具開放性架構的控制器軟體元件設計概念

工研院機械所＼許明景

關鍵詞

開放性架構　Open Architecture
軟體元件　Software Component
物件導向　Object Oriented

摘要

透過標準化軟體元件的力量，開放性架構有能力整合不同的應用案例，其主要的關鍵在於將一些共同的功能分解成獨立的單元，並定義成標準的軟體元件（Component）。而模組即是由一群軟體元件所集合而成，模組化的能力造就”隨插即用”（Plug-and play）的可能性，我們可以僅單純地替換一個元件或模組，卻不會影響到系統的其他部分。只要經過適當的設計，軟體元件或是模組就可以在不同的系統中重複使用。控制軟體元件化或模組化的能力可以讓工程師更容易且更有效率地從事設計與開發。

1. 前言

PC-Based 控制器順應著 PC 產業的發展，漸漸成為控制器產業中一股不可忽視的力量。但是在 PC-Based 控制器中，除了遵循標準的 PC 架構為平台之外，對於控制器所使用的硬體模組和軟體架構，是否已經有了可以共同遵循的標準？是否需要一個可供遵循的標準？這個問題似乎是值的我們考量的。

以運動控制中最常使用的驅動器和馬達為例，各廠商的控制介面、訊號定義、控制腳位以及接頭的型號都沒有統一的標準，各有各的規格，甚至同一廠商所生產的不同系列產品，也都存在著不同的差異性。例如以脈波輸入型或步進馬達驅動器為例，所接受的控制命令是差動（differential）型態的輸入脈波，而速度型或扭力型式的驅動器，則是以類比電壓作為控制命令的輸入訊號（例如+/- 10V 相對於+/-3000RPM）。以現今全數位化的馬達驅動器與運動控制軸卡而言，將控制命令經由 DAC 轉換至類比電壓，透過控制線傳送至驅動器，再由驅動器將類比電壓轉換回數位訊號作為控制命令的方式，無疑是一道不必要、也是影響系統整體性能表現的程序。

為何不將數位的控制命令直接傳送到數位式的驅動器，省卻中間這道類比和數位的轉換程序？

由德國機械工具協會、電氣製造協會與許多工具機、伺服器廠商針對數位運動控制所制訂，並於 1995 年獲的通過成為 IEC 61491 國際標準的 SERCOS（**SE**rial Real-time **CO**mmunication **S**ystem）或許可符合上述的需求。SERCOS 是目前國際上唯一成為 IEC 國際標準的運動控制專用網路。以下約略地簡述其特色：

1. SERCOS 是全數位化的網路：SERCOS 提供一個全數位化的介面，無須數位/類比轉換器，不僅節省數位/類比轉換器的成本，利用全數位式的特性，可以製作智慧型控制器。
2. SERCOS 是開放式架構：SERCOS 是一個由許多製造商共同制訂發展的架構，為一開放式架構，任何人都可以依據其架構設計自己所需的產品。
3. SERCOS 具有即插即用（Plug and Play）的特性：SERCOS 在制訂過程中就以即插即用的概念設計，將許多上下游間的協議明確地定義下來，以確保產品間即插即用的特性。
4. SERCOS 具有跨廠商相容的特性：SERCOS 制訂了一些標準的系統介面，只要依據此介面規格設計產品，就能夠具備跨廠商相容的特性。

對於驅動器的軟硬體控制介面來說，SERCOS 的確可以作為開放性的標準架構。但是對於控制器的系統軟體架構來說，卻沒有一個可供依循的標準。當我們可以任意替換 PC 週邊的滑鼠、鍵盤、螢幕、光碟機而並不會影響整個硬體結構與系統軟體的執行時，我們是否也可以更換不同廠牌的運動控制卡而不會影響系統的運作？是否可以替換不同公司的馬達驅動器而不需要更改機台的配線？

不可諱言的，以現在的產業情況，以上這兩個問題的答案在大部分的系統中仍然是否定的。我們是否可以提出一個解決的方案，除了讓以上兩個問題的答案成為肯定之外，最重要的，是讓整個控制器產業中無論軟體或硬體的架構有個可供遵循的標準介面。

2. 開放性架構的背景與優勢

大部分的 CNC(Computer Numerical Control)控制器和一些可程式邏輯控制的應用都會遭遇一種情況：就是應用系統的開發必須整合不同功能或不同廠商的產品。為了使用這些不同型態、不同公司的產品，不論是自行研讀技術文件或接受產品的使用訓練，開發人員都必須投入相當多的時間和精力，以熟悉不同產品的使用介面。如果每一次的系統開發都必須一切從頭開始「重新發明輪子」的話，那開發的效率與時程的掌握自然不高。相反的，在具有標準化、開放性的控制器架構中，由於模組間的介面一定，不同的模組可以更新或替換，也可以針對功能或性能上的需求重新規劃。因此只要針對某個模組作局部修改就可以產生不同的功能，或提供不同的性能層次，而整體的系統架構卻不會因為這樣的變動而改變。從使用者的角度來看，我們認為一個具有開放性、模組化的控制器架構，應包含下列的優點：

a. 允許系統整合者修改或更新所使用模組的能力，甚至可以替換整個模組。
b. 提供控制器整合其他智慧型裝置的能力
c. 增加 3rd-party 廠商針對軟體改進的空間。例如，以控制法則而言，如果 3rd-party 廠商發展性能更好的控制法則或是參數調整的機制，應可容易地取代原有的控制法則而不影響其他程式模組的運作。
d. 將使用者介面與控制的程式模組分開，讓資料的取得與呈現有更好的彈性。這樣就可使用 3rd-party 的人機介面取代原有的人機介面。允許在同一個程式架構下，提供不同的操作介面。

透過標準化軟體元件的力量，開放性架構有能力可整合不同的應用模組，其主要的關鍵在於將系統中共同的功能分解成獨立的單元，並定義成標準的軟體元件（Component）。

而功能相似的軟體元件就可結合成功能性的模組，而模組化的能力造就"隨插即用"（Plug-and play）的可能性，我們可以僅單純地替換一個元件或模組，卻不會影響到系統的其他部分。只要透過適當的規劃與設計，軟體元件就可在不同的系統中重複使用。這種只要將原有元件或模組經過些微修改或加入使用者特殊需求的功能之後就可重複使用的特性，對於系統的開發是非常有幫助的。工程師可以利用元件化的概念快速地設計系統，同時又可達成不同功能的特殊需求。總而言之，控制軟體元件化或模組化的能力讓工程師可以更容易且有效率地從事應用系統的設計開發與整合，同時這種以元件為基礎的技術思考邏輯，也讓系統架構明確，複雜度降低。

這種利用元件可重用性所的到的效率是非常可觀的，根據 QSM Associates 發表在 1996 年 8 月 Software Magazine 中的調查發現，當高度重用性的經驗實現時，產品開發上會有 900%的增加，開發時程將減少 70%，並同時減少 84%的成本。

另一個元件化的好處在於可以在整合開發環境(IDE)中以圖形化方式進行系統設計。如 Figure 1 所示，就如同硬體設計一樣，軟體元件之間透過拉線的方式將介面相互連接。

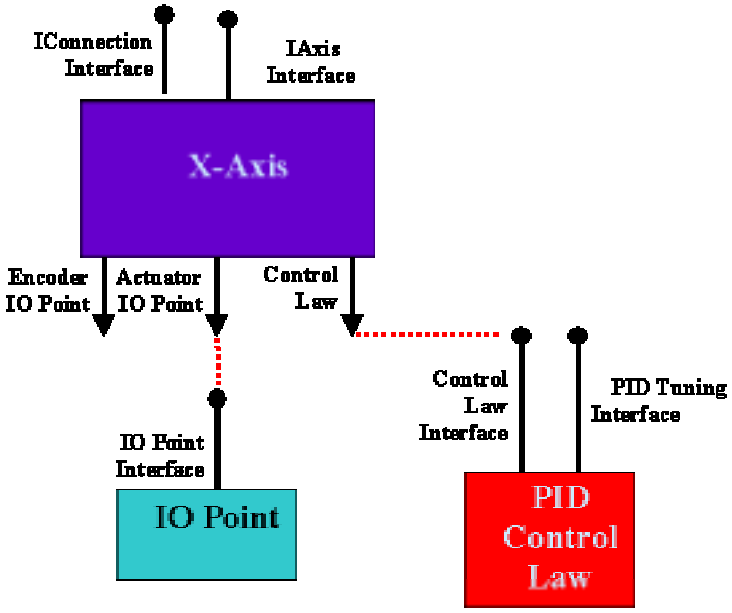

Figure 1: Component wiring

3. 開放性架構的參考模型(Reference model)

為了達到開放性架構所闡述的目標，有必要為所謂的「開放性架構」提出一個更詳盡的描述，並訂定開放性架構的**應用程式介面(API: Application Programming Interface)**。接下來，我們由下列不同層次的抽象結構來描述開放性架構 API 的參考模型。

- 基礎類別（Foundation classes）
- 模組（Modules）
- 架構設計（Architecture design）
- 細部設定（Detailed Design）

Foundation classes 是將一般型態的控制器分解成不同的類別(classes)架構，形成控制器的類別階層。相似功能的 Foundation Classes 結合成模組(modules)，成為可以即插即用的元

件。一個控制器控制軟體的開發就在於結合不同的模組與 Foundation classes 的實作而來。一個控制器系統的設計可因此而分成兩個不同的階段，第一階段稱為架構設計(Architecture Design)，就是將系統分解成不同的功能模組，第二階段稱為細節設計(Detailed Design)，實際針對功能模組內個別元件的屬性（attributes）與執行函式（method）撰寫程式。在細部設計時就需要使用具有開放性架構的 API 來完成系統程式的開發。

3.1 Foundation Classes

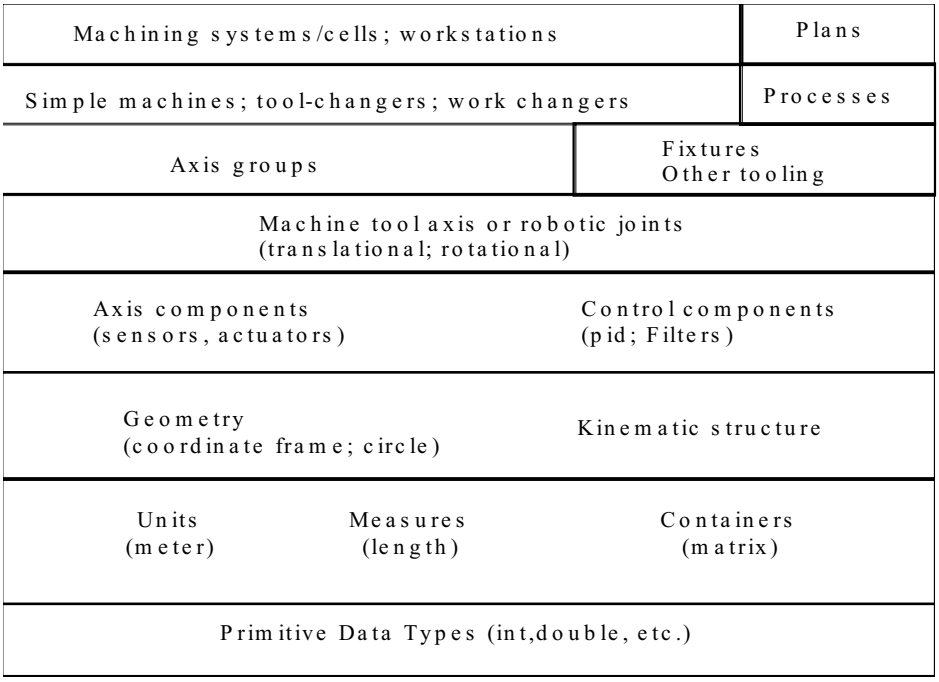

Figure 2: Controller Class Hierarchy

就是將控制器的軟體架構分解成不同層次的抽象類別，包括運動控制(Motion Control)與可程式邏輯控制(Programmable logic)。Figure 2 所描述的就是由控制器所分解的類別階層圖。越底層，Foundation classes 所定義的就越一般化，可以在不同的模組中使用，而越上層，Foundation classes 所定義的就越符合裝置（Device）的抽象層次，例如像感應器、致動器、或 PID 控制法則的等運動控制的元件。由於抽象化的結果，由 Foundation classes 所產生的軟體元件並不一定有相對應的實體裝置，像 Axis groups 就僅是邏輯上的個體(logical entities)。透過物件導向的繼承機制，我們可以根據系統功能的需求衍生出不同的**衍生類別**(derived classes)。開放性架構 API 就利用繼承的特性來控制不同層次的複雜度。

3.2 模組(Modules)

Figure 3 呈現一個控制器軟體中可能出現的模組定義，要構成一個模組，開放性架構 API 定義必須具備下列的特徵：

1. 必須集合功能相似的基礎類別或衍生類別
2. 功能定義完善的應用程式介面(API：Application Programming Interface)
3. 對於模組運作時，內部狀態以及狀態的轉移必須有明確的定義。

在控制器軟體的運作中，模組可能會靜態或動態地產生多個實體物件(instance)以處理不同的組態。舉例來說，如果一個系統有三個軸 X、Y、Z 和一個 Spindle，假設我們產生三個 Axis Group 的物件，ag1、ag2、ag3，分別規劃為：

ag1：x, y, z

ag2：spindle

ag3：x, y, z, spindle

當大部分的機械加工，運動控制與 spindle 較無直接相關時，使用 ag1，ag2。當 Spindle 與運動控制必須同步的時後，就使用 ag3。

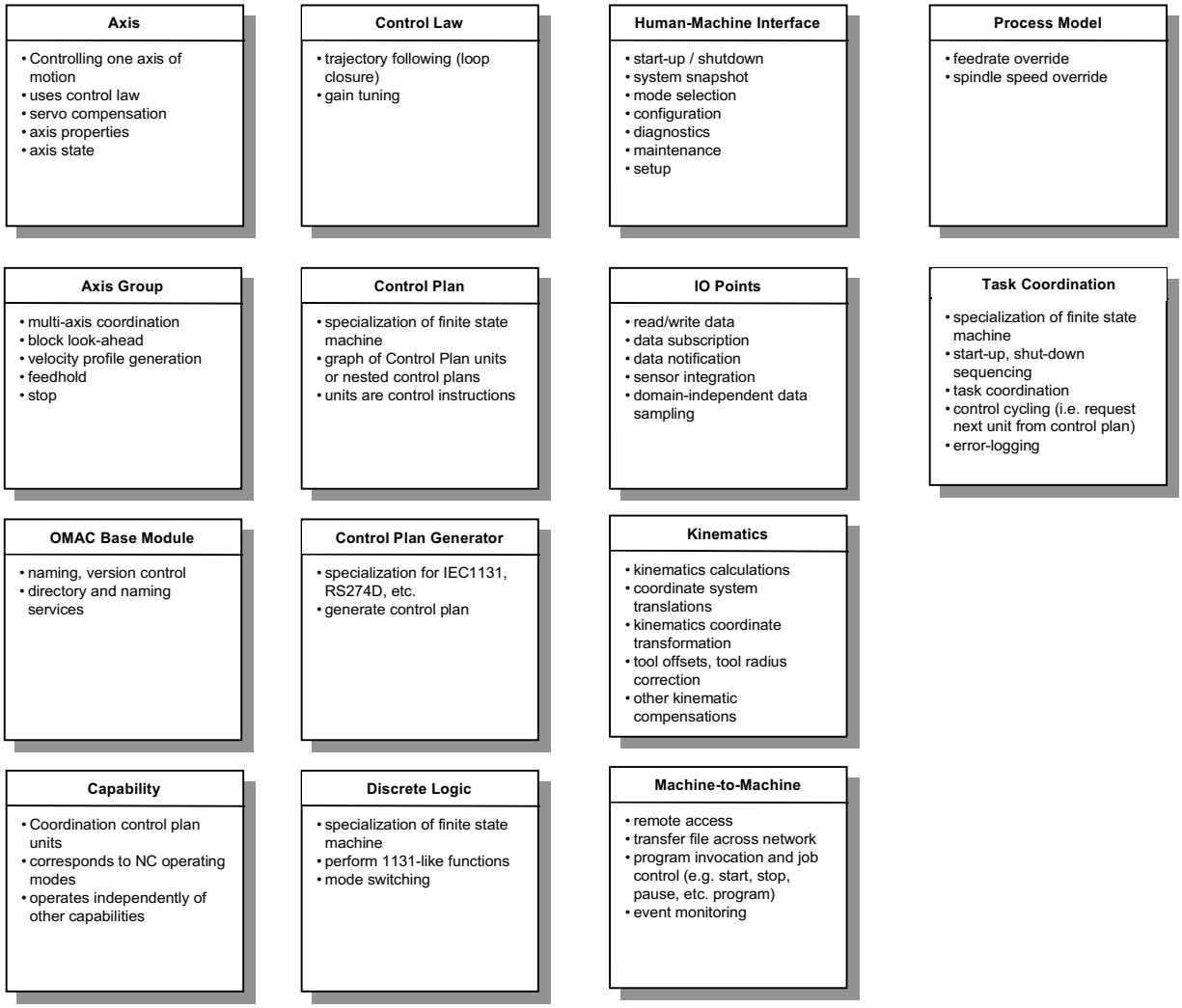

Figure 3: Modules

3.3 Architecture Design(架構設計)

接下來，我們以"單軸的手動控制"及"可程式邏輯控制"兩個實際的應用範例來說明如何利用開放性架構的模組來建構應用系統：

(1) One Axis Manual Control

在 CNC 控制器的發展過程中，手動控制是一項基本的功能，像 Jogging 和 Homing 就是最主要的代表。Figure 4 展示一個單軸系統的架構圖，由"AXIS" 模組結合"IO POINTS "與" Control law"模組，其中使用兩種控制法則，一種是位置迴路的 PID 控制，一種是速度迴路的 PID 控制。而 IO POINTS 模組中的組成可能包括 PWM 馬達驅動器，驅動器的 Enable 控制訊號與錯誤狀態訊號，AB Phase 型態的編碼器迴授訊號以及原點與極限開關等。

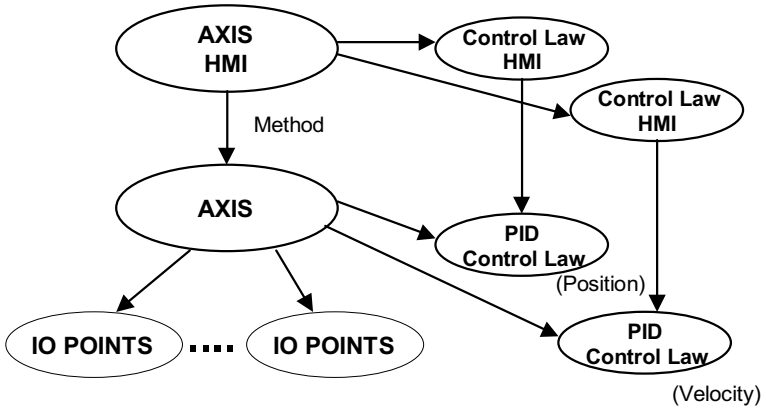

Figure 4: Simple, Single Axis, Jog/Home Only System

(2) Programmable logic

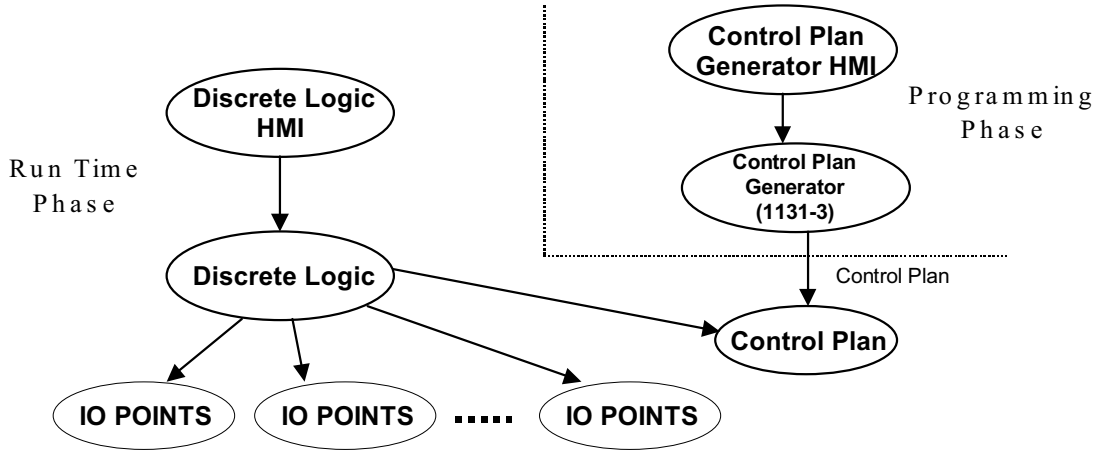

Figure 5: Programmable Logic Control

在 Figure 5 中顯示出兩個部分：Programming Phase 與 Runtime phase。

在 Programming Phase：

(a). 發展符合 IEC-1131 規範的程式，將 IO 輸出入裝置對應至不同的邏輯功能。

(b). 由解譯程式產生對應的邏輯控制單元。

在 runtime phase：

(a). “Discrete Logic”模組依照所指定的掃瞄頻率(Scan Rate)執行邏輯控制單元。
(b). HMI 或是 IO 所形成的 Client 端，產生不同的事件（event）通知”Discrete Logic” 模組執行相對應的工作。

4. 規格

要為不同的模組定義標準的 API 以符合控制器軟體元件的發展，在技術上，應採用下列的方式：

1. 使用物件導向的技術
2. 模組之間使用 Client / Server 的溝通模式
3. 使用 Proxy / Stub 的機制達到跨行程或跨機器平台的資料傳遞
4. 以 FSM（Finite State Machine）模型化控制的流程與狀態的轉移

4.1 API 的規格

定義標準的 API 對於系統複雜度的降低是非常有幫助的，但是為了因應不同開發環境的需求，例如不同的程式語言有不同的函式呼叫格式，因此 API 的定義必須要與開發的程式語言環境及使用平台無關。為了所定義的 API 可以跨平台，跨語言而不受影響，可採用 IDL(Interface Definition Language)語言來解決這個問題，它是一般性的介面描述語言，與使用的開發環境無關。在 IDL 中，介面的描述包含屬性(attributes)與操作函式(method)，並支援大部分的物件導向技術。如果在 Microsoft 的開發平台上，可利用 MIDL 編譯器可以將 IDL 中所描述的介面轉變為符合物件導向技術的 C++語言或非物件導向的 C 語言。IDL 經過 MIDL 編譯之後會產生 header file 及 proxy/stub 的檔案，可以直接使用在應用程式的開發之中。

4.2 物件導向技術

封裝，繼承，動態連結是物件導向技術中最重要的特性。開放性架構 API 使用物件導向的方法透過 class 來定義模組的 API。在類別定義中完整地封裝所擁有的資料與操作函式（method）。繼承的特性對於發展過程中資料的抽象化是非常有幫助的，具物件導向特性的類別可以從另外的類別中繼承它的資料與 method，這種被繼承的原有類別我們稱為基礎類別，而透過衍生的方式所的到的類別則稱為衍生類別。我們可以在衍生類別中加入自己的資料與操作函式，這樣除了可以建立符合自我需求的類別之外，對於基礎類別的重複使用也變的非常容易。

```
Class Axis
{
   virtual void setX(float x);
private:
   double myx;
}

application()
```

```
{
    Axis ax1;
    ax1.setX(10.0);
}
```

為了擴展基礎類別的功能，自 Axis 類別中衍生中 **myAxis** 類別，並在衍生類別中加入額外的 offset 資料。

```
class myAxis : public Axis
{
        virtual void setX(float x){ x= x + offset; Axis::setX(x); }
private:
        double myx;
        double offset; // set elsewhere for offset calculation
}

application()
{
        Axis ax1;
        myAxis ax2;
        double val=1.0;
        double offset =10.0;

        ax1.setX(val+offset);   //setX( )需明確指定 offset 值
        ax2.setX(val);          // 透過繼承的特性，將 offset 資料隱藏在 setX( )中
}
```

3.2.2 Specialization 特殊化、專門化

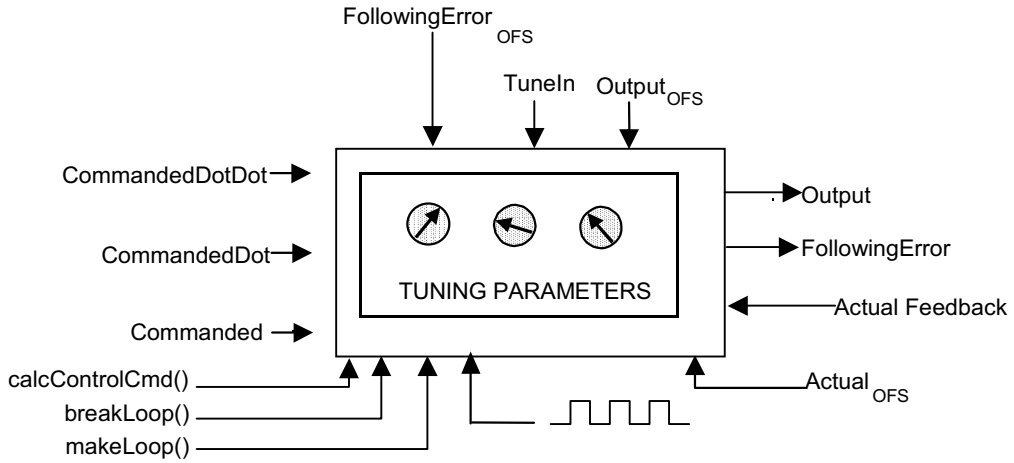

Figure 6: General Control Law

```
interface IcontrolLaw
{  // Parameters
  void setCommanded(double setpoint);
  double getCommanded();

  void setCommandedDot(double setpointdot);
  double getCommandedDot();

  void setCommandedDotDot(double setpointdotdot);
  double getCommandedDotDot();

  void setOutput(double value);
  double getOutput();
```

```
        void setFeedback(double actual);
        double getFeedback();

        void setFollowingError(double epsilon);
        double getFollowingError();

        // Offsets
        void setFollowingErrorOffset(double preoffset);
        double getFollowingErrorOffset();

        void setOutputOffset(double postoffset);
        double getOutputOffset();

        void setFeedbackOffset(double postoffset);
        double getFeedbackOffset();

        void setTuneIn(double value); // enable with breakLoop
        double getTuneIn();

          void breakLoop();
          void makeLoop();
          void calcControlCommand();

      };
```

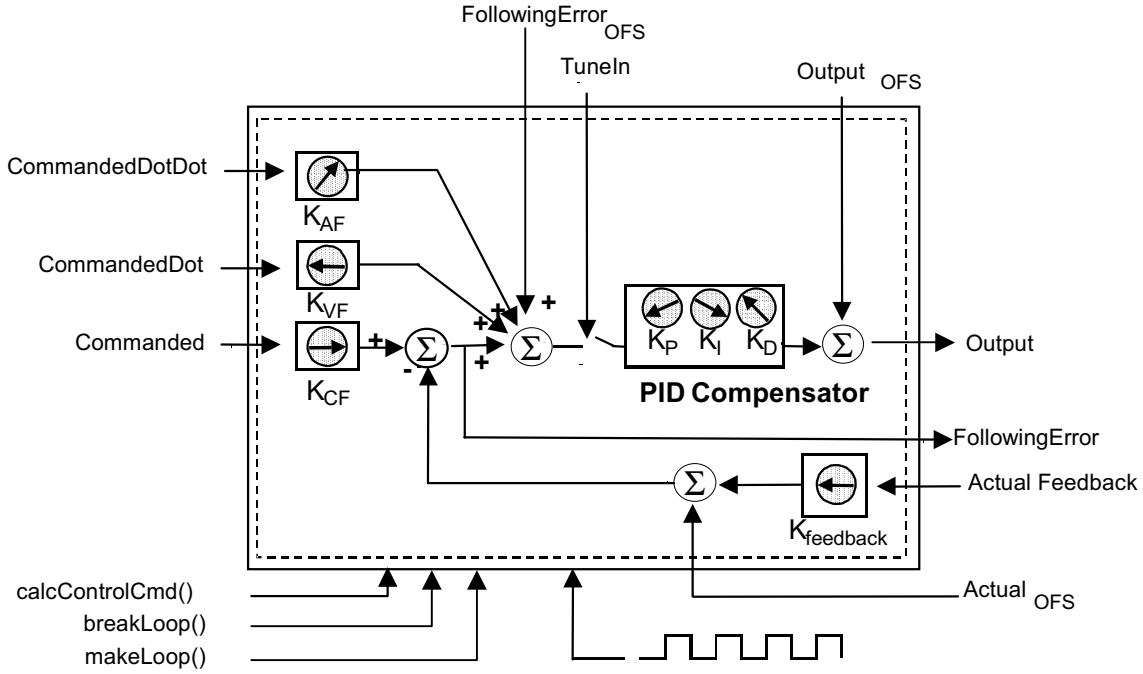

Figure 7: PID Control Law

```
        interface IPIDTuning: IcontrolLaw
      { // Attributes
        double getKp();
        double getKi();
        double getKd();

        void setKp(double val);
        void setKi(double val);
        void setKd(double val);

        double getKcommanded();
```

```
        double getKcommandedDot();
        double getKcommandedDotDot();
        double getKfeedback();

        void setKcommanded(double val);
        void setKcommandedDot(double val);
        void setKcommandedDotDot(double val);
        void setKfeedback(double val);
      };
```

IPIDTuning 介面是繼承 IcontrolLaw 而來。

4.3 client /server 行為模式

開放性架構 API 在元件及模組的溝通上採用 client /server 的架構。client 端發送訊息至 server 端，提出服務需求，server 端會產生相對應的回應。client 端的需求可能來自相同或不同的執行緒，下圖所示即為一個 server 擁有兩個來自不同行程（process）的 client 端，Axis Group 負責提供各軸的運動控制命令，而另一個週期性的 scheduling updater 程式可能位於不同的執行緒中執行，負責對 Axis Module 提供執行時序、同步化等控制程序。這個 scheduling updator 模組可能與部分的硬體裝置緊密的結合，例如計時器，以保證週期式的執行動作是正確的。

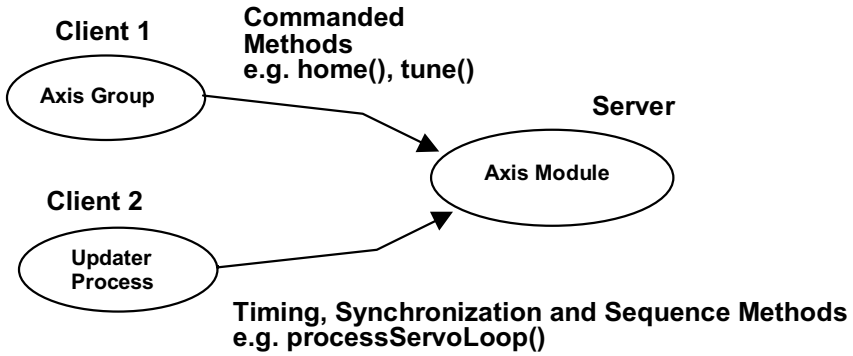

Figure 8 : Multiple Threads of Control

4.4 採用 proxy 技術

client/server 的相互作用可能發生在近端(Local)或是遠端(Remote)。所謂的近端，指的是 client 與 server 端的程式位於相同的行程（Process）中，由於具備相同的邏輯位址空間，資料的存取與函式的呼叫較為簡單且快速。如果 client 端與 server 端的程式分散於不同的行程上或是分處不同的機器，我們就稱此為遠端的 client/server 架構，這就需要 proxy agent 技術來作資料的包裝與轉換。這種 proxy agent 的機制可以透過不同通訊方式來達成，包括 network、share memory、message queue、serial line 等。

4.5 執行平台與開發環境

所謂的執行平台所提供的服務包括如計時器，中斷處理，行程內通訊。而作業系統則

提供各種軟硬體的服務，包括程式執行的控制，資源的配置，輸出/輸入裝置的處理，檔案管理等。而一個具有即時作業系統（Real-time Operation System），則必須額外提供像 semaphoring，具有優先權的先佔式多工排程的能力。開發工具就包括程式的編譯，連結和除錯工具。

在執行平台、作業系統、開發使用的程式語言與環境上，並無特別的限定。舉例來說，proxy agent 的技術，就可以使用由 Microsoft 發展的 DCOM（Distributed Common Object Model）或是在 Linux 平台上的 CORBA（Common Object Request Broker Architecture）技術來實作。

4.5 狀態(state)轉移的行為模式

模組的執行行為是由內部的 Finite State Machine(FSM)所控制。每個模組都有一個主要的 FSM 來描述模組運作時的行為模式，執行時會依據不同的狀況轉移到特定的狀態（state）。而模組間在執行的過程中所發生的 event 也可能造成狀態的轉移。Figure 9 表示模組中一般性的狀態轉移機制。

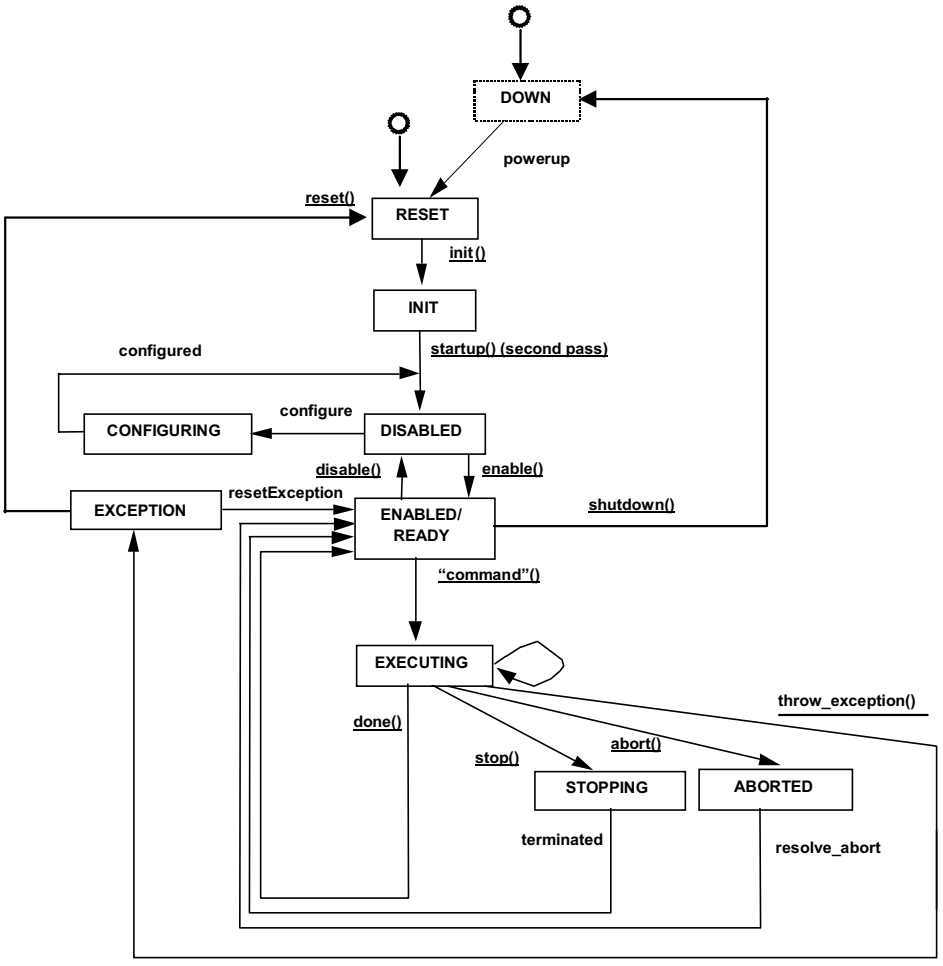

Figure 9: Generalized State Diagram

5. 模組概觀

5.1 Task Coordinator

負責初始化並決定何時啟動系統中的模組，The Task Coordinator is an FSM，.在整個控制系統中 Task Coordinator 扮演最上層 FSM 的角色。

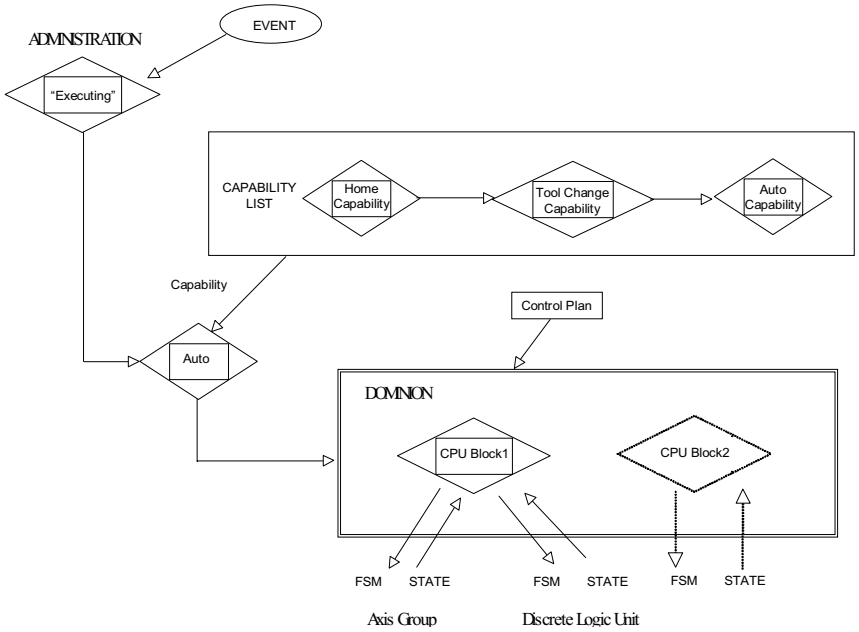

Figure 10 : Task Coordinator Computational Model

對一個控制器系統而言，可能有各式各樣的功能存在，當執行某一個功能時，就是負責反應來自操作者的需求。由上圖來看，在控制器電源開啟並啟動人機畫面時，可能置於 Manual 模式，當操作者按下 AUTO 的按鍵時，HMI 模組將執行 Manual 功能的 stop( ) 函式，並將 AUTO 功能從 Task Coordinator 的佇列（queue）中載入，當使用者此時執行加工程式載入時，程式將被載入到控制單元產生器（Control Unit Generator)中以解譯程式，按下「開始」鍵，就引起 state machine 動作並開始執行程式的解譯以產生欲執行的控制單元（Control Unit）。

5.2 Programmable logic

Programmable logic 模組和 Task Coordinator 模組相同，也是透過 FSM 的機制來統籌邏輯控制單元的執行。一般來說，Programmable logic 的 FSM 可以將任何以 IEC-1131 語言寫成的 Soft-PLC 程式解譯成可執行的邏輯控制單元。

5.3 Axis

負責處理每一個運動軸的伺服控制，搭配適當的 IO 裝置或控制法則，執行閉迴路或開迴路的位置、速度或扭力控制。

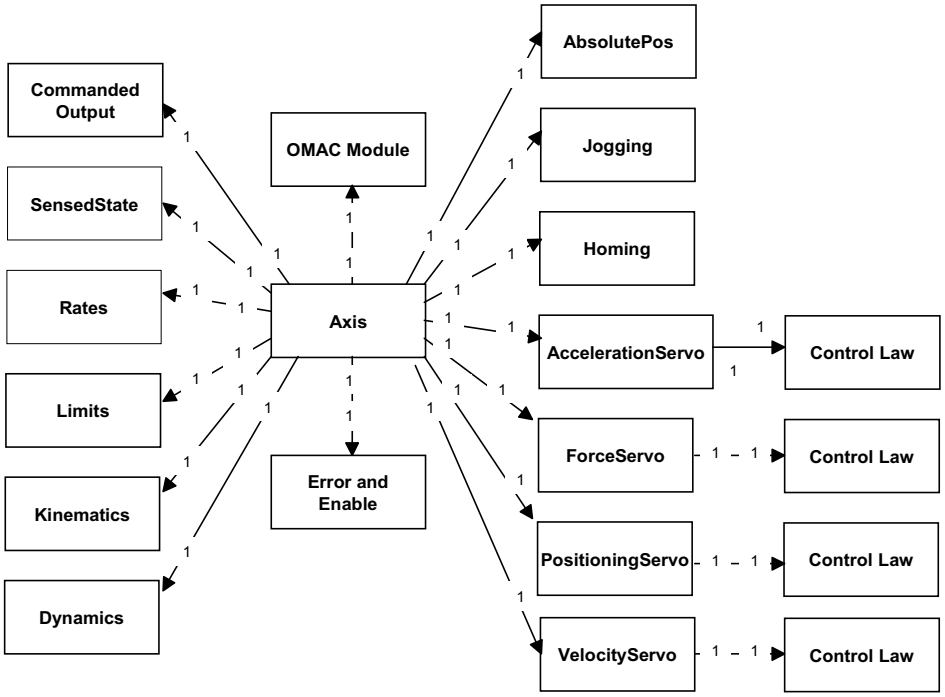

Figure 11: Axis Class Diagram

5.4 Axis Group

Axis Group 模組負責將 G-M Code 所定義的運動命令解譯成各控制軸以時間單位為主的伺服控制命令。

5.5 Kinematics

負責控制系統中運動學與反運動學的計算，座標的轉換。同時也產生 offset 和補償值。

5.6 IO System

IO 模組的主要目的是為系統中實際或虛擬的輸出入裝置提供一致的介面，以隱藏實際 IO 硬體的動作。以系統中的 DAC(Digital to Analog)為例，假設我們定義它是一個廣義的 IO 輸出裝置，IO 的基礎類別將提供一個一般性的讀取與寫入的介面，包含 ReadValue( )與 WriteValue( )函式。

5.7 Control Plan Generator

控制單元產生器負責載入加工程式並解譯，產生獨立的控制單元，如 Line，Arc，Circle 等。有可能一次就解譯整個加工程式或每個時間僅執行一個加工的指令。

5.8 Human Machine Interface

人機介面模組負責連接控制器與人機監控系統。

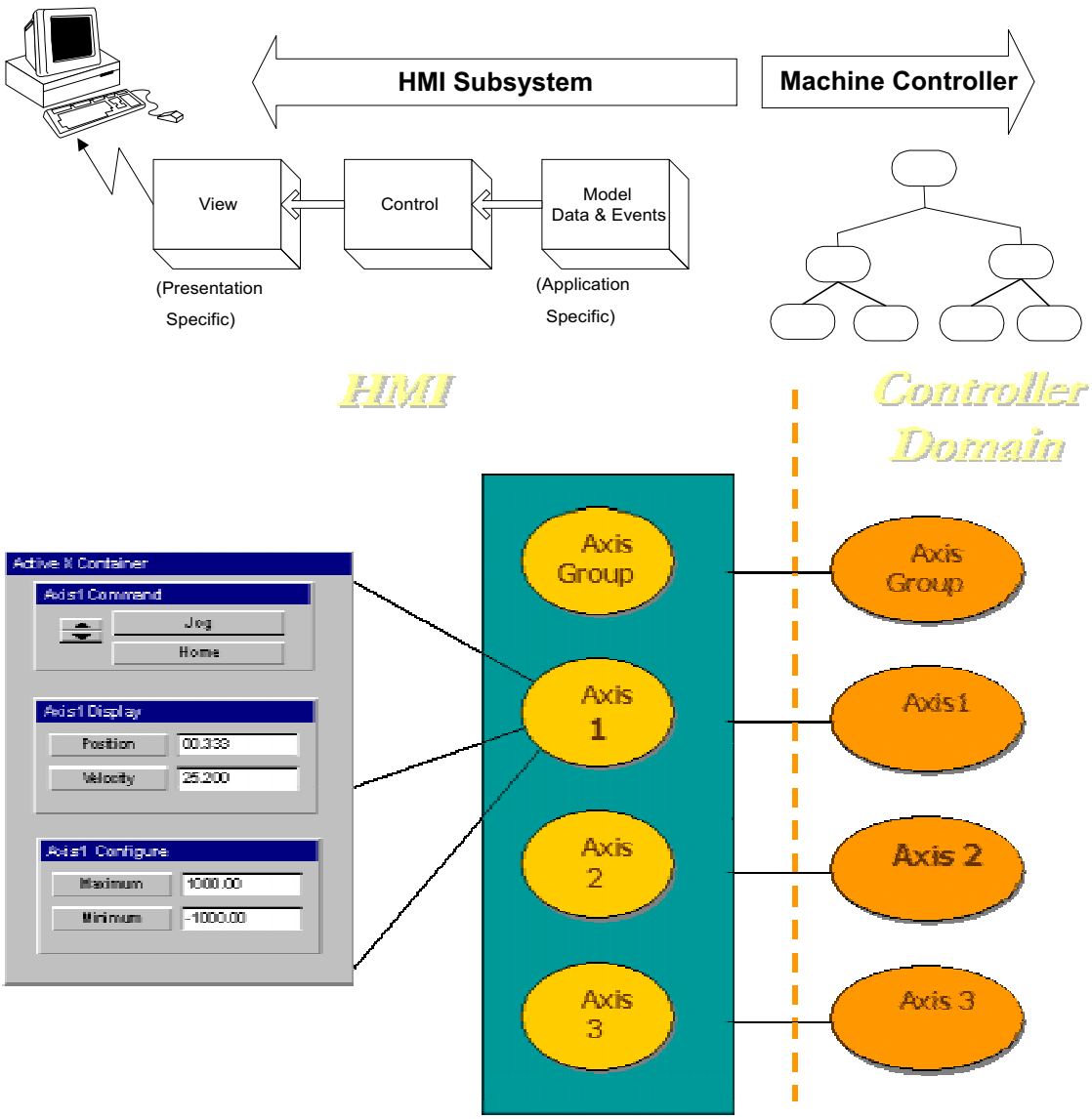

Figure 12: Human Machine Interface

5.9 Machine To Machine Interface

為了整合各種不同的裝置，例如 PCL，NC 等，開放性架構 API 定義控制器之間的通訊介面，提供遠端監視、診斷與控制的能力。

6. 結論

觀察ＰＣ產業一路走來的歷史，我們可以深刻地了解，一個架構開放的系統，因為有了標準化的規範，除了使用者在系統建構時的便利性增加之外，也由於開放的因素，生產廠商有明確的規格可供依循，並且在技術與人才都較容易取的的情況下，投入的意願也相對提高。而使用者也會因為廠商的積極投入、標準化產品的出現，而免除對單一廠商、單一產品的依賴與風險。因此，如同 PC 產業一般，一個開放性架構的系統也無形中擴大了產業的利基與規模。

如同前面的章節所敘述的，開放性架構 API 的目標在於定義標準化的控制軟體元件，

讓系統的架構更為結構化。雖然不同的廠商對所謂開放性的定義不盡相同，但是為了實現產品隨插即用的理想(plug-and-play)，開放性無疑是該努力的方向也是未來的趨勢，雖然這僅是邁向 plug-and-play 理想的第一步而已。要定義一個理想的開放性架構 API 的確是不容易的。要成功的技術關鍵在於：

1. 像 PC 硬體產業那樣，即插即用的軟體元件技術
2. 最大的設計與元件重用性，以降低成本
3. 提供一種可以利用相似的開發模式來處理不用應用系統的彈性

我們期待開放性架構 API 可以改善今日工業自動化軟體的開發架構，更能因應軟、硬體因為逐步地開放與整合所帶來系統複雜度的提高的挑戰。