可，如此一來便使得此系統的彈性更提高了。在現今發展軟體的過程都已朝"以軟體元件為基礎"的方向朝進，而不再像以前一切都從零開始，致使一些工作老是重複的一做再做，不但浪費人力更浪費時間，也因以軟體元件為基礎的軟體發展過程已成為現今的主流，所以本計劃的軟體架構在將 Hardware Primitive 用 IDL 包裝成軟體元件 (Software Component) 的基礎下使用 ORB 的概念，便可製作出一個軟體架構出來，讓各系統做一些 Server 端的 Hardware Primitive 更改即可在此軟體架構下運作，減輕軟體發展者的工作，更可達到軟體再使用的功能。 故將 ORB 概念帶入軟體架構對此系統而言是壹項非常重要的概念。

3.3 運用使用按例圖(Use Case Diagrams)來模式化物件請求仲介者 (Object Request Broker, ORB)

在 UML 中首先就於使用案例觀點 (需求的說明)，找出在 Radar DSP Software Architecture 中 ORB 所會引用到的使用案例 (Use Case) 及相關的行為者 (Actor)。其中使用案例是行為者與系統之間的互動情形，可以取決於整個系統所會處理到的 functions；而行為者是使用者在於系統當中所扮演的一種角色 (role)，其可為實行使用案例的人或其他系統。而後畫出使用案例圖 (use case diagrams)，並且以情節 (Scenarios) 描述出每個使用案例的實現方法 (realization)。

我們採取兩個步驟來建構一個使用者案例圖 : 1) we 描述整個系統的 problem statement:首先所有的 server Component 都必須先對 ORB 作註冊 (register)的動作，以便讓 ORB 知道在系統中有哪些 Sever Component 可以使用、及各 Server Component 有提供哪些發法可供 Clien t端呼叫。接著 Client 端便可下 request 給 ORB。ORB 再去尋找有提供 request 所要求的 service 的 Server Component。當 ORB 找到適合的 Server Component 後便將之啟動，並啟動 Client 端所呼叫的相對應的 Server Component 中的 Object Implementatio 。由被啟動的 Object Implementatio 執行 request，並將執行結果傳回給 ORB，再由 ORB 將結果傳回給 Client 端。2) 找出每個使用案例(Use Case) 與使用者(Actors)，再畫出 use case diagrams。 Figure 5 即為在 Radar DSP Software Architectur 中 ORB 的 UML Modeling 的 use case diagra 的結果。

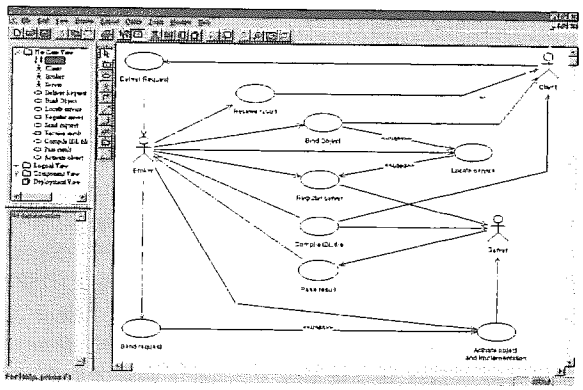

Figure 5. Use case diagram of ORB

在 Figure 5 中有三個使用者(Actor)及九個使用案例(use case)。三個使用者(Actor)描述如下：

一、 Client: 呼叫 server 中的某個 method 的 application
二、 Broker: 根據由 client 端所下的 request，去尋找在系統中有哪些適合的 Server Component，並進而啟動 clien 所呼叫相對應的 Object Implementation
三、 Server: 提供 clien 所須使用的 method,以供呼叫

九個使用案例(use case)描述如下：

一、 Compile IDL file 由 IDL Compile 編譯(compile)包裝 Server Implementation 的 IDL fil ，編譯(Compile)後會產生 Client 端相對應的 Client Stub

和 Server 端相對應的 Server Skeleton 以及 Client 與 Server 所要 include 的 Head file，以便作傳送 request 中參數的包裝 (marshal) 及解包裝 (unmarshal)的動作，

二、 Register server: 當 Server Componen 提供一個或多個 Object Implementatio 的程式完成後，便需將有關 Server Implementation 的相關資訊儲存註冊到 Implementation Repositor y中，以便提供將來在作尋找及啟動相關 Object Implementation 時可以參考的相關資訊

三、 Deliver Request 接收由 Client 端所傳送過來的 request，並將之經由 Client Stub 作參數包裝的動作後傳給 Broker 去處理此 request

四、 Bind Object: 當 Broker 經由 Deliver Reques 此 use case 接收由 Client 端所傳的 request 後，Broker 必須先 Bind 到一個相對應的 Object Implementation 來執行此 request

五、 Locate service: 經由參考 Implementation Repositor 中的資訊去尋找在系統中有哪些 Server Implementatio 適合由 Broker 所接收到的 request

六、 Send request: 將 request 傳給所選定的 Server Implementation 但在這之前 Broker 需先藉由 Implementation repository 的資訊尋找到適合並決定使用哪個 Server Implementation，接著將所選定的 Server Implementatio 啟動後，才能將 request 經由 Server Skeleton 作參數解包裝的動作後傳給給相對應的 Object Implementation

七、 Activate object and implementation 處理啟動所選定的 Server Implementation 及 Object Implementation 的動作，並在啟動後告訴 Broker 已經啟動了且通知 Broker 它們也可以開始接收 request 了。

八、 Pass result: 將 Object Implementation 的執行結果經由 Server Skeleton 作執行結果之值的包裝動作後傳回給 Broker

九、 Receive result: 接收由 Broker 所得到的執行結果，並將之經由 Client Stub 作執行結果的解包裝動作再傳回給 Clien 端

3.4 情節及順序圖(Sequence Diagrams)的模式化

一個情節(scenario)的定義是一個使用案例(use case)的一個實例(instance)，且根據每個使用案例(use case)的情節(scenario)我們可以進一步的推衍出一個順序圖(sequence diagram)。一個情節(scenario)描述執行使用案例(use case)的先決條件(precondition)及提供一個正常處理(basic course)來執行這個 use case。除此之外，一個非主流處理(alternative course)是處理正常處理(basic course)狀況下的例外處理。原始的使用案例(Original Use Case)是表示這個使用案例(use case)是處理一般的功能，而用擴展的使用案例(Extension Use Case)來表示此使用案例(use case)是處理在某個使用案例(use case)下擴展的功能。另外使用進一步的流程(Subflow)來解釋在正常處理(basic course)狀況下各種情況的進一步流程。的我們拿 Bind Object 這個使用案例(use case 的情節(scenario)來做範例。完成 Bind Object 這個使用案例(use case)的先決條件(precondition)是執行完成 Locate Service 這個使用案例(use case)，而在此例中的正常處理(basic course)狀況並無非主流處理(alternative course 與進一步的流程(Subflow)，且在此例的使用案例(use case)下並無擴增的功能所以就沒有擴展的使用案例(Extension Use Case)。

Scenario of ***Bind Object*** Use Case
Precondition：Complete ***Locate Service***
Original Use Case
Basic Course

1. 當 Broker 接收到 Clien 端所傳的 request 時，Broker 根據由先前必須先執行的 Locate Service 此 Use cas 的執行結果，來獲知要由系統中的哪個 Server Componen 執行 request。
2. Bro er 產生由 Locate Service 此 Use Case 中所決定執行 request 的 Server Component 的 Object Reference。
3. Broker 將所產生的 Object Reference 傳給 Client 端。

Alternative Course

根據上述的情節(scenario)，我們推衍出 Bind Object 此使用案例(use case)的一個順序圖(sequence diagram，此順序圖(sequence diagram)如 Figure 6 所示。而在本系統中有九個使用案例(use case)所以基本上應有九個相對應的順序圖(sequence diagram)，因我們只拿 Bind Object 此使用案例(use case)來作範例，所以只畫出 Bind Object 此使用案例(use case)的順序圖(sequence diagram)，至於其他的使用案例(use case)的順序圖(sequence diagram)沒有列出來了。

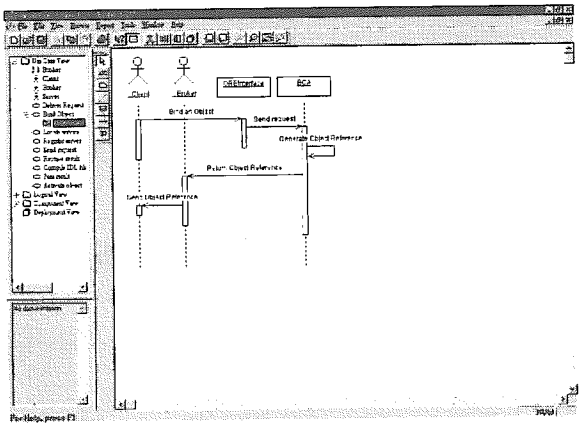

Figure 6. Bind Obje c 此使用案例(Use Case)的順序圖 (Sequence Diagram)

4. 開發 ORB 應用程式

我們運用在第二節所提出的開發ORB應用程式的步驟來發展我們的雷達數位信號處理器上的應用程式。以下是我們的數信訊號處理器上應用程式的實作範例。

Step1、先將 RTC, IBU, VSP 用 IDL 包裝起來

// 例如：RTC 這個 Server Componen 的 IDL fil 的撰寫方式如下

```
Interface RTC
{
    // 宣告在此介面(interface)的各個屬性(attribute)
    // 宣告形式 ：
```

```
    // [readonly] attribute attribute_type attribute_name

    readonly attribute string str_1
    ……
    // 宣告在此介面(interface)的各個方法(method)
    // 宣告形式 ：
    // method_return_value_type method_name
     (in/out/inout   parameter_type
     parameter_name)

    float mtehod_1 (in float var_1 , out unsigned long
var_2)
    ……
}
```

Step2、用 IDL Compile 去 Compil 包裝好的 RTC, IBU, VSP

```
// 執行 IDL Compile 去編譯(compile) ID  file
// 執行命令的形式 ：
IDL_Compiler_name Server_IDL_file_name
// 例如 ： 以 visibroker[3]這個 ORB 中的 IDL Compiler
為例，它的下達命令方式如下
idl2cpp IDL_filename
```

Step3、 實做 RTC, IBU, VSP

```
// 將 server 端的所有 method 皆寫在一個 class 中，且此
class 必須繼承所 include 的 head fi l 中的定義 skeleton
的 class

Class RTCImpl : public _sk_RTC
{
  //宣告此 class 中的變數及實做此 Server 所提供的方法
(method)

  string str;
……
  float mtehod_1 (in float var_1 , out unsigned long var_2)
  {
  //實作方法
  ……
  }
……
}

// 實作 Server 的 main routine
void main(argc,argv)
{
  try
  {
  //先對 ORB 及 BOA 做 initialization

   CORBA::ORB_ptr orb = CORBA::ORB_init(argc,
   argv);
   CORBA::BOA_ptr boa = orb->BOA_init(argc, argv);

  //宣告 Impl class 的 object

   RTCImpl  RTC("傳入參數");

  //輸出所宣告新的 object，並告訴 BOA 所宣告的這個
    object 已準備就緒可以讓 clien 端去提出要求
    （request）了。

   boa->obj_is_ready(&RTC);
   cout << "Account object is ready." << endl

  //通知 BOA 在 server 端已準備好 Object(一個或數個)
   來接收 request，並等待從 Client 端所提出的
   request。

   boa->impl_is_ready();

  }

  //處理例外
  catch(const CORBA::Exception& excep)
  {
   ……
   }
}
```

Step4、實做 RCC request

```
Void main()
{
  try
  {
   //先對 ORB 做 initializa t i o 的動作

    CORBA::ORB_var orb = CORBA::ORB_init(argc,
    argv);

   //Client 端必須 bind 到 Server Implementatio 中的一
    個 object

     RTC_var  rtc = RTC::_bind();
     ……

   //invoke method

     CORBA::Float RTC_method = rtc->method_1();
     ……
   }

   catch(const CORBA::Exception&  e)
   {
    ……
    }
}
```

5. 結論及未來工作

CORBA 是由物件管理組織 (Object Management Group, OMG)所定義的一個在分散式環境下規範各種情況下各種軟體元件之間互相溝通的標準機制，以便能真正達到軟體重用(software reuse)的理想。CORB 規格利用介面定義語言(Interface Definition Language 分離介面與實作，而客戶端(Client)是藉由介面與伺服器端(Server)的實作做溝通，所以 Client 端無法直接的去存取到伺服器端(server)的程式及資料，如此一來便可保護到伺服器端(Server)的程式及資料不會受到別人惡意的破壞。利用 CORB 規格來發展在分散式環境下的應用程式時，你只需說明需要何種服務，至於有那些軟體元件提供這種服務，那些軟體元件位於何處，使用何種語言去實做這些軟體元件，這些軟體元件在那種作業平台下執行等等

的事情，你都可以不用關心，這些事情都可以由 CORB 規格中的 ORB 來幫你處理即可，如此一來便可簡化系統發展者的工作。

因為使用 CORB 具有以及下列的優點：1.增加系統上的彈性 。 2.可達到軟體重用(software reuse)的概念。 3.簡化系統發展者的工作，即可減少所花的人力資源成本。 所以我們將 CORB 的概念引入本計畫中以使其具有上列的優點。首先我們對雷達數位信號處理系統之軟體架構以統一化模式語言 (Unified Modelin Language, UML) 進行模式化，以模式化雷達數位信號處理系統中的 ORB。一方面可瞭解在雷達數位信號處理系統中的 ORB 的架構另外一方面可用來評估 ORB 的效能。而 UML 是一種用來描述系統藍圖的模式語言，其中所定義的符號不但具有豐富的語意並且讓系統分析與設計者便於溝通瞭解。UML 承襲物件導向分析與設計(OOA&D)方法，所以擁有著重複使用（reuse）及方便維護的特點。是一種用來描述系統的藍圖的標準模式語言。且 UML 已通過物件管理組織 (Object Management Group, OMG) 的標準認證，成為現在最普遍被採用的標準模式語言了，UML 所使用的符號不但容易了解且具有豐富的語意所以容易與使用者作溝通，因此使用 UML 來模式化雷達數位信號處理系統中的 ORB 可讓使用者更容易瞭解 ORB 的架構並方便系統發展者作系統的維護。所以我們採用 UML 來作雷達數位信號處理系統中的 ORB 的模式化。

在本計畫的目前進度上並未將即時(Real-Time)限制考慮進去，但因為雷達數位信號處理系統有即時(Real-Time)因素的限制，所以本計畫未來的工作進度上會朝將即時(Real-Time)限制因素加入 ORB 的功能上，並使用 UML 搭配 Rational Rose 這套工具來對具有即時(Real-Time)限制因素性質的雷達數位信號處理系統中的 ORB 進行模式化。而物件管理組織 (Object Management Group, OMG)在 1999 年 2 月 12 日提出了 Real-Time CORBA 1. 的規格，此 Real-Time CORBA 1. 的規格是根據 CORBA 2. 規格(1998/12/01)及 CORBA Messagin 規格(1998/05/05)加上即時(Real-Time)限制因素的擴增。因此在未來的進度上我們將會根據 Real-Time CORBA 1.0 規格來將即時(Real-Time)限制因素加入系統上的 ORB。

6. Acknowledgements

This research is supported by National Science Council in Taiwan under grate NSC 88-2213-E-008-005

7. References

[1] Cheng Chang and Tai-Chung Wang. "Component-Oriented Digital Signal Processors", In International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '99), pages 1263 – 1269, June 1999.

[2] Object Management Group, "CORBA/IIOP 2.2 Specification," 1998, http://www.omg.org/corba/corchpCB.htm

[3] INPRISE Corporation, "Installation and Administration Guide : VisiBroker for C++ 3.3," 1998, http://www.inprise.com/techpubs/books/vbcpp/vbcpp33/index640x480.html

[4] INPRISE Corporation, "Programmer's Guide VisiBroker for C++ 3.3," 1998, http://www.inprise.com/techpubs/books/vbcpp/vbcpp33/index640x480.html

[5] INPRISE Corporation, "Reference : VisiBroker for C++ 3.3," 1998, http://www.inprise.com/techpubs/books/vbcpp/vbcpp33/index640x480.html

[6] 周瑞, ActiveX / OLE / COM 程式設計, 1997

[7] 黃俊翔, ActiveX / OLE 技術手冊, 1997

[8] 梁德容, 羅文聰, 袁賢銘, "CORBA2. 標準簡介及展望,"

[9] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, *Addison Wesley Longman, Inc.* USA, 1999.

[10] M.R. Cantor, Object-Oriented Project Management with UML, Wiley, Inc. 1998.

[11] B.P. Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems, *Addison Wesley Longman, Inc.* USA, 1998.

[12] M. Fowler and K. Scott, UML Distilled: Applying the Standard Object Oriented Modeling Language, *Addison Wesley Longman, Inc.* USA, 1997.

[12] D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," *ACM Trans. On Software Engineering and Methodology*, vol. 5, no. 4, pp. 293-333, Oct. 1996.

[14] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, Object-Oriented Software Engineering: Use Case Driven Approach, *Addison Wesley Longman, Inc.* USA, 1992.

[15] W.J. Lee, S.D. Cha, and Y.R. Kwon, "Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering," *IEEE Trans. on Software Engineering*, vol. 24, no. 12, pp. 1115 -1130, Dec. 1998.

[16] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, Object-Oriented Modeling and Design, *Prentice-Hall International, Inc.* New Jersey, 1991.