Java 應用於 PC-Based 控制器之探討

黃申在
國立屏東科技大學資訊管理研究所
sthhuang@mail.npust.edu.tw

摘要

Java被視為PC-Based控制器系統及應用軟體邁向物件化、元件化並強化網路功能與跨平台特性的重要解決方案,然而以嵌入式即時系統的角度來看,仍有許多欠缺之處,而改善方法則呈現多元競逐的局面;本文試圖以評論式的綜覽觀點,呈現個中思維脈絡,提供各取所需的組合式方案之選擇與定位參考。

關鍵字: Java, PC-Based控制器, 嵌入式系統, 即時系統,物件導向
Keywords: Java, PC-Based Controller, Embedded System, Real Time System,Object Oriented

一、前言

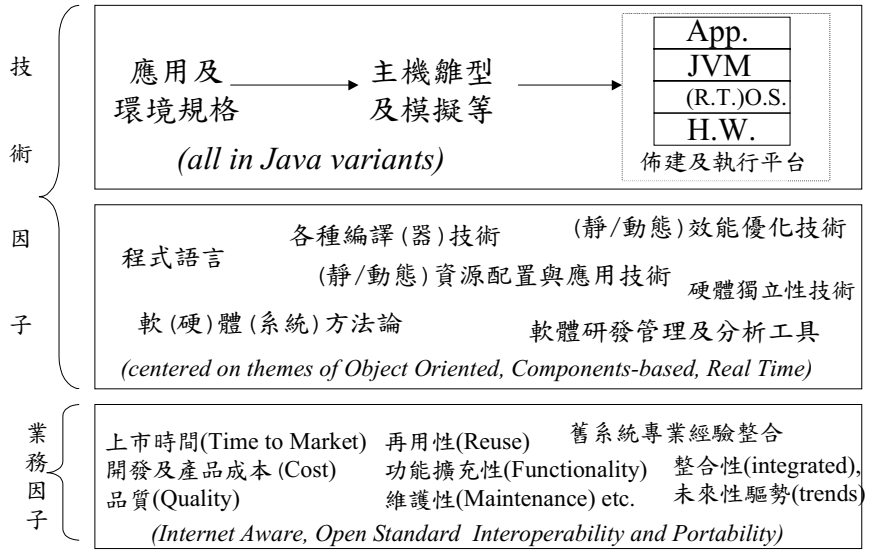
隨著 32 位元微電腦成本之降低,及軟體在產品與系統所佔比例漸增、彈性化及可重組性等非功能性需求的增加、通信能力與網路化監控能力之必要性等,在製造業精密儀器、自動化設備、程監控程序、遠端維護、虛擬廠房、企業整合等問題上扮演前線地位的 PLC 控制器無疑地是關鍵性的一環; 然而這些需求與趨勢已非傳統的 8 位元或 16 位元微控制器可負擔,而需以具有管控硬體及提供軟體開發與執行能力之作業系統的 PC-Based 控制器為之。隨著系統軟體開發成本之增加、上市時間縮短、品質與持續操作性、穩定性及可維護性等, 物件技術被視為 PC-Based 控制器軟體的趨勢[14,15,17]; 而隨著網際網路的風行草偃,網路能力、跨平台、標準化互通能力等,也不免成為 PC-Based 控制器軟體企慾的理想。不管是應備功能、非功能需求、趨勢或者理想, 總要有落實及展現的舞台及工具; 在舊有的 C/C++及組合語言無法完整有效地滿足這些需求與考量的狀況下,只有尋求新的技術解決方案。Java,原本著眼於(嵌入式)消費性電子產品的物件導向程式語言及平台, 卻在以桌上型電腦為基礎所聯結的網際網路上,以全球資訊網應用而發跡,更進一步多元擴展至商業伺服器及資訊家電等領域,成為網際網路程式設計之語言與平台, 也回頭成為 PC-Based(或嵌入式)控制器軟體開發及執行平台所期待的解答[10,20,22]。

由於 Java 原先就沒有支援即時性之設計, 且桌上型電腦應用之環境也不同於嵌入式之有限資源環境,因此樂觀支持擁抱者之外,亦不乏懷疑論者[9],當然也有折衷混成論者;總之,在潛在利基誘因及技術觀點競爭下,呈現百家爭鳴、多元發展的局面[6,7,21,23]。本文透過文獻的研習彙整,配合解析評論式的綜覽觀點及要點式的量測, 試圖呈現個中思維脈絡,並分享對 Java 應用於 PC-Based 控制

器相關議題的一些初淺心得與看法,限於篇幅,特別著眼於剛即時性(Hard Real Time)能力的面向。

二、問題綜覽

Java 是從 C 和 C++所簡化衍生而來的平行式物件導向程式語言，希望改善 C 和 C++的缺失，刪除了 C/C++功能如 enumerated constants、指標、傳統函數、結構、union、多重繼承、goto 敘述、運算子超載、preprocessor directive 等複雜、易錯、或不安全的部份;而因應物件導向語言特性之物件實體動態記憶需求、動態配置及多型動態鍊結,及隔離使用者對記憶體管理細節之涉入,對記憶體資源採自動回收方式;此外,由於設定為網際網路運算平台,因此跨平台特性、架構中性化、網路支援、資料庫取用支援、安全機制等皆因此成為內建類別封裝介面式功能,其中跨平台與架構中性化之考量, 使得 Java 以虛擬碼(bytecode)形式傳送並須透過虛擬機器以直譯方式將虛擬碼轉成機器指令，再予以執行,並以開放標準化透通地取用主機作業平台的各項系統資源及各 Java 特性功能套件之實作機制。而這些彈性與特色, 雖然可透過良好的語言設計及程式設計方法(如物件導向技術與方法)及其編譯器最佳化技術與執行環境系統實作技巧達成, 然而額外代價如較多的系統資源或較長的反應時間等。無疑的這是一種權衡, 而權衡的基準則視整體目標而定, 如關鍵技術、執行效率、軟體生產力、市場普及度等。了解這類的權衡與整體目標考量, 有助於理解與討論技術的焦點、改善或整合

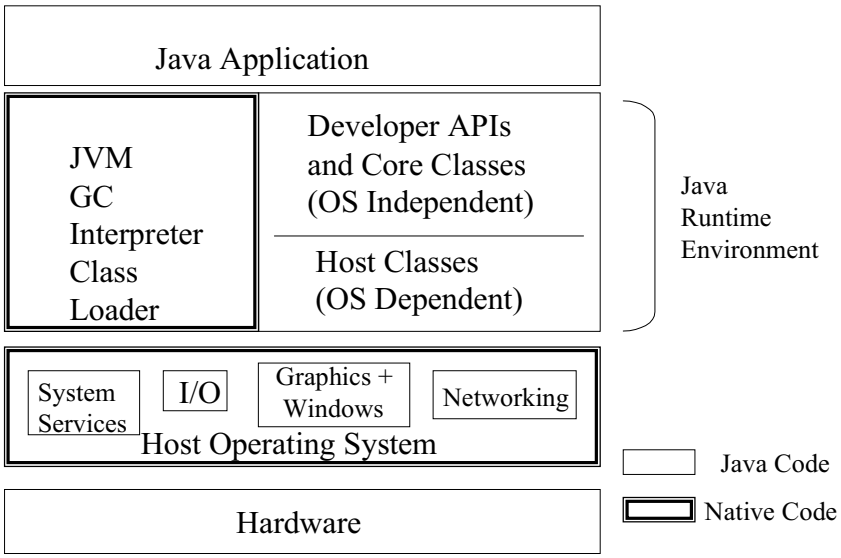


圖一 軟硬同體之嵌入式系統與即時性應用研發各考慮因素綜覽

的方向、各解決方案定位與比較等問題。因此,我們以兼顧業務及技術因子、考量研發工作之一般程序與產品及使用的技術與工具等的視野,以圖一簡單彙整這樣的思維架構,將之置於嵌入式系統及即時性應用之情境,而將 Java 應用於 PC-Based 控制器系統視為此情境之個案,圖二則為主機系統 JDK 佈建及執行平台的

一般架構圖。逐一解說各項細目將耗用篇幅亦非本文主題,在此僅就與後續討論有關的幾點綜合性問題略述:

首先,圖一突顯的是完整解決方案(Total Solution)及整體成本(Total Cost)的觀點,涉及策略組合式而非單點決定式; 舉列言之,現有的 C 及 16 位元在技術因子的平台執行效率上,明顯優於 Java 方案, 但我們不會選擇該方案, 反過來, 會因



圖二 主機作業系統式JDK平台架構

業務因子及其它技術因子等考量, 而致力於 Java 佈建及執行平台功能及效率的提升, 而目標也不會是效率最佳化的追求,而是以總體符合度或階段性為基準,而且使用的方法,也不會只鎖定執行平台內部(更何況具有多種組合可能),而會輔以如編譯技術(如虛擬碼優化(Bytecode optimization))或製程方式調整(如 Java 開發但以 C 及原生碼執行,即交互編譯(cross compilation)型式)等;此外目前廠商多致力於柔即時性應用的嵌入式系統技術,技術因子的剛即時性難度固是原因,但預期市場及其現階段應用需求,柔即時性即可滿足等業務因子才是主導因素。

另外,就傳統控制器及 PC-Based 控制器本質來看,圖二平台架構中,其實只要上層應用程式及下層硬體(及其驅動程式)即可,中間的作業系統及 Java 平台皆是額外的負荷,最多以 MS-DOS 為之即可;的確, 如果控制對象只是一般的感應器(Sensor)及致動器(Actuator)單元,不是簡化成 DI/DO,AI/AO 搭配事件驅動之中斷機制及取樣頻率即可?這樣的觀點是以控制對象的各類(電、化、光、物理、機械等)週邊硬體單元及終極使用者角度而言,若觀點移往控制策略主體的系統軟體(及其研發)與高階精密設備整體控制(多個高取樣頻率及反應需求), 則如何善用低成本 32 位元微處理器之強大的 CPU 及有限記憶體資源, 超越週邊之個別物理限制,整合地控制週邊硬體及提供給應用程式(及其開發工作)更多的計算與控制能量,才是 PC-Based 控制器存在的意義所在。這樣的角色即是資訊科技領域內的作業系統範疇,嵌入式即時作業系統是針對無硬碟與有限記憶體的環境需求的特殊設計,虛擬碼與虛擬機器,可視為與原生碼(Native codes)在跨平台、可執行

位元相容性(Binary Compatibility)及效率與空間折衝(Java 原始碼平均對應於1.8byte 長的 Bytecode 遠小於一般 C 原始碼對應之原生碼,在記憶體佔用空間及傳輸下載時較優)等構面的邏輯選擇項。在實作層面, 我們尚有如結合硬體(如picoJava)、省略平台作業系統(如 JavaOS)、融合虛擬機器與即時作業系統(JVM=RTOS)、Java/C 混成式(如 PersonalJava[1]應用環境(PJAE)架構)等等技術組合的選擇。作業系統對 CPU 資源運用上,由於 CPU 操作速率遠高於 I/O 硬體,而一般應用(如控制軟體)皆是計算與輸出入錯雜,利用計時器及 CPU 中斷機制將 CPU 操作時段分割(Time-sliced),按特定排程法則,依序分配給個別的控制單元並在其取樣頻率與反應時間限制內完成個別控制工作,藉由 CPU 與 I/O 硬體操作在時軸上的同時性重疊,單一 CPU 資源可抽象化成多執行緒,分時同時運作多個控制硬體,不受限於單一控制器之物理限制。這項時間驅動的時間分割式多執行緒是現代 PC-Based 控制器作業系統必要的致能機制,而執行緒為 Java 語言的內建物件,其功能及執行效率無疑地影響 Java 語言及平台支援即時性的能力,當然也是研發改善的主要焦點之一[2,8,11]。

三、Java 支援 Real Time 之問題

Java 技術含括兩部份: Java 程式語言及 Java 工作平台;後者包括虛擬機器(JVM)及應用程式介面(Java API)等(如圖二), 這些介面皆類別化並依相關性予以群組套件化,可針對不同應用領域特性及需求,依循開放性及互通性標準,或封裝平台相關及領域相關之細項,進行模組化組裝或擴充。Java 程式語言為一高階語言, 具備簡單(Simple)、架構中立(Architectural neutral)、物件導向(Object oriented)、可攜性(Portable)、分散式(Distributed)、高效能(High performance)、直譯式(Interpreted)、多執行緒(Multithreaded)、豐富性(Robust)、動態式(Dynamic)、安全性(Secure)等特性。

如圖二綜覽之觀點所示, 程式語言與執行平台並不需有絕對的連動性,理論上,只要程式語言原始碼編譯後是標準的 Java 虛擬碼,而執行平台的啟動輸入的也是 Java 虛擬碼, 則二者是可視為以虛擬碼為唯一交換介面的兩獨立模組。實際的情況當然無法如此簡化,因此,Java 應用於 PC-Based 控制器之即時性支援的問題, 也應分 Java 程式語言及 Java 平台兩方面來看。這樣的觀點也突顯我們認為, Java 於即時系統與其應用,嵌入式與否,特殊領域如 PC-Based 控制器或其一般性應用與否, 並不能只考慮執行平台的效率與即時性而已[7,16,19]。

3.1. Java 語言支援 Real Time 之問題

Java 語言是從 C 和 C++所簡化衍生而來的平行式物件導向程式語言，希望改善 C 和 C++的缺失，透過良好的語言設計、虛擬機器規格設計及編譯器實作,具有簡單及多緒的語言特性、物件導向之軟體模組化與可再用性、可攜性與架構中立所產生之跨平台特性加上開放性標準之網際網路應用程式介面,還有軟體生產力之提升與廣大的市場普及性,使得傳統控制器應用研發人員與廠商對 Java

技術於 PC-based 控制器及自動化製造應用上有很高的期許[10,17,22]。

然而就如 Fortran 專長於科學運算, COBOL 善長於資料處理的商業運算, Mathematica 專用於數學代數解析,而 C 被認為是系統程式的高階語言,Java 的動機雖是嵌入式系統,發展上卻是標定為網際網路程式設計語言。嵌入式環境如 PC-Based 控制器系統及應用對硬體、CPU 及有限記憶體資源運用之要求高,屬性上為具即時性之系統軟體,因此,在語用(Pragmatics)及本體論(Ontology)層次是有匹配阻抗落差的;此外, Java 語法上雖有多執行緒,卻無時間性或資源控制指令,也無法表達工作單位的資源需求及資源使用狀況[7,9,11],沒有特殊輔助工具時,只能完全仰賴編譯器及虛擬機器。

在[22]彙整了 Java, C, C++及組合語言在嵌入式系統軟體的優劣處:

	優點	劣處
Java	高度可攜性,物件導向,易於了解,網路功能強,與網際網路緊密整合,瀏覽器內可用	未支援即時程式設計,開發工具最少,語言仍不成熟,非完全的"Write Once, Run Anywhere"
組合語言	以最小最緻密的軟體對事件及時間等提供最精確控制	難寫難改難解讀,開發週期最長,不具可攜性,開發工具亦少, 通信機制最差
C	標準高階開發語言,很多開發工具,可具可攜性,通信機制佳	可能會如組合語言般難瞭解,非皆可攜
C++	具物件導向但未強迫,可具可攜性,通信機制佳,開發工具佳	只具部份物件導向特性,不必然可攜,可能寫得比 C 更難瞭解

Java 語言雖未直接支援即時系統程式,不過其強型態(Strong typing) 、明確的語意、以物件為單位之記憶體管理及簡短的虛擬碼等,使得在編譯優化技術(如 bytecode optimization, constant propagation, strength reduction(如以移位替換乘法),loop unrolling, in-lining 等)[3,4,23]有更多發揮處, 此外一般作業系統為保護位址空間及程序管理等所加的負載亦可省卻,這些特性雖已有初步研發成果,如 PicoJava 及 JavaOS 等,其實際效果與市場接受度猶待考驗。另外,即時功能與其程式界面,亦可能物件套件的方式(如同 C 的程式庫)加入或覆蓋之 Java Core API 中(Real Time Java API 的標準化是美國國家技術標準局(NIST)正致力推動中),當然,其即時效果的發揮仍要視 Java 平台支援 Real Time 的成效而定,也是下一小節之主題。

3.2. Java 平台支援 Real Time 之問題

即時系統及應用,不管是剛性或柔性,在定義上不只要求程式邏輯與功能規格之正確性,而且該正確性具有時間特質,亦即功能執行時必需在要求時程規範內完成方可,因此對系統資源運用的效能及系統單元反應時間之控制皆須有相當的可預測性及區間限制[19]。這些問題在如 PLC 等嵌入式系統上,由於 CPU 等級更低、記憶體容量較低及缺乏硬碟等限制，加上自動化控制功能需求,更顯跼蹐。

狹義而言, 一般探討 Java 即時性問題,指的就是 Java 平台,特別是 JVM 的部份,不過以圖二所示,Core Classes (如 Java.lang.*) API 實作的方式亦會影響執行效

率。請注意, Java 平台雖有一序列版本: JavaCard, EmbeddedJava, PersonalJava, Java, Enterprise Java 等,JDK 也從 1.0 進展到 2.x,其差別多在於所界定的 Core 及 Standard Classes 範圍之差異, JVM 技術本身則只在 HotSpot[2,3,4]出現才有較大的變動; 故此處的分析固然以嵌入式系統之即時性為主, 大部份亦可適用於其它版本的 Java 平台。

Java 平台應用於如 PC_based 控制器之即時性嵌入式系統的障礙,簡言之,即是太胖、太慢、無法時控(或缺乏可預測性)及可排程性四大問題的交互影響。太胖是記憶體資源運用及 Core Classes 減肥塑身選擇的問題,太慢即一般所謂 JVM 效率問題[5],是 CPU 如何扮演 JVM 執行虛擬碼的問題,亦即 CPU 資源與執行效率問題; 執行速度慢尚可,但若無法估算或控制最壞的情況(而非平均狀況), 便無法保證剛即時性;最後,即時性質乃需要耗用 CPU 及記憶體資源方可達成,即使有辦法估算及控制某一段工作於最壞情況下所需耗用的資源,在資源有限之前題下,是否可以滿足所有工作,包括周期性、非週期性及不可預期事件?此即可排程性問題[12,13],此問題原則上並非 Java 語言及平台獨有的問題,而是任何即時系統設計及即時性分析的共通理論與實務問題[19],即使以組合語言實作或 PLC 亦然。

在記憶體資源運用方面:

首先靜態上,Java 執行平台及輔助套件等所佔的記憶體空間過大, 在沒有硬碟及虛擬記憶體機制或有限實體記憶體空間的系統環境是不可行的; 動態方面,Java 虛擬機器內部的資料分為四大部份:類別定義區、Java 堆疊區、物件雜堆(Object Heap)及原生方法堆疊區; 後三者皆為動態變動的,然而 Java 語言的設計(包括無指標及無 malloc()功能)使即時程式設計者完全無法控制其配置,無法得知可用數量,無法控制記憶體斷裂(fragmentation)[7];Core Class API 實作中有過多又無法重複使用物件雜堆配置, 如 Socket 物件及 FileInputStream 等[18]; 而回收為自動式的垃圾收集,固然免除程式設計師之負荷,然而其實作方式(不定時、低優先序、全鎖定式)所造成執行時間之不可預測性及記憶體資源運用之低度可控制性,為 Java 即時性質的首要缺陷。

回收機制與記憶體運用間關聯性可簡單估算如下[7]:假定總可用記憶體為 M bytes,完整(Stop-and-Wait)之記憶體回收需要耗用 CPU 時間為 S 秒,所有的即時性工作所需的記憶體為 U bytes, 總記憶體配置速率為每秒 V bytes,而 R 代表用於記憶體回收所耗用 CPU 時間比率; 分析連續兩次記憶體回收間記憶體運用狀況, 在最差情境下, 要支援這些即時性工作負載所需之總記憶體 M 需大於 U + 2V(S/R); 換句話說, 耗用於記憶體回收之 CPU 時間之最小比率 R 之下限為 2VS/(M – U),與記憶體配置耗用速率 V 成正比,與可用及所需記憶體之差額(M-U)成反比。據此分析,沒有即時的回收機制時,程式設計上可行的策略是降低 V 值, 即以靜態配置取代動態配置,而其效果反應於 JVM 上,就是多運用類別定義區,少用其它三區。

在 CPU 運用及執行效率方面:

Java 虛擬機器為堆疊導向架構(Stack-based),與現有 CPU 架構之記錄器導向(register-based)是另一高匹配阻抗處,也間接造成虛擬碼對應到機器碼的困難,也較無從運用硬體的特性,如 register sets, pipelining, caching 等;反之,優化動作全交給編譯器及虛擬機器,在 Java 強型態及明確語意下,相關技術的推進(如 bytecod 優化及動態編譯等)將比程式員手工調製好。根據 Sun Micro.自己的統計[3],JVM 執行典型 Java 應用程式時,平均而言,記憶體配置與回收、執行緒同步、原生方法及解譯迴圈等分別佔 20%、19%、1%及 60%之執行時間。因此解譯迴圈效能之改善最有助於縮短虛擬碼的執行時間,常用的改善技術如 Just-In-Time Compiling、動態編譯(Dynamic Compilation)、虛擬碼優化、效能側寫以分析執行樣版等。而針對記憶體回收及執行緒同步,則採世代式回收演算法(generational garbage collection)並增強即時回應性及快速原生執行緒,如 HotSpot[2,3,4]所採行。另外,增加原生碼比率,透過 JNI 以 Java/C 共工的方式提升單元指令的執行時間,唯此法將降低可攜性。

而為了增進 CPU 效率, 特別是與 I/O 運算交互重疊時(在 PLC 控制系統軟體上特別明鮮,如 DI/DO, AI/AO 等), 可利用 Java 內建之多執行緒機制; 由於 Java 語言強型態之因素,多執行緒的負載在 Java 是相當輕微的。

不過,多執行緒若只在利用分時平行特性以增進執行效益,縮短單元指令的執行時間,這是可行的(唯仍須顧及 Java 堆疊區記憶體之增加); 但涉及同步問題時,將因鎖定機制而降低效率,或不定的擋阻導至反應時間的不可預測性,進一步危害 Java 即時性的支援。此外,Java 為強調各公共程式介面為執行緒安全性(thread safety),造成過多的執行緒鎖定與等待(如 StringBuffer)[],亦損及整體執行效率。

時間、時序控制機制方面:

Java 執行緒同步機制是即時性的另一要害。有幾個原因:優先序設定未必反應在核心執行緒上、同步指令(wait(),yield()、notify()等)無法確切指定對象、monitor 機制等待之不確定性、deadlock 是可能的、優先序逆轉(priority inversion)問題等,皆使得多執行緒之即時性支援功能無法發揮,反而惡化之。

此外, 多執行緒及同步機制只能提供相對次序性時間,無法提供絕對性的時間控制,如無法限定某一程式區塊的的執行時間,而 sleep()的時間遲延也會因 lock 問題而失真; 此外,Java 所提供的時間解析度只有 55msec (or 1/18.2 sec) (由 System.currentTimeMilis()決定),對控制較高頻取樣及反應區間是不足的。另外,Java 的例外處理機制亦開啟了非週期性負載的可能性,而且例外處理的反應時間也不具可預測性,一部份是其處理路徑的不等(受例外參數影響), 一部份則因物件覆載與多型機制所導致的動態變異性。

其它方面: 如排程負載、可排程性、動態環境因子等

即使克服前述的效率問題及反應時間的不可預測性,在應用程式之工作單元層次(task unit level),仍將面臨可排程性理論[12,13]的結構性限制。執行緒的個數影響排程所需十間,可用記憶量及斷裂與回收效率相關[7],而執行緒也需耗用 java 堆疊記憶體,故而執行緒的個數與可用記憶體多寡等動態環境因子,會交互影響降低可預測性。

有關可排程性的理論,在廣義頻比增排程理論架構中的三個定理可作為討論的機礎。定理假設頻比增(Rate Monotonic), 固定優先序(fixed priority), 非搶佔式(non preemptive)之排程機制, 並據此證明理論上多工可即時排程之的條件; 在此, 每一工作時間 C_i 及其週期 T_i 皆須被量測外, 工作總數 n 也是有限制的。 然而, 除了 C_i 及 T_I 等之不確定性外, 我們無法限制使用者動態產生執行緒數目(增大 N 值),特別是獨立的週期性工作。

定理一 [Rate Monotonic Schedulability]: 一組 n 個獨立的週期性工作, 以頻比增演算法排程時, 在下述情況下, 可確保其可排程性使個工作皆可於限期內開始: $\sum_{I=0,n} C_i/T_i \leq n * (2^{1/n} - 1)$.

定理二 [Extended Rate Monotonic Schedulability]: 一組 n 個獨立的週期性工作, 以頻比增演算法並採用 priority ceiling 協定進行排程時,在下述情況下,可確保其可排程性使個工作皆可於限期內開始:

$$if\ \forall i,\ 1 \leq i \leq n,\ (\sum_{I=0,I-1} C_i/T_i) + (C_i + B_i + E_i)/T_i \leq i * (2^{1/i} - 1),$$

其中,$0 \leq E_i = T_i - D_i$ 且 B_i 為工作 τ_i 被較低優先序工作所延遲的時段.

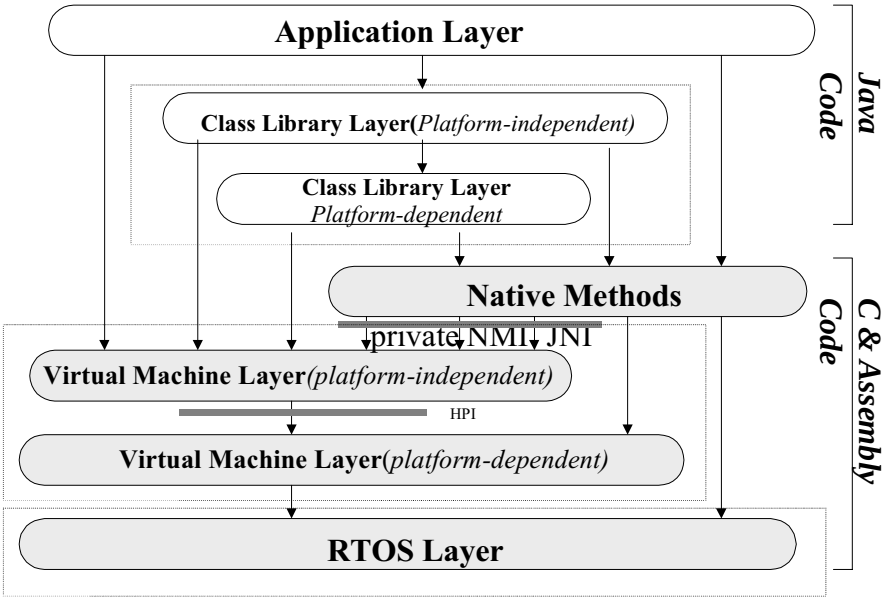
定理三 [Equivalence of Sporadicity] 一組含工作 τ_i 可排程的周期性工作, 在將 τ_i 換成具相同周期與執行時間之 sporadic server 後,仍是可排程的.

定理的陳述亦指出, 即時系統涉及(最差狀況)時間量測(即 C_i)、可排程性及執行工作所需(保留的)資源等,因此需對重要的 Java 運作指令與時間行為,特別是與 JNI 及執行緒有關之機制,進行實際量測; 且不論量測的精準度, Java 語法若無法表達這些資訊, 虛擬機器也無從判斷可排程與否。

此外, 物件導向各種動態機制與環境, 如動態記憶體大小、封裝與否、物件個數、繼承層級、多型介面呼叫等皆會影響定理的參數值。

四、Java 應用於 PC-based 控制器之建議

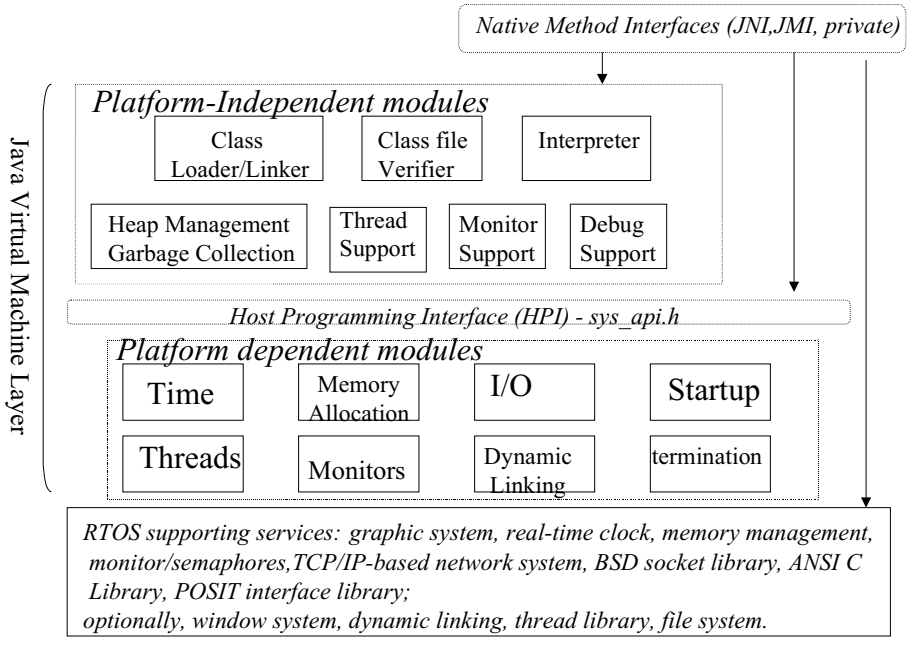
依據上述的探討,任何應用 Java 於 PC-based 控制器的研發計畫,特別是作為剛即時性嵌入式系統軟體及應用,而非只是人機介面的應用,在現階段仍是相當具挑戰性的工作,必須結合系統軟體及控制應用領域的知識與技能,選擇所需技術與架構組合, 一方面掌握 Java 平台對 CPU 及記憶體資源運用的控制, 一方面恰當地



圖三 PJAE Architecture

分離出控制策略所需的工作, 進行即時性分析並予以配置執行緒及其它資源, 同時也需小心處理環境變動因子等。

在主機平台架構選擇上, 我們建議採用 Java/C 混成式的 PersonalJava 應用環境架構規範(如圖三)[1],一方面其架構針對嵌入式系統環境與應用限制做了瘦身,使整個執行平台縮小,回應了記憶體資源運用之限制,而 Class API 也作修改或使具有彈性組態式選擇可調適於不同的硬體環境;執行動態上亦強化 JIT 及 Native Compiling 及 Romizer 等機制以回應 CPU 運用及執行效率方面; 另外它以即時作業系統為平台,且提供虛擬機器 Porting 後的 Timer API 及 Kernel Thread API,



圖四 PJAE JVM細部架構與RTOS支援

強化對絕對時間及相對時序控制機制,提供剛即時回應的門路, 配合 JNI 機制,使 Java/C 共工的方式可視需求彈性組配; 至於 Deadlock 及排程問題, 一方面有新增的 Thread 機制可運用,另一方面,核心工作緒可採 Deadlock free 之協定設計,避免死結偵測或解除之負荷。

依據圖三之規範,選擇恰當的硬體與即時作業系統, 以建置或轉換方式架構出如圖四之虛擬機器及即時作業系統所支援服務之細節;其中有幾點值得說明; 首先是即時作業系統(RTOS)及其底層所隔離之硬體與週邊, 很明顯地, 在嵌入式系統中我們需要一微核心以提供基本的時間、行程、中斷、Boot、低階輸出入甚至簡化的記憶體管理等; 第二, 在 Java 完全的平台獨立性與 RTOS 間, 我們逐層由平台相關的部份累進過渡到平台獨立的部份, 當然過程中為保持即時特性, 保留有各類存取及控制低階資源之捷徑(如 Private, HPI, NMI, JNI 等)配置及保留必要的資源; 整體架構也突顯了 Java 與 C 不同程度分工的必要性; 最後, 仍有很多的工作須在 JVM 層完成,包括動態的 JIT 與 GC 等。

最後,為進行即時性分析,應在預定的平台量測最差狀況時間、評估可排程性及估量執行工作所需(保留的)資源,小心的設計應用程式之執行緒數目與同步行為, 及控制其反應時間與週期期性,並對排程負載、動態環境因子等方面有所掌控,方能達成整體系統的即時性。最差狀況時間量測(即 C_I),代表基本工作單元的(內在)執行時間, 在控制器應用時函蓋驅動程式執行部份(C 程式碼)及 Java 碼直譯部份, 而各工作單位 T_I 則以執行緒代表之。

在缺乏效能側寫工具時或只須粗估時,亦應在選定圖四平台上,了解不同型態及參數個數對 JNI 的負荷變異,了解 Thread 之建置(new) 、啟動(start) 、文本變換(context switch)同步方法、鎖定等之負荷及多個 Thread 時之影響,了解不同型態及大小與數量之物件記憶體之配置與回收變異,了解物件導向相關特性的差異(如繼承層次是否影響等)。

五、結語

在對 Java 支援即時性進行以上廣泛而簡短的探討後,針對 Java 應用於 PC_Based 控制器的問題,可以感受到技術方案正多元化迅速發展,架構已成形及關鍵技術明確標定,甚至初步解決方案已超越雛型階段,逐漸進入技術市場領導地位競逐之階段。Java 應用於 PLC 控制器之技術可行性已可確定,而產品組件成本更可望進一步降低, 因此,新興利基市場、新應用的標定與彈性化的技術模組策略與組合機制,將易形重要; 畢竟新技術若只是用以重複或改善現有解決方案,而無法創造出新價值或競爭優勢,是無法在企業及市場產生足夠的研發動力的。確信 Java 語言及技術可取代 C/C++及組合語言,扮演 PLC 控制器剛即時性應用之系統程式設計語言及執行平台後,我們應回頭思索 Java 語言及技術之原始定位: 物件導向網際網路程式設計語言, 如何善用其物件導向特性、元件化及再用性及各項開放性標準化網路應用技術, 以開創出以網路為中心(Network

Centric) 、元件化的各類以 PC_based 控制器為基礎之高階工廠自動化自動化應用及企業整合,應會是另一個更重要的議題; 這也是我們提出彙聚綜覽(圖一)軟硬同體之嵌入式系統與即時性應用研發各考慮因素的用意。

六、參考文獻

1. Persona Java Home Pages, Sun Microsystems Inc. (URL: http:/www.javasoft.com/products/personaljava/)
2. White paper : The Java HotSpot Performance Engine Architecture, (URL: http://java.sun.com/products/hotjava/whitepaper.html)
3. Erik Armstrong, HotSpot: A New Breed of Virtual Machine, Java World, March 1998, http://www.javaworld.com/javaworld/jw-03-1998/jw-03-hotspot.html
4. Stevn Brody, Can Hotspot Jumpstart your Java Applications?, Java World, June 1999, http://www.javaworld.com/javaworld/jw-06-1999/jw-06-hotspot_p.html
5. John Neffenger, The Volano Report: Which Java Platform is fastest, most scalable? A JavaWorld Exclusive, Java World, March 1999, http://www.javaworld.com/javaworld/jw-03-1999/jw-03-volanomark_p.html
6. Kelvin Nilsen，Embedded Real-Time Development in the Java Language.
7. Kelvin Nilsen, Adding Real Time Capabilities to Java, CACM, June 1998.
8. Yan Gu, B.S. Lee and Wentong Cai, Evaluation of Java Thread Performance on Two Different Multithreaded Kernels,
9. Benjamin M. Brosgal, A Comparison of the Concurrency and Real Time Features of Ada 95 and Java, ACM SIGAda, pp.175-192, 1998.
10. Deepak Mulchandani, Java For Embedded Systems, IEEE Internet Computing, May/June 1998.
11. Akihiko Miyoshi and Takuro Kitayama, Implementation and Evaluation of Real-Time Java Thread, IEEE, 1997.
12. C.L. Liu & J.W. Layland, Scheduling Algorithms for Multi-Programming in a Hard Real Time Environment, JACM, Vol.20, No.1, 1973.
13. Lui Sha, Ragunathan Rajkumar and Shrish S. Sathaye, Generalized Rate Monotonic Scheduling Theory: A Framework for Developing Real Time Systems, Proc. of the IEEE, Vol.82, No., Jan. 1994.
14. Mark T. Hoske, Objects Make Software Behave Like Hardware, Control Engineering, Oct. 1998.
15. Robert Atherton, Java Object Technology Can Be Next Process Control Wave, Control Engineering, Oct. 1998.
16. Shen-Tzay Huang, Theory and Practices of Real Time Support for LAN-based Set-Top Box Runtime. Unpublished scripts, 1998.
17. U.Brinkschulte, C. Krakowski, J. Kreuzinger and Th. Ungerer, A Multithreaded Java Microcontroller for Thread-Oriented Real Time Event Handling, IEEE, 1999
18. Allan Heydon and Marc Najork, Performance Limitations of the Java Core Libraries, Java'99 San Francisco, ACM 1999.
19. John A. Stankovic, Real Time and Embedded Systems, ACM Computing Survey,

1996.
20. Pual Tyma, Why are we using Java Again?, CACM, June 1998.
21. Edy Bertolissi and Clive Preece, Java in Real Time Applications, IEEE Transactions on Nuclear Science, Vol. 45, No.4, Aug 1998.
22. Rick Cook, Java Embeds Itself in the Control Market, Java World, Jan. 1998, http://www.javaworld.com/javaworld/jw-01-1998/jw-01-embedded_p.html
23. Jurgen Tryggvesson, Torjun Mattsson, Hansruedi Heeb, Jbed: Java for Real Time Systems, Dr. Dobb's Journal, Nov. 1999.