行政院國家科學委員會專題研究計畫 成果報告

多核心嵌入式軟體之模型驅動整合開發環境-VMC--總計畫(2/2)
研究成果報告(完整版)

計 畫 類 別 ： 整合型
計 畫 編 號 ： NSC 99-2220-E-029-001-
執 行 期 間 ： 99 年 08 月 01 日至 100 年 07 月 31 日
執 行 單 位 ： 東海大學資訊工程學系

計 畫 主 持 人 ： 朱正忠
共 同 主 持 人 ： 熊博安、楊朝棟、張志宏、薛念林、石志雄
孔崇旭

報 告 附 件 ： 出席國際會議研究心得報告及發表論文

公 開 資 訊 ： 本計畫可公開查詢

中 華 民 國 100 年 10 月 31 日

中文摘要：　模型驅動開發的應用越來越廣泛，主要的概念是獲取所有系統設計的重要資訊，利用正規或是半正規的模型描述系統不同的抽象層級，將複雜的問題簡化，並且透過自動化程式碼產生的技術，產生出實際的程式碼，有效的加速系統開發，降低開發的成本，其中物件管理組織所定義的模型驅動架構是目前最接近此一概念之架構，統一塑模語言更成為其主要的核心語言，它可以描述系統不同面向的靜態或是動態觀點，這些觀點彼此相依並且有所交集，但是統一塑模語言並未定義模型間的關聯，且缺少檢驗模型之間的關聯性或是一致性的機制，當不一致的模型資訊出現時，更會影響模型驅動架構下程式碼產生的結果。

在本計畫中，我們分析統一塑模語言的儲存格式，並根據彼此之間的關聯性，在可延伸標記語言為基礎的整合模型中，建立其水平及垂直的一致性，且透過相似度的量測，比對不同模型，找出潛在的關聯來建議系統開發人員，以達到軟體模型的追溯性，最後我們在 Eclipse 的環境下，建構此一外掛工具，協助系統開發人員進行塑模。

英文摘要：　Multi-core architectures have proliferated embedded processor designs； however, few embedded software engineers know how to program them. The current state-of-the-art technology in multi-core programming is based on the use of language extensions or libraries, such as OpenMP and Intel Threading Building Blocks (TBB) that require parallel computing expertise. To accelerate the adoption of parallel programming technologies by embedded software designers, we extend our tool, Verifiable Embedded Real-Time Application Framework (VERTAF), for multi-core embedded software design and verification. Our primary goal is model-driven architecture (MDA) development for such software. In this article, we focus on the code generation for multi-core embedded software using TBB which is a C++ library and helps us to leverage multi-core processor performance. TBB also represents a higher-level, task-based parallelism for performance and scalability. We will use an example to illustrate the integration of TBB into VERTAF.

Several issues crop up when developing a model-driven architecture for multi-core embedded software. First of all, how much and what kinds of explicit parallelism must be specified by a software engineer through system modeling. Second, how can we automatically and correctly realize the user-specified models into multi-core embedded software code? Third, how do we test and validate the generated code. Finally, how do we apply a software engineering process to the development of multi-core embedded software? We will try to provide partial solutions to the above issues, which are still open to more research work.

The VERTAF/Multi-Core (VMC) framework takes SysML models as input which contains user-specified modellevel explicit parallelism and automatically generates corresponding multi-core embedded software code in C++, which are scheduled and tested for a particular platform such as ARM 11MPCore and Linux OS.

The VMC project has been divided into seven subjects, including VMC_REM, VMC_MUM, VMC_DSS, VMC_SYN, VMC_AM, and VMC_TMS. These subprojects cooperate together to solve the problems aforementioned.

附件一

行政院國家科學委員會補助專題研究計畫 ■ 成 果 報 告 □期末進度報告

多核心嵌入式軟體之模型驅動整合開發環境-VMC

計畫類別：□ 個別型計畫　　■ 整合型計畫
計畫編號：NSC99-2220-E-029-001
執行期間：2010 年 08 月 01 日至 2011 年 07 月 31 日

計畫主持人：朱正忠
共同主持人：熊博安、楊朝棟
計畫參與人員：張志宏、薛念林、石志雄、孔崇旭

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：東海大學資訊工程與科學系

中華民國一〇〇年十月三十一日

目錄

摘要

VERTAF/Multi-Core (VMC) 為一針對多核心嵌入式架構之整合性軟體開發環境，開發者可藉由本環境以SysML描述其系統需求，以SysML標準符號塑模其系統設計，並將該設計套用樣式結構自動轉換為更高品質的設計，透過優良的設計模型自動產生程式碼，最後並可對程式碼加以測試。因此，本系統具下列優點：（1）支援多核架構嵌入式軟體之設計與驗證；（2）提供一個整合性的開發環境；（3）因本環境乃基於SysML標準所建構，所以易與目前既有之SysML開發工具整合;（4） 支援既有之多核架構開發函式庫，如：OpenMP 以及Intel Threading Building Blocks 。

總計畫包含七項子計畫：子計畫一以SysML為基礎之多核心嵌入式系統需求塑模環境，子計畫二支援多核心嵌入式軟體設計之多重觀點整合模型與可再用元件庫，子計畫三多核心嵌入式軟體設計之設計支援系統，子計畫四多核心嵌入式軟體之合成與程式碼生成，子計畫五多核心嵌入式軟體設計工具系統之架構支援，子計畫六多核心嵌入式軟體設計工具系統之平行程式優化支援，以及子計畫七多核心嵌入式軟體設計之測試支援系統。各子計畫關係圖如下：

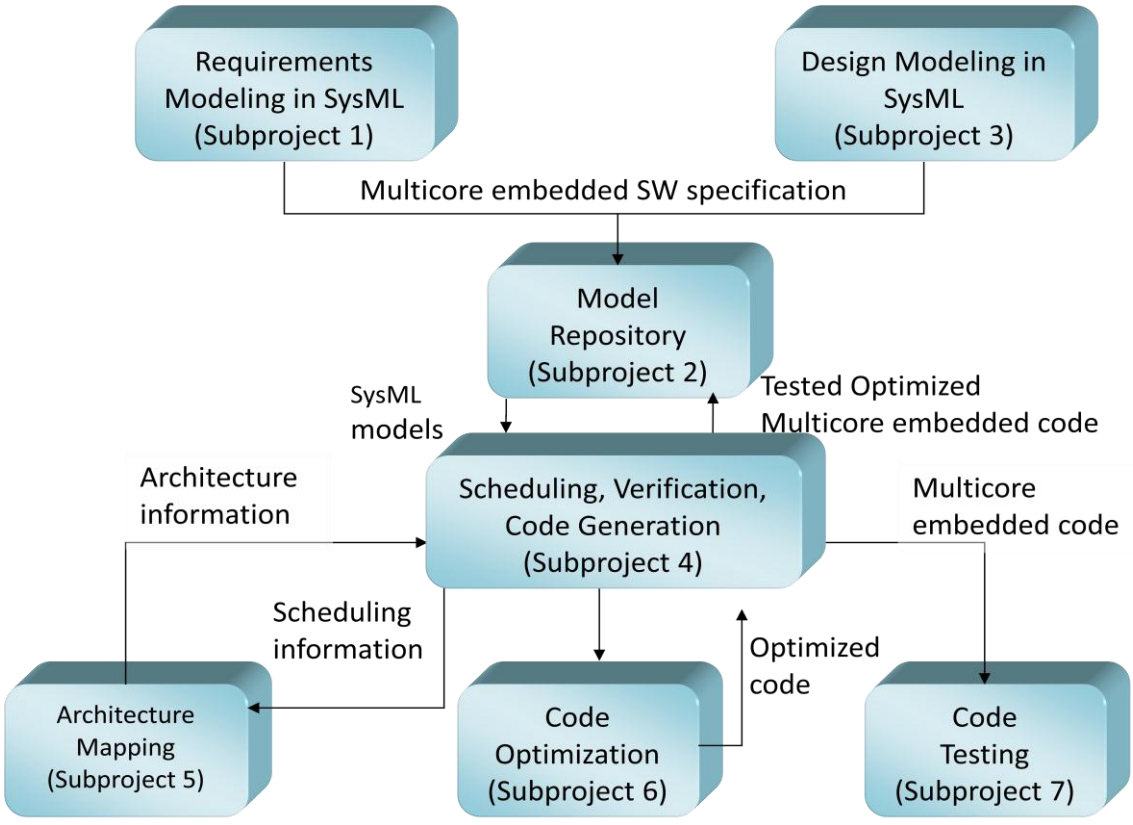


圖1.

圖 1 各子計畫之間關係圖

各子計畫內容摘要如下：

I. 子計畫一: 以SysML為基礎之多核心嵌入式系統需求塑模環境：

本子計畫主要在需求階段解決因自然語言所描術的需求常有模糊不清、不一致等問題，使得訂定規格所帶來的效益減低,成本增加。在執行嵌入式軟體開發的過程中有很多的問題，而其重要的問題之一，就是如何幫助系統開發人員從先前開發過的案例中找尋符合目前系統的可重用軟體需求，並利用重用技術減少系統開發成本、加快研發速度。然而這個問題不容易解決的原因在於缺乏能夠提供系統開發人員瞭解需求重用的資訊，因而需要人工逐一檢視需求文件。本研究中將結合需求訪談與編輯、樣板及SysML圖三種元素到軟體中，利用介面引導搭配專家樣板實現需求重用,並讓系統開發人員了解使用者需求的正確與否，預期將可減少開發人員思索如何定義需求屬性的時間，提高開發效率。

II. 子計畫二:多重觀點整合模型與可再利用元件庫:

本子計劃主要目的在支援多核心嵌入式軟體設計之多重觀點整合模型與可再用元件庫(VMC_MUM)，分析了XML-based Unified Model 在嵌入式系統中之限制及元件再利用，以物件導向技術(OO Technologies) 作為基礎，主要原因是目前物件導向分析及設計語言如UML 及相關物件導向程式語言，如Java，C++已被廣泛的接受與使用，本子計劃分析了重觀點整合模型與可再用元件庫： 第一是使用先前在XML為基礎的軟體標準整合模型研究的成果，定義出一個以模型為基礎之XML 系統文件描述模型，稱為XML-based Unified Model Metamodel。 二是使用Reusable Design Component Repository，藉由元件式資料庫之再利用性來提高軟體品質及產能。最後透過本子計畫的Unified and Integration Model 作各子計劃之間的銜接與資訊交換。將各子計畫所採用不同之標準整合後，透過一致的表現方式以及整合鏈結(Integration Links)的串接，有效達到跨階段、觀點間相關連資訊的串接與再用。

III. 子計畫三：設計樣式支援系統:

本計畫重點著重於系統開發者在設計方面的支援，即VMC之設計應用子系統（Design Supporting Subsystem for VMC，簡稱為VMC-DSS）主要功能的設計與開發，包含樣式塑模模組、操作定義模組以及硬體資訊定義模組，並與其他子計畫系統做進一步的整合，且以一嵌入式系統實例驗證本研究之實用性。本計畫預期提供系統開發者一個完整的UML系統設計流程環境，即選定硬體規範、設計與套用樣式、定義操作功能，用以提供其他子計畫銜接之所需的設計相關資訊。

IV. 子計畫四：多核心嵌入式軟體合成及程式碼生成:

本子計畫(VMC-SYN)的主要目標在延伸VERTAF程式碼自動生成的能力，使用者透過UML工具來描述系統模型，讓程式碼生成器來自動生成具有可平行工作、可平行處理資料和具有可平行處理資料流的程式碼。我們以第一年設計的DVR系統為例，我們接收子計畫一(VMC-REM)針對DVR系統所設計的requirement diagrams，以及子計畫三(VMC-PSS)畫的class diagrams、block definition diagrams、和state machine diagrams，根據我們所設計的一套流程來自動產生可平行化的程式碼。好處除了可減少程式碼的撰寫之外，也可提供程式設計師在多核心嵌入式系統上開發專案時一套完整的自動化流程。

V. 子計畫五：架構及效能調校支援實作

本子計劃(Subproject 5)主要著重於架構對應及整合後工具組之實際應用，首要之目標為設計一多執行緒分析模組(multi-thread analysis module)以針對合成多核心嵌入式程式碼時，解決多執行緒與多核心分配對應之問題。目的在提昇多核心嵌入式程式碼之效能及避免 Racing 及 deadlock 現象。此階段將利用 Intel 之所提供之軟體工具如 Intel® VTune™ Performance Analyzer, Intel® Thread Profiler 及 Intel® Thread Checker 等進行分析 。因最後之多核心嵌入式程式碼將支援 Intel Threading Building Blocks (Intel TBB)語法，此階段亦將配合提供 TBB 所需之程式庫及系統調配指標，進行反對應模組設計。形成一漸進式迴圈(iterative loop)達成整體最佳效能。

VI. 子計畫六：平行程式優化支援實作

本子計畫主要設計與實作多核心嵌入式軟體之平行程式優化支援。有效的迴圈切割與排程可以顯著地減少程式在多核心處理器系統環境中的整體回應時間，特別是針對具有許多迴圈的應用程式與針對新興的多核心嵌入式系統環境，其優化處理也更為困難。第一年(2009/8~2010/7)，子計畫六將使用編譯器技術如資料相依性分析與軟體管線化技術，完成指令階層平行化與平行迴圈切割與迴圈排程優化支援。子計畫六將整合各個部分功能並建立與其他子計畫的介面，其中包含模型解析器(與子計畫二的介面)、軟體之合成器及程式碼生成器(與子計畫四的介面)、架構對應分析器(與子計畫五的介面)、及輔助測試的相關程式碼(與子計畫七的介面)。第二年(2010/8~2011/7)，子計畫六將擴充VMC 平行程式優化介面支援至更多的架構平台與應用，如支援多核心程式庫例如OpenMP及Intel Threading Building Block及強化與其他子計畫的介面。同時將建置一套在多核心嵌入式系統上平行程式優化支援模式適用於OpenMP 及Intel Threading Building Block 應用程式介面。

VII. 子計畫七：多核心嵌入式軟體之測試支援系統

在多核心嵌入式軟體的軟體開發的過程中，軟體測試是一項龐大的工程，更是軟體產品執行品質控管的重要關鍵。然而軟體測試工作往往需要花費很多的人力及物力，加上嵌入式系統的硬體資源的限制，使得嵌入式軟體的測試較為困難。此外多核心嵌入式軟體之 concurrent program 存在很多不確定性，且相對於 sequence program 也較難除錯。然而，目前大部分嵌入式軟體開發環境也較缺乏自動化的軟體測試工具來輔助。為了減輕這些問題，本子計畫建構了一個支援嵌入式軟體開發的 cross-testing 的自動化測試環境系統。能提供自動執行多回合測試機制；降低測試工程師的負擔，增進嵌入式系統執行測試工作的效率。以「支援平行程式自動化測試」及「object testing」為主找出平行程式在資料同步時潛在的錯誤，藉由檢查多執行緒共同存取共享資源 synchronization lock 的一致性及監控所有 synchronization primitives 事件執行的順序來判斷是否有潛在的 race condition。而平行程式的效率也是開發人員所關注的項目，本子計畫針對多核心來平行程式效能的測試，可以即時監控 CPU 各個核心的利用率、程式執行時間。此外，我們也將發展如何提昇平行效能的方法及發展配合多核心嵌入式軟體的測試方法，例如配合 OpenMP 及 TBB 的測試方法。

關鍵詞: 嵌入式系統, 多核心, 統一塑模語言,系統塑模語言,統一塑模,自由軟體, 軟體設計框架

Abstract

Multi-core architectures have proliferated embedded processor designs; however, few embedded software engineers know how to program them. The current state-of-the-art technology in multi-core programming is based on the use of language extensions or libraries, such as OpenMP and Intel Threading Building Blocks (TBB) that require parallel computing expertise. To accelerate the adoption of parallel programming technologies by embedded software designers, we extend our tool, Verifiable Embedded Real-Time Application Framework (VERTAF), for multi-core embedded software design and verification. Our primary goal is model-driven architecture (MDA) development for such software. In this article, we focus on the code generation for multi-core embedded software using TBB which is a C++ library and helps us to leverage multi-core processor performance. TBB also represents a higher-level, task-based parallelism for performance and scalability. We will use an example to illustrate the integration of TBB into VERTAF.

Several issues crop up when developing a model-driven architecture for multi-core embedded software. First of all, how much and what kinds of explicit parallelism must be specified by a software engineer through system modeling. Second, how can we automatically and correctly realize the user-specified models into multi-core embedded software code? Third, how do we test and validate the generated code. Finally, how do we apply a software engineering process to the development of multi-core embedded software? We will try to provide partial solutions to the above issues, which are still open to more research work.

The VERTAF/Multi-Core (VMC) framework takes SysML models as input which contains user-specified modellevel explicit parallelism and automatically generates corresponding multi-core embedded software code in C++, which are scheduled and tested for a particular platform such as ARM 11MPCore and Linux OS.

The VMC project has been divided into seven subjects, including VMC_REM, VMC_MUM, VMC_DSS, VMC_SYN, VMC_AM, and VMC_TMS. These subprojects cooperate together to solve the problems aforementioned.

Keyword: multicore, Embedded system, model driven, UML, SysML, Open Source, framework.

一、 前言

本整合型計畫網羅了各方面的專家，包括了需求分析、軟體設計、軟體工程、嵌入式系統、平行化計算、軟體測試、系統整合等，共同開發設計出一套整合式多核心嵌入式軟體之模型驅動整合開發環境–VERTAF/Multi-Core (VMC)。VMC對於目前國內嵌入式軟體產業缺乏支援自動化的軟體發展整合開發環境現象，提供一個符合開放原始碼(open source)規範的選擇，進而提升國內嵌入式產業軟體設計能力與品質。

本整合型開發計畫(簡稱 VMC 計畫)為二年期長期計畫，橫跨五所大專院校合作，因此若沒有一致的專案管理方式與技術整合機制，會造成資源浪費、時程落後、品質不齊、無法整合等問題。因此總計畫依據 Light-weight CMMI 的方式，溝通、協調與管理整個 VMC 計畫，如圖二所示，總計畫的重點在於架構協調、專案管理、規格控制、與品質控制等議題。

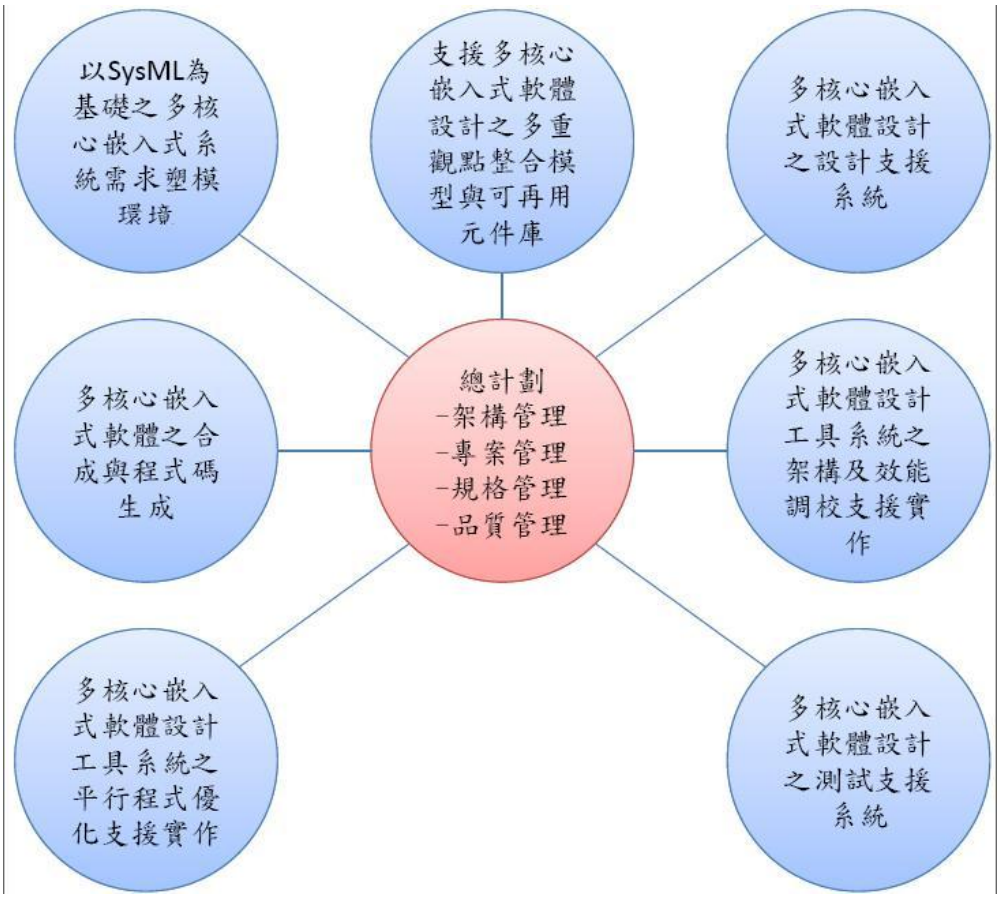


圖 2 總計畫與其它子計畫關係圖

A. **架構管理**: 主要在確認各子計畫對 VMC 系統有一致的架構認知，特別是在介面的溝通上。

B. **專案管理**: 則廣泛的包含CMMI中的專案計畫、專案監控、建構管理與量化分析等項目。

在專案計畫中我們會展開更細部的工作分解圖，估計每一項工作的工作量、並依此建立專案的時程，並依此進行專案的進度追蹤。建構管理主要在維護重要的工程文件、程式碼或資料的一致性，避免各子計畫因為版本的錯誤而造成整合錯誤。量化分析則定期的收集數據資料以作為後續專案管理的依據。

C. **規格管理**: 主要包含需求規格與設計規格的建立與維護，主要的目的在協助團隊對整個計畫的產出有明確、一致、完整的認識，並維護其一致性直至專案結束。在此過程中，我們將會產出需求規格書、系統設計規格書以及需求的追溯文件。

D. **品質管理**: 包含流程、文件、程式碼、及系統的品質管理。程式碼及系統的品質主要透過程式碼檢視(code inspection)、技術審查(technical review)及動態測試(dynamic testing)來完成，其他重要的流程與產物則透過稽核的方式來檢驗其品質，以確保團隊是在高效率、高品質的流程下作業。

二、 二年研究項目及成果

VMC 計畫目前執行最後一年。VMC 計畫第一年有五個子計畫組成，Bottom-up 的設計方式,先實作 Digital Video Recorder (DVR) System 做為第二年整合所有子計畫的基礎，第一年成果如下：

1. 延伸系統塑模語言（SysML）以及相關需求塑模方法，以達成支援多核架構之嵌入式系統需求文件，並產生一個以XML 為基礎之整合式系統文件，便利各子系統間的整合。
2. 支援設計樣式的套用，以提升嵌入式軟體設計的品質與效能。本計畫將應用模型驅動開發方式的模型轉換技術以自動化的方式提升設計的品質,並提供使用者自行設計樣式的環境。
3. 設計與實作多核心嵌入式軟體之程式碼生成器,以提升嵌入式軟體工程師的生產力以及提高程式碼的正確性。為了達到延伸VERTAF 的平行程式碼自動化生成，我們將借助二項成熟技術的整合，亦即將Quantum Platform（QP）的核心技術整合到Threading Building Blocks（TBB）的平行技術，減少程式碼自動合成時，上層系統塑模與TBB 平行技術及平行計算模型之間的差距，同時也減少程式設計者的程式碼撰寫及避免設計者投入太多心力在解決因設計及規劃平行程式時所帶來的問題。
4. 以軟體品質提昇控管為出發點，提昇開發多核心嵌入式軟體的品質，開發出軟體測

試的環境，方便軟體測試的進行，增加測試的效能，提高測試的coverage，降低軟體的錯誤率。另外，預計發展出提昇Code Coverage 的方法及配合多核心嵌入式軟體的測試方法。

就第一年的成果而言，需求塑模利用需求模板(Requirement template)對 DVR 例子分析重要資訊，設計樣式則針對 DVR 中的平行模型，例如 software pipeline 等定義了樣式模板。而 DVR 系統也以 QP 及 TBB 完成整合及實作並在 PC 端做展示，其中包含系統功能測試及監測部份功能，擷取系統執行期資訊並加以分析，以上成果亦有論文發表於會議及期刊，此成果將會在面後章節詳列。

在第二年的計畫中增加了二個新的子計畫分別為多重觀點整合模型與可再用元件庫(VMC-MUM)以及平行程式優化支援實作(VMC-PPO)。 透過子計畫二，各子畫目前已按 DVR 例子提供 input/output 範例進行資訊交換整合。此外 VMC 計畫利用 Trac project 建置專案管理網站(http://140.123.105.205/projects/vmc)。Trac 搭配 SVN 來使用，含括文件，時間軸追蹤，產品路線圖，瀏覽原始碼，新增/檢視待辦事項，搜尋等功能) ，提供計畫進度管理、會議紀綠等相關文件及各子計畫之設計產出之管理，並提供各子計畫交換資訊及解決問題的地方。

下圖 3 為 wiki 文件管理畫面，包含文件如下：

1. VMCMeetingMinutes:

   存放各子計畫每次開會的會議紀錄。

2. Documents：

   主要為 light weight CMMI 文件管理

3. DVRSourceCode

   存放三年實作之 DVR source code

4. DVRModels

   存放以 papyrus 設計之 DVR 塑模

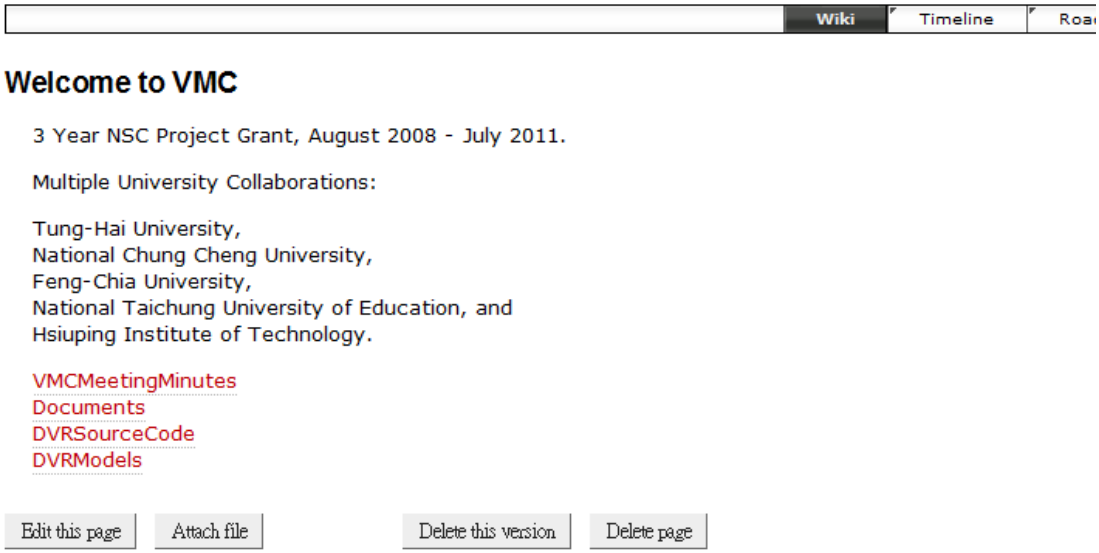


圖 3 wiki 文件管理

圖 4 為 Timeline 畫面，主要用來追蹤各子計畫對於專案修改的狀況，以維護系統內容版本的控管。

圖 4 Timeline 畫面

圖 5 為 roadmap 畫面，主要用來管理專案進度，目前以 CMMI 文件撰寫及繳交做為製定各階段的里程碑。可搭配 Ticket 系統，向各子計畫發出通知，並控管進度。

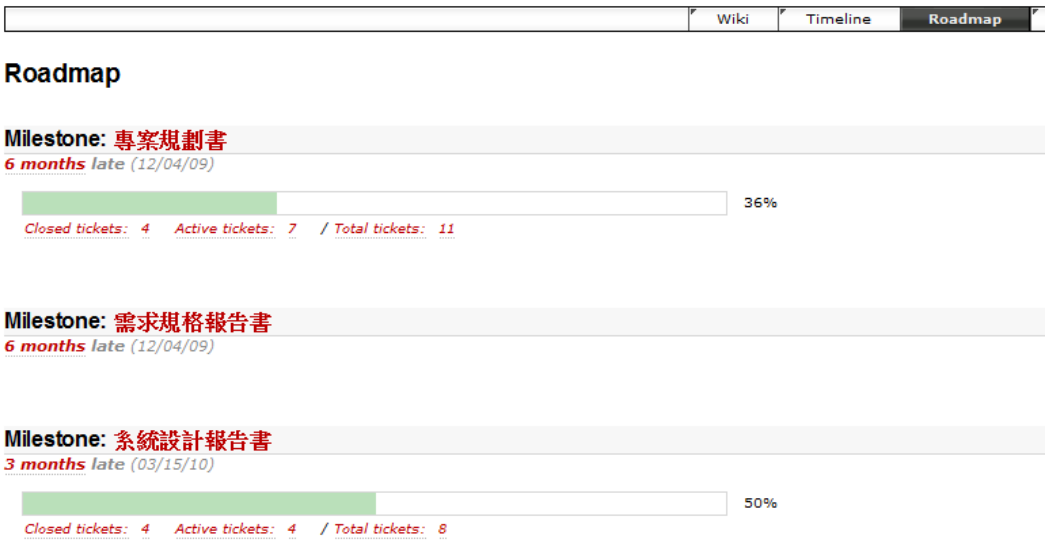


圖 5 Roadmap 畫面

圖 6 為源碼瀏覽畫面，主要是利用 SVN 為後端版本控制系統，前端 Trac 提供使用者查詢及管理，SVN logs，可以看到修改紀錄， 版本比對等資訊。

圖 6 源碼瀏覽及版本控管畫面

為了第二年各子計畫之間的整合及塑模之間資料一致性的控管及追蹤，各子計畫實作三項計畫整合工作，包含：API 介面制定及 Data dictionary。就 API 制定的目的而言，各子計畫制訂的 API 交由子計畫二統一實作，提供子計畫間訊息及資料的交流及傳遞。交流的技術主要以 XML 格式為主。另外由於系統的 model 之間彼此往往會具有相依性，需要為 model 之間建立鏈結，以維護不同設計階段各 model 之間的關係，我們透過 data dictionary 制訂，維護不同 model 之間資料的一致性，達到整合的目標及 VMC 功能的正確性。

本章節我們將針對各子計畫之研究及開發成果做說明：

I. VMC-REM：以 SysML 為基礎之多核心嵌入式系統需求塑模環境

A. 研究目的

一個標準程序的軟體開發流程可由四個階段所組成，分別為需求分析階段(Requirement Phase)、規劃設計階段(Analysis and Design Phase)、實作階段(Implementation Phase)、以及維護階段(Maintenance Phase)，如圖7所示。一個軟體流程由需求階段開始逐一往下，直到維護階段可做數循環的開發流程，其間各階段間也可經由結果的度量做往上修正的動作，或對原先設定的目標做修正。

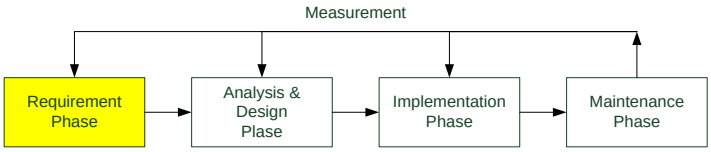


圖 7 軟體開發流程架構圖

而嵌入式系統的開發程序也與瀑布模型差異不遠，由於在開發過程中，會產生許多文件，而這些文件間又存在著參照、衍生等關係，為能達到提升文件整合及管理，甚至軟體開發整體效益有效地提昇，陸陸續續有多種標準化的模型與技術被提出以解決各階段的問題，例如，物件導向分析與設計觀念之於新軟體架構分析與規劃的改變；標準化模型語言之於系統整體架構、設計、狀態控制、與測試分析；設計樣板(Design Patterns)之於問題定位與有效設計模式的導入；元件再使用(Component Reuse)之於回收應用高效益高認同之軟件與資料模組或片段等等。企業對於這些標準及技術片段的使用，對於軟體開發流程定義片段的設計規格，未考量軟體設計各個階段的連貫性與追蹤機制，大多必須以人工方式檢視並協調來自不同工作小組的規劃與設計的片段，而類似的問題，將進一步導致當處理跨越階段與階段時，因不同設計之間的統合、細部分工等作業程序採用異質模型與規範而衍生的對應(mapping)與連結(linking)的相關問題，使得軟體開發與研究者無法以一個一致性、協同性的開發方式進行軟體系統的開發作業，這將使得整個軟體開發成本居高不下，而這個模組/元件間缺乏共通聯繫與互動追蹤機制的問題，於該系統未來進行維護(maintenance)、演進(evolution)、與再生工程 (re-engineering) 等必要作業時，更將益發地增添問題的困難度與複雜度。

在軟體開發流程(Software Process)的每一個階段(Phase)，從需求(Requirement)、設計(Design)、流程(Process)、實作(Implementation)一直到測試(Testing)、維護(Maintenance)，都有其適用的Model及技術，例如模型驅動架構(Model-Driven Architecture，MDA)、敏捷軟體開發(Agile)、等等有效率及可執行的方法。在需求擷取方面，雖然也有需求工程(Requirement Engineering)來幫助工程師獲得客戶要求(Customer Needs)的相關資訊，再經由人工作業，來轉換整理成顧客的需求(Customer Requirements)，然後得到最終的軟體需

求規格(SRS，Software Requirement Specification)。但是由於將顧客要求轉換成顧客需求必須建立在經驗的基礎上，也就是說如果工程師缺乏相關的經驗，在將要求轉成需求的過程當中，有可能會遺漏相關的資訊或是必要之功能。

在開發流程之需求階段時，及時地規劃出使用者需求的系統流程以及動態，而後開發者得以從介面延伸出類別、方法以及類別圖。而其他的需求，尤其在Non-Functional的方面的需求，如Security、Performance、Reliability，則需要開發者在設計階段訂定。近年來嵌入式軟體不論從絕對數量或複雜度而言都是處於蓬勃發展階段，需要開發的軟體越來越多樣化，包括手機、PDA、GPS或是嵌入式伺服器(Embedded Server)都是全球廠商非常看好的一塊市場。同時，在嵌入式系統中，透過軟體實現的功能有日益增加的趨勢。傳統的先硬體後軟體的設計方法對這種複雜的嵌入式系統開發已不再適用。因為無論從品質或是從生產力的角度來看，軟體和硬體都必須同時開發和驗證。因此在需求分析階段取得精確、完整的需求將會是降低軟體開發成本的關鍵，其重要性也是顯而易見。

本計畫兩年期間藉由所開發之需求塑模環境，提供使用者方便建立完整且量化之需求、提高嵌入式軟體需求的重用性與可靠度，並且建立需求之間的雙向追溯，節省系統開發的成本，並能應變市場環境以及多樣客製化等因素，成為針對多核心嵌入式軟體一套完整的輔助開發環境。

B. 研究成果

在需求分析，針對嵌入式系統的特性，在需求階段特別強調量化及模型導向，使用SysML來描述需求。為了提供輸入需求資料時更好的組織化，並且正規化資料，避免缺漏或模糊等情況，我們設計一個系統模式的畫面呈現給需求建立者，包括利用模板來編輯需求，以及匯入匯出等功能。接著，藉由專家樣板的引導，讓需求建立者能夠清楚瞭解需要輸入什麼資訊以達到需求的重用性。

為了提高嵌入式軟體元件的重用性(Reusability)與元件之間的關聯性，我們提出以XML為基礎的專家樣板(expert template)，並開發了XML為基底並結合參考硬體限制的重用嵌入式軟體需求資料庫。

當需求輸入完畢後，會將資訊輸出成XML檔案存入需求庫，成為經驗樣版的儲存體，以利日後的使用者重用，若未來需求的內容有所改變，也可以經由此系統模式更新需求，並以版本控制作為解決需求重複或不一致的情況產生。使用者並不瞭解自己真正需求的元件時，就可以依照一套標準的專家樣板來引導使用者篩選出需要的元件類型，將自己所需要的元件一一的選取。

1. 模型導向需求塑模

在系統開發的第一個步驟是需求文件撰寫，需求文件的品質將影響專案流程的進度，我們希望使用者在需求圖形的塑模過程中，能夠讓每個需求模型都附帶完整需求資訊，圖 8 為基於 MDA 架構的需求資訊建立流程，在 CIM 階段，使用者在需求訪談之後，套用我們所制定的需求樣版，將所得到的需求資訊輸入需求樣版，產出初步的 SysML 需求圖，並且透過需求樣版所得到的資訊來繪製使用案例圖，這個階段為需求的雛型(Prototype)，接下來 PIM 階段，建立 SysML 狀態圖(State Diagram)與區塊定義圖(Block Definition diagram)以及其所對應的需求樣版內容，透過狀態圖與區塊定義圖的分析內容可以分別得到系統靜態(Static)與動態(Dynamic)的需求，最後回到 Prototype，與原始 SysML 需求圖所提供的資訊整合之後，形成一份完整的 SysML 需求圖(Requirement Diagram)。

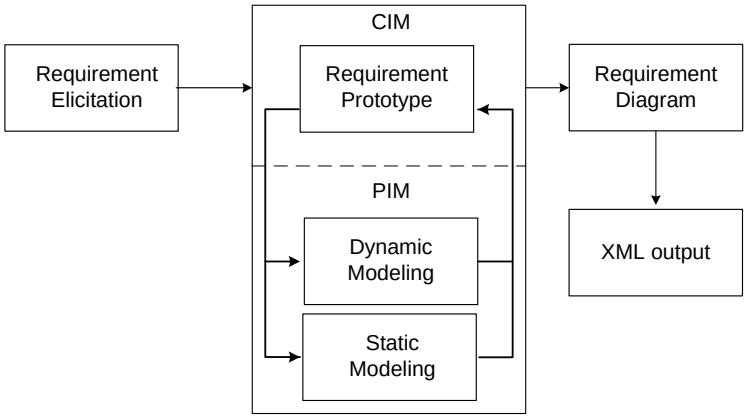


圖 8 需求資訊建立流程圖

需求擷取與分析的推導流程，如圖 9 所示經由一連串的需求擷取分析與 SysML 的塑模方式，從使用者需求推導到系統需求。

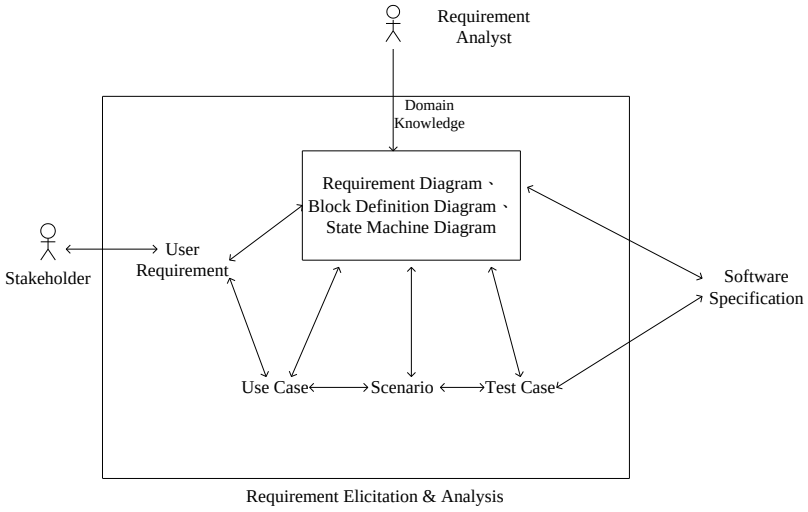


圖 9 需求擷取分析流程

SysML 的需求圖有助於組織使用者的需求，並且能夠有效地表現出不同需求間的關

係。需求圖可以顯現出需求與需求之間的階層(hierarchy)，讓系統設計師清楚地看出需求階層關係與系統架構；還有數種表示方法來找出需求間的關係，包括 derive (衍生)、copy(複製)、satisfy(滿足)、verify(驗證)、refine(提煉)、trace(追蹤)等，通常應用於比較大型且複雜系統方面。

SysML 可以應用於表示各種領域的模型元素，靠著需求圖與視覺化的關係。事實上，SysML 的需求圖為 UML 類別圖(class diagram)的擴充原型(stereotype)，如圖 10 所示:

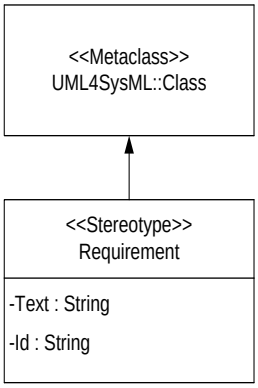


圖 10 需求圖擴充原型

如圖 11 為需求推導流程，由使用者需求到系統需求的推導。與專案關係人的需求訪談以及系統分析師的知識領域分析，來產生使用者需求。接下來由 SysML 的塑模，來產生需求圖，此需求圖為初階的需求圖圖形。然後由分析使用者需求來產生 Use Case 與 Scenario，Use Case 與需求之間會有 refine 關係產生。可以從 Scenario 的細部描述，來得到使用者需求的 external 與 internal interface，而形成第二層的需求圖，會有 derive 關係產生；另外一方面，可以藉著 Use Case 與 Scenario，來推得使用者需求的 Test Case，需求與 Test Case 會有 verify 關係。由於已經從 Scenario 得知細部的需求，加上第二層需求圖的內外部介面動作，所以可推得需求的類別屬性與操作，而產生 Block Definition Diagram，類別與需求之間有 satisfy 關係。在推導得知需求最細部的類別之後，階層式的完整需求圖圖形也已經完成，然後再描繪出系統的 State Machine Diagram。這就是由使用者需求到系統需求的推導流程。

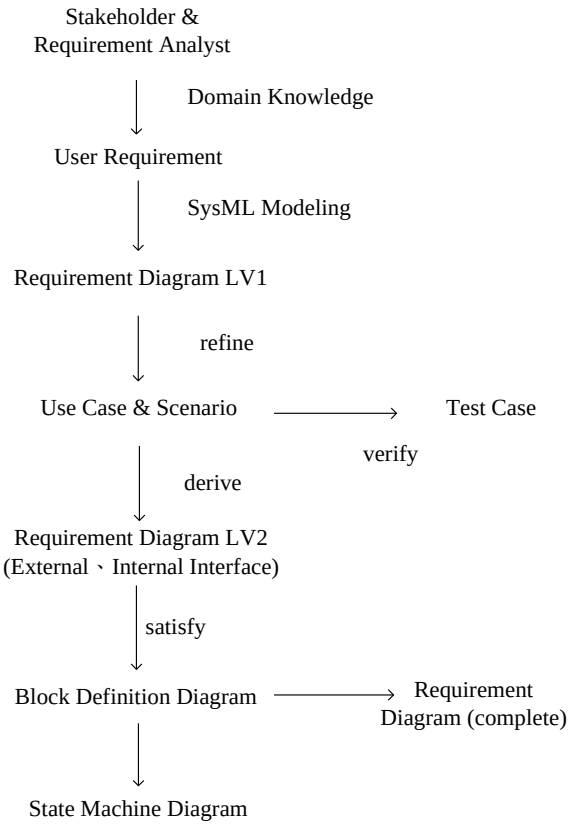


圖 11 需求推導流程圖

2. CIM 階段

在 CIM 階段分析師根據需求問題樣版內容針對每個需求分別繪製出 SysML 需求圖(圖 12),每個需求類別(Requirement class)都擁有各自對應的樣版 Profile，如表 1 功能性需求樣版 Profile 是功能性需求的樣版 Profile。

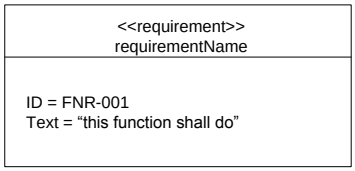


圖 12Requirement diagram element

表 1 功能性需求樣版 Profile

Profile	Content
需求種類	Functional
需求名稱	Connection Server
需求編號	VSS-FNR-002
專案名稱	VMC_SYN
需求建立者	Joe
需求建立日期	2009.5.1
….	…

透過分析初步的需求問題樣版與初步的需求圖，了解初始需求之間的關聯性，將 SysML 需求圖的雛型依照需求關聯來描繪，再配合使用案例圖將有助於快速釐清系統範圍以及功能，促進

需求的完整性，同時也便於估算開發成本與時程。

3. PIM 階段

在 PIM 階段我們根據分析狀態圖來有說明關於人員、流程、系統的運作，以及其狀態的轉移的規則；接著為狀態圖內部各狀態之間轉移的動作，定義需求的操作(operation)，以及為狀態的轉換行為定義屬性(Attribute)，所得到的資訊如表 2，將被參照使用於區塊定義圖來表達系統內部的靜態結構，區塊定義圖(Block Definition Diagram 以下簡稱 BDD)，如圖 13，通常會是系統設計的時候，被拿來當作重要設計藍圖，分析師經過套用需求樣版，得到初步的系統靜態結構，也就是區塊定義圖，而透過分析前面的狀態圖與需求樣版來得知區塊定義圖尚需加入的屬性與操作種類，並可透過每個區塊所屬的 Profile(表 3)來描述元件運作方式。

表 2 定義狀態轉移動作時的操作

系統名稱	
A.轉移動作 B.轉換行為	a.操作定義 b.屬性定義

表 3 Connection Server BDD Profile 內容

Profile	Content
區塊名稱	Connection Server
區塊編號	1
描述	提供連線給 RMC，並且接受 client 端的指令，經過指令分析之後提供相對應的服務。

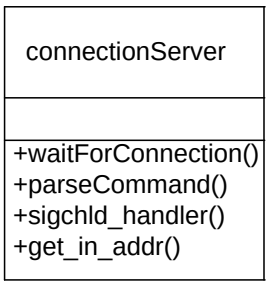


圖 13 Connection Server Block Definition Diagram

在分析狀態圖以及子系統內每個元件所屬的區塊，瞭解子系統內一群物件的互動情形之後，分析師將完整的區塊定義圖繪製出來，分析師在 CIM 階段取得系統環境以及初步的需求內容，經過分析之後，於 PIM 階段將系統內部細節結構化並且產出 SysML 圖形，經過擷取系統的靜態以及動態需求的過程，就能得到繪制出完整的 SysML 需求圖所需要的需求資訊，而分析與整合這些需求資訊之後所得到的需求圖，是經由原始的使用者需求不斷的被延伸(derive)、細分(refine)、滿足(Satisfy)、 驗證(verify)、追蹤(trace)之後，方能完成一個表現出需求之間關連性的 SysML 需求圖。

4. 需求追蹤

需求圖的追蹤(Traceability)是系統設計中很重要的一部分。在需求工程中，需求追蹤是極具重要性的一環，在某些需求改變之後，必須經由需求追蹤以便找出需求與需求

之間關係的變動與關聯，以及該次變動是否對系統整體造成影響，一般而言，需求建立過程中會使用需求表(requirement table)來做需求追蹤。

SysML 需求表分為階層需求表與需求關係表兩種格式，階層需求表可以表示需求間的階層關係，而需求關係表則用來表示需求與需求之間的關聯性，表 4 與表 5 分別為上述的兩種範例表格:

表 4 SysML 階層需求表

Id	Name	Type
需求代號	需求名稱	需求類別

表 5 SysML 需求關係表

Id	Name	Relates To	Relates How	Type
需求代號	需求名稱	相關需求	關聯性	需求類別

在開發大型與複雜的系統的過程中，系統需求通常可以被細分為多個子需求，叫做群集需求(grouping requirement)，在 SysML 中的 User requirement 可被編成 SysML requirement sub-package，如圖 14 SysML 群集需求圖所示，一個需求(Requirement A)可再被細分、延伸為數個子需求，子需求彼此之間具有前面我們所提到的衍生、提煉等關聯性類型。

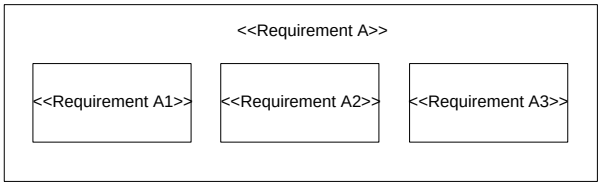


圖 14 SysML 群集需求圖

為了避免因為人為的疏忽，例如工程師的資歷深淺不一，而導致需求不完整等問題發生，我們在這裡定義了一個需求問題模組樣版，需求樣版(Requirement Template)的好處就是可以導引使用者依循制定的需求欄位來描述需求，避免因為前面所陳述因為需求的模稜兩可與不完整而造成需求品質的下降，提供系統分析師來進行使用者或是客戶的初步需求擷取，定義出一系列的需求欄位，以擷取精確以及量化的需求為目標，來定義每個選項，以及進而對使用者在編輯需求的同時對系統需求做出完整的需求導引，並且找出軟體的主要需求、功能性與非功能性需求，提供 CIM 以及 PIM 階段所需的基本需求資料，勾勒出系統的主要架構。

5. OCL 條件限制

我們利用 OCL 來規範使用者在輸入 Profile 時的敘述方式，這樣可以藉由驗證使用者所輸入的需求內容是否符合該欄位所制定的 OCL 條件，避免使用者輸入不當的數據或是需求文件內容，如表 6 是我們應用 OCL 做出條件限制的範例之一。

表 6 OCL 條件限制

樣版內容	資料型態	條件限制	OCL 定義
客戶端數目	整數	數值不得小於 1	**Self**.CleintNum>0

6. 重用需求資料庫

同時，我們運用重用資料庫(Reuse Library)進行需求重用，以便有效率地需求重用及引導使用者建構需求。考慮到嵌入式開發專案多種開發特性，像客制化、領域差異性高、開發時程緊湊等等，我們的需求庫存放所有開發中或已開發的重用嵌入式軟體需求資料，針對硬體規格差異性小、使用者需求相近、系列產品去進行需求重用，以利於縮短開發週期，進一步降低開發成本。

系統設計架構如下圖 15，此以物件導向概念開發的重用資料庫系統將產生不同類型的文件，分別代表了各自的意義，以下分別說明。這些文件會經由本系統檢查是否符合格式，進而經由 XML Generator 產生 XML 文件記錄相關資訊，最後將元件與 XML 文件一起存入資料庫中，將來檢索元件時也依據這些 XML 記錄資訊來進行條件檢索。

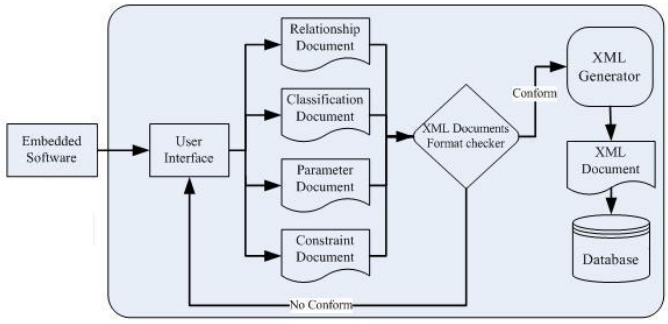


圖 15 需求重用資料庫系統架構

本研究依據上述的系統概念，將文件分為幾個部份：

Classification(分類)：此部分主要依產品類別為基礎由使用者在設計需求介面決定分類，需求依照所選取的參考進行分類。

Relationship(關聯性)：內部可能包含一個至數個需求，讓使用者能夠經由此需求所屬之介面，進而瞭解此需求之間的來源與關聯，在本系統藉由此介面增加需求之間之關聯性，並且能夠記錄良好定義需求的方式。

Constraint(限制)：限制主要因為環境限制，由於嵌入式系統軟體開發受到環境上的限制可能無法隨心所欲的設計軟體，所以在本研究加入了環境限制的考量，讓軟體開發流程符合嵌入式系統的領域。

本系統提供一個需求重用介面給使用者使用，進而開發一套重用嵌入式軟體需求庫系統。在軟體的重用上，使用者常常因為不完全瞭解其屬性如何設定而增加需求重用的困難性，此系統提供良好使用的樣版屬性設定，讓使用者更容易且快速瞭解各種需求屬性設定以及使用方式，以及利用這些屬性讓使用者更快找尋到需要的專家樣板。

7. 轉換 SysML 模型

利用需求訪談、樣板及系統塑模語言(Systems Modeling Language, SysML)三者結合，能讓使用者更直觀、快速有效將需求轉換成設計階段所需的資訊，於是利用這些資訊，初步地讓使用者確認未來的系統是否正達到使用者想要的。在這方面，我們利用 OMG MOF、DI 以及 JAXB 技術來達成模型間轉換的的目標，以產生對應之 XML 格式檔案，交付給子計畫 2 進行管理與整合。

8. 系統實作

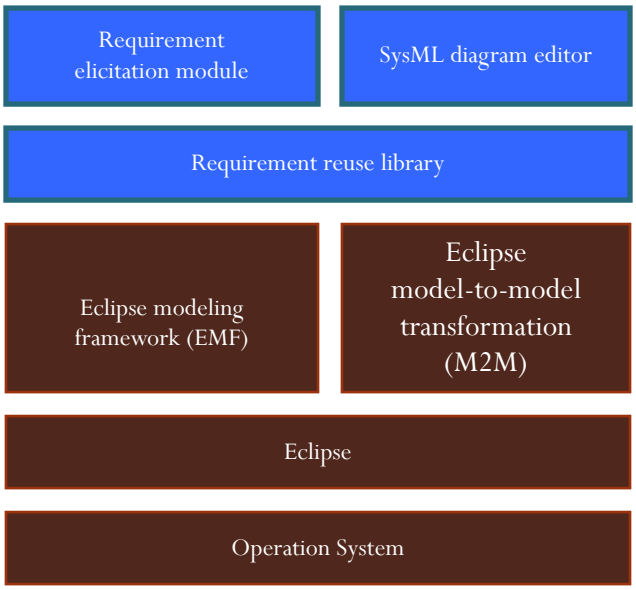


圖 16 VMC_REM 系統堆疊圖

目前計畫之VMC_REM系統堆疊圖如圖 16，其中包含：需求擷取模組(Requirement elicitation module)、SysML圖形編輯器(SysML diagram editor)、重用需求資料庫(Requirement reuse library)。

為了輔助軟體需求之擷取，利用量化的需求來輔助使用者，如下表 7所示：

表 7 量化需求範例

回應時間	**5 秒**

傳輸速率	**10 frame/s**
客戶端數量	**10 個**

我們採用「需求模板(Requirement Template)」(如圖 17)來協助使用者思考所需要的各類需求。此模板是基於功能及操作面去制定的，並透過系統內部的資料結構來補足必要的硬體資訊或參數資訊。

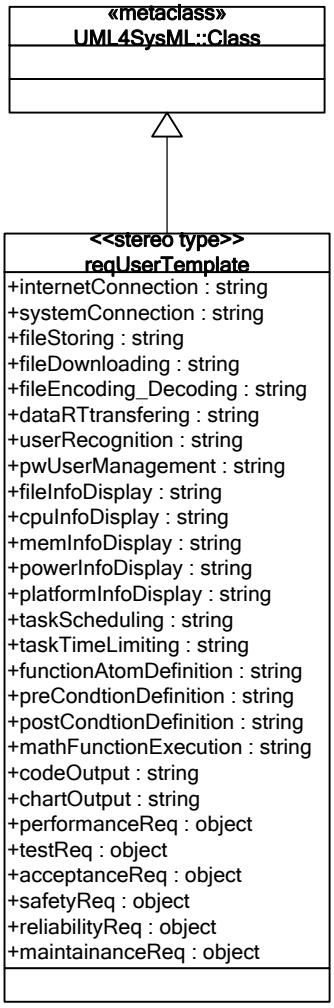


圖 17 需求模板

以下是VMC-REM系統操作畫面截圖。首先使用者透過圖 18畫面，將訪談到的使用者需求，輸入到VMC-REM。

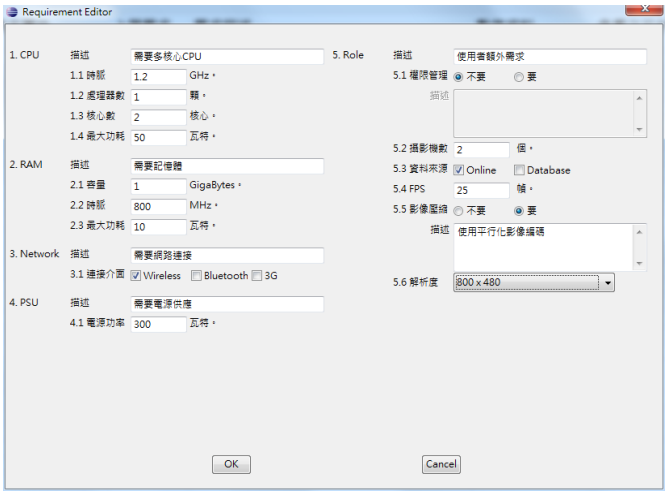


圖 18 新增基本需求

填寫完成後，按下OK，產生 需求列表(圖 19)

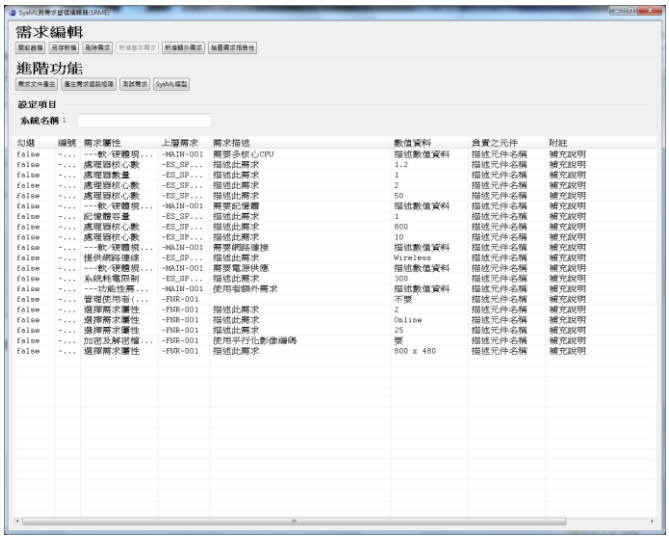


圖 19 系統需求列表

按下另存新檔，選擇塑模資訊放置位置(圖 20)

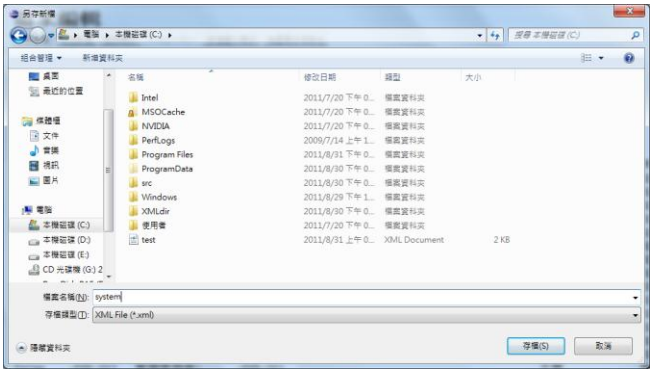


圖 20 塑模資訊存檔

若需要勾選要刪除的項目，刪除需求(圖 21)

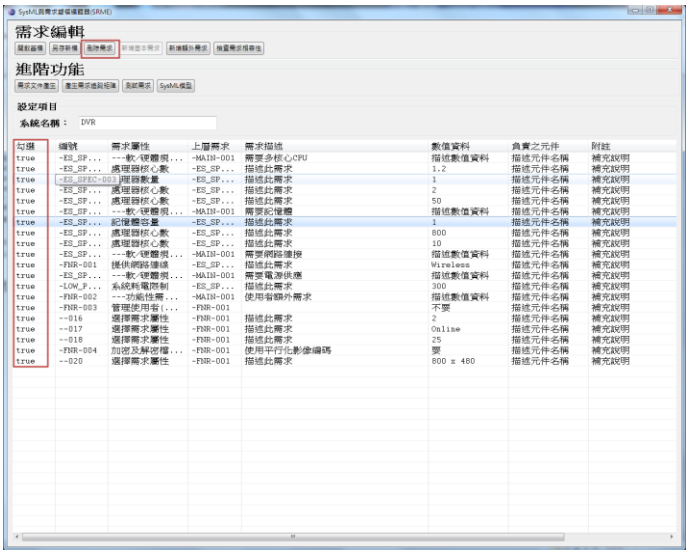


圖 21 需求刪除管理

使用者可透過VMC-REM，將以系統需求產生對應的需求追蹤矩陣(圖18)。

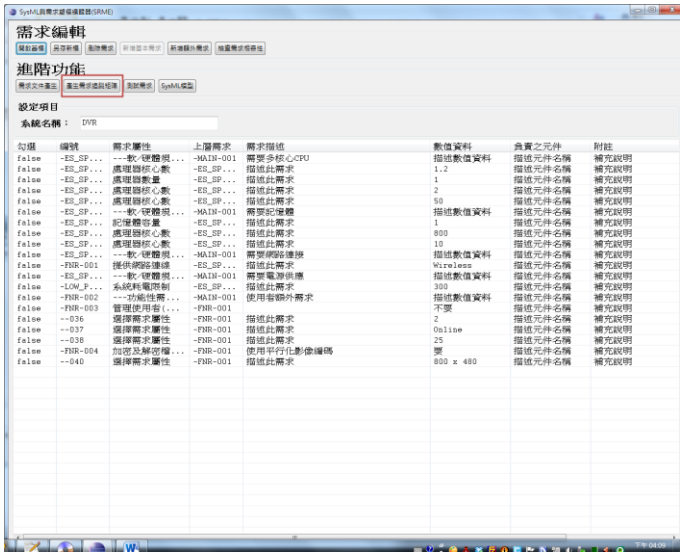


圖 22 產生需求追蹤矩陣

使用者可亦在矩陣介面上勾選，建立新關聯(圖 23)

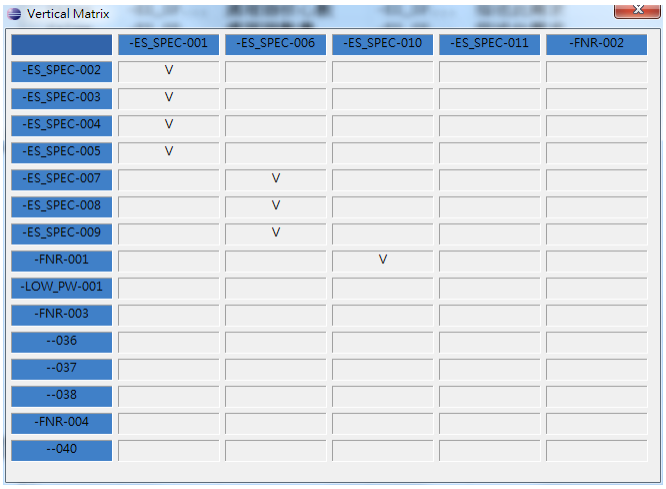


圖 23 與圖 18 對應之需求追蹤矩陣

使用者也可以透過VMC-REM產生需求文件(圖 24、圖 25)

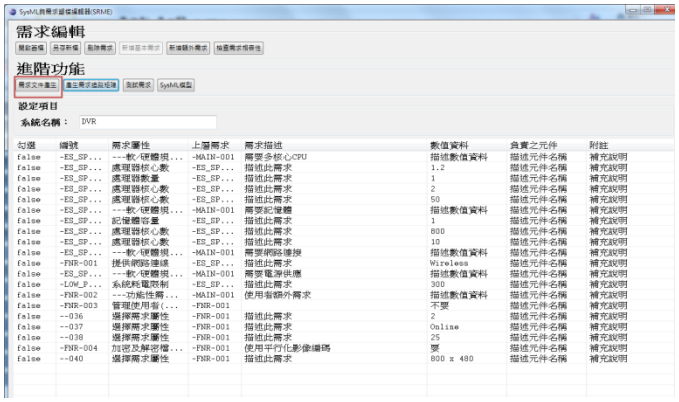


圖 24 產生需求文件

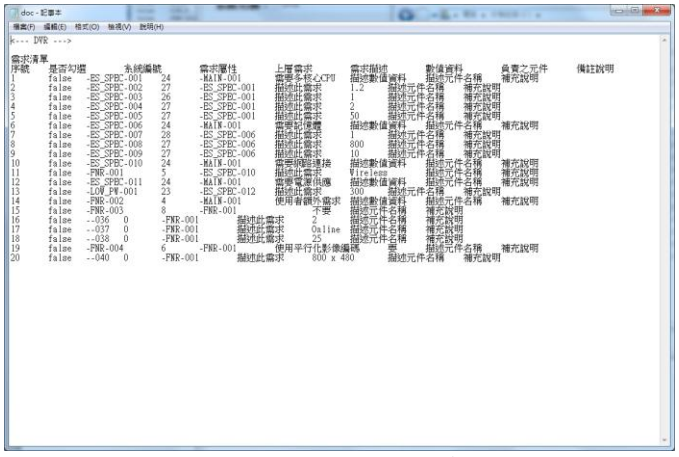


圖 25 圖 19 對應之需求文件

VMC-REM系統可以對所建立的需求產生對應SysML需求圖模型(圖 26、圖 27)。

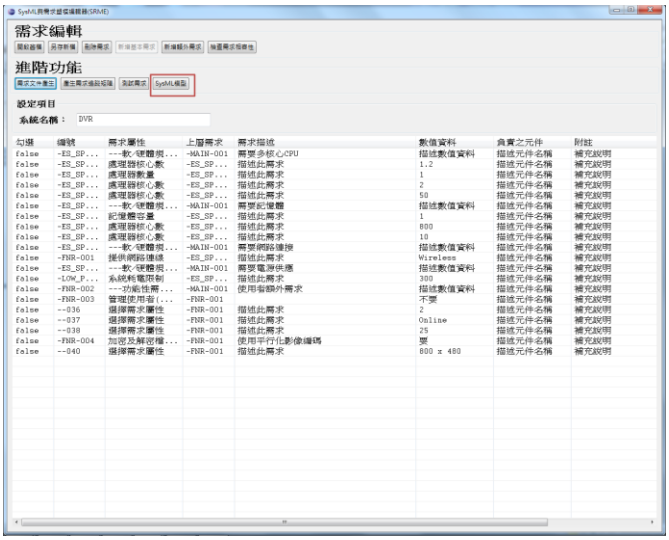


圖 26 產生 SysML 需求圖模型

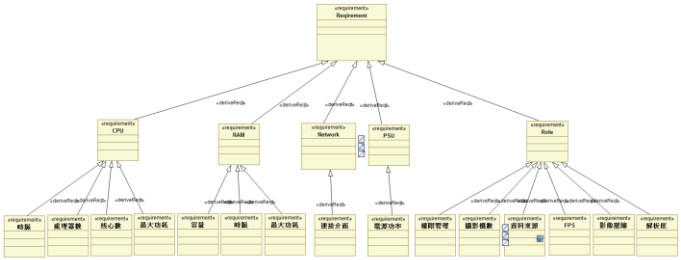


圖 27 對應產生 SysML 需求圖

使用者也可以產生輸入測試需求接受條件，提供給子計畫七產生自動測試使用(圖 28)。

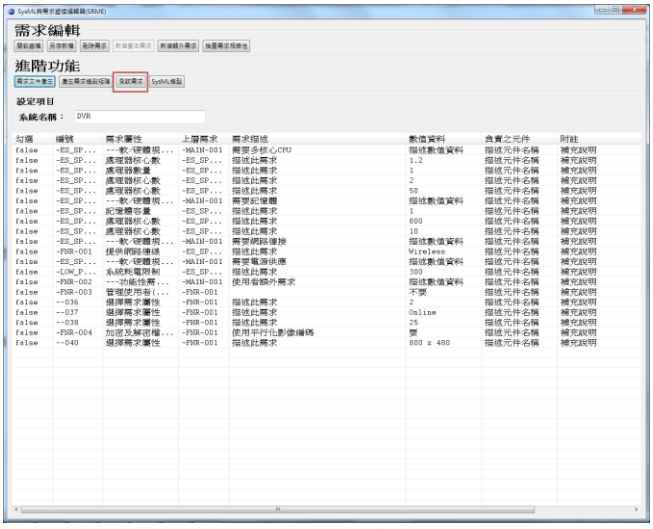


圖 28 測試需求產生

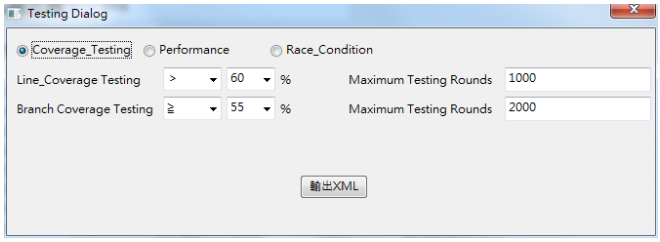


圖 29 測試需求接受條件輸出

按下輸出XML，產生XML檔案，位於eclipse根目錄下。

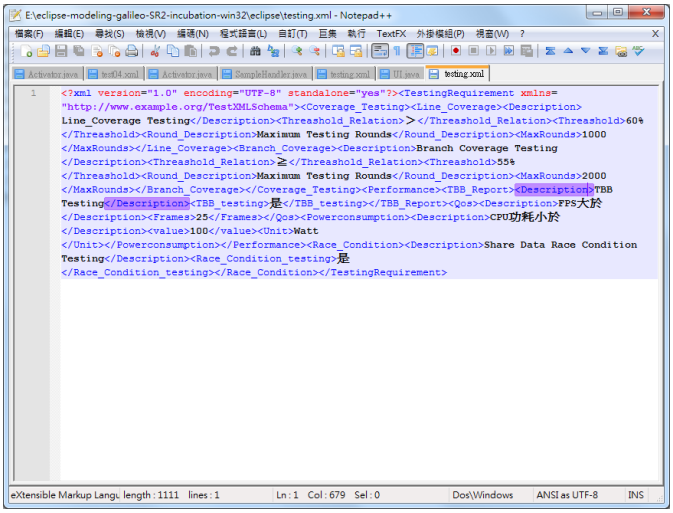


圖 30 XML 文件轉換

C. 結論

在這計畫中，我們分析了以模型導向架構為基礎之多核心嵌入式系統設計需求分析流程，透過這一個流程，需求分析師可以逐步建立相關的嵌入式系統需求條件，減少因為需求分析師經驗不足造成需求訪談不確實引發需求分析不正確的機會。

此外我們也建立了一套 SysML 多核心嵌入式系統需求塑模環境 VMC-REM 雛型，透過這套環境，使用者可以在其中輸入系統需求相關資訊，並將相關可再用需求儲存/擷取，同時可以轉換/繪製 SysML 相關塑模圖形的，提供一套視覺化的需求塑模環境，相關需求分析資訊也可以輸出成 XML 格式提供給其他子計畫/工具，做後續設計、程式碼產生、最佳化與測試使用。

此外，系統也可以將需求間的關係自動建立對應需求追蹤矩陣，幫助專案需求管理。

目前關於需求塑模分析的研究，多著重於需求擷取與管理，對於需求塑模分析仍相當缺乏，總結本計劃執行三年下來，初步達成本計畫當初設定之目標，完成需求擷取與圖形化塑模環境的建立。

在成果國際合作方面，目前已與 Dakota State University 的 Dianxiang Xu 教授達成合作協議，希望進一步將其 ISTA (Integration and System Test Automation)程式碼自動產生工具(關於徐教授 ISTA 教授 ISTA，曾發表於 PETRI NETS 2011, LNCS 6709, pp. 308–317,

2011，目前此工具已被三星電子應用於部分電子與數位媒體產品測試)，與本計畫的 SysML-based 需求塑模環境整合，達成自動化測試目標。

II. VMC-MUM：支援多核心嵌入式軟體設計之多重觀點整合模型與可再用元件庫

A. 研究目的

軟體的生命週期一般而言包含了需求、設計、實作、測試及維護五個階段，在這些階段裡可以利用許多的軟體標準以降低軟體開發本身的複雜度。然而當我們使用某個標準於某階段時所造成的改變，可能會帶來其他階段所使用的標準也連帶的需要作改變的問題。在整個軟體開發週期裡，開發團隊裡的成員會根據特殊的需求或個人的經驗法則導入不同的軟體標準。但是所引發的結果卻是標準與標準之間的不相容(Incompatibility)，這樣的結果往往會帶來許多問題，若使用的軟體輔助開發系統工具互相整合有難處，彼此間無協調整合機制，所帶來的不確定因素讓開發者之發展環境持續在改變而使其無所適從，軟體開發方法、軟體開發流程與軟體輔助開發工具無法相互整合，以致於不能發揮整體功效。軟體的開發與維護相當的不易，也因此至目前為止已有許多的組織及相關領域的專家制定並提出了軟體標準，其目的就是提昇軟體的品質。但沒有一種軟體標準及模型可以涵蓋軟體開發過程中的每個階段，因此我們必須要開發一套專屬於嵌入式系統的軟體工程方法來處理上述可能發生的問題。

綜觀目前整個嵌入式系統軟體設計方面各個階段之間的關係，在整合技術方面並無一套標準步驟以及應用的工具，會在開發的過程中將會產生許多不必要的成本，就有可能導致產出功能與原始需求不盡相同的產品仍勉強上架。並且也須考慮到跨各階段間的連續性與一致性，往往以人工方式檢視並協調整合來自不同的標準、方法及應用及工具之來源資料，更因為開發經驗難以學習累積，使得嵌入式系統開發的成本居高不下且品質浮動。並且，不同設計之間的統合、細部分工等作業程式，使得嵌入式系統開發與研究者無法以一個一致性、協同性的開發方式進行軟體系統的開發作業。此外因為系統會因不同硬體、不同 OS、不同平台而有所變化，功能亦因價格、硬體資源問題有所增減及不同，而使得整個軟體開發及維護困難重重。

不好的開發流程，已經被認定為是造成成本浪費及進度落後的主要原因，在任何新的專案的生命週期裡，最重要的一個部份就是要整合系統各階段之不同的標準，因此我們提出的 Unified and Integration Model，運用先前在 XML 為基礎的軟體標準整合模型研究的成果，定義一個以模型為基礎之 XML 系統文件描述模型，稱之為 XML-Based Unified Model Metamodel 來描述，將相關的資訊表示成元件(Components)、關連(Associations)

以及整合鏈結(Integration Links)，藉此有效地將不同的標準整合在一起，透過一致的表現方式以及整合鏈結(Integration Links)的串接，有效達到跨階段、觀點間相關連資訊的串接與再用。且軟體再使用(Software Reuse)，無疑地是一個目前用來提高軟體品質(Quality)及產能(Productivity)的熱門研究技術。基本上，所謂的軟體再使用，其範疇涵蓋可再使用的軟體元件(Software Components)的辨識(Identification)、取得(Extraction)、表示方法(Representation)、擷取(Retrieval)、調適(Adaptation)、整合(Integration)及其整個過程。

在實例中，我們觀察到近年來嵌入式軟體中的專案進度是工程師們最在乎的事項。造成不好的開發流程的原因有許多，特別是軟體在本質上具有高度的彈性，當不同公司與不同團隊採用各別的方法、Models 及 Data Schema 描述方式，便會造成進行軟體開發(Development)、整合(Integration)、再利用(Reuse)時的極大困難，最後導致軟體整體開發時的耗費人工、低效率、高價格等不利現象不斷出現。以往在開發嵌入式系統時並無導入軟體工程，會使其在開發中未能注意到更多的層面，進而增加錯誤的產生，以及人力和資源成本提高，造成開發時程的延誤。

目前自由軟體相關學者都認為嵌入式應用技術需要一個很方便使用的開發平台，然而在此方面人才還是相當缺乏，嵌入式系統並非要求全能，但必須能夠依據系統設計規格，有效率的發揮出硬體的運算功能，使得產品達到效率/價格比的最佳化。大多數的學者都是應用免費授權的 Embedded Linux 作業系統來開發新的系統，但他們著重在於嵌入式系統軟體實作(Implementation)的功能方面。整體來考量，像是整合(Integration)的問題，例如怎麼讓嵌入式系統符合軟體工程生命週期中的產物(Artifact)做整合及產物之間之追蹤(Traceability)及一致性(Consistency)。因此，實際上大多數的嵌入式系統採取的方式為土法煉鋼，依照客戶之需求開發系統，較無一套流程規範。且使用自然語言(Natural Language)的文字格式撰寫需求檔，此類需求文件檔案會充滿模擬兩可(Ambiguity)、不一致(Inconsistency)、不精確(Imprecision)及不完整(Incompleteness)的情形，造成整合的困難。除此之外，由於缺少精確的模型能力(Modeling)，使得想要整合或追蹤其他階段的產物只能倚賴人工(Manual Effort)，而造成時間上的浪費以及容易發生錯誤。為了讓軟體工程能夠有效的應用在實際的嵌入式系統之案例上，多核心嵌入式系統程式設計之整合模型(Model)需要做深入的探討與研究。

B. 研究成果

1. 一致性的分類

過去的研究指出 UML 模型的一致性可以分類為下面幾種：

垂直一致性(Vertical Consistency)：又稱交互模型(Inter-Model)一致性，指在不同的抽象層級中，不同的模型應該保持一致性。在軟體開發的程序中，當規格被轉換到更細部的規格時，一致性應要能被確保，隨著軟體程序這樣的動作會不斷的重複。

水平一致性(Horizontal Consistency)：又稱內部模型(Intra-Model)一致性，意指在相同抽象層級中不同的模型間應該保持一致性，例如類別圖、循序圖以及狀態圖彼此之間應該保持一致。

演化一致性(Evolution Consistency)：驗證不同版本的相同模型應保持一致性。

語意的一致性(Semantic Consistency)：在 UML 的 Metamodel 中，語意的意義應該被驗證。

語法的一致性(Syntactic Consistency)：在 UML 的 Metamodel 中，UML 模型的規格應該被驗證。

在本計畫中，我們利用 XUM 的理論，達到 UML 垂直與水平的一致性，整合使用案例圖(Use Case Diagram)、類別圖(Class Diagram)、狀態圖(State Machine)、循序圖(Sequence Diagram)等模型，這四種圖型最被常用來描述系統的不同觀點，因此優先被探討。

2. 整合模型 XUM

UML 可以描述系統不同面向的靜態與動態觀點，但本身並未定義彼此之間的關聯，在過往的研究當中探討了不同的 UML 模型間的關聯性，可以歸納如圖 31，系統的描述由使用案例開始，表明了系統的功能面向和使用者與其之關係，再更進一步的細化設計則利用類別圖顯示系統的靜態觀點，當某個類別圖的邏輯過於複雜時，狀態的轉移可能分散在不同的使用案例中，則由狀態圖針對事件以及狀態做統一的描述，而循序圖分別可以描述使用案例以及類別圖的動態觀點，以進行驗證，最後這些模型會產生明確的程式碼去實作。

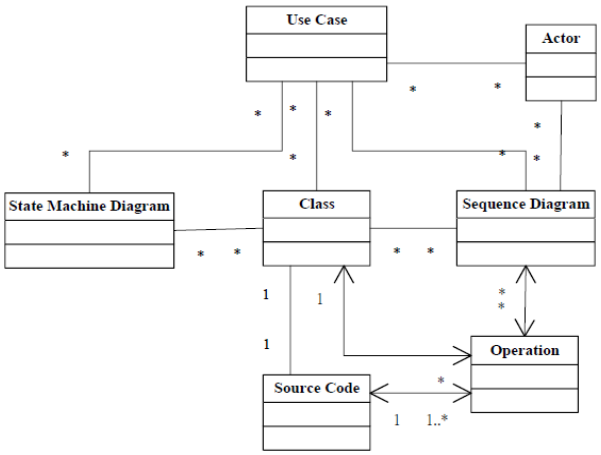


圖 31 UML 之間的關聯

為了建立模型之間的追蹤性，我們運用 XML 的標準建立整合模型，記錄必要的模型資訊，在模型驅動的架構下，目前的 UML 模型資訊都以 XMI 的標準進行儲存，以圖形交換(Diagram Interchange)的格式記錄圖形的名稱、位置、關係、大小等圖形化的資訊，所以要建立完整的整合模型，必須從這兩個標準的檔案中取得相關的資訊。圖 32 顯示類別圖在 XMI 格式下的表示方式，以 packaged Element 的標籤記錄不同的 UML 模型，包含 xmi:type 用來識別此 UML 的類型，另外具有唯一的識別 ID 和此類別的名稱。而類別所在的圖形名稱則是從圖形交換的格式中獲取，如圖 33 所示，在樹狀結構的最上層包含類別所在的圖形名稱，而樹狀結構內，可以找出符合相同 ID 之屬性值。

```
<packagedElement xmi:type="uml:Class" xmi:id=" VlIh8PcDEd-tF4FYV-UJQA" name="EncodedDataBufferManager">
  <ownedAttribute xmi:id="_BpYIsPcEEd-tF4FYV-UJQA" name="buffer1" type="_vaLNUPcDEd-tF4FYV-UJQA" isUnique="false"/>
  <ownedAttribute xmi:id="_DSie4PcEEd-tF4FYV-UJQA" name="buffer2" type="_vaLNUPcDEd-tF4FYV-UJQA" isUnique="false"/>
  <ownedOperation xmi:id="_8zND8PcDEd-tF4FYV-UJQA" name="EDTI"/>
  <ownedOperation xmi:id="_88-D8PcDEd-tF4FYV-UJQA" name="RTS"/>
  <ownedOperation xmi:id="_9CYvcPcDEd-tF4FYV-UJQA" name="SFS"/>
  <ownedOperation xmi:id="_9E0_cPcDEd-tF4FYV-UJQA" name="SVF"/>
  <ownedOperation xmi:id="_AQ89MPcEEd-tF4FYV-UJQA" name="getfilename"/>
</packagedElement>
```

圖 32 XMI 中類別的描述

```
<di2:Diagram isVisible="true" fontFamily="Arial" lineStyle="solid" fontColor="255:255:255" foregroundColor=
"255:255:255" backgroundColor="255:255:255" borderColor="255:255:255" position="0:0" name="VSS Class Diagram">
                                        ⋮
    <semanticModel xsi:type="di2:Uml1SemanticModelBridge" presentation="TextStereotype">
      <element xsi:type="uml:Class" href="VSS Class Diagram.uml# VlIh8PcDEd-tF4FYV-UJQA"/>
    </semanticModel>
                                        ⋮
</di2:Diagram>
```

圖 33 圖形交換格式的描述

當我們從上述的兩種標準獲取相關的資訊時，即可將需要的資訊寫入至整合模型之中，是 XUM 整合模型中類別圖的定義，分別從 XMI 的檔案中取得類別的 ID、名稱以及來源的檔案，並寫入指定的屬性，從圖形交換格式取得此類別所屬的圖形名稱，另外定義三種不同的鏈結記錄與其他 UML 的關聯性。

在每一種模型的表示下，定義了整合關係(Unification Link)，整合關係是用以整合各階段相關產出物的機制。其中包含了抽象鏈結(Abstraction Link)、整合鏈結(Integration Link)與程式碼鏈結(Source Code Link)，如

圖 34 定義了抽象鏈結，包含了幾個屬性，分別是此鏈結的 ID、來源的圖形名稱以及類型、指向的圖形名稱以及類型、此鏈結的建立是否由使用者所決定和此鏈結建立的時間。

```
<xsd:complexType name="abstraction_linkType">
    <xsd:attribute name="date" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="toType" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="to" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="fromType" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="from" type="xsd:string"></xsd:attribute>
    <xsd:attribute ref="id"></xsd:attribute>
</xsd:complexType>
```

圖 34 抽象鏈結的定義

.

抽象鏈結是用來串聯較抽象的軟體需求與較具體的軟體設計，軟體的生命週期中，在需求分析的階段，會利用使用案例圖獲取系統的需求，描述不同的系統功能面向或是目標，明確的指出系統該做的事情，而不是如何去做，並從較高的層級來觀看整個系統，這些被識別出的功能會在設計的階段，使用多個類別圖描述系統構成的物件以及其間的關係，再細化成循序圖與狀態圖描述動態的結構和事件，如圖 35 顯示使用案例圖與類別圖之間的關聯，兩種圖形分別位於不同的系統抽象層級，其中使用案例的描述，可能會被定義成類別的名稱或是透過特定類別內的方法去實現，透過抽象鏈結，將這些位於不同系統抽象層級的資訊進行整合，以達到垂直一致性。

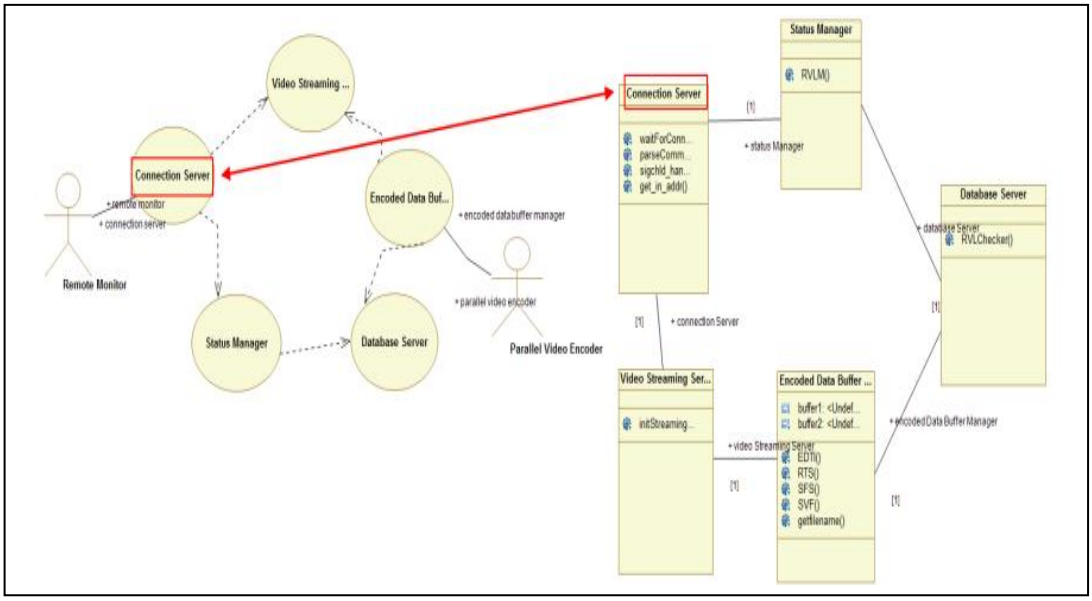


圖 35 抽象鏈結的示意

整合鏈結的目的在於將相同的抽象層級或共用性質的軟體模型做關聯，顯示出物件共享，定義如圖 36 所示，同抽象鏈結包含了相同的屬性來記錄鏈結的資料。在設計的階段，類別圖被用來表明靜態的結構，但為了清楚表示使用案例的詳細流程或是物件之間的動態操作關係，會使用循序圖來表示，循序圖主要由生命線以及生命線上的訊息呼叫所構成，循序圖上的生命線或是訊息，應該在類別圖中定義過，彼此間應有多對多的對應關係。而不同於循序圖用來表示多個物件之間的互動行為，狀態圖會被用來顯示單一特定類別的事件、狀態轉移情形，但並不是所有的類別都一定會需要狀態圖的表示，只有某些特定的行為才需要。

圖 37 則顯示這樣的關聯，如 Database Server 類別分別屬於 UML 類別圖中的類別、循序圖中的物件，而 Status Manager 類別另有狀態圖描述事件的狀態，因此彼此有整合鏈結的關係，透過整合鏈結可以達到相同抽象層級的水平一致性。

```
<xsd:complexType name="integration_linkType">
    <xsd:attribute name="date" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="toType" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="to" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="fromType" type="xsd:string"></xsd:attribute>
```

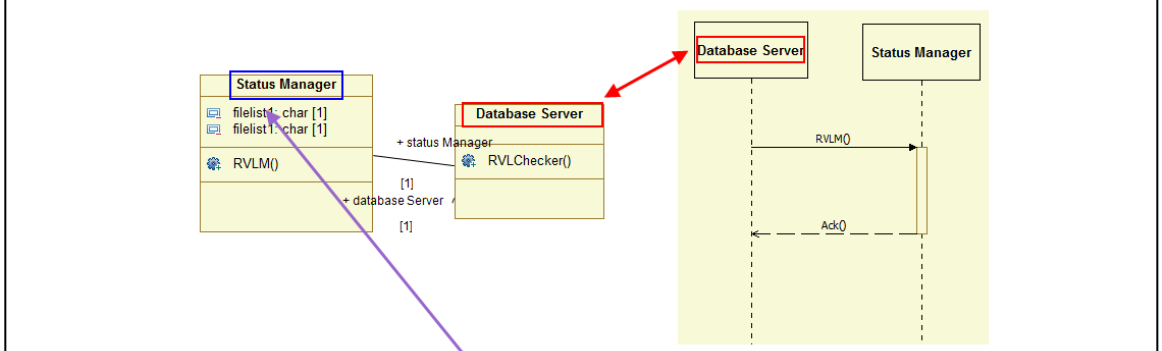


圖 37 整合鏈結的示意

程式碼鏈結則是用來串聯設計模型資訊中的類別與實作的程式碼，如圖 38 是其定義，主要包含此鏈結的 ID、來源的類別圖到指向的程式碼、此鏈結的建立是否由使用者所決定和此鏈結建立的時間，圖 39 顯示由程式碼擷取出的資訊，類別與程式碼之間，從名稱、變數到方法，彼此之間應該有著一對一的關係，任何一邊都不可以只有單獨的定義出現。

```
<xsd:complexType name="sourcecode_linkType">
    <xsd:attribute name="date" type="xsd:string" />
    <xsd:attribute name="userdefined" type="xsd:string"></xsd:attribute>
    <xsd:attribute name="to" type="xsd:string" />
    <xsd:attribute name="from" type="xsd:string" />
    <xsd:attribute name="id" type="xsd:string" />
</xsd:complexType>
```

圖 38 程式碼鏈結的定義

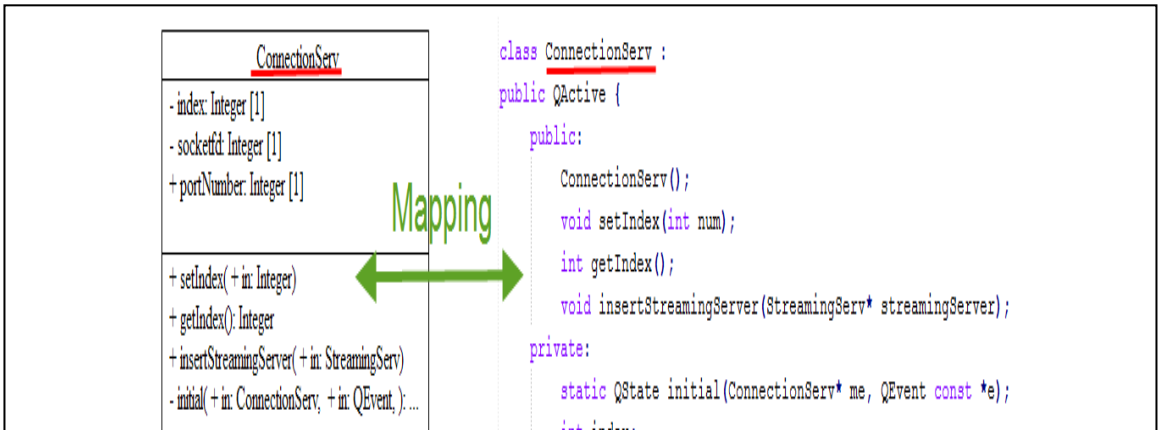


圖 39 從程式碼中擷取出的資訊

3. 模型相似度比對

在資訊檢索中，常見有幾種方法來量測文字的相似度，包含 Jaccard Coefficient、Bleu Coefficient、Dice Coefficient 等方法。在本計畫中利用文字相似度比對的方法來進行軟體產出物的比對，找出可能不一致的部分，發現潛在的錯誤，更能進一步建議使用者建立可能的關聯性，Dice Coefficient 在過往的研究顯示其成效較不受出現次數多寡的影響，且計算簡單，因此在本計畫中被使用，定義式 1 所示：

式 1:

$$s = \frac{2n_t}{n_x + n_y}$$

nx 代表字串 X 的雙元字串(Bigram)數目，ny 代表字串 Y 的雙元字串數目，nt 則是分別出現在 X 與 Y 字串的雙元字串數目，計算的結果會出現介於 0 到 1 之間的值，分別代表完全不同與完全相似。以類別圖與原始碼的類別名稱比對為例，表 8 顯示幾個不同計算區間的結果，套用 Dice Coefficient 的計算，除了有量化的數值依據，更可以省去對於文字比對所需要做的前置處理，例如字串的切割等，能夠找出潛在的相同或是相似名稱之軟體產出物，另一方面當使用者寫入錯誤之名稱時，也有利於校正，在本計畫的系統中，我們預設比對顯示的結果是大於 0.5 小於 1 之間，以避免使用者得到過多不必要之資訊。

表 8 類別圖與原始碼類別名稱的文字相似度比對

類別圖	原始碼類別名稱	相似度
ConnectionServ	ConnectionServer	0.92
	Connection	0.81
	CnnServ	0.52
	Server	0.33
	Quantization	0.25
	StreamingServ	0.24

	Camera	0.11
	HuffStatistics	0.07

4. 可再用元件庫

針對各相關功能進行模組化之關聯性探討、模組分析與設計、整合分析、原型發展後，設計一套具轉換性質的平台。

我們設定的再使用元件庫為一 XML 資料庫，其中的再使用軟體元件(Reusable Components，RC)，是蒐集程式的知識並產生原始述語(Primitive Predicate)，我們用來替 Reusable Components 建立正規化的知識，使其成為正規化軟體元件(FRC)及定義相關之推理規則，並將程式 Artifacts 及 FRC 存入相對的 Reuse Component Library 中，藉由搜尋樹產生 Artifacts，正規化之再使用元件(Formal Reusable Components, FRC)，然後儲存至再使用程式庫中，並以一有效率的元件存取機具(Library Retrieving Tool, LRT)，提供後續擷取所需。

此系統另外其中包含了二個組件：

a. 再使用程式庫和資料庫擷取協尋工具(Reuse Library and Library Retrieving Tool，LRT)

使用元件程式庫(Reuse Library)是用來存放處理中或處理後的再使用元件。

我們透過 OO 觀念，配合 XML 資料庫使用 QUERY 語法索引，目標是呈現一套使用者透明化、低維護需求、組織良好之軟體再使用元件高效能資料庫與資料管理系統。可提供使用者與系統工具間對軟體再使用元件之增減修改等等操作，可作為讓各階段各工具存取資訊之中間儲放所。

資料庫擷取協尋工具(LRT)，它的作用在於提供一套正規化的元件擷取(Retrieving)協助方式，經由述語編碼的量化資訊，將使用者所需要的元件搜尋出來。

b. 元件再利用編輯器(Reusable Component Editor，RCE)

此編輯器可列出元件及它們間的相互關係，從 XML 資料庫中抓出所需的項目列表，

可以幫助我們達到元件再利用性及各階段之模型鏈結以達到模型之一致性。除了可檢視元件關係外，使用者也可以在編輯器中進行修改元件相互關系，並再儲存進元件程式庫中。

5. 系統實作

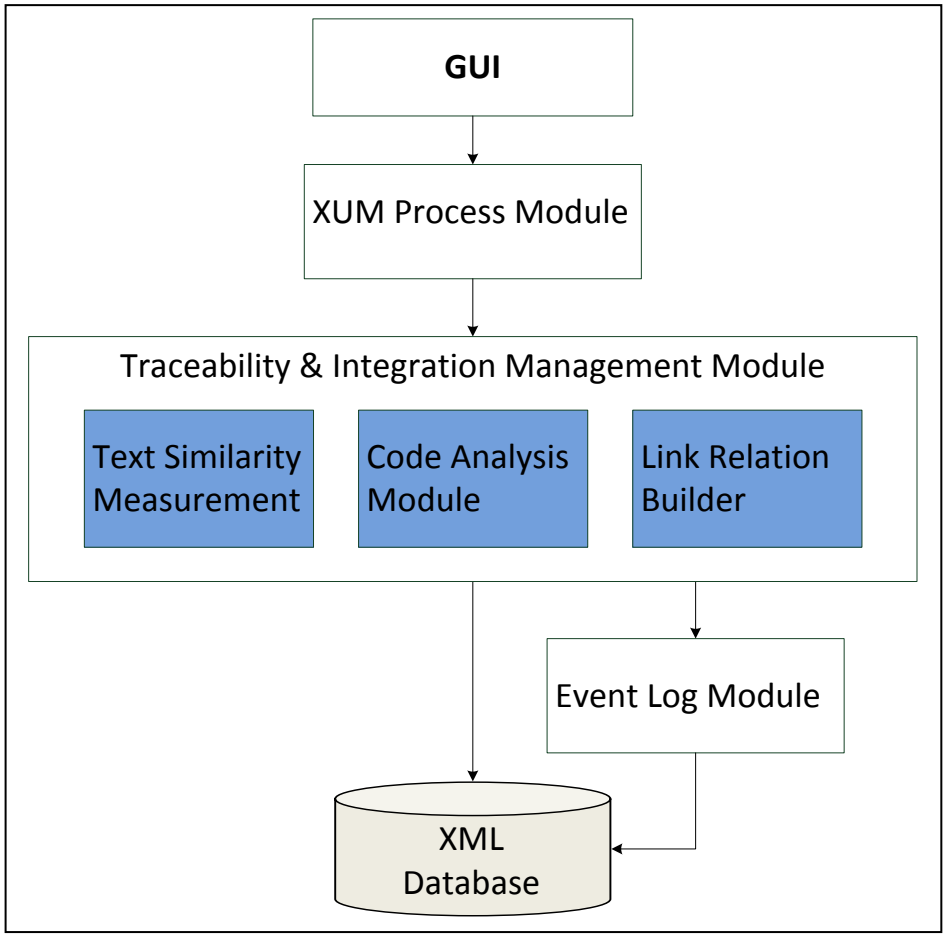


圖 40 系統架構

本系統之系統架構如圖 40 所示。

主要可以分成五個部分，簡述如下：

- 使用者介面部分：提供使用者視覺化的介面，觀看模型間的關聯性，事件的紀錄等。
- XUM 處理模組：針對 XUM 檔案不同之情況，進行寫入寫出之動作。
- 追蹤整合管理模組：比對不同模型間之關聯，找出相同或相似之字串。
- 事件記錄模組：記錄操作過程中產生之事件。
- XML 資料庫：儲存所有的模型資料、XUM 檔案、事件記錄等。

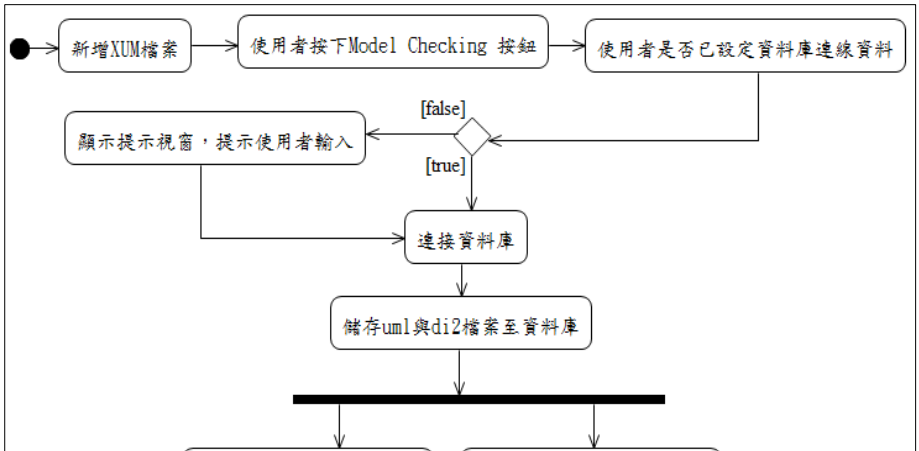


圖 41 系統活動圖

本系統的流程如圖 41 活動圖所示，使用者必須先在 Papyrus 的專案下，建立一個副檔名為 XUM 的檔案，接著按下檢查的按鈕，系統會抓取專案下所有.uml 與.di2 的檔案，並儲存至資料庫內，接著將 UML 的模型與 XUM 的整合模型進行比對，將相關的資訊寫入 XUM 檔案中，最後顯示給使用者。

以下是 VMC-MUM 系統操作畫面截圖。首先使用者可以透過本系統之新增檔案的 Wizard，加入 XUM 的檔案，如圖 42 所示。

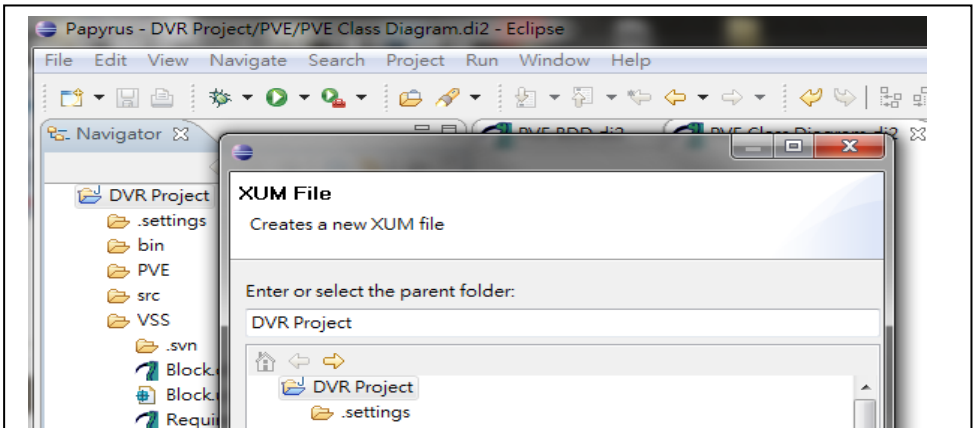


圖 42 新增 XUM 檔案 Wizard

在 XUM 的檔案建立後雙擊此檔案，Eclipse 的環境會根據此副檔名開啟對應的編輯器或是工具列按鈕，如圖 43，本系統包含了編輯器、觀點等功能，在首次開啟時，其內容並無任何資料存在，使用者必須藉由點擊 Model Checking 的按鈕，進行分析，系統會抓取在此專案目錄下，所有.uml 與.di2 的檔案並儲存至資料庫內，再進一步寫入部分必要之資訊以及彼此之關聯到 XUM 的檔案中。

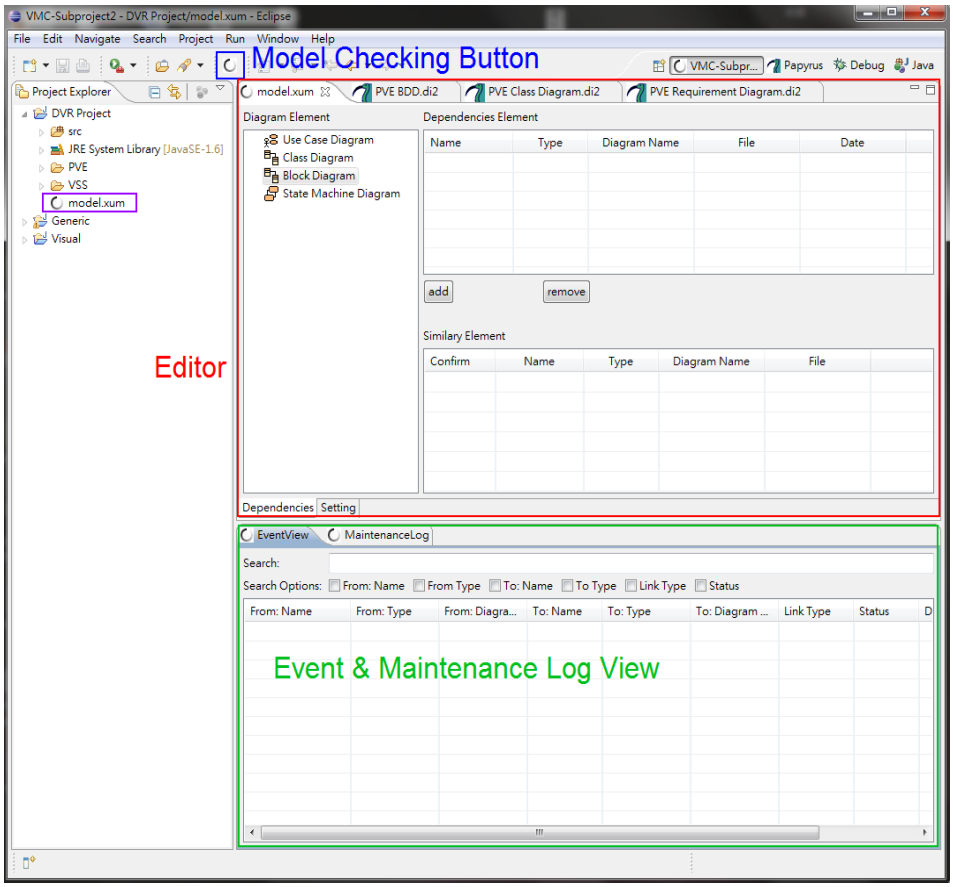


圖 43 系統執行畫面

在模型分析完成後，編輯器會自動更新資料，如圖 44 所示，左邊的樹狀結構顯示所有的 UML 項目，點擊其中任一項目，在右邊的上方表格顯示與此名稱一致之項目，右邊的下方表格，則是利用相似度比對，找出可能潛在結果，使用者可以在右下方的表格，透過勾選 Checkbox 進行確認。另外，考量到使用者可能需要進行手動變更的需求，使用者也可以透過 Add 的按鈕自行手動加入相關連性的項目，透過 Remove 的按鈕將不相關的項目移除，如圖 45 所示。

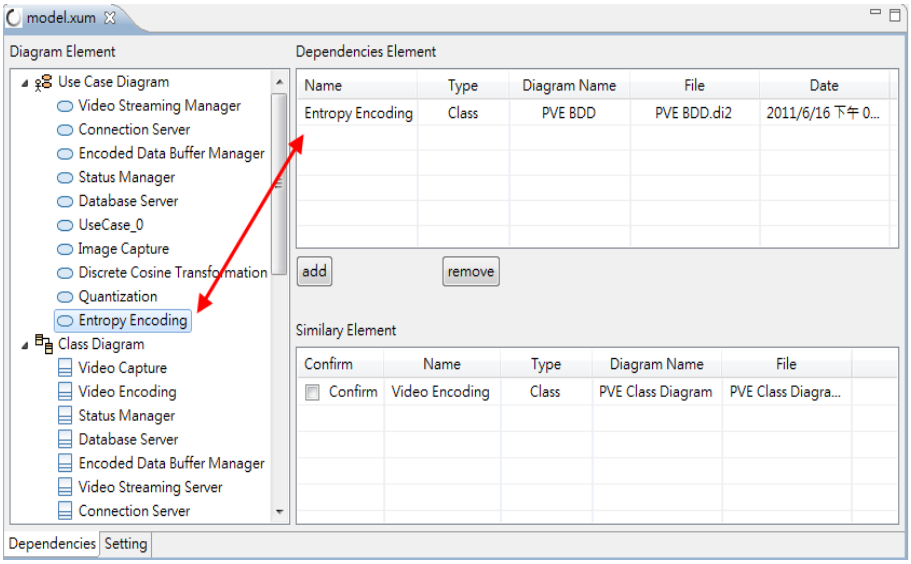


圖 44 模型的關聯

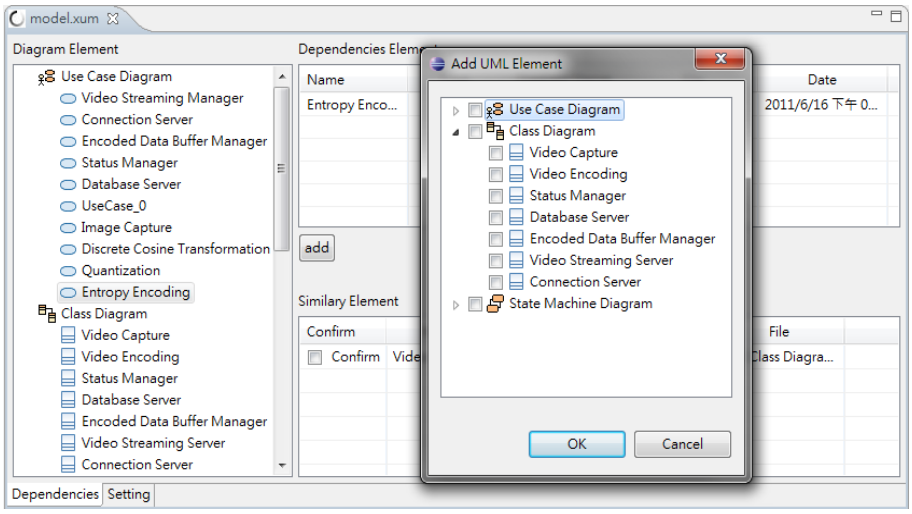


圖 45 手動修改關聯

系統分析模型的過程中，所發生的事件會自動的記錄下來，如圖 46 所示，由 Link Event 的 View 去顯示。使用者可以搜尋或是排序特定的欄位，來獲取不同的訊息，

如圖 47，使用者經由搜尋 From: Type 的欄位，了解所有 Use case 與其他 UML 模型間的關係。

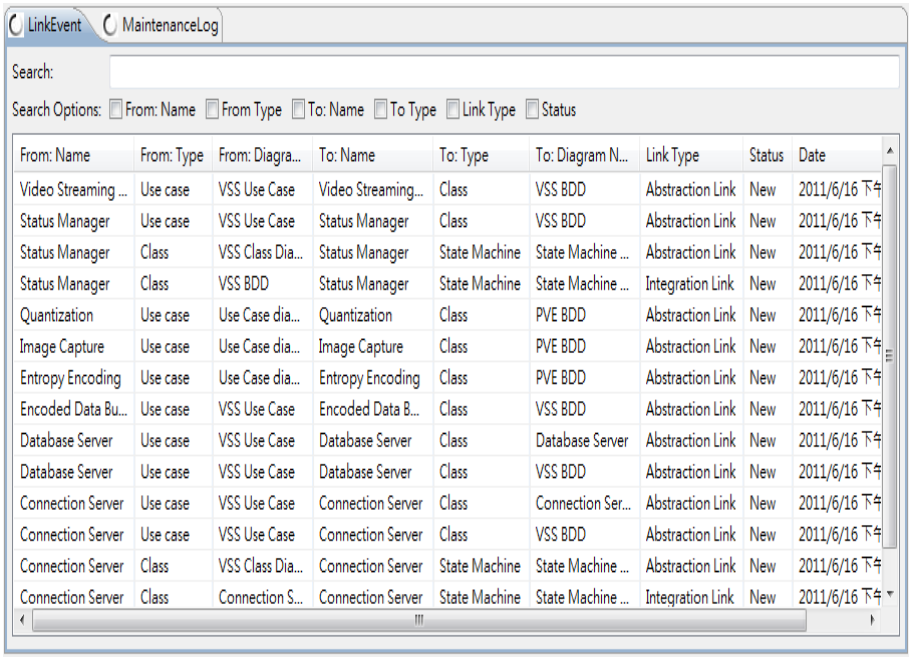


圖 46 Link Event View

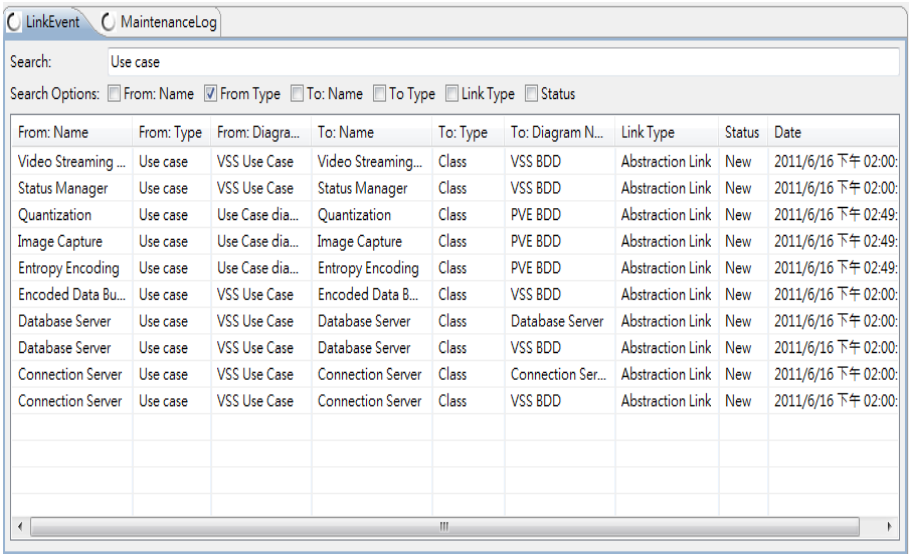


圖 47　搜尋特定 Link Event 欄位

圖 48 則顯示了使用者針對 UML 模型修改的紀錄，這樣的紀錄有助於使用者了解前後版本的差異。

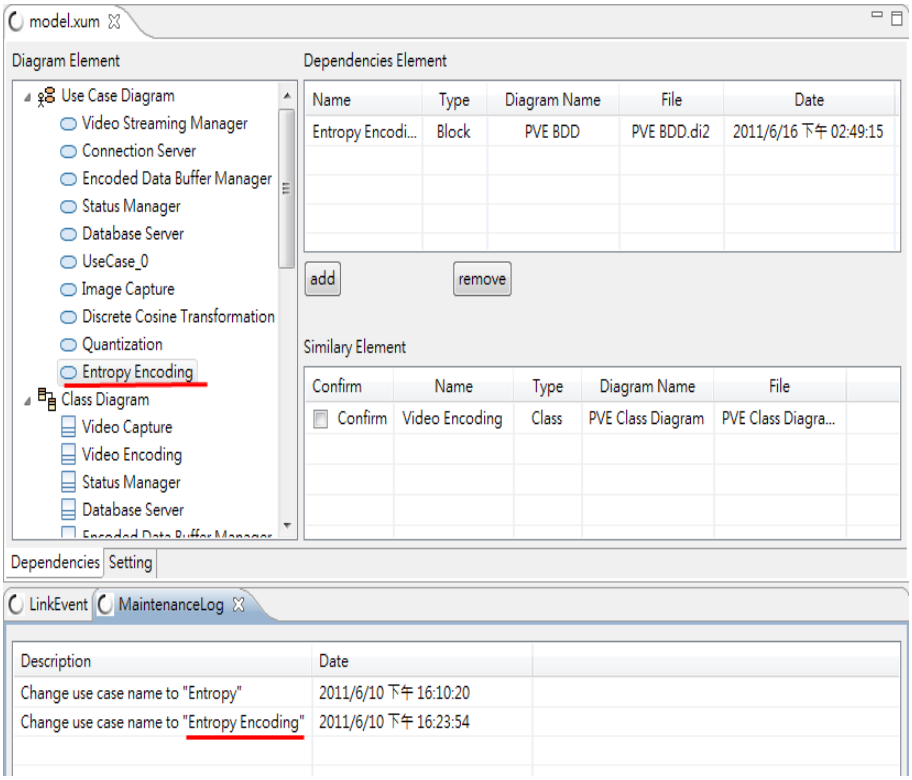


圖 48Maintenance Log View

C. 結論

在本計畫中，我們建立的多重觀點整合模型與可利用元件庫，分析 XMI 以及圖形交換格式，將所需的資訊擷取並儲存到 XUM 的模型，建立垂直與水平的一致性確認機制，針對不同的抽象層級，使用不同的鏈結型態進行串連，並且在分析的過程中，不同事件的發生都會被記錄下來，使得系統開發人員能發現系統開發過程中潛在的錯誤，達到軟體的追溯性，利用 MODEL 的特性將軟體的知識加以儲存利用，使得組織在開發專案時，能

夠將模型套用到嵌入式系統軟體的開發流程上，使得組織的經驗以及知識能夠被累積與再使用。

此外，本計畫與 VMC 其他各子計畫成員曾於中正大學分別舉辦過 Workshop 與今年 8 月 17，18 日為期兩天的 Summer School，分享 VMC 整合計畫的系統開發過程經驗，讓其他對於 Multi-core 嵌入式系統開發有興趣的開發者或在學學生對此相關領域做進一步的了解與學習。

III. VMC-DSS：設計樣式支援系統:

A. 研究目的

隨著多核心處理器架構漸漸取代傳統單核心處理器，開發軟體的方式與技巧也必須跟著轉變，而傳統的循序式執行的方式已經不足以完全發揮多核心處理器的效能，必須導入平行處理的概念才能有效利用多核心的優點。然而，對於熟悉傳統循序式設計的軟體開發人員而言，實現平行運算的技術勢必遭遇到許多的困難。因此，我們提出模型驅動及樣式支援的方法來加速多核心軟體的開發，並且降低開發人員設計多核心軟體的門檻。此外，如何同時確保多核心軟體的品質也將會是一大挑戰。在軟體工程的領域中，應用設計樣式一直是一個提升軟體品質的有效方法，並且有許多學者致力於這方面的研究，各種不同的設計樣式更是推陳出新。樣式的概念是由 Alexander 在 1977 年所提出，並且由 Gamma 等四人將樣式的概念應用在軟體設計上，他們提出了多種解決不同情境問題的設計，稱之為設計樣式。設計樣式是一種被分析並且歸類的設計框架，用以解決軟體設計的特定問題。在過去多核心軟體的開發往往都要從程式碼階層開始，這種開發方式造成開發者的視野被侷限於程式碼的邏輯上，無法綜觀整個多核心軟體設計的全局。近年來，有多位學者提出模型驅動架構（Model-Driven Architecture）的概念，其目的是為了使軟體的實作更趨近於軟體的設計以便於開發及維護，利用自動或半自動的轉換方式從高階的設計塑模產生不同階段的設計乃至於程式碼[14-16]。藉由模型驅動的優點，我們希望能夠加速多核心軟體的開發，並且經過套用樣式來提升多核心軟體的品質。另外，在過去多核心軟體開發的討論較多集中於函式庫所提供的 API，如何將多核心軟體相關的 API 提升至設計階層，並歸納出一些可供套用的設計樣式也是本研究想要解決的問題之一。

B. 研究成果

1. 規格設計

如圖 49 所示，本研究希望能夠協助多核心軟體開發者從初始設計（Raw Design）套用樣式，使塑模成為進階設計（Enhanced Design），接著藉由模型轉換產生平台

規格化設計（PSM Design），並利用多核心的平行觀察框架延伸設計系統，使得開發出來的系統能夠更容易被監控其平行化的效率。

樣式（Pattern）規格設計

如圖 50 所示，從結構面來看一個樣式規格包含了三個元素，分別是：

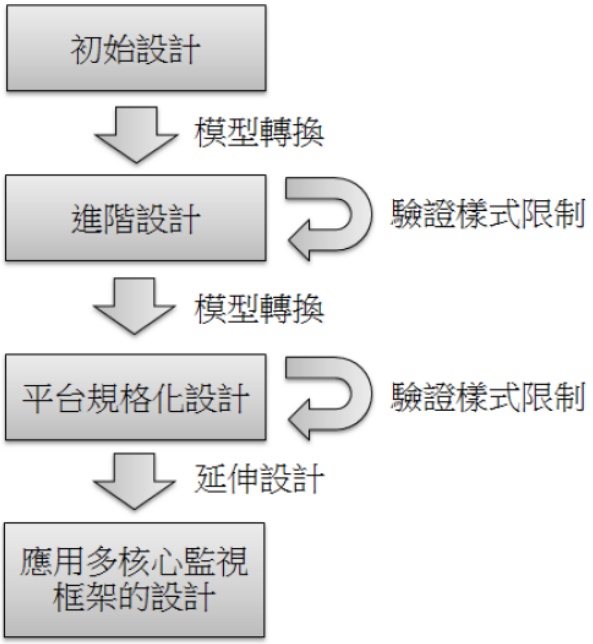


圖 49 研究方法的應用流程

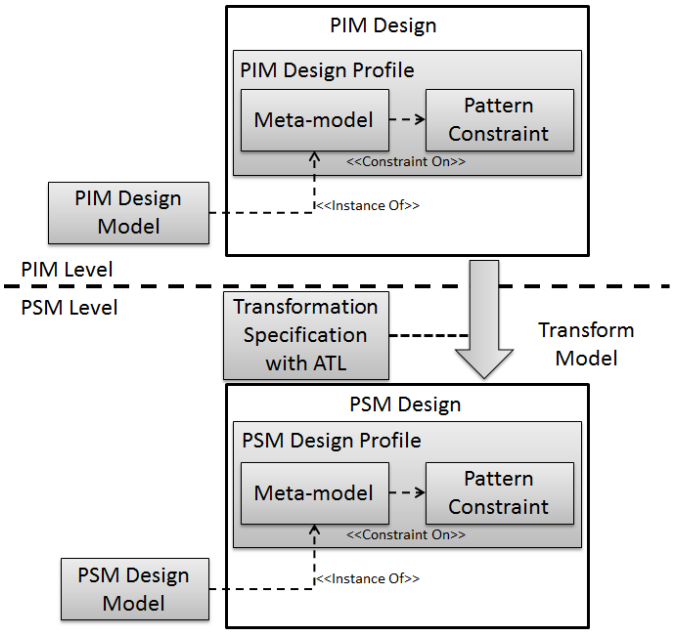


圖 50 樣式規格化 MDA 設計架構

- **超模型（Meta-model）**：在樣式的超模型階層中，我們定義樣式中所有的角色及屬性，根據樣式的結構訂定刻板 (Stereotype)以及標籤值 (Tagged Value)，並

且針對每個角色之間的關係建構出刻板與刻板之間的關聯 (Relationship)。超模型將會與限制一同被包裝在 UML Profile 中。

- **轉換規格 (Specification Transformation)**：不論是從初始設計轉換到進階設計；亦或是從 PIM 轉換到 PSM，都需要藉由轉換規格來進行模型的轉換以進入到下一個設計階段。在我們的方法中使用了 ATL 來描述樣式模型轉換的規則。
- **樣式限制 (Pattern Constraint)**：樣式限制用來表示樣式中角色的屬性及角色之間的關係限制，在實作上我們使用物件規範語言來限制超模型中刻板的標籤值以及刻板之間的關係，當模型在模型階層應用了刻板之後便隱含了在超模型中所訂定的樣式限制。

樣式規格化步驟分別是：

- 定義超模型：識別來源及目標的樣式模型結構並為樣式角色定義刻板及標籤值。
- 定義轉換規則：定義來源及目標之間的轉換規則。
- 定義驗證規則：根據樣式限制描述樣式驗證規則。

我們將會在後面的 5.4 節以一個 TBB 管線樣式範例來解釋樣式規格化的流程。

2. 樣式套用方法

套用樣式的方法分為兩種，分別是樣式模型轉換以及引用設計樣板，以下將對這兩種套用方式及使用時機做更進一步的說明。

樣式模型轉換

樣式模型轉換是將一個樣式區分為來源設計以及目標設計兩個階段，即是圖 51 中的 Source 與 Target 兩個部分，這兩個設計階段分別使用超模型描述其擁有的角色與結構，並且由一個轉換規格 (Transformation Specification)來定義其轉換的規則。樣式模型轉換可以應用在 PIM 轉換到 PSM；以及初始設計轉換到進階設計這兩種不同的來源與目標。

引用設計樣板

引用設計樣板是為了補足樣式模型轉換的使用限制，當樣式沒有一個明確的初始設計時，我們可以根據樣式的規格定義產生出一個基礎的設計樣板供使用者套用。設計樣板是一個模型層的設計模型，而該模型會套用超模型層的樣式規範，並且符合超模型層所定義的 OCL 規則(如圖 52)。

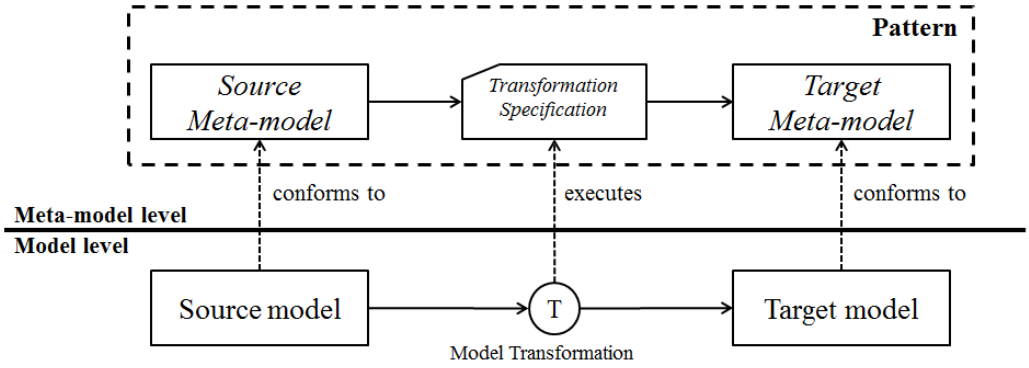


圖 51 樣式模型轉換方法

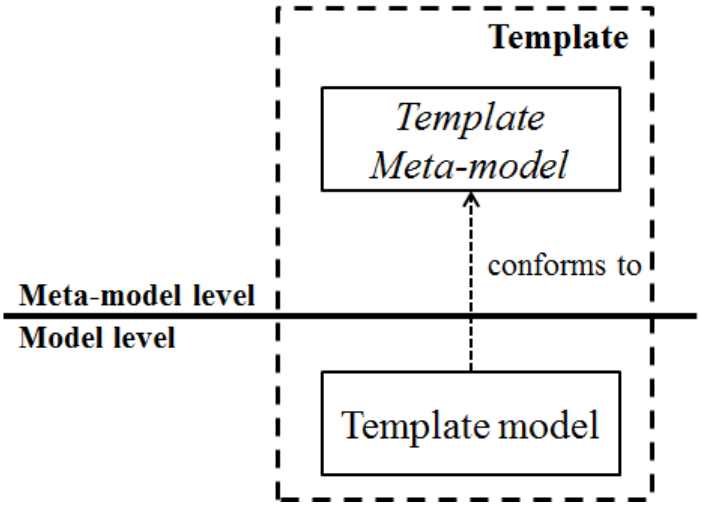


圖 52 引用設計樣板方法

3. 樣式應用後之驗證

在樣式被應用到設計之後，開發者會針對需求修改其設計，如此將可能會產生不符合原本應用該樣式目的的變更。所以我們在樣式規格化時，根據樣式的目的與限制，利用 OCL 撰寫樣式的規則，以確保樣式在被應用之後還能保有其應用之功能。OCL 是一種敘述性的查詢語言，它可以被定義在超模型層 (M1 level)或模型層 (M2 level)上，而我們選擇將樣式的規則定義在 M1 層是著眼於它的可重用性，只要 M2 層的元素套用了 M1 層的刻板，便能夠利用 M1 層的 OCL 描述來限制 M2 層的規則，如此一來使用者在設計時便不需要重複定義 M2 層的 OCL 描述，只要依靠事先描述好的 M1 層 OCL 敘述，便能夠達到驗證的目的。圖 53 描述了一個樣式驗證的架構。

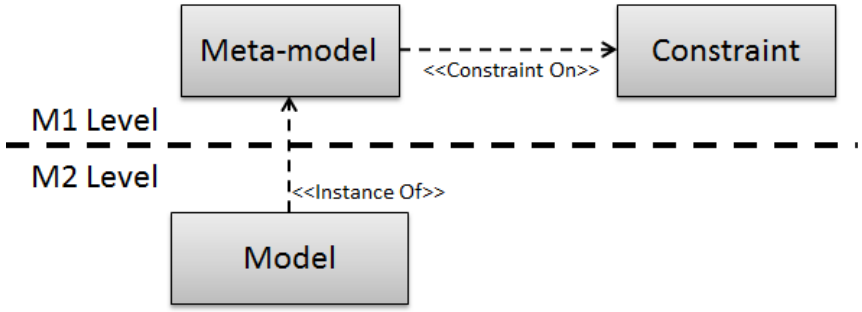


圖 53 樣式驗證架構

4. 範例說明：TBB 管線樣式

本節以 TBB 管線樣式來說明本研究方法的樣式規格化流程，以及如何從 PIM 階層的設計轉換到 PSM 階層的設計。圖 54 表示了一個管線運作的範例，其中橫軸的部分代表不同的工作階段 (Stage)，而 TBB 程式庫將一個管線中的工作階段稱做一個篩選器 (Filter)，縱軸的部分表示了時間區間，色塊的部分代表了每個工作階段所需要完成的工作內容 (Token)。

由這個範例我們可以看到在 Time0 時紅色色塊在橘色色塊之前進入一個線性的篩選器 FilterA，所以 FilterA 必須先處理完紅色色塊的工作內容之後才能繼續處理橘色色塊的工作內容。而由於 FilterB 是平行的篩選器，所以在 Time3 的時候，橘色色塊比紅色色塊提早進入 FilterC 是有可能的。一個線性的篩選器必須依照工作內容進入的先後順序來處理工作內容，而一個平行的篩選器則可以平行處理所有進入的工作內容。

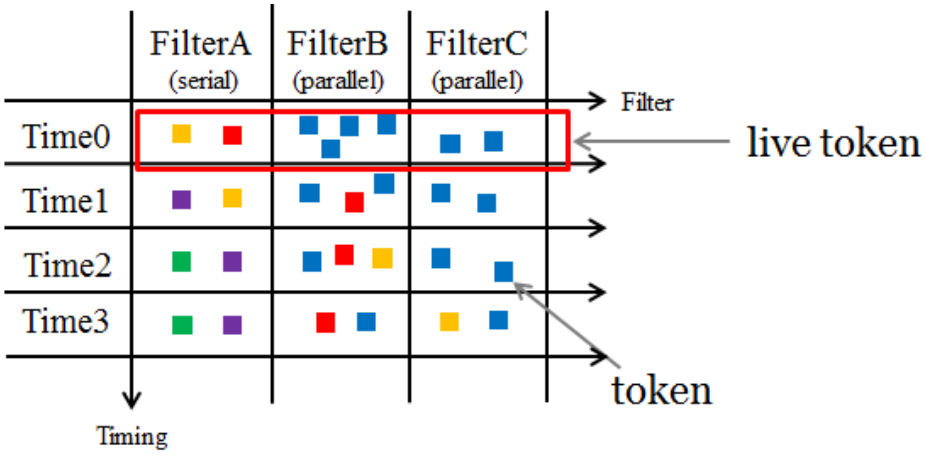


圖 54 管線運作示意圖

a. 樣式與樣板規格化

樣式規格化

步驟 1：定義超模型

圖 55 描繪了一個 TBB 管線樣式平台獨立設計的超模型，由於 TBB 管線樣式平台獨立設計主要是由 Client 及 Delegate 等類別所組成，所以其超模型將會包含 Client 及 Delegate 等角色，其中 Client 的 Operation 會將工作委派給 Delegate 的 SubOperation 去執行，類似管線中不同的工作階段。而表 9 則表示了這個超模型的參與角色以及擴展了那些 UML 超模型。

圖 56 則描繪了一個 TBB 管線樣式平台規格化設計的超模型，而表 9 表示了這個超模型的參與角色以及擴展了那些 UML 超模型。我們可以看到 TBB 管線樣式平台規格化設計主要由 Client、tbb:pipeline 及 tbb:filter 等類別所組成。值得一提的是在 TBB 管線中，呼叫 run 函式必須傳入一個 max_number_of_live_tokens 的整數參數，用以限制所有工作階段中能夠處理的工作內容總和，因此，在 run 這個刻板中將會有一個標籤值來表示這個參數。另外，每個 filter 刻板也會有一個 order 的標籤值來識別工作階段的先後順序。

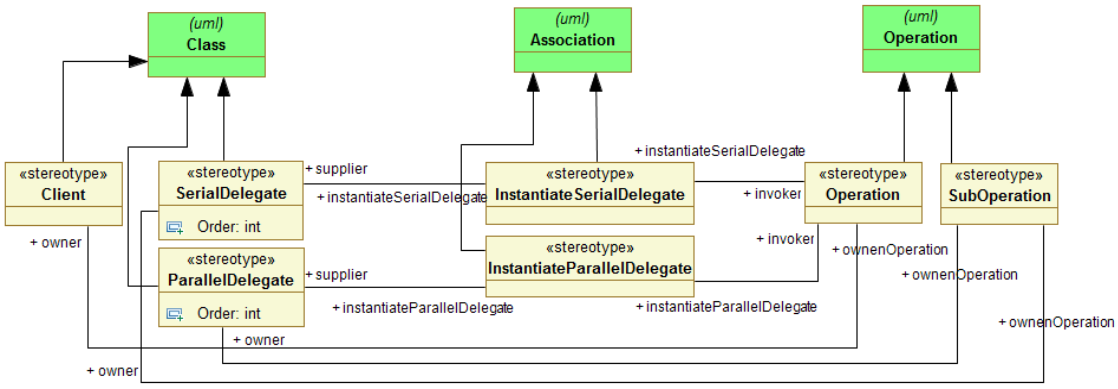


圖 55 TBB 管線樣式平台獨立設計的超模型（TBB-Pipeline PIM meta-model）

表 9 TBB 管線樣式平台規格化設計的參與者資訊

UML meta-model	Stereotype	Tagged-value	Description
Class	*client*		管線樣式中的使用者端。
	tbb::pipeline		定義 TBB 管線的角色。
	tbb::filter		定義 TBB 工作階段的角色。
	ConcreteFilter	*filter_order*: int *filter_mode*: mode	定義具體工作階段的角色，其中 *filter_order* 定義了該工作階段的運作順序，而 *filter_mode* 則定義了該工作階段的運作模式是平行或是線性的。
Generalization	*ConcreteFilter ExtendFilter*		具體工作階段繼承 TBB 工作階段的一般化關係。
Association	*InstantiateFilter*		由 TBB 管線使用到 TBB 工作階段的關係。
Operation	*run*	*max_number_of_live_tokens*: int	TBB 管線執行管線任務的操作，其中 *max_number_of_live_tokens* 定義了所有工作階段中能容納的工作內容總和。
	clear		TBB 管線清除所有管線任務的操作。
	add_filter		TBB 管線加入新的管線任務的操作
	FilterConstructor		工作階段的建構子操作。
	operator		工作階段所執行的工作內容操作。

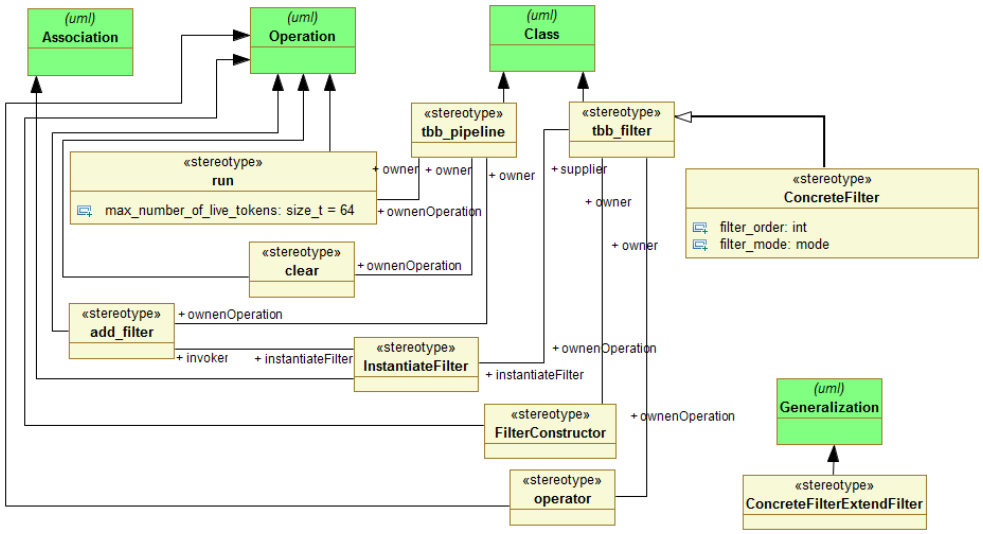


圖 56 TBB 管線樣式平台規格化設計的超模型 （TBB-Pipeline PSM meta-model）

步驟 2：定義轉換規則

在定義完來源及目標的超模型之後，我們將根據樣式的需求使用 ATL 來描述樣式模型轉換的規則，在表 11 中表示了 TBB 管線樣式從平台獨立設計轉換到平台規格化設計的角色對應關係。套用了圖 55 中 Client 的實例在經過模型轉換之後會產生套用圖 56 中 Client、tbb:pipeline 以及 tbb:filter 等刻板的實例。而套用 serialDelegate 及 parallelDelegate 的實例分別會產生出套用 ConcreteFilter 的實例，

並且設定其對應的標籤值。圖 57 使用 ATL 程式碼描述了 serialDelegate 的部分轉換規則。

步驟 3：定義驗證規則

針對 TBB 管線樣式的特徵，本節列舉幾個該樣式的平台規格化設計所要遵守的規則：

- 在 TBB 管線中的每個工作階段都必須繼承 filter 類別。
- 在一個 TBB 管線中，工作內容的總數必須大於等於工作階段的數量。
- 工作內容的順序編號不能重複。

對應以上三條規則，圖 58 使用 OCL 語法來描述 TBB 管線樣式平台規格化設計的限制規則。

表 10 TBB 管線樣式平台規格化設計的參與者資訊

UML meta-model	**Stereotype**	**Tagged-value**	**Description**
Class	*client*		管線樣式中的使用者端。
	tbb::pipeline		定義 TBB 管線的角色。
	tbb::filter		定義 TBB 工作階段的角色。
	ConcreteFilter	*filter_order*: int *filter_mode*: mode	定義具體工作階段的角色，其中 *filter_order* 定義了該工作階段的運作順序，而 *filter_mode* 則定義了該工作階段的運作模式是平行或是線性的。
Generalization	*ConcreteFilter* *ExtendFilter*		具體工作階段繼承 TBB 工作階段的一般化關係。
Association	*InstantiateFilter*		由 TBB 管線使用到 TBB 工作階段的關係。
Operation	*run*	*max_number_of_live_tokens*: int	TBB 管線執行管線任務的操作，其中 *max_number_of_live_tokens* 定義了所有工作階段中能容納的工作內容總和。
	clear		TBB 管線清除所有管線任務的操作。
	add_filter		TBB 管線加入新的管線任務的操作
	FilterConstructor		工作階段的建構子操作。
	operator		工作階段所執行的工作內容操作。

表 11 TBB 管線樣式角色對應關係

	來源模型參與者	**標籤值**	**目標模型參與者**	**標籤值**
參與者	*client*		*client*	
			tbb:pipeline	
			add_filter	
			run	*max_number_of_live_tokens*: int
			clear	
			tbb:filter	
	serialDelegate	*order*: int	*ConcreteFilter*	*filter_order*: int *filter_mode*: mode (=serial_in_order)
	parallelDelegate	*order*: int	*ConcreteFilter*	*filter_order*: int *filter_mode*: mode (=parallel)

```
rule SerialDelegate {
    from s : UML2!"uml::Class" (s.hasStereotype('SerialDelegate'))
    to concreteFilter : UML2!"uml::Class" (
        package <- thisModule.Global_UmlModel,
        name <- s.name
    ),
    gen : UML2!"uml::Generalization" (
        specific <- concreteFilter,
        general <- thisModule.Global_filter
    )
    do {
        concreteFilter.applyStereotype(UML2!Stereotype.allInstancesFrom('OUTPRO')->
            select(p | p.name = 'ConcreteFilter')->first());
    }
}
```

圖 57 TBB 管線樣式 ATL 程式碼 （serialDelegate 轉換部分）

Rule 1

context ConcreteFilter inv:

self.base_Class.generalization.target.getAppliedStereotypes()
-> exists(name='Filter')

Rule 2

context tbb:pipeline inv:

self.max_number_of_live_tokens >= (tbb:filter.allInstances()
-> size())

Rule 3

context tbb:filter inv:

not((tbb:filter.allInstances() -> excluding(self)).filter_order
-> includes(self.filter_order))

圖 58 TBB 管線樣式平台規格化設計的 OCL 驗證語法範例

在 Rule 1 中限制套用<<ConcreteFilter>>的類別必須繼承套用<<filter>>的類別，所以一般化的目標必須存在一個叫做 filter 的刻板。而在 Rule 2 中則定義標籤值 max_number_of_live_tokens 必須大於等於所有繼承套用<<filter>>實例的總和。在 Rule 3 中定義所有的標籤值 filter_order 的值必須是不重複的，所以除了執行檢查本身的實例 (Instance)之外 (excluding (self))，其他的實例不能包含相同的 filter_order 值 (includes (self. filter_order))。

樣板 (Template)的規格化

步驟 1：定義超模型、2：定義轉換規則

由於樣板的規格化步驟 1、2 與樣式的規格化步驟 1、3 的目標超模型及目標樣式限制雷同，故在此不再贅述。

步驟 3：應用樣板刻板

我們在樣板規格化的最後一個步驟將會根據先前所定義好的超模型產生相對應的模型，並且儲存起來作為樣式樣板使用。圖 59 則是套用了 TBB 管線樣式平台規格化設計超模型的模型。

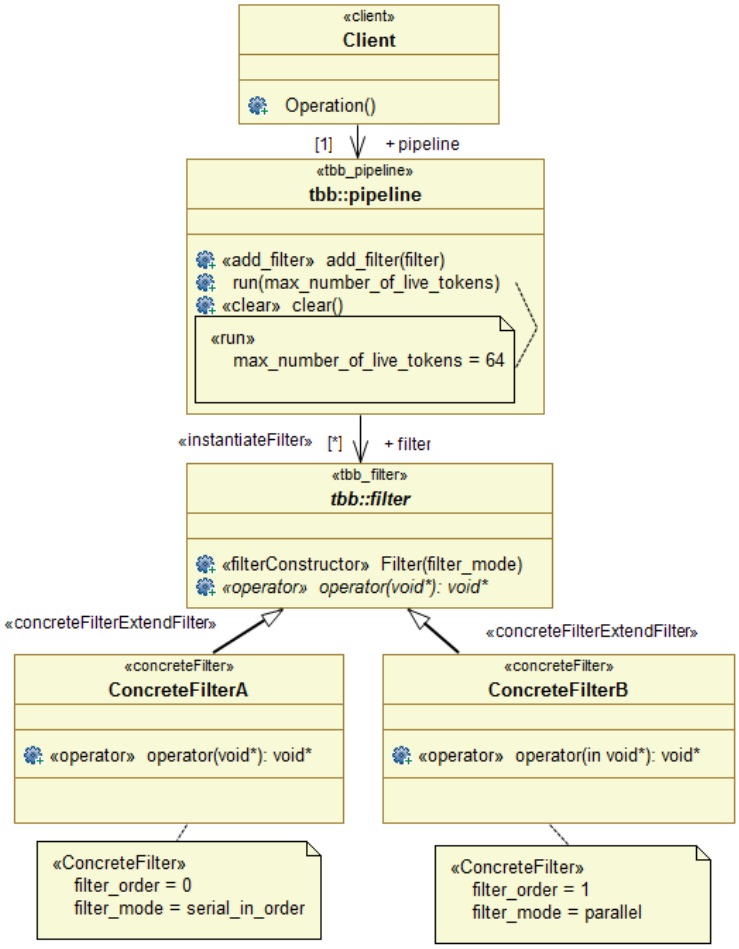


圖 59 套用 TBB 管線樣式平台規格化設計的模型 (TBB-Pipeline PSM model)

b. 樣式與樣板套用

圖 12 描繪了套用 TBB 管線樣式平台獨立設計的模型結構範例，經由執行樣式模型轉換後將會產生圖 61 描繪的模型結構。如果來源模型越複雜則越能夠顯現出樣式模型轉換的優點，因為有越多的來源參與者實體則能夠轉換出相對應數量的目標參與者實體。

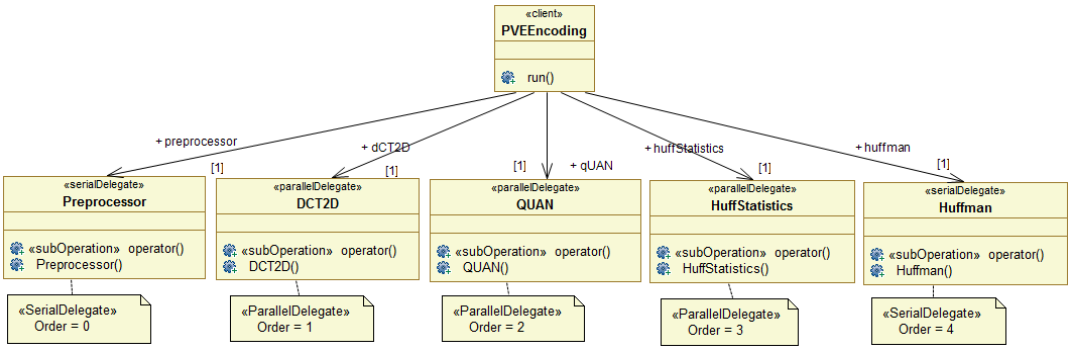


圖 60 TBB 管線樣式轉換前的模型範例

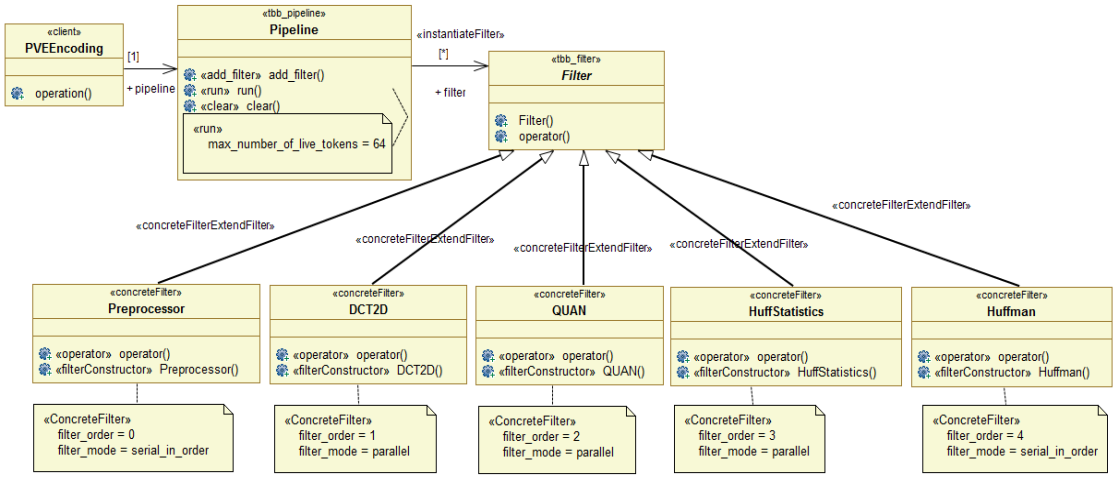


圖 61 管線樣式轉換後的模型範例

C. 結論

多核心軟體設計支援系統是建構在 Eclipse 平台上的一個插件 (Plug-in)工具，主要擴充了 Papyrus 的塑模功能，並且藉由了 ATL 模型轉換語言的幫助，使得套用樣式能夠更直覺迅速，從而建構一個符合 MDA 流程規範的開發環境。

本系統主要由五個子模組所組成，分別是：使用者介面、樣式管理、樣式應用、樣式驗證以及平行系統監控框架。如圖 62 所示，使用者介面及樣式管理主要是由 Eclipse 所提供的 UI 來呈現，而樣式應用則使用到 ATL 及 Papyrus 的部分功能；樣式驗證是修改 Papyrus 的 OCL 驗證功能而成；平行系統監控框架是一個獨立的軟體框架，用以提供實作系統的被監控能力。

由圖 63 可以更清楚地看到系統中的模組運作流程，首先我們透過樣式管理模組產生樣式的規格化文件，並且將其相關文件存進一個樣式儲存庫中。接著使用樣式應用模組將事先定義好的樣式引入工作區的設計模型中，並且進行細部設計。最後當設計完成時，我們會利用樣式驗證模組來檢驗被套用的樣式是否完整，並產生一份驗證報告。

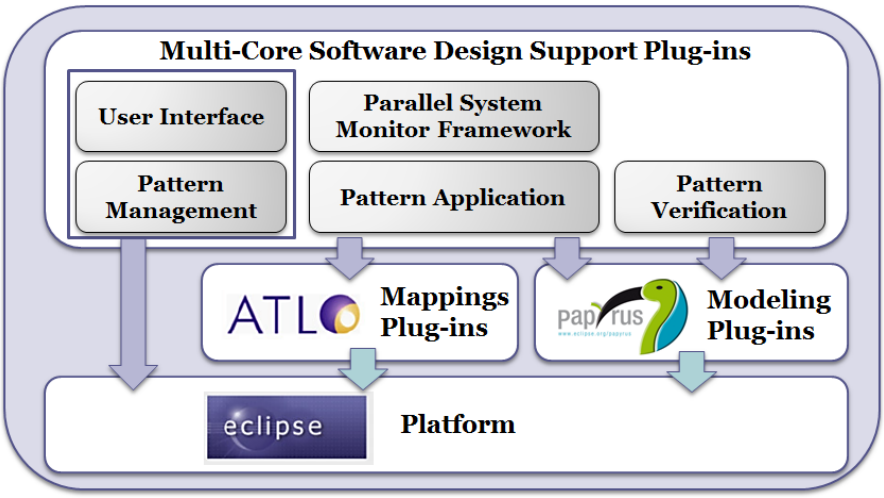


圖 62 多核心軟體設計支援系統架構圖

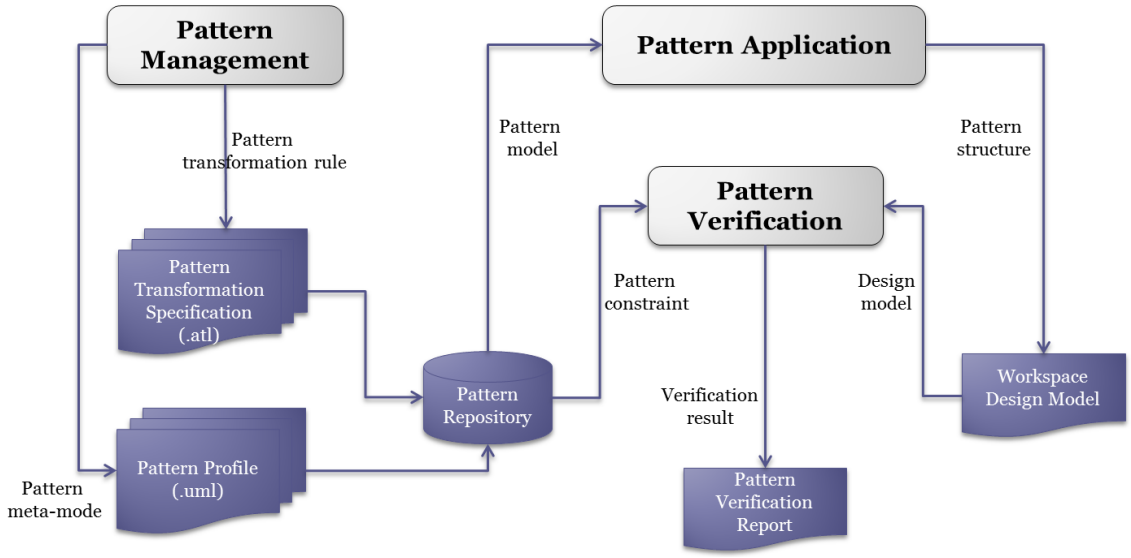


圖 63 多核心軟體設計支援系統運作流程圖

在本研究中介紹了一個模型驅動及樣式支援的方法用以幫助多核心軟體設計，我們對樣式進行抽象化，藉由以樣式為基礎的轉換方法，讓開發者達成快速開發的目的。藉由此方法減低人為錯誤的引發機會又可讓設計維持一定的品質。而這些抽象化後的樣式資訊與轉換規格儲存在樣式儲存庫後，在之後的開發專案中可被重複使用，可重用性大幅提昇。

在支援系統實作的部分，我們提供了一個無縫的設計環境來支援多核心軟體的開發，透過本系統可以降低開發人員設計多核心軟體的門檻，並且增加應用樣式的機會以提升軟體的品質。我們同時也說明了樣式的規格化流程，以增加日後系統的可擴充性，本研究也設計了多個多核心相關樣式供開發人員使用，藉由超模型層級定義轉換規則並進行自動化的轉換或是直接引用設計樣板的方式，使得這些規則可以不斷的被重複使用。

IV. VMC-SYN: 多核心嵌入式軟體合成及程式碼生成

A. 研究目的

本計畫(VMC-SYN)為國科會自由軟體研發推動計畫的第三年計畫。第二年計畫以第一年度從 bottom-up 為設計主軸的數位影像串流(Digital Video Recording, DVR)系統為基礎，我們設計了相關的平行模型(Parallel Model)以及設計模式(Design Pattern)，以應用在程式碼生成器(Multi-Core Embedded Software Code Generator)中，結合二大仲介軟體 Quantum Platform (QP)及 Intel Threading Building Blocks (TBB)這兩套 Libraries 來實現 Parallelized C++ Code。在去年度完成大部分的 Code Generation 功能後，第三年度我們重點放在加入 Monitoring Mechanism 來監控系統資源(Resource)與執行效能(QoS)，以期符合使用者給定的需求(User-specific Requirements)。我們也透過介面的制定說明與其他子計畫之間資訊的交換及所需的功能性互動，達到從上層 model，程式碼生成後到多核心目標平台上執行的完整性。

本團隊針對多核心嵌入式軟體的程式碼生成器設計了一套完整流程。根據第二年計畫的開發經驗，我們修改了程式碼生成器的流程，如圖 64 所示。基本上還是沿用了前兩年計畫所使用的二大仲介軟體 QP 及 TBB 來輔助產生程式碼，第三年修正後的程式碼生成器的主要功能如下：Model Parser (MP)、Model Compiler (MC)、Parallelism Implementer (PI)、Monitor Generator (MG)、Tree Translator (TT)。圖 65 為次要功能的關係圖。

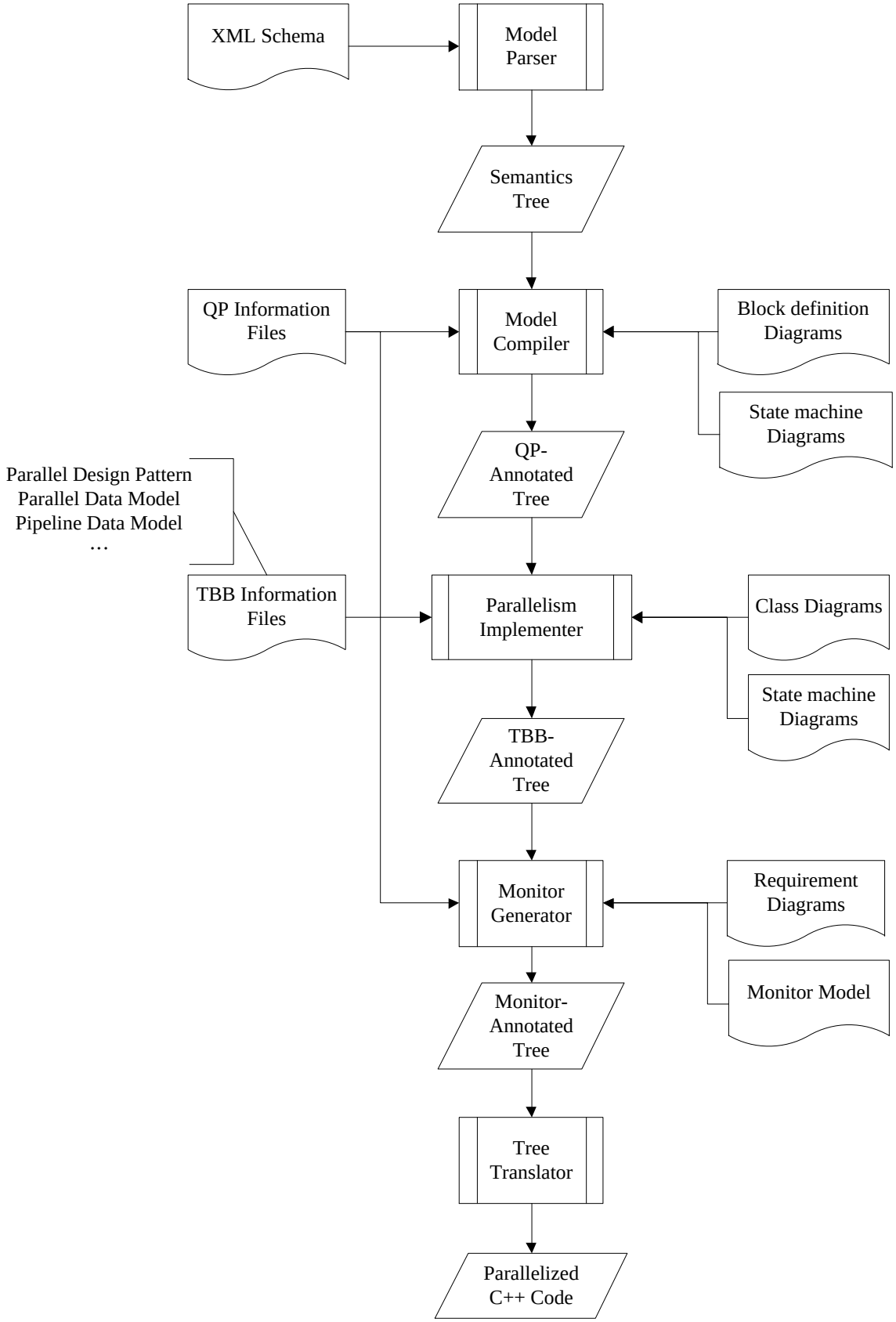


圖 64 程式碼生成器流程

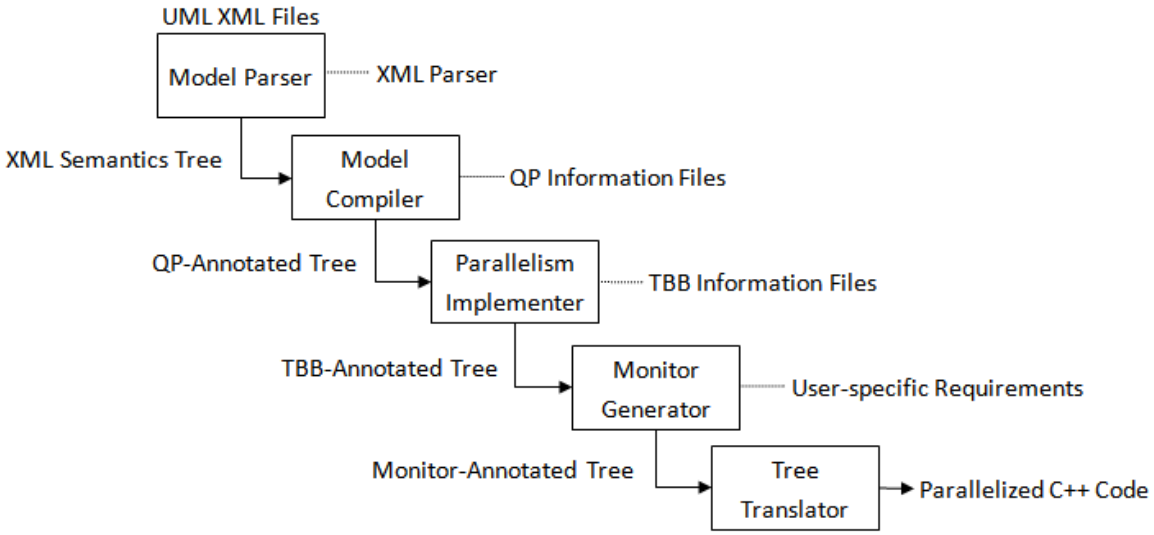


圖 65 程式碼成生器次要功能關係圖

Model Parser 主要負責從 High-Level Model 取得系統設計相關資訊，並將這些資訊建立出一個與程式碼生成相關的樹狀結構(XML Semantics Tree)。Model compiler 負責追?由 Model Parser 產生樹狀結構的節點，從中擷取系統設計所產生的 State Machine Diagram 的資訊，並將這些資訊轉成相對應的 QP 程式碼附加到適當的節點之中，此時稱該樹狀結構為 QP-Annotated Tree。而 Parallelism Implementer 則是負責在 QP-Annotated Tree 中找出使用者指定系統可平行化的 Model，並幫使用者轉出相對應的 TBB Code，並附加至相對應的節點之中，而完成該階段之後，我們稱此樹狀資料結構為 TBB-Annotated Tree。接下來的階段則是透過 Monitor Generator 來針對使用者需求，產生監控用的 Monitor Active Object 資訊，附加到相對應的節點中；完成這個步驟的樹狀結構，稱之為 Monitor-Annotated Tree。最後一個階段則是由 Tree Translator 走訪 Monitor-Annotated Tree，產出平行化的 C++ Source Code。

B. 研究成果

本計畫依照程式碼生成器流程來做設計，主要有 Model Parser (MP)、Model Compiler (MC)、Parallelism Implementer (PI)、Monitor Generator (MG)、Tree Translator (TT)五個功能，依序介紹如下：

Model Parser (MP)主要使用現有的 libxml2 這套 XML C Parser 負責從 High-Level SysML Model 取得系統設計相關資訊，並將這些資訊建立出一個與程式碼生成相關的樹狀結構(XML Semantics Tree)。

Model Compiler (MC)，是將使用者設計的 state machine diagrams 和描述系統功能的 class diagrams 翻譯成以 QP 程式架構的樹狀資料結構。每一個 state machine 在 QP 裡分別用一個 Active Object (AO)實現。為了要把 semantics tree 翻譯成帶有 QP 的程式架構，就需要 QP Information Files 的資訊，如此一來才能便於程式碼的自動生成。例如：使用者設計了一個 state machine，由 QP Information Files 得知把 state machine 對應成 AO，就實作來講，即為把此 state machine 宣告成 C++ Class，並繼承 QActive。

MC 的流程主要可分為以下三個部份：Tree Traverser、QP Transformer 以及 QP-Annotated Tree Generator。圖 66 為 MC 的架構圖。

MC 走訪 MP 所建的 semantics tree 上的每個節點，之後由 QP Transformer 根據每個節點的資訊來轉換出所相對應的 QP 程式碼。例如：遇到一個 state machine 的節點，就可以產生 QP Initialization 的程式碼來初始化 AO 和 AO 所需要用到的訊號；遇到 state machine 上的 state，就根據 QP Information Files 來產生相對應的 QP 程式碼；遇到 state machine 上的 transition，則是將 transition 上的 event 條件和 actions 依據 class diagram 所描述的功能，一併轉換成 QP 程式碼。

最後，每個節點透過 QPc Transformer 所轉換出來的 QP 程式碼，由 QP-Annotated Tree Generator 嵌入到 semantics tree 上，即為 MC 的輸出：QP-Annotated Tree。

Parallel Implementer (PI)主要功能為將系統設計時指定的平行模型(Parallel Model)，以 TBB 為應用介面，生成相對應的平行演算法程式碼。我們主要針對 Data flow parallelism 來設計平行程式碼之生成。

圖 67 為 PI 的設計架構圖，主要包含三個部分：QP-Annotated Tree Traverser、Parallel Model Translator、TBB-Annotated Tree Generator。PI 將會接收從上層 MC 而來的 QP-Annotated Tree 以及外部加入的 TBB Information Files，其中 TBB Information Files 包含使用者描述的 parallel model 用來產生 TBB parallel code，最後 PI 產生出 TBB-Annotated Tree 供下層的 Monitor Generator (MG)使用。

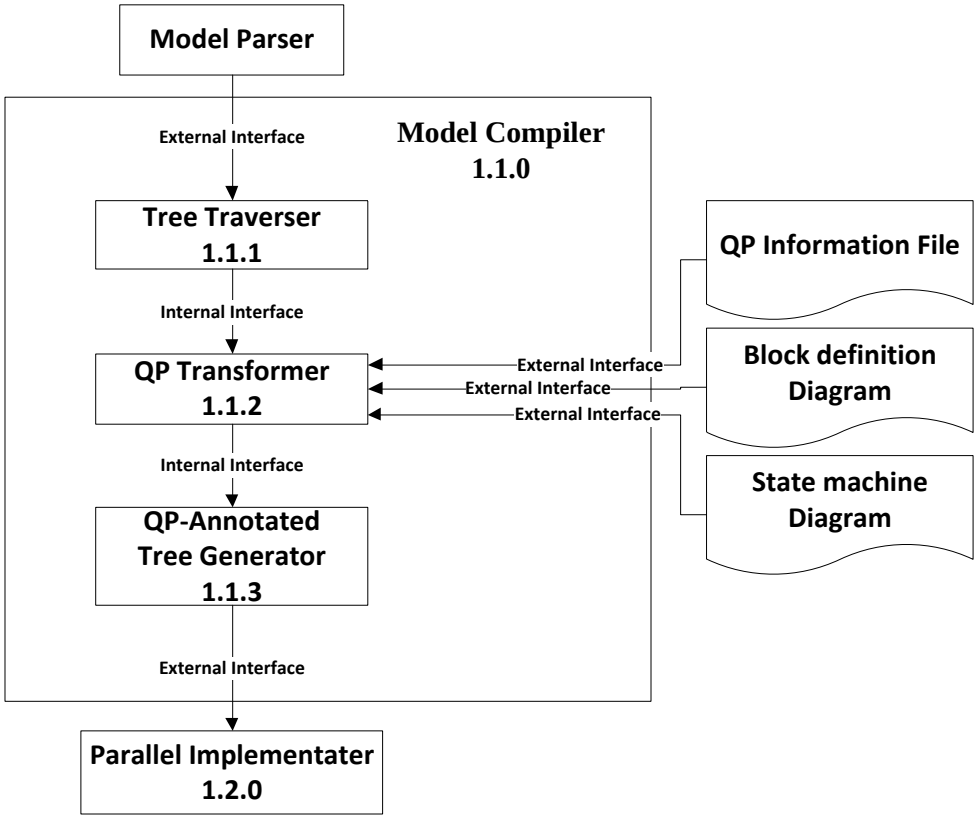


圖 66 MC 架構圖

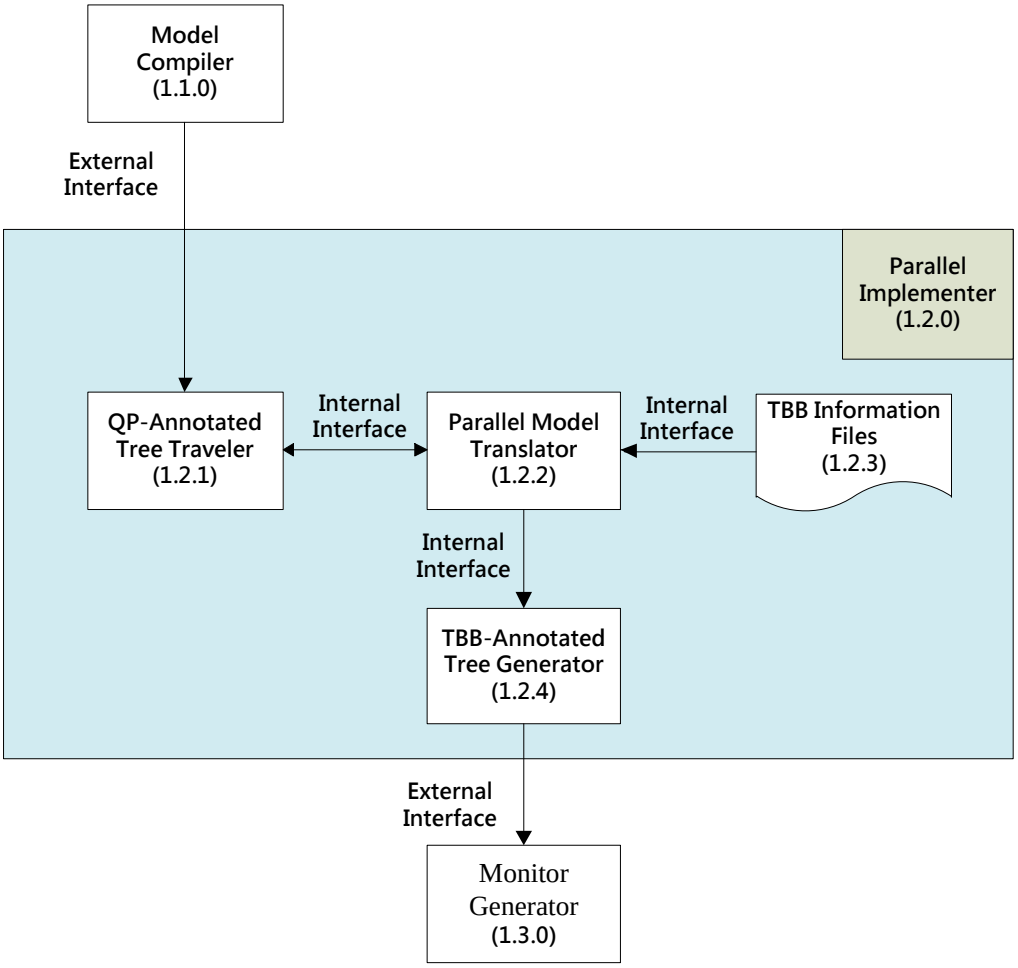


圖 67　PI 架構圖

Monitor Generator (MG)，將會接收從上層 Parallelism Implementer(PI)而來的 TBB-Annotated Tree 以及外部加入的 QP Information Files、Monitor Model 和 Requirement Diagrams。其中 Requirement Diagrams 包含使用者描述要 monitor 的資訊。主要的功能是要產生一個程式執行時期可以對 resource 做觀測的 Monitor active object，從使用者描述的 Requirement Diagrams 了解要 monitor 的 resource，依照 Monitor Model 來將 resource 和 Monitor 做對應並將程式碼加入 TBB-Annotated Tree 內，最後輸出 Monitor-Annotated Tree。

我們仍然使用 Quantum Platform (QP)來實作 Monitor，也就是說，Monitor 也會是一個系統中的一個 AO。這樣一來，Monitor 和系統中原有的 AOs 在溝通上可以沿用 QP 提供的溝通機制，使得各 AOs 和 Monitor 之間的互動 API 較為方便設計與整合。

MG 包含了三個部分：TBB-Annotated Tree Traverser、Monitor Model Translator 和 Monitor-Annotated Tree Generator。圖 68 為 MG 的架構圖。

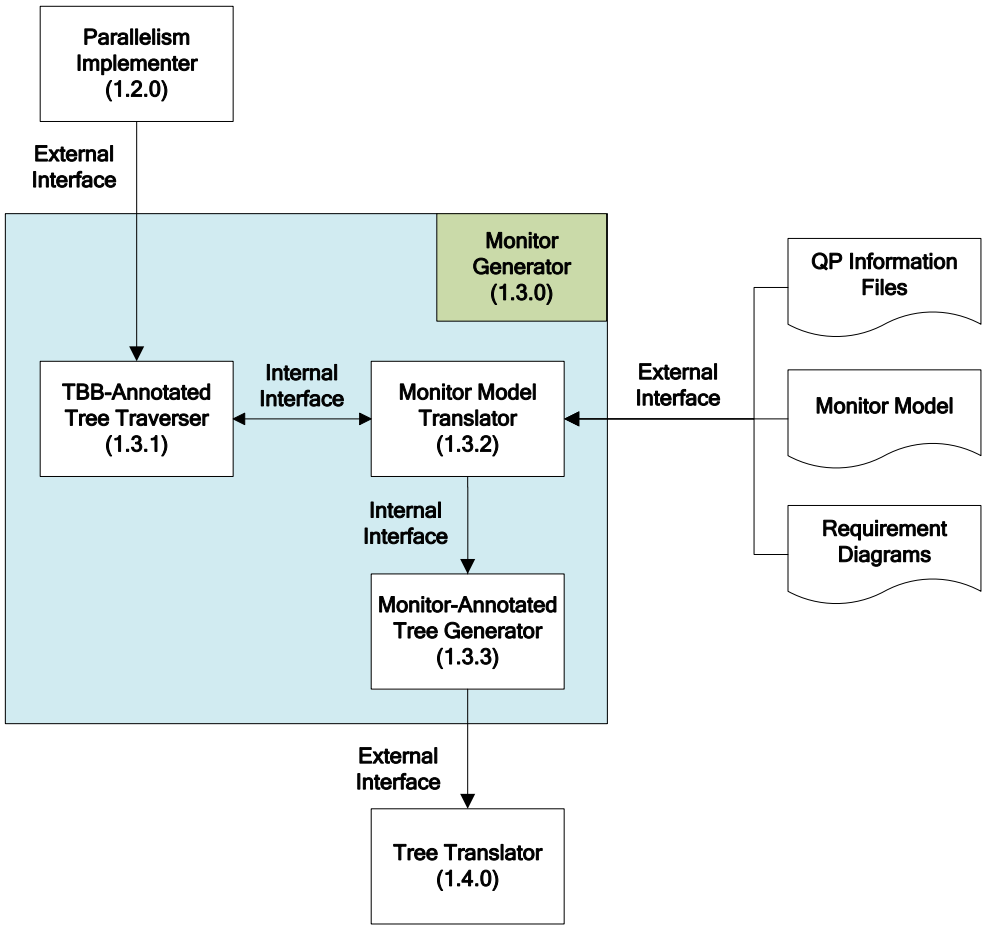


圖 68　MG 架構圖

Tree Translator (TT)，是讀入 Monitor Generator 階段所產生的 Monitor-Annotated Tree 並實際產生 Parallel C++ Code 的階段。這也是我們 Code Generation 子計畫的最後階段。TT 包含了兩個部分：Monitor-Annotated Tree Traverser、Code Generator。圖 69 為 TT 的架構圖。

在整個 Code Generation 子計畫流程中，使用者所設計的 SysML 檔案經過了 MP、MC、PI 和 MG 等階段之後產生名為 Monitor-Annotated Tree 的樹狀結構，在此樹狀結構中已儲存了足夠的資訊，包括 QP、TBB 和 Monitor 程式碼、使用者定義的程式碼和微調的參數等資訊，而 TT 主要的目的就是要將這棵完整的 Code Tree 實際轉換成程式碼檔案，稱為 Parallelized C++ Code。

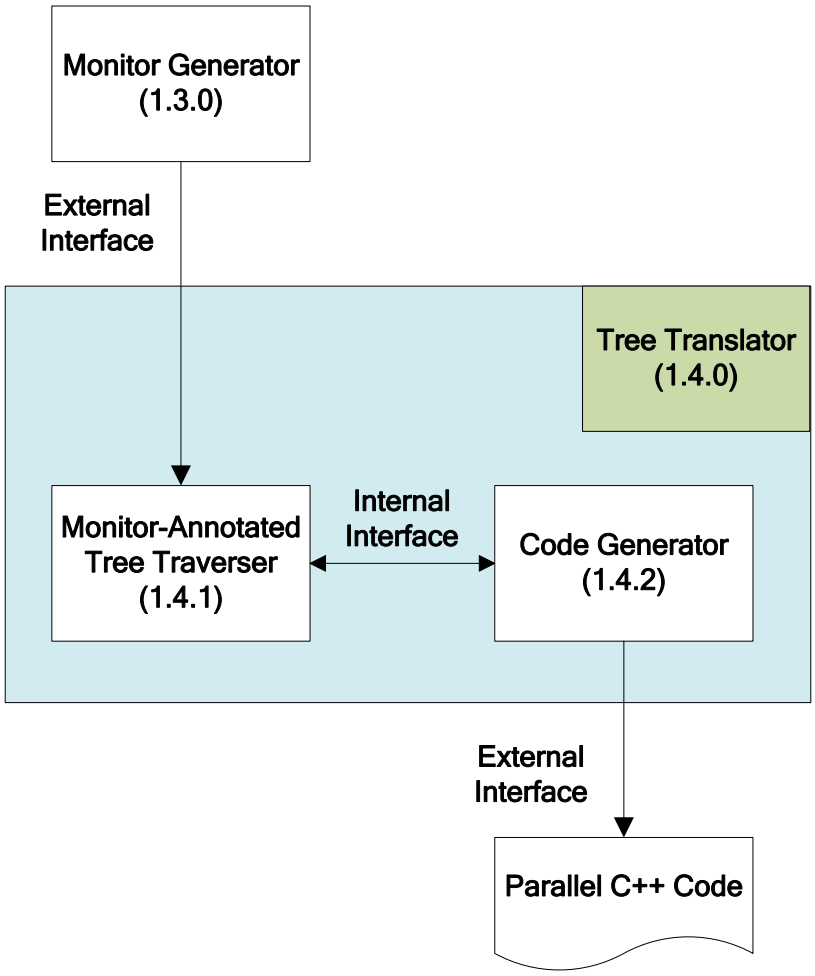


圖 69 TT 架構圖

介紹完整個設計流程後，本計畫以第一年度所設計的 DVR 系統為基礎，來實作程式碼生成器。目前以 DVR 系統中的 PVEEncoding 為例子來進行實作，圖 70 和圖 71 為 PVEEncoding 的 Block Definition Diagram 和 State Machine Diagram

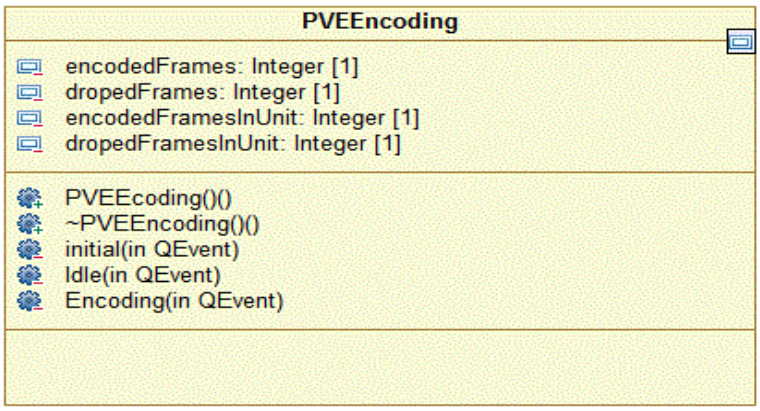


圖 70 PVEEncoding 的 Block Definition Diagram

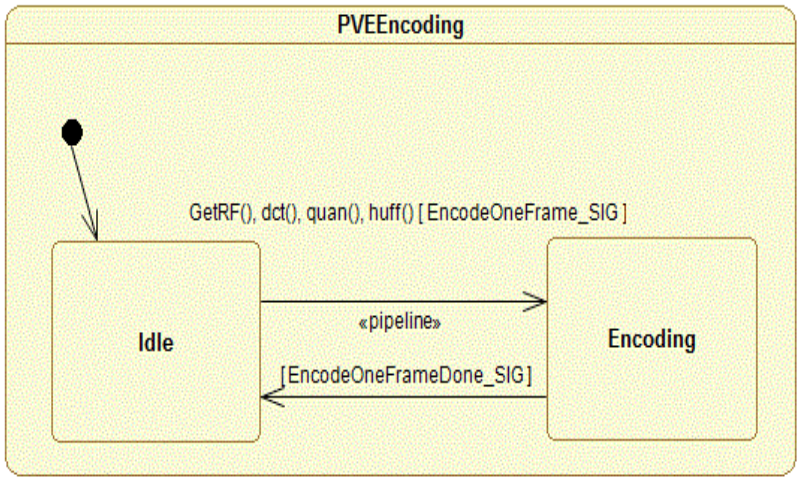


圖 71　PVEEncoding 的 State Machine Diagram

有了使用者定義的 PVEEncoding 的 block definition diagram 和 state machine diagram 後，Model Parser 可產生 PVEEncoding 的 semantics tree。MC 從 semantics tree 中獲得關於 PVEEncoding model 的相關資訊，在走訪每節點時，會根據 QP Information Files 貼上相對應的程式碼，例如：當遇到 state machine 的標籤時，會在此節點貼上 Active Object 的相關程式碼，因為 QP 裡的每個 state machine 都是一個 AO，就物件導向程式語言的角度來說(以 CPP 為例)，就是讓 PVEEncoding 物件來繼承叫作 QActive 的物件；而在 state machine 裡的每個 state，在 QP 裡都是 PVEEncoding 物件的 member function。圖 72 和圖 73 為 MC 所產生的 header file 和 cpp files。

以 PVEEncoding 的 state machine 為例(如圖 71 所示)，transition 上有叫作<<pipeline>>的 stereotype，PI 會參考 pipeline 的 parallel model 來產生相對應的 TBB code。圖 74 是 pipeline 的 parallel model。

```
Class PVEEncoding : public QActive
{
  public:
    PVEEncoding();
    ~PVEEncoding();
  private:
    void initial(QEvent const *e);
    QState Idle(QEvent const *e);
    QState Encoding(QEvent const *e);
};
```

圖 72PVEEncoding.h

```
QSTATE PVEEncoding::Idle(QEvent* const e)
{
  switch(e->sig)
  {
    case EncodeOneFrame_SIG:
      Q_TRAN(&PVEEncoding::Encoding)
      return 0;
  }
  return (QSTATE)&PVEEncoding::top;
}

QSTATE PVEEncoding::Encoding(QEvent* const e) {
  QEvent* pe;
  switch(e->sig){
    case Q_ENTRY_SIG:
  /* user-given encoding code or TBB code goes here*/
      pe = Q_NEW(QEvent, EncodeOneFrameDone_SIG);
      QF::publish(pe);
      return 0;
    case EncodeOneFrameDone_SIG:
      Q_TRAN(&PVEEncoding::Idle);
      return 0;
  }
  return (QSTATE)&PVEEncoding::top;
}
```

圖 73PVEEncoding.cpp

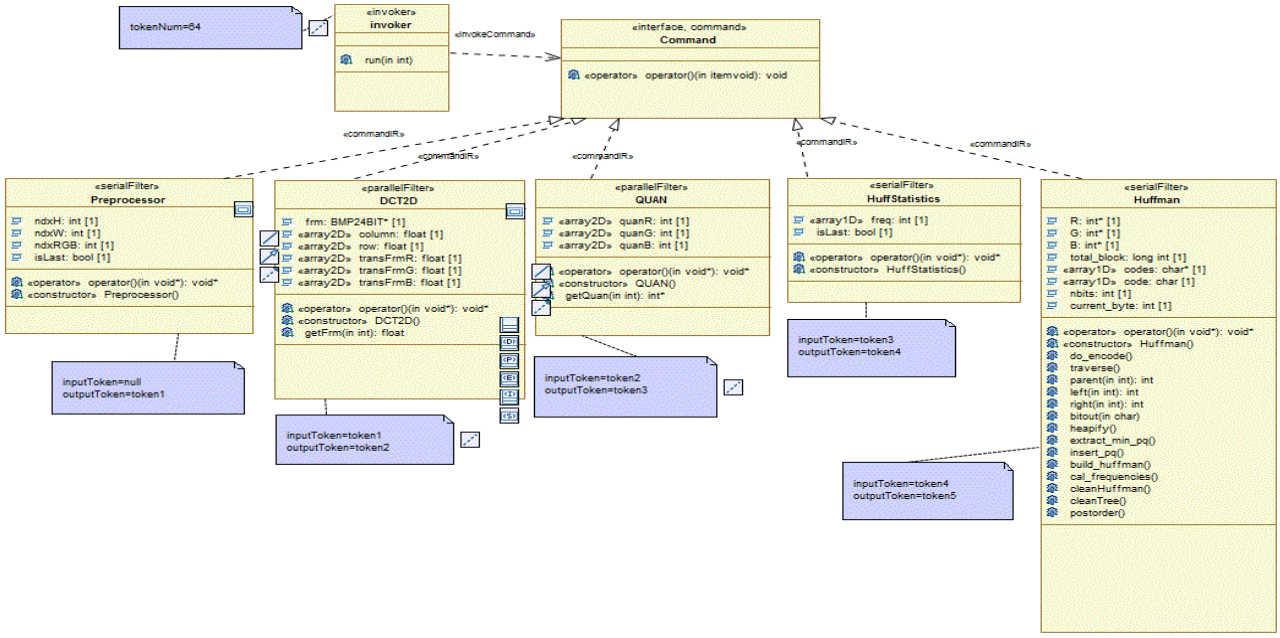


圖 74 Pipeline 的 Parallel Model

基於多核心嵌入式系統之動態可適性效能監視機制(Adaptive Performance Monitoring for Multicore Embedded Systems**)**

由於多核心處理器的蓬勃發展，軟體在運行的效能上，已經又被推向更高的一個層級，多核心處理器已經漸漸普及，不像以往僅有高階的伺服器會使用，一般家用的桌上型電腦也已經能夠輕易發現多核心處理器的存在，近年來在嵌入式平台上，更有許多多核心處理器的發表與採用，例如 Nvidia Tegra、Qualcomm Snapdragon 等。因應多核心處理器的發展，在軟體設計上，也必須發展出新的演算法，才能夠有效的發揮出多核心處理器的效能，但是這是一個非常困難的問題，因為許多常見的效能瓶頸問題，例如 Resource sharing problems、Unbalanced workload、Power consumption 等瓶頸，在多核心平台上發生時，會嚴重的影響到軟體的執行效能，但是這些問題有往往不容易被發現，因此我們發展出一套基於多核心嵌入式系統之動態可適性效能監視機制，希望藉此來輔助程式設計人員可以能夠輕易地發現軟體效能的瓶頸，進而做適當的修正來取得最大效能，充分發揮多核心平台的運算能力。

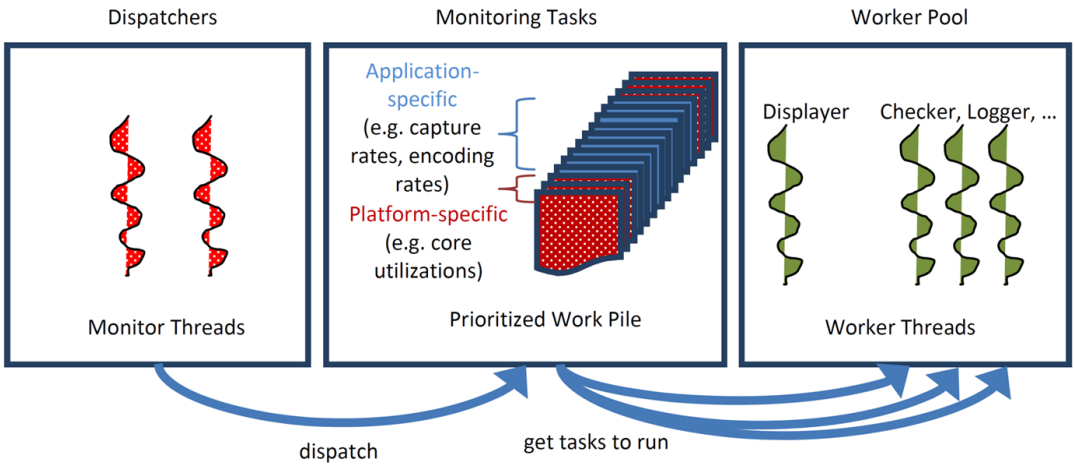


圖 75 動態可適性效能監視機制架構

在動態可適性效能監視機制中，共可分為三大部分，如圖 75 所示，分別為 Dispatcher、Monitoring Task、Worker Pool。在 Dispatcher 當中，監控執行緒(Monitor Threads)將會依照使用者所訂出的監視項目，將 Monitoring Task 加到一個具優先順序的工作堆疊當中，並且會依據系統的運作狀況，調整加入工作的數量，避免因為效能監視機制造成系統額外的效能負擔，導致監視所取得的資料失去準確性。在 Worker Pool 中，具有許多工作執行緒(Work Threads)，等待執行 Monitoring Task，當工作執行緒閒置時，便會檢查工作堆疊中是否有監試工作尚未被執行，如果有，則將工作取出並且執行，已取得所需要監控的資料。透過此動態可適性效能監視機制，使用者將可以軟體執行時，以最小的額外效能負擔，動態取得軟體運行的狀況及資訊，如此便能夠作為評估的依據，針對軟體的效能瓶頸做改善，提升軟體效能，充分利用硬體平台所提供的處理能力。

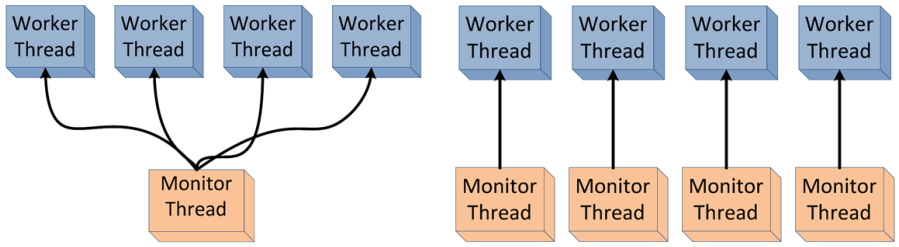


圖 76 監控執行緒工作分配方式

監控執行緒工作的分配方式，可分為兩種基本分配方式，分別為一對多模式(One-to-all)與一對一模式(One-to-one)，如圖 76 所示。不同的分配方示，將對系統產生不同的額外負擔(Overhead)，並且對於監視所取得的資訊之正確性(Accuracy)與即時性(Immediacy)，也有很大的關聯，例如採用一對多模式時，利用單一監控執行緒來監控所以監控目標，

由於系統僅需額外執行一個執行緒，因此對於系統的額外運算負擔(Computation Overhead)很低，但是卻會引起額外的資料溝通負擔(Communication Overhead)，並且當使用者監視的資訊越多時，可能導致監控執行緒工作過於忙碌，導致正確性與即時性下降，而取得無效的資訊。

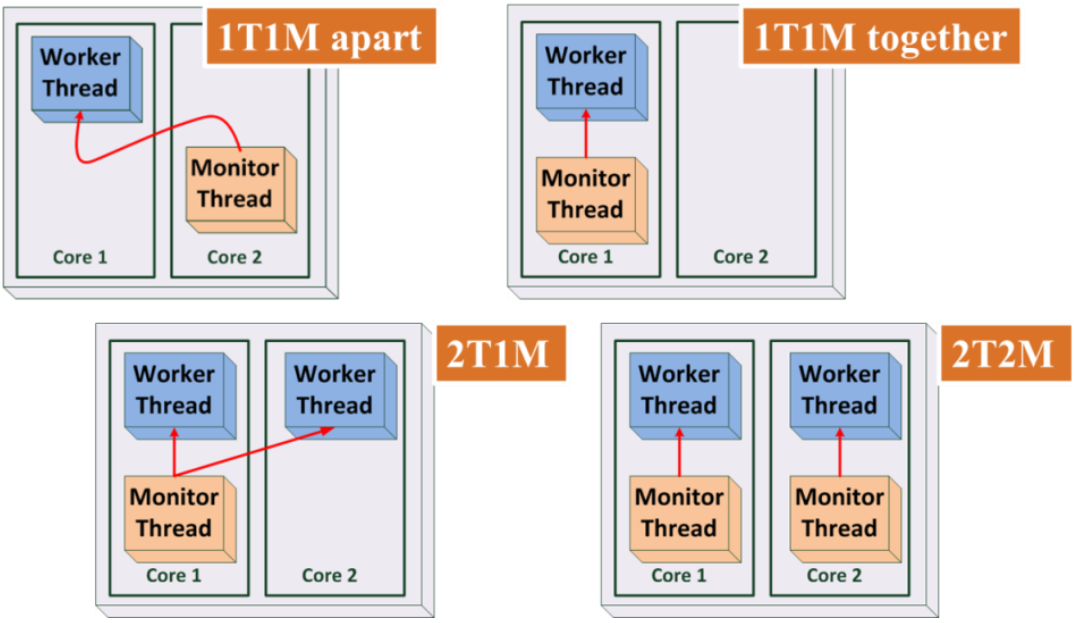
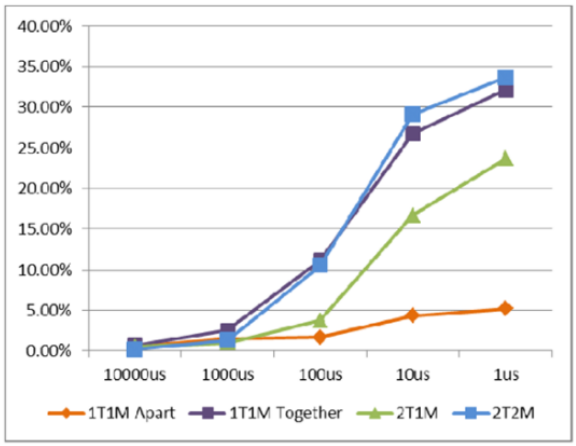
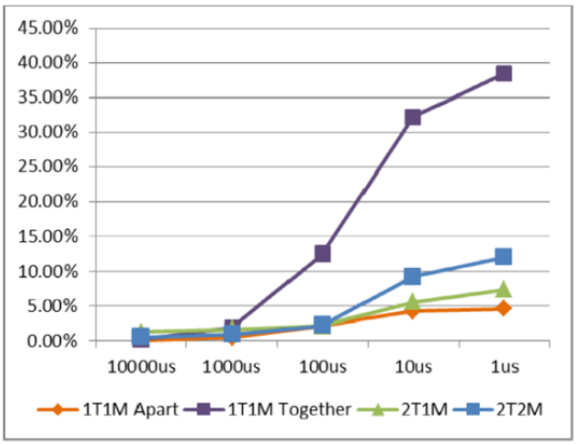


圖 77 四種監控執行緒分配情境

在本實驗當中，分別設計了四種工作分配情境，如圖 77 所示，其中 T 代表工作執行緒，M 代表監控執行緒，而工作執行緒根據兩種不同工作內容，可分為受裝置存取限制工作(IO-bound task)與受運算能力限制工作(CPU-bound task)。本實驗監控執行緒將在不同的時間區隔，分別為 10000us,1000us,100us,10us,1us，不斷的監測工作執行續的工作狀況，取得已完成的運算數量資訊。本次實驗分別運行於兩個實驗平台，第一個平台為 Intel Core2 Duo SU7300, 4GB DDR3 SDRAM, Linux 2.6.35，第二個平台為 ARM 11 MPCore, 512MB SDRAM, embedded Linux 2.6.35，本實驗都是使用 GNU gcc 4.4.5 編譯器，並且關閉所有編譯器最佳化功能。



(a) Monitored IO-bound task(s)

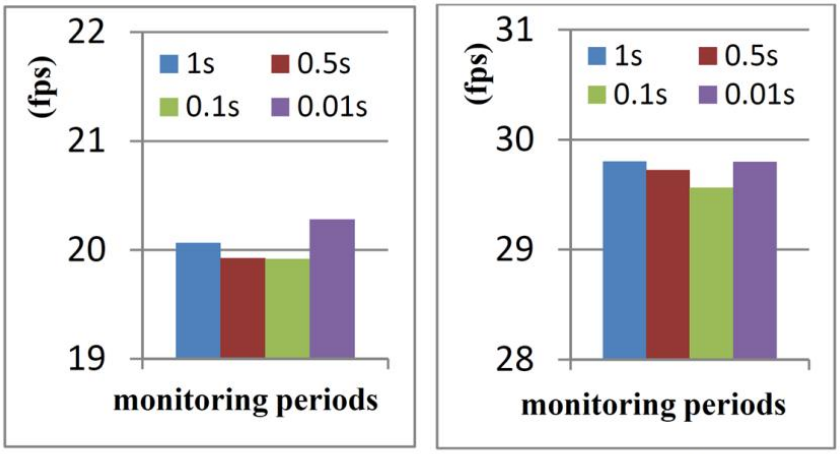


(b) Monitored CPU-bound task(s)

圖 78 四種情境之額外負擔(Overhead)評估圖

實驗結果如圖 78 所示，其中縱軸表示由監控執行緒所引起的額外負擔，橫軸為不同的監控執行緒監測時間區隔，根據實驗我們可以發現，在監測時間區隔大於 500us 以上時，監控執行緒所造成的額外負擔，其實都是在 5%以下，這表示效能監視機制是可行的，並不會造成嚴重的效能負擔，導致收集的資訊都是無效的。另外，我們發現在不同的監測時間區隔，採用不同的監控執行緒分配將會有不同的額外負擔，例如在圖 78 (b)中，我們可以發現 2T1M 與 2T2M 兩條曲線在時間區隔為 100us 時交叉，在時間區隔大於 100us 時，2T2M 所造成的額外負擔較大，但在時間區隔小於 100us 時，卻又是 2T1M 所造成的額外負擔較大，相同的情況在圖 78 (a)當中也可以發現到，因此我們發現，針對不同的監控目標，必須使用不同的監測時間區隔，才能夠獲得最好的資訊。

我們的實驗也在 Digital Video Recording System(DVR)系統上運行，DVR 為一個多人多監視器之線上即時影像串流撥放系統，在這個實驗中發現，不當的的監測時間區隔，將為使得監控執行緒過度的過取資訊，甚至速度超越真實的資訊改變速率，因此會造成監測所取的資訊失去意義，如圖 79 所示，當監測時間區隔超越真實資料改變速率(30fps)時，將會導致監測所取得錯誤的資料，完全失去意義。



(a) Encoding Rates (b) Capturing Rates

圖 79 不同的監測時間區隔所取得之資訊

在這篇論文當中，我們提出了一套基於多核心嵌入式系統之動態可適性效能監視機制，希望藉由此一機制來輔助程式設計人員，能更輕易地發現軟體效能的瓶頸，做出適當的修正來取得最大效能，充分發揮多核心平台的運算能力。在實驗中，我們針對監控執行緒的分配方式提出了分析，並且對於監控執行緒數量與分配方式做出了建議，並且發現在監測之資訊正確性與即時性之間的取捨，必須適當的設定監測時間區隔，才能夠獲得最準確的資訊。

以多核心系統偵測與追蹤籃球比賽視訊串流中的球 (A Multicore System Design for Basketball Detection and Tracking in Sports Competition Video Streaming)

由於籃球為熱門球類之一，且籃球比賽包含了動機之所有部分，因此我們將目標球類設定為籃球。而系統首要目的為正確追蹤籃球位置，唯有達到此目的才能實踐其餘目標。同時系統也將使用多核心之分散處理達到即時運算，得以跟上比賽節奏。另外，為了達到減少漏網鏡頭的目的，系統從兩方位錄製比賽，並自動切換攝影畫面。

圖 80 為自動化球類即時追蹤系統之架構。架構中包含四個主要功能:球場邊線偵測、籃球位置偵測、籃球追蹤及影像切換。系統中，我們匯入 OpenCV 進行影像處理。

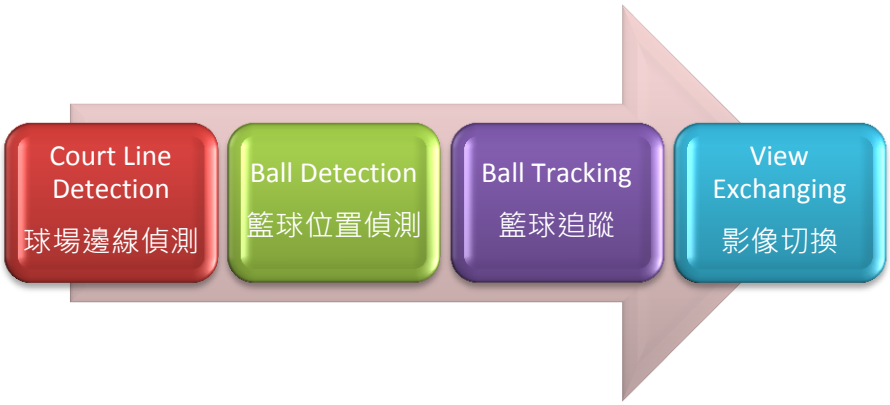


圖 80 自動化球類即時追蹤系統之架構

Court Line Detection:

為了確定籃球在球場上之確切位置以便攝影機切換，我們必須偵測球場邊線並定位球場中線。當讀入遠景影片第一張影像時，系統便會執行 Court Line Detection 偵測球場上的線並完成定位。

Ball Detection:

我們將近景影像和遠景影像經過顏色、集合大小、形狀等條件篩選，最後找出籃球位置，並回傳籃球 XY 座標，提供 Ball Tracking 使用。而因 Ball Detection 處理時間和運算量較大，因此需要 Ball Tracking 在定位籃球後做快速追蹤。Ball Detection 是由 Color Filter、Grouping Function、Shape Filter、Circle Ball Function 線性組合。Color Filter 將讀入之影像利用 OpenCV 提供之 cvCvtColor() function 轉換成 YCbCr 顏色空間，並顏色符合籃球之點標記為黑色，其餘標記為白色，輸出黑白影像。因為點與點間判斷顏色範圍沒有相依性，因此在 Color Filter 中可將影像切成數個區塊，交由 CPU 分別運算。Grouping Function 為 Ball Detection 中最重要的步驟，因為此步驟讓各點聚集成集合，形成物體的概念，以利後續篩選。Grouping Function 承續 Color Filter 之黑白影像，將影像由上而下、由左而右聚集，相鄰之黑點視為同一集合。我們給定每集合一個代碼(Group Number)，並記錄此集合的各種資訊。例如集合中包含多少點，集合之中心座標，集合半徑等。最後我們將點數過多或過少之集合刪除。Shape Filter 依據各集合的資訊，把非球之集合刪除。Shape Filter 首先判斷集合的長寬比，如果長寬比差距太大則代表為其他雜訊，則刪除此集合。接下來，由中心點出發計算上下左右之點數，太少也代表為雜訊(例如:手、腳)，予以刪除。此步驟中也可平行化判斷集合資訊，加速程式進行。最後，Circle Ball 將判定為球之集合用 cvCircle()依半徑大小圈出位置，並回傳 XY 座標，以利 Ball Tracking 追蹤。

Ball Tracking:

現在我們有籃球的確切座標。在 Ball Tracking 中我們將使用 Diamond search algorithm(DSA)追蹤籃球。每個影像都會經過以下三步驟，最後定位籃球。第一個步驟為 LDSP 搜尋:以前張影像之籃球 XY 座標為中心建出 LDSP(Large Diamond Search Pattern)，並在 LDSP 中找出九個點做為搜尋基準點。以此九個基準點為中心搜尋，範圍各為一菱形，大小為籃球的三倍寬。在九個菱形中各自標記符合籃球顏色之點，並將點集結成集合，依照各集合之形狀和集合內點數給予權重。九個菱形各自選出最高權重之集合比較，最後以最高權重集合所在之菱形，也就是 MBD(Minimum Block Distortion)做為下次搜尋中心。第二步驟為 LDSP 移動:得知 MBD 後我們將判斷 MBD 在菱形的何點上。MBD 之位置有三種可能，分別為角落、邊界和中心。如果 MBD 在菱形之角落或邊上，我們將以 MBD 為中心點重複運算 Step1，直至 MBD 落在菱形之中心；如果 MBD 位於菱形之中心，則進入第三步驟執行。第三步驟是 SDSP 搜尋:當判斷 MBD 位

於菱形中心時，系統會將搜尋範圍從 LDSP 縮小為 SDSP(Small Diamond Search Pattern)，並重複執行 Step1，進行精準定位。而此步驟求得之 MBD 即為籃球之最後所在之範圍。我們認定此次 MBD 中權重最高之集合為籃球，計算平均座標並在影像中圈出。如果在 Step3 仍無法獲得籃球確切座標，經過三個影像之容忍期後，跳回 Ball Detection 重新定位籃球。圖 81 表示可以看到 MBD 在角落或邊上時，演算法繼續以 LDSP(藍色圓形)為搜尋範圍，當 MBD 在菱形中心時，將縮小為 SDSP(綠色正方)搜尋。

View Exchanging:

當近景影像偵測到籃球時，將影像輸出切換為近景；當近景影像連續無法偵測到籃球時，將影像輸出切為遠景。

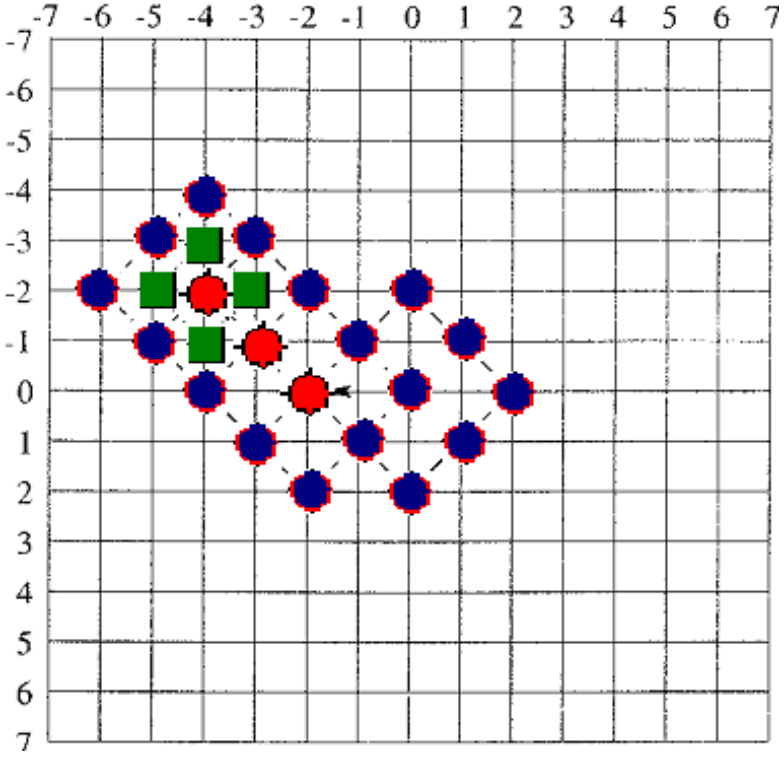


圖 81 Diamond search algorithm(DSA)示意圖

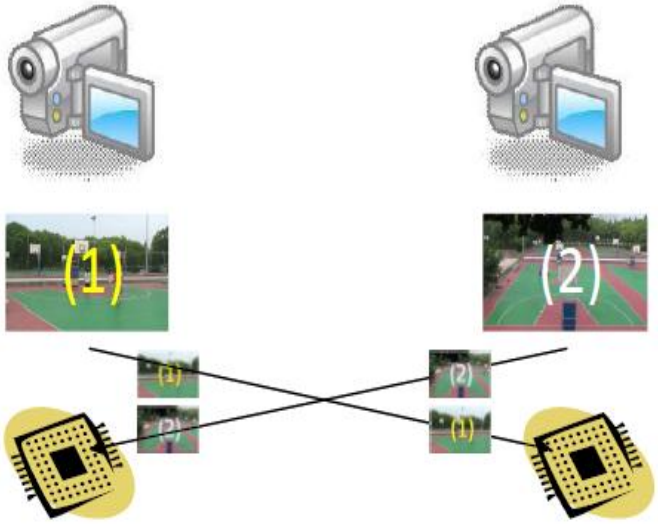


圖 82 Task Parallelism 示意圖

以下是 TBB 之平行化方法:

1)多台攝影機影像輸入之平行化

<1>使用任務平行(Task Parallelism)方法實做平行化。

<2>如圖 82 所示，將多台攝影機拍攝之畫面交由不同處理器平行處理。

2)切割畫面之平行化

<1>使用資料平行(Data Parallelism)方式

<2>將每單位輸入之畫面切割成數等份，配合 Ball Detection 和 Ball Tracking 架構，利用 thread 平均運算，即時追蹤。如圖 83 所示。

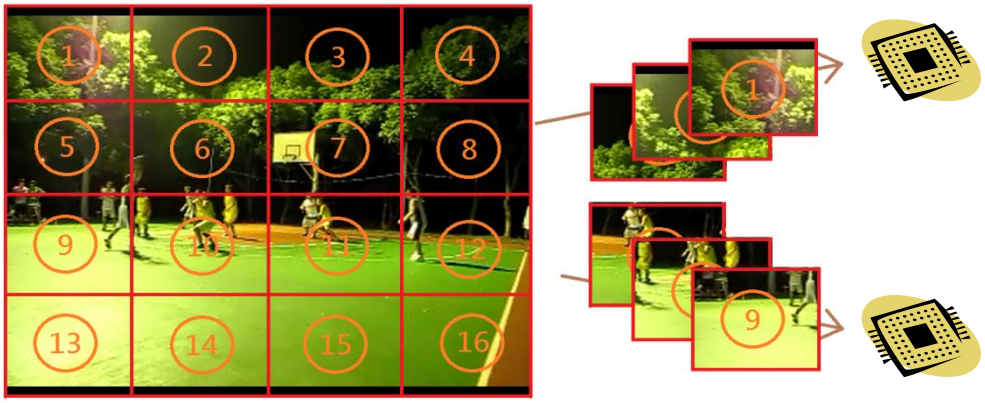


圖 83 影像切割平行處理

3)Group 篩選平行化

<1>使用資料平行(Data Parallelism)方式

<2>由於形成集合後，可依照篩選條件同時判斷各 Group 是否符合條件，加快判斷時間。

4)流程平行化

利用 Ball Detection 和 Ball Tracking 之架構，將流程組成一管線(pipeline)，利用 pipeline 平行化各 function，縮短整體時間。

目前僅針對 Detection 的部分做平行化，以下是實際實驗數據結果:

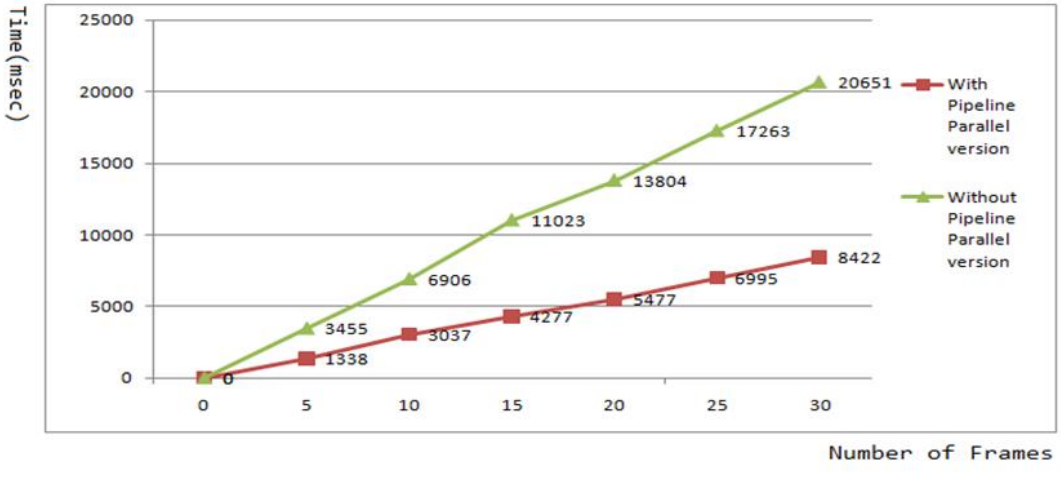


圖 84 平行化前後執行時間比較圖

由圖 84 觀察得到的結果來分析，發現利用 TBB Pipeline 所得到的結果是減少了原本執行時間的 59.22%。

C. 結論

三年計畫的第一年主要是以 bottom-up 為設計主軸，來設計數位影像串流(DVR)系統，並從中獲得更多關於多核心嵌入式軟體的發展經驗與議題;第二年計畫的目標著重於多核心程式碼生成器的設計流程並支援更完整的平行模型(Parallel Model)提供系統更多平行化程式碼生成的能力;第三年則重點放在加入 Monitoring Mechanism 來監控系統資源(Resource)與執行效能(QoS)，以期符合使用者給定的需求(User-specific Requirements)。我們也透過介面的制定說明與其他子計畫之間資訊的交換及所需的功能性互動，達到從上層 model，程式碼生成後到多核心目標平台上執行的完整性。

我們已經透過 DVR 系統這個例子和二大仲介軟體 QP、TBB Libraries，實作出自動化平行程式碼生成器，並也已發展出一套完整的自動化流程。

目前的成果有兩大技術優越性，其一是使用 Model-driven 的開發流程，特點是可以重複使用且容易作系統驗證，另一是多核心程式碼的開發，目前多核心系統愈來愈普遍，能自動產生多核心平行化程式碼勢必是一大優勢。目前成果可以提供業界一套多核心嵌入式系統的自動化平行程式碼生成流程，也可以提供學界一套在多核心嵌入式系統上的 Model-driven 開發流程，例如可以用在嵌入式軟體工程的課程教材上。

三年計畫已告一個段落，當初我們規劃的目標已大部分實現，我們接下來會繼續延伸 Monitoring Mechanism 的研究，朝著動態 monitoring 並依使用者需求作系統效能調校的方向努力，之後也會繼續往異質多核心系統作研究。

V. VMC-AM: 架構及效能調校支援實作

A. 研究目的

本子計劃五(Subproject 5)主要著重於架構對應及整合後工具組之實際應用，首要之目標為將子計劃一與三產生之嵌入式多核心軟體規格，在經由子計劃四之 thread-mapping、thread-scheduling 和正式的驗證以後，實作一自動淬取及合成核心(extraction and mapping kernel)。子計畫五之目標為設計一多執行緒分析模組(multi-thread analysis module)以針對合成多核心嵌入式程式碼時，解決多執行緒與多核心分配對應之問題。目地在提昇多核心嵌入式程式碼之效能及避免 Racing 及 deadlock 現象。此階段將利用 Intel 之所提供

之軟體工具如 Intel® VTune™ Performance Analyzer, IntelR Thread Profiler 及 IntelR Thread Checker 等進行分析 。因最後之多核心嵌入式程式碼將支援 Intel Threading Building Blocks (Intel TBB)語法，此階段亦將配合提供 TBB 所需之程式庫及系統調配指標。

最後將以研發之多核心嵌入式軟體工具系統實作三維實境行車導航系統，在此系統中將以開發工具實際模擬開發一虛擬實境行車導航情境。其中會使用到貴會支援所研發之光流演算法處理影像串流，此一演算法非常適合大量平行化運算，因此適合用於此計畫之測試。個人在結合光流演算法尋找特徵點及相機定位及反投影貼圖之相關經驗，正好可用於此三維實境行車導航系統之實作。圖一綜整本子計劃之主要目標及預計之達成方法。

B. 研究成果

第三年街景實境導航之研究程詳述如下。本研究之輸入有二。第一為全球定位之 3D 城市高度模型為一建物高度模型，此為一外形簡化之實體模型。第二為車載攝影機，車上以 GPS 在行進間隨時定位。一般情形下，GPS 可提供車體之中心位置及方位。一旦車體中心位置及方位已知，相機由於係安裝於車體上，相機中心位置及方位亦可得知，經由投影計算，建物模型及路面之可見面邊緣線可投影於相機影像平面，此時可繪置立體導航資訊如路名，門牌號碼，商家資訊等，與即時拍攝之行車街景結合，可形成一逼真之導航效果。但在某些情形下，GPS 信號無法即時被接收，如高樓蔽暲，或由於無線傳輸機構限制，在此情形下本研究提出一多核心光學演算法輔助之程序，可及時校正相機方位。以先前接收之 GPS 信號時計算之建物特徵點，加上經光流計算之位移量穫取新的特徵點，以反推相機之內外方位。進而進行反投影貼圖達成原目標。

立體導航圖形與實體街景影像套合，是使用車載即時視訊（ video sequence）做為輸入利用第一階段所產生的網格模型參考資訊與光流計算，計算攝影機之內外參數。進而求取立體導航圖形投影於攝影機影像平面結果。本研究的特色在於環物影像的拍攝並不需要精密的相機定位儀器，拍攝方位可以不預知，在此特別提出每一行車路段皆有一對應可見網格組合。在剛進入此路段時，需由 GPS 提供相機中心及方位。由此方位計算影像中之建物之 landmark lines and points 像素。之後的即時視訊則採用此初始 landmark lines and points 像素利用光流原理計算每張影相與類近影相的相素位移。經與粗體模型的對應即可求得相機的內，外方位，進而計算３Ｄ網格相對位置。至於上述各階段內部

與彼此之間的詳細規劃情形，我們將在以下的章節中逐一說明。首先對不使用 GPS 之相機定位法如直接線性轉換及光流演算法進行說明。

第二階段不使用 GPS 情況下，產生定位相機外參數，以做為輸入之持續拍攝視訊（ｖｉｄｅｏ ｓｅｑｕｅｎｃｅ）疊合第一階段所產生的粗體模型。本研究的特色在於環物影像的拍攝並不需要精密的相機定位儀器，拍攝方位可以不預知，利用光流原理計算每張影相與類近影相的相素位移。經與粗體模型的對應即可求得相機的內，外方位，進而計算３Ｄ網格之線性投影。至於上述各階段內部與彼此之間的詳細規劃情形，我們將在以下的章節中逐一說明。以下介紹本階段重要之理論部份:直接線性轉換及。此直接線性轉換理論在第二階段會使用到。

DLT 理論簡介:給定 *N* 對之對應 *x,y,z* 及 *u,v*。式(1)可表達如下:

(1)

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & u_1x_1 & u_1y_1 & u_1z_1 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & v_1x_1 & v_1y_1 & v_1z_1 \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ x_N & y_N & z_N & 1 & 0 & 0 & 0 & 0 & u_Nx_N & u_Ny_N & u_Nz_N \\ 0 & 0 & 0 & 0 & x_N & y_N & z_N & 1 & v_Nx_N & v_Ny_N & v_Nz_N \end{pmatrix} \cdot \begin{pmatrix} L_1 \\ L_2 \\ \vdots \\ L_{11} \end{pmatrix} = \begin{pmatrix} u_1 \\ v_1 \\ \bullet \\ \bullet \\ u_N \\ v_N \end{pmatrix}$$

上式為典型之線性系統方程式，可使用最小平方差矩陣計算 L_1 到 L_{11}。一旦計算出 L_1 到 L_{11}，x_0,y_0,z_0 可由式(2)計算獲得。旋轉矩陣(T)之元素 $r_{11},r_{12}\ldots r_{33}$ 可由式(3)計算獲得。

(2)

$$\begin{bmatrix} L_1 & L_2 & L_3 \\ L_5 & L_6 & L_7 \\ L_9 & L_{10} & L_{11} \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} -L_4 \\ -L_8 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} L_1 & L_2 & L_3 \\ L_5 & L_6 & L_7 \\ L_9 & L_{10} & L_{11} \end{bmatrix}^{-1} \begin{bmatrix} -L_4 \\ -L_8 \\ -1 \end{bmatrix}$$

(3)

$$T_{I/O} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = D \cdot \begin{bmatrix} \frac{u_0L_9 - L_1}{d_u} & \frac{u_0L_{10} - L_2}{d_u} & \frac{u_0L_{11} - L_3}{d_u} \\ \frac{v_0L_9 - L_5}{d_v} & \frac{v_0L_{10} - L_6}{d_v} & \frac{v_0L_{11} - L_7}{d_v} \\ L_9 & L_{10} & L_{11} \end{bmatrix}$$

藉由已經由 GPS 定位之影格 k 邊界框架，收集所有未處理的相鄰影格，接著利用 Fast Optical field計算目前影格的畫素相對於連線影格之位移,對每一個邊界框架內之相素，對鄰近影格更正這個新的邊界框架座標(boundary patch frame)。檢查邊界框架是否改變了，若為是，則重行用 GPS 定位；若為否，則將一個鄰近影格給定一個新的邊界框架像素座標及相對應的三維網格座標，並且計算轉換矩陣座標。使用ＤＬＴ（Direct Linear Transformation）計算相機的內外方位參數，然後即可給定兩個像機內外參數及對應之相數座標，計算所有的線性投影。

C. 結論

本研究順利完成以下主要工作項目:

1. TBB library support on PC and Linux configuration file
2. 行車導航系統建置(共同主持人協助)
3. 使用快速光流原理計算特徵點追蹤。
4. 與第一、二年實作系統整合測試，輸入連續影像，結合 GPS 定位及光流位移，以 DLT 進行相機內外參數計算，最後 3D 立體路標反投影至影像平面。

第一工作項目先將跟子計劃四 code gen.時所需之硬體系統資源相關訊息建入一嵌入式多核心軟體規格，以 deployment diagram 方式呈現。如圖一所示。此圖中包括了由子計劃一中 user 對硬體系統之需求規格,如板子的核心數(core number)、事件反應速度、OS 種類、硬體成本等。另外再加入子計劃四 code gen.時所需之多核心運算程式庫支援，包括程式庫安裝資訊、及程式庫種類等。本研究接著將此一 deployment diagram 產出 xml 檔，以便在後續階段轉換為系統支援所需之 automake 檔.

第二工作項目為立體導航圖形與實體街景影像套合，是使用車載即時視訊（ video sequence）做為輸入利用第一階段所產生的網格模型參考資訊與光流計算，計算攝影機之內外參數。進而求取立體導航圖形投影於攝影機影像平面結果。圖六為整合圖一之各子模組後呈現之結果。由該圖可看到車體內有一 CCD 相機負責即時抓取影像，一實體建築物即時影像顯示於嵌入式開發版之顯示螢幕。該建物之英文路標顯示於建物之左上方。

第三工作項目採用之快速光流計算係基於 structure from motion 之原理。空間中一點 P，相對於一固定相機所進行運動時，其空間速度 V 相對於影像速度之關係推導。p，之相對應運動三分向量結果。可知 z 向之分向量抵消為零。將 P 之速度分向量以，t，及 ω，及代入 u,v,可得一展開式如圖二之流程圖。由此方程式可知 u 是以 x,y,xy,及 x^2 構成之多項式，v 是 x,y,xy,及 y^2 構成之多項式。此多項式即為區域光流 local optical flow 計算之基礎。圖

三為對一實體建物特徵點進行光流追蹤結果。由此圖可看出隨著相機之移動，光流計算出正確的特徵點位移量，並顯示於實體建物之角落點。

第四工作項目結合 GPS 定位及光流位移，以 DLT 進行相機內外參數計算，最後 3D 立體路標反投影至影像平面。給定第三工作項目產出之特徵點位移量(2D)，再加上處於該物體之模型頂角之特徵點(3D)，即可以 DLT 理論計算出相機之外參數，進而進行 3D 路標之反投影。最後將路標投影至正確的平面位置，圖四為 3D 路標加建物模型在各個角度顯示之結果。圖五為 2D 路標與平面影像重疊後之結果，由此圖可知經由光流追蹤計算之結果是正確的，2D 路標可以持續標示於建物附近，而且會隨著相機移動而變動。

目前對光流計算效率評估是以核心數進行調整，研究對效能及耗電進行比較。結果顯示當核心數增加時核心數增加時，效能(performance)有獲得提昇，顯示本計劃產出之 Makefile 可正常運作，同時分析結果可幫助提昇效能(performance)，但耗電量確在提昇效能(performance)之後有所降低。此為下階段研究極待努力克服之議題。

VI. VMC-PPO: 平行程式優化支援實作

A. 研究目的

有效的平行迴圈切割與排程可以顯著地減少程式在多核心處理器系統環境中的整體回應時間，特別是針對具有許多迴圈的應用程式與針對新興的多核心嵌入式系統環境，其優化處理也更為重要。本子計畫的研究重點主要是設計與實作多核心嵌入式軟體之平行程式優化支援。在本計畫中，我們實作一套基於 OpenMP 的自動平行化工具，以產生能運行在多核心系統上之平行程式碼，其中我們使用 ROSE 開放原始碼編譯器為核心，針對 ROSE 深入探討並實作一個使用者介面以簡化使用之複雜度。本計畫完成一套自動程式碼平行化整合系統，系統包含如何去判斷該程式是否適合平行化及所需要的背景知識以及技巧等內容。並介紹如何讓使用者透過我們提供的 GUI 圖形使用者介面或是使用 Eclipse 外掛(Plug-In)，透過圖形操作方式，來使程式碼自動平行使得自動平行更加的容易。在多核心嵌入式系統上如果只運行普通的程式將無法達到應有的效能，所以透過適當的平行化轉換，會節省許多的時間。本計畫實驗有兩個部分，首先，我們證明了這些自動平行化的工具可行性與正確性。接著，分別於一般的處理器和嵌入式系統上運行並討論其效能比較。最後，我們將展示如何應用此工具來使得程式在多核心系統上有更佳的效能。

本子計畫主要設計與實作多核心嵌入式軟體之平行程式優化支援。有效的迴圈切割與排程可以顯著地減少程式在多核心處理器系統環境中的整體回應時間，特別是針對具有許

多迴圈的應用程式與針對新興的多核心嵌入式系統環境，其優化處理也更為困難。第一年(2009/8~2010/7)，子計畫六將使用編譯器技術如資料相依性分析與軟體管線化技術，完成指令階層平行化與平行迴圈切割與迴圈排程優化支援。本計畫將整合各個部分功能並建立與其他子計畫的介面，其中包含模型解析器(與子計畫二的介面)、軟體之合成器及程式碼生成器(與子計畫四的介面)、架構對應分析器(與子計畫五的介面)、及輔助測試的相關程式碼(與子計畫七的介面)。第二年(2010/8~2011/7)，本計畫將擴充 VMC 平行程式優化介面支援至更多的架構平台與應用，如支援多核心程式庫例如 OpenMP 及 Intel Threading Building Block 及強化與其他子計畫的介面。同時將建置一套在多核心嵌入式系統上平行程式優化支援模式適用於 OpenMP 及 Intel Threading Building Block 應用程式介面。

B. 研究成果

本子計畫範圍包含建置下列主系統與各項子系統，主系統為：

- 平行程式優化支援實作-VMC_PPO(Implementation of Parallel Program Optimization Supporting for VMC–VMC_PPO)

各子系統分別為:

- Source Program Parsing Subsystem
- Parallelism Subsystem
- Optimization Subsystem
- Logging Subsystem

隨著處理器從單核心進展至雙核心、四核心，甚至部分應用於伺服器的八核心之計算平臺，如何有效利用多核心的特性，並使應用程式於多執行緒的架構下加速運行，就成了開發人員極需研究的課題。關於嵌入式計算領域，也從單核心成長至多核心，為使原本效能備受限制之嵌入式裝置，透過多核心同步運行之機制，盡可能提高其運算效能，本計畫將建置一高效能平行編譯器，載入既有之程式碼，透過逐行逐段分析，輔以高效能平行演算法，基於 OpenMP 及 Intel Threading Building Block 多核心函式庫，產出平行程式碼，使其達到多核心同步執行，加速其運算速度，並達到最大之產出，其概觀如圖 85 所示

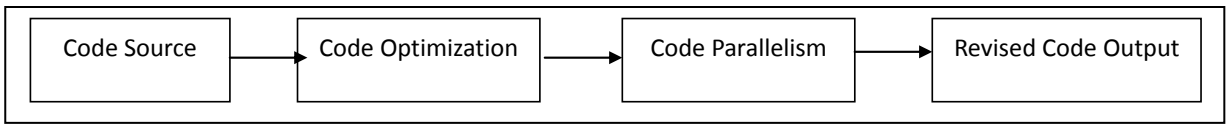


圖 85 高效能平行編譯器概念圖

本子計畫所開發，結合了平行技術與程式碼優化演算法，提供程式碼優化的基本功能和學習模式。本計畫建置一多核心嵌入式軟體之高效能平行編譯器 (PPO)，載入既有之程式碼，透過逐行逐段分析，輔以高效能平行演算法，基於 OpenMP 及 Intel Threading Building Block 多核心函式庫，產出平行程式碼，使其達到多核心同步執行，加速其運算速度，並達到最大之產出。有鑒於以往程式優化系統的全面性，我們採取了區塊優化，也就是經過學習模式就每個迴圈區塊做一次優化並記錄下來，以迴圈執行效率做為比較，去判定運算迴圈是否需要程式優化，因此本系統利用 Intel TBB 與 OpenMP 以及 ROSE 來達到程式碼平行與優化的效果。

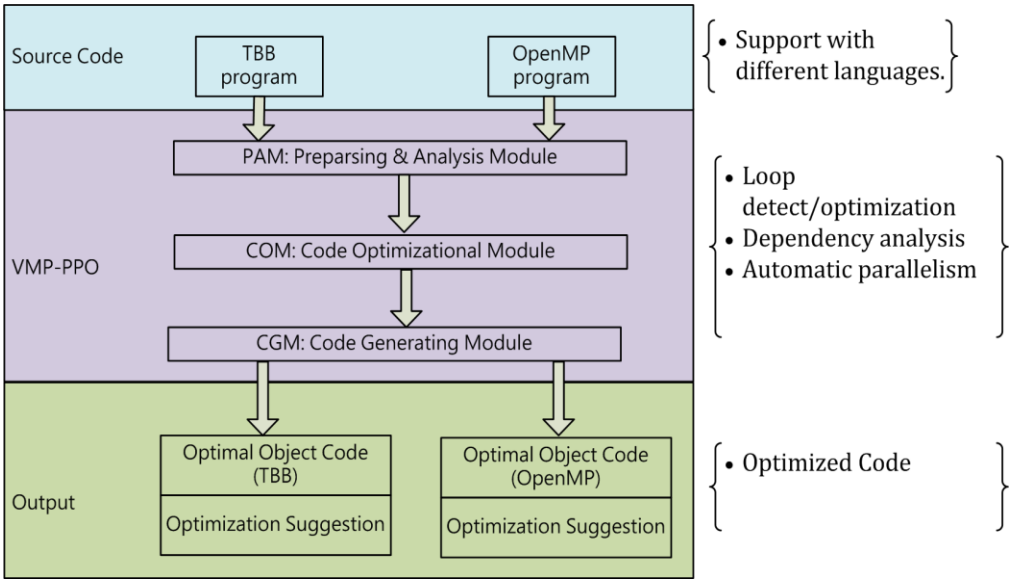


圖 86 VMC_PPO Architecture

圖 86 為本子計畫的系統架構示意圖，Source Code 部分是由子計畫四所產生的程式碼為 Input 支援 TBB 與 OpenMP，中間部分為我們子計畫的核心部分，分別是 PAM、COM 與 CGM。Output 部分是要連接子計畫七並讓其做測試。

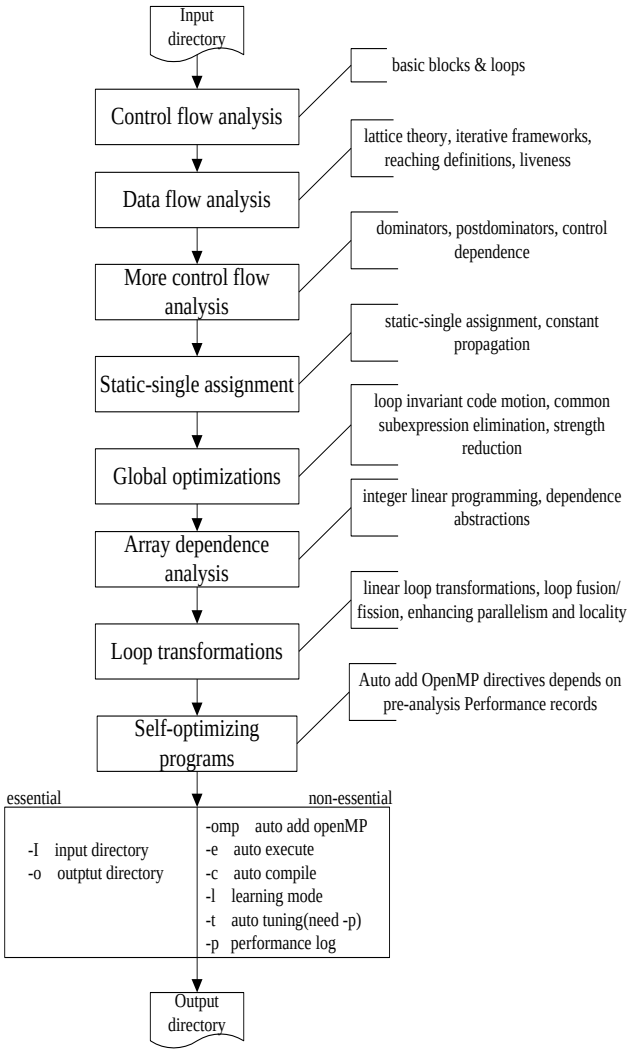


圖 87 系統流程示意圖

圖 87 系統流程示意圖為系統執行流程，當程式碼由子計畫四接收後，中間共會經過八道處理程序，以下一一列舉說明：

1. Control flow analysis

   分析程式中的內部區塊為階層式與計算迴?影響值。

2. Data flow analysis

分析程式中變數的生命周期。

3. More Control flow analysis

   分析程式中控制相依性。

4. Static-single assignment

   常數影響係數、靜態單一賦值。

5. Global Optimizations

迴圈不變碼的移動、常見子表示消除與簡化。

6. Array dependence analysis

整數線性程式化、相依性抽象化。

7. Loop transformation

   線性迴?轉換、迴?的融合/分裂與增加平行化和局部化。

8. Self-optimizing programs

根據預先分系的效能記錄自動添加 OpenMP 標籤。

本系統可區分為四個子部份系統，其個子系統之間的關係如圖 88 各子系統關係圖所示。

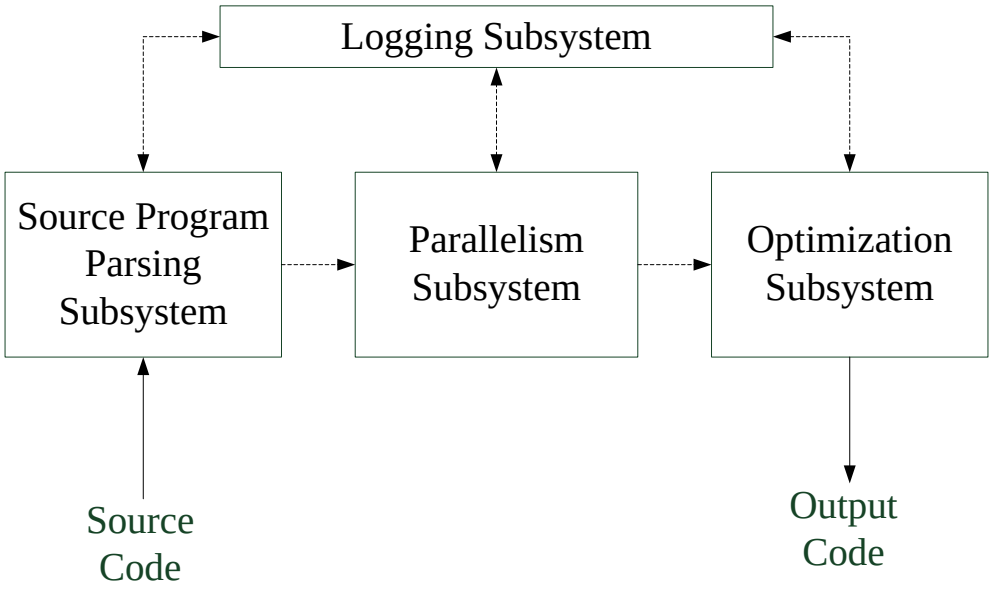


圖 88 各子系統關係圖

以下會各別對每個子系統做詳細敘述，如下：

1. 程式剖析子系統(Source Program Parsing Subsystem, SPPS)

   SPPS 為 PPO 系統之前端，用以語法剖析系統接受程式碼之後，逐行剖析，找出所有可平行化之部份，捕捉與程式結構區塊有關之符號，審視程式上下文，判斷符號與關鍵字之前後配對架構，濾出關鍵符號及框架，最後交付給後端之平行子系統。圖 89 為 SPPS 子系統架構圖，SPPS 包含下列兩個子部份：

   A. UI：提供使用者操作與設定介面

   B. Parsing：程式碼語意整理

   C. Analysis：程式碼語意分析

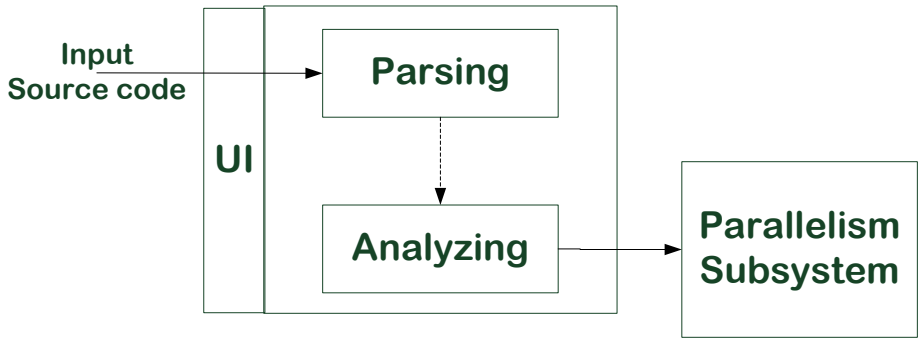


圖 89 SPPS 子系統架構圖

接受使用者輸入之原始程式碼，逐行進行剖析，主要鎖定多數可平行運行之迴圈，以平行程式撰寫時常用的關鍵字與平行化之架構為基礎，針對輸入程式之上下文，以及各種區塊符號，判斷出各個程式區塊，最後交付後端之平行優化子系統。

由於本系統所採用的 ROSE 是由 C 語言所撰寫的故在使用者介面上，我們使用 Visual Studio C#進行開發，但是在執行上我們是用 Python 去進行改寫，以適應跨平台的環境。

2. 平行子系統(Parallelism Subsystem, PS)

當接受前端程式剖析子系統之結果，針對剖析子系統所判斷出各個可平行化之標記部份，予以新增符合 TBB 或 OpenMP 之函式庫導引器，使其達到平行優化之目標。圖 6 PS 子系統架構圖為 PS 子系統架構圖。

本子系統接受的輸入為剖析子系統之剖析紀錄檔，針對使用者指定之平行函式目標(TBB 或 OpenMP)，於各個可平行化之部份，增加函式導引器，最後產生平行程式碼。

圖 90 PS 子系統架構圖

3. 優化子系統(Optimization Subsystem, OS)

將上一個子系統所產生出來的程式碼，更進一步去做最佳化。就是在除了平行化的程式碼外，再去尋找可以使之增進執行效率的程式碼區塊，加以最佳化並產出最後的完成碼。圖 91 為 OS 子系統架構圖，本子系統之組成如下：

Scanning：提供程式碼掃視

Analysis：程式碼優化區塊分析

Modify：處理欲優化程式區塊行為

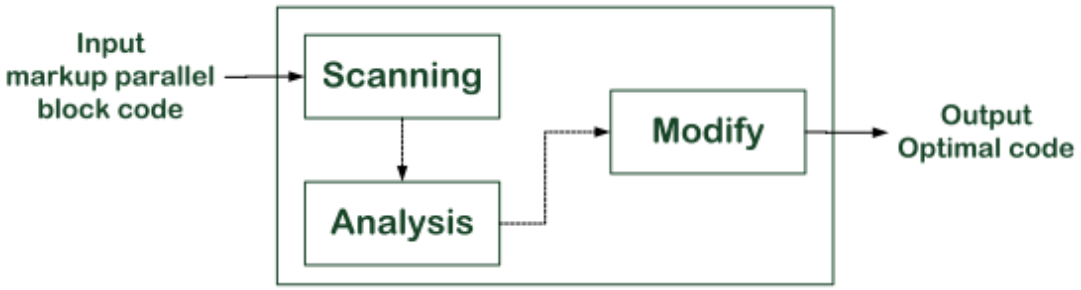


圖 91 OS 子系統架構圖

4. 紀錄子系統(Logging Subsystem, LS)

LS 功能為紀錄程式編譯優化過程中，所標記之程式碼行號以及所加入之程式導引器，並將之輸出以做檢驗或查詢用途。圖 8 LS 子系統架構圖為 LS 子系統架構圖，本子系統之組成如下：

A. Markup code log：紀錄程式碼被剖析時的所有 SPPS 行為

B. Parallel code log：紀錄程式碼在平行化時所有 PS 行為

C. Optimize code log：紀錄程式碼優化其間 OS 的所有行為

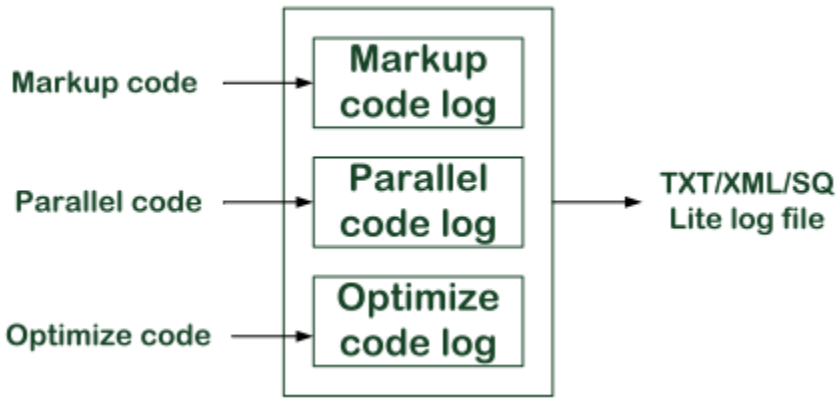


圖 92 LS 子系統架構圖

在 LS 中，於編譯程式執行過程中，紀錄程式分析紀錄，包含所標記之行號，新增加之平行函式導引器及其位置，最後分別輸出 TXT、XML、以及 SQLlite 等格式，供日後追蹤分析。

我們利用了 ROSE 計畫中兩大部分 Program Analysis 與 Program Transformation and Optimizations 中的 Recognizing Loops、Generating Control Flow Graphs、Dataflow Analysis、

Partial Redundancy Elimination、Loop Optimization、Tailoring The Code GenerationFormat……等等函式類別，以 Pyhton 程式語言去撰寫與呼叫 ROSE 原有的程式以達到我們子計畫六所需要的應用。子計畫六主要執行程式名稱為 vmc-ppo.py

我們 vmc-ppo.py 主要是以兩大函式為主，Auto Parallel 跟 Loop Optimizer。

Auto Parallel 支援多執行緒的 source-to-source 轉換，也就是加上 OpenMP 的標籤在程式碼之間，圖 93 為呼叫 Auto Parallel 函式的畫面。呼叫的參數定義為：

-i　　指定為優化/平行原始碼的資料夾位置

-o　　指定優化/平行過後原始碼輸出的資料夾位置

-omp　　是否執行自動平行功能

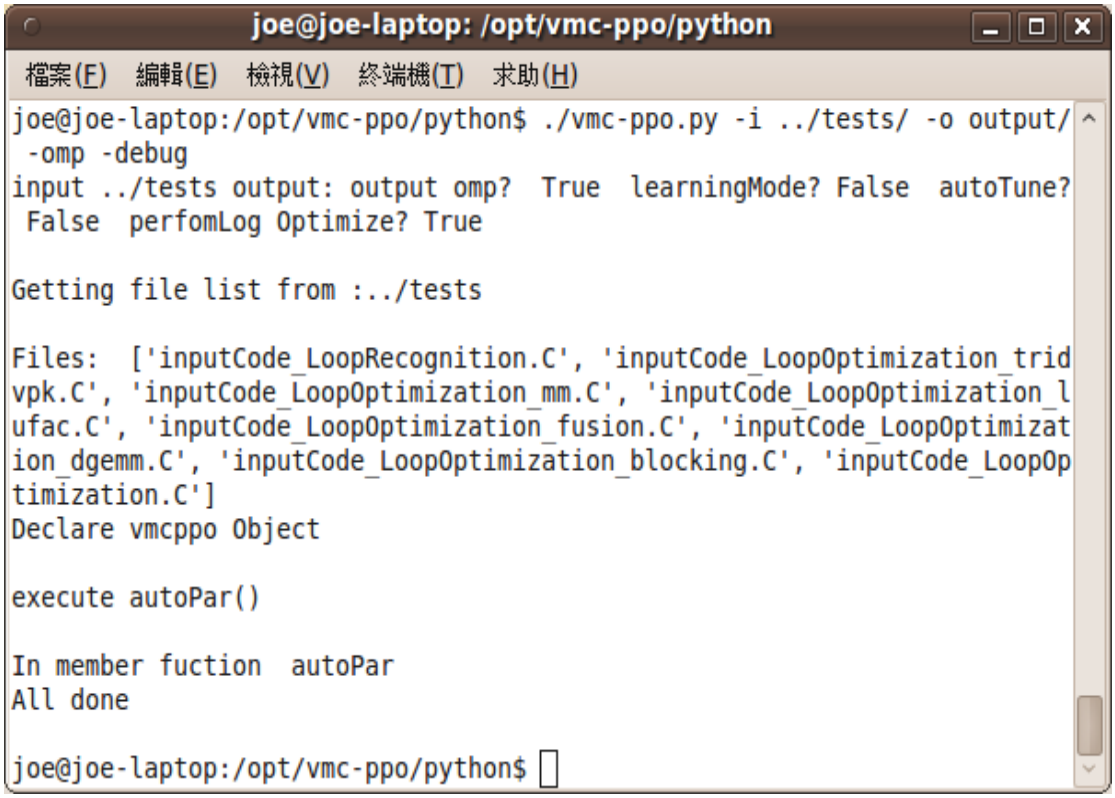


圖 93 Auto Parallel 執行畫面

圖 94 為執行 Loop Optimizer 的畫面，呼叫的參數定義為：

-i　　指定為優化/平行原始碼的資料夾位置

-o　　指定優化/平行過後原始碼輸出的資料夾位置

-optimize 是否執行優化功能

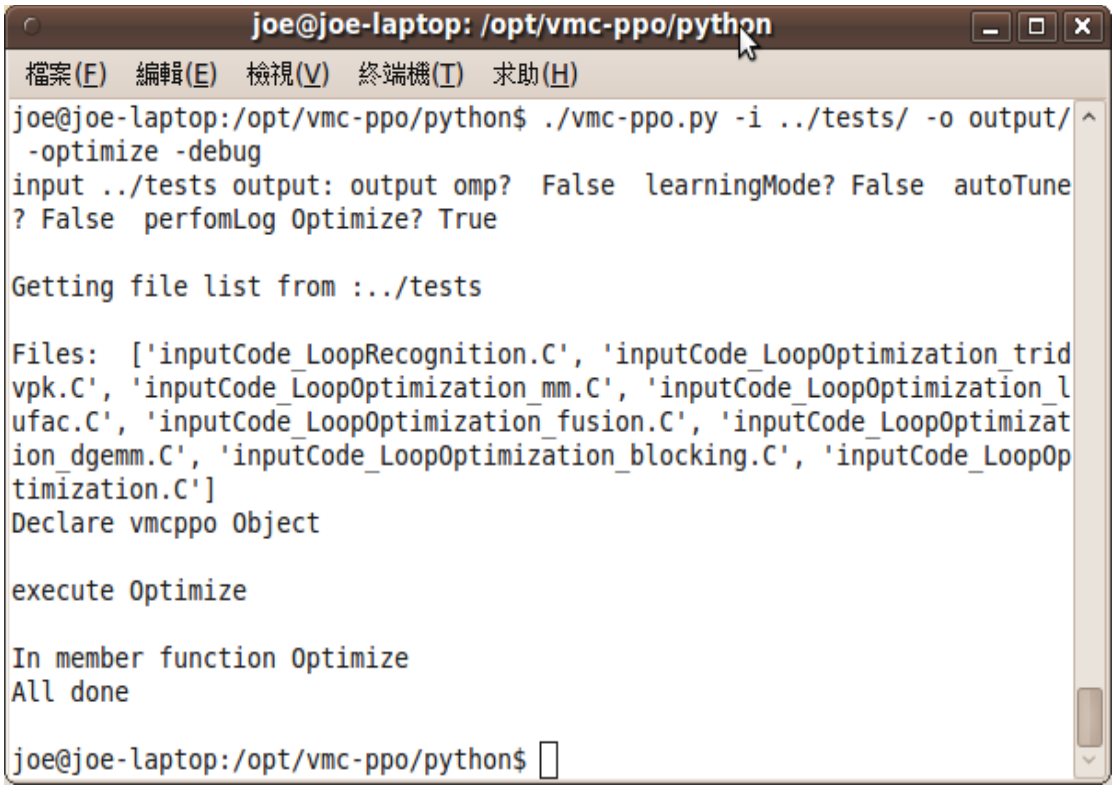


圖 94 Loop Optimizer 執行畫面

我們這支程式利用 Auto Parallel 與 Loop Optimizer，進一步去達到迴?剖析、分析資料相依性、變數區域性與效能最佳化分析。除了文字命令介面，我們還以 C#實做出使用者圖形介面，讓使用者可以利用點選的方式去執行與操作。C#介面如圖 95。

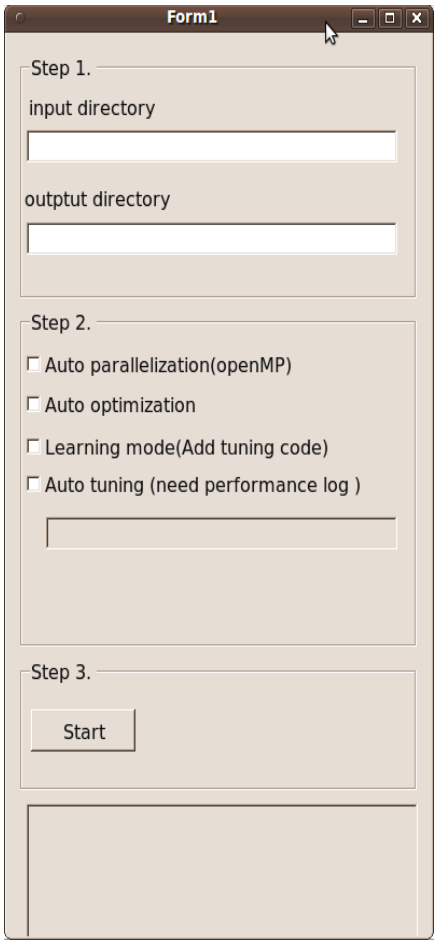


圖 95 C#使用者圖形介面

接下來我們將我們的函式庫做成 Eclipse 的 plugin，整個流程如圖 96

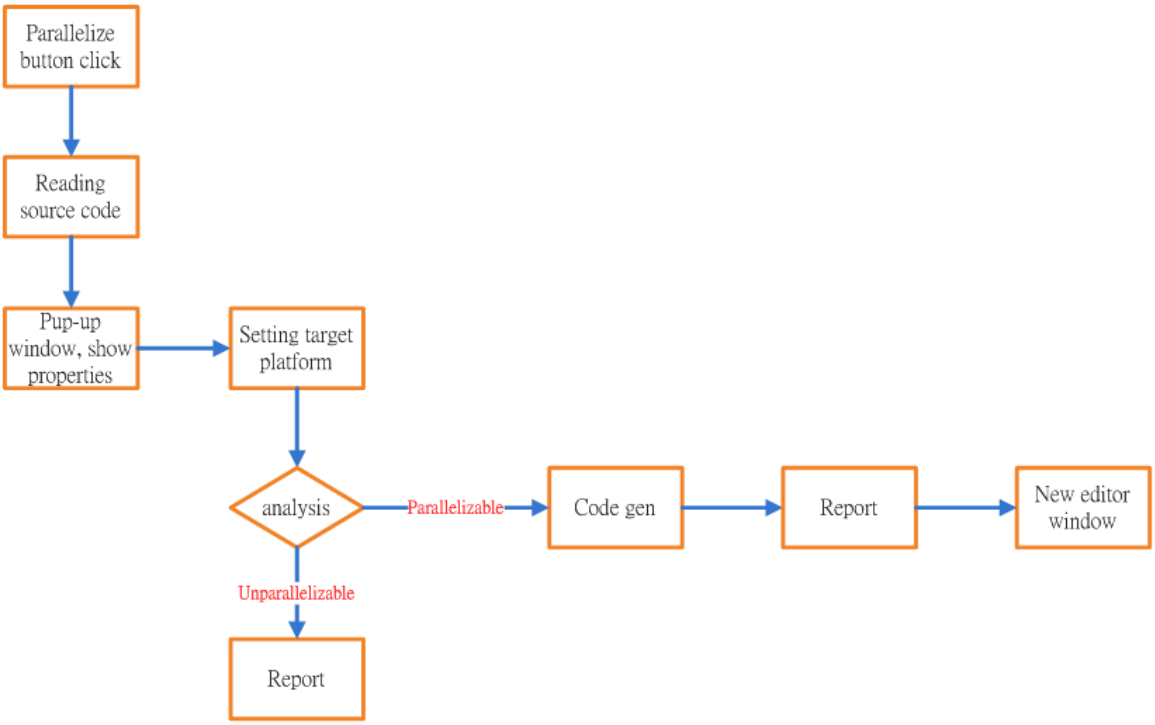


圖 96 判斷流程

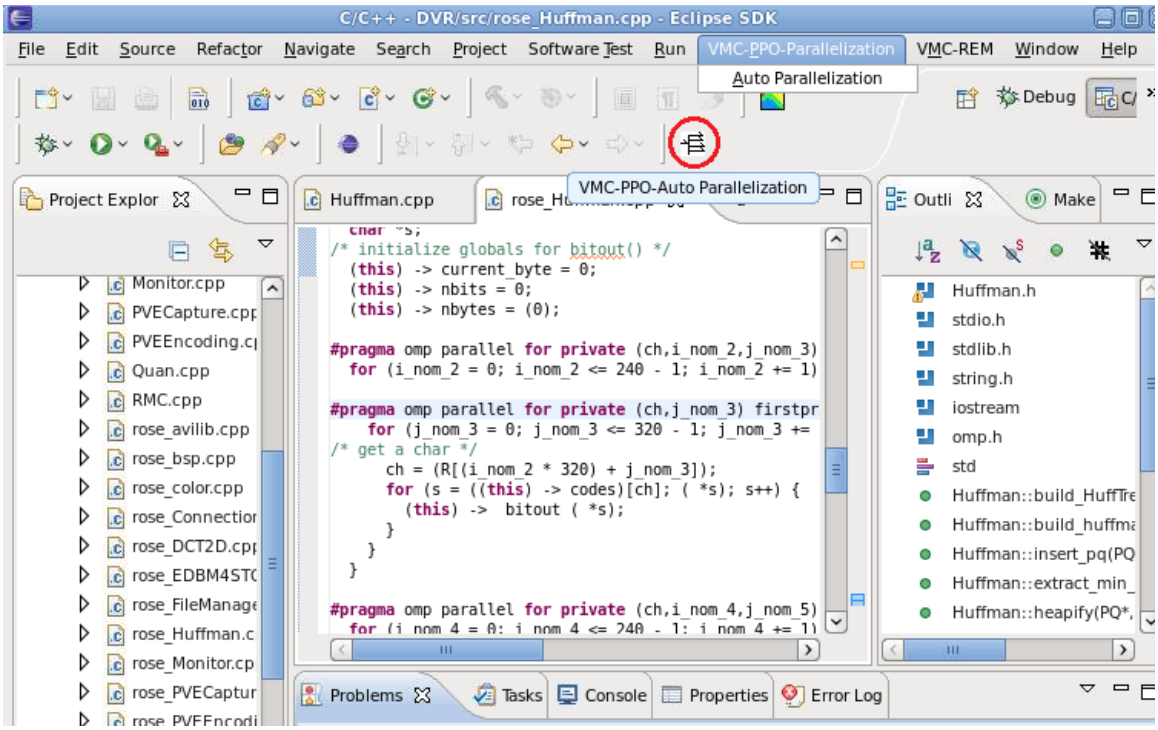


圖 97 Eclipse 使用者圖形介面

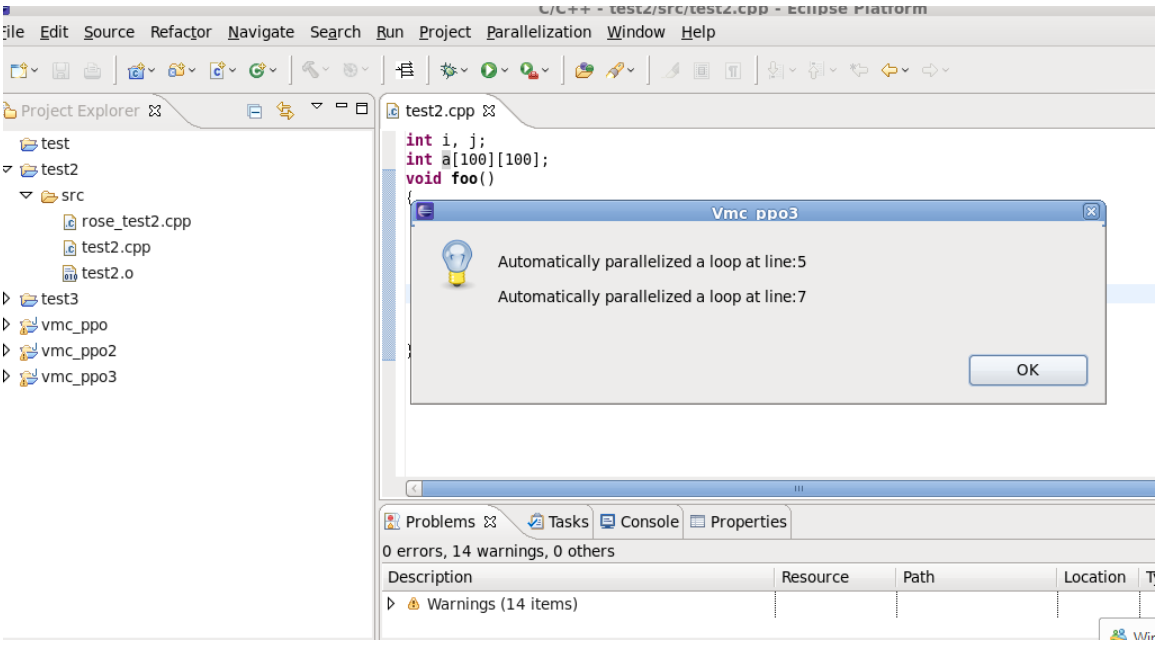


圖 98 平行化通知視窗

如通知視窗所表示，針對第五行及第七行使用自動平行化。

C. 結論

本計畫完成一套自動程式碼平行化整合系統，系統包含如何去判斷該程式是否適合平行化及所需要的背景知識以及技巧等內容。並介紹如何讓使用者透過我們提供的 GUI 圖形使用者介面或是使用 Eclipse 外掛(Plug-In)，透過圖形操作方式，我們所建立的程式

成功的使程式碼自動平行使得自動平行更加的容易。在多核心嵌入式系統上如果只運行普通的程式將無法達到應有的效能，所以透過適當的平行化轉換，會節省許多的時間。首先，我們證明了這些自動平行化的工具可行性與正確性。接著，分別於一般的處理器和嵌入式系統上運行並討論其效能比較。最後，應用此工具來使得程式在多核心系統上有更佳的效能。

此外，於民國 100 年 8 月 17、18 日藉國立中正大學舉辦 Summer School，分享整個 VMC 團隊的系統開發過程與經驗分享，讓其他對於 Multi-core 有興趣的開發者或是同學能夠參與其中，並做進一步的學習及瞭解。

VII. VMC-TMS: 測試支援系統

A. 研究目的

本計劃主要是以軟體品質提昇、品質控管為出發點，針對「多核心嵌入式軟體之模型驅動整合開發環境」結合軟體開發流程之需求分析、設計及撰寫程式碼各階段，做不同層次的軟體品質測試及測試的管理，以降低嵌入式平台執行測試負荷及減少測試工程師的負。本專案為針對多核心架構開發的「多核心嵌入式軟體之模型驅動整合開發環境 -VMC－子計畫七:多核心嵌入式軟體設計之測試支援系統」有關多核心嵌入式軟體測試的子計劃，在本計畫中，我們發展了單元測試、覆蓋率測試、平行程式效能量測，以建構「自動化測試工具」來支援測試工程師不同的需求，提供多核心嵌入式軟體自動測試服務。

本研究計劃建構了一個支援 cross-testing 的自動化測試工具，其主要區分為功能性測試，包含單元測試、覆蓋率測試，非功能性測試包含平行程式的效能量測，其目的說明如下：

1. 建構 Host-Side 及 Target-Side 的測試環境來實現自動化嵌入式軟體測試的工作。
   - 所有測試的工作不必完全在嵌入式系統平台上執行，可增進測試的效率，有效地利用嵌入式硬體資源，減輕測試工程師的負擔。
2. 發展一個支援 C/C++複雜資料型態(primitive type、structure type 、object type)，自動產生 test data、test case、test driver。
   - 讓自動化單元測試、覆蓋率測試不受資料型態的限制。
3. 自動執行多回合 line coverage 測試、branch coverage 測試。
   - 測試工程師不必重覆多次相同的測試動作。
4. 可在 runtime 時去監控程式在多核心嵌入式平台上執行的效能及自動化多回合平行程式(TBB)效能量測。

- 使測試工程師容易得知軟體在嵌入式平台上執行的效能。

5. 以文字及視覺化介面輔助呈現每回合的測試結果。

- 使測試工程師容易理解測試報告。

B. 研究成果

1. 自動化測試方法

在本章節中,我們將說明本研究實現包含功能性及非功能性自動化測試主要的方法,在功能性測試中,我們將說明在單元測試中自動產生支援 C/C++複雜資料型態的 test data、test case、test driver 方法,讓自動化測試能支援複雜的資料型態,並且介紹在覆蓋率測試中結合自動化多回合測試機制的方法,可減少測試需重覆多次執行的負擔;最後在非功能性測試中,則說明 TBB 平行程式效能量測方法,幫助測試工程師找出適合 TBB pipeline 平行化所需的參數值。

a. 自動化單元測試

在單元測試中,我們將測試程式中每一個 function 是否會發生錯誤,在進行測試前,必須先產生測試所需的 test data、test case 及 test driver,首先,我們說明如何自動產生支援 C\C++複雜資料型態的 test data,其資料型態包含 primitive type、structure type、object type 的 test data,此外也可以產生 array type 的 test data,而產生的 test data 將作為 test case 及 test driver 的輸入資料,圖 99 為自動產生 test data 的演算法,在演算法中相關名詞說明如下:

(1) Function-List

儲存待測程式中所有 function 的資訊,包含名稱、參數型態及回傳型態。

(2) Object-List

儲存待測程式中所有 object type 的屬性。

(3) Parameter-List

儲存所有 function 的參數型態。

(4) TestInputDataGenerationMainFunction

擷取待測程式所有 function 的參數型態及回傳值型態。

(5) TestInputDataGenerator

產生與 function 參數型態及回傳值型態的 test data。

演算法執行的步驟:

Step 1：TestInputDataGenerationMainFunction procedure 會從 function list 中取得 method name、parameter type 及 return type，並將 parameter type、return type 代入 TestInputDataGenerator procedure。

Step 2：TestInputDataGenerator 根據 parameter type。

Step 2-1：若為 primitive 則直接產生對應型態的 test data。

Step 2-2：若為 array type，則會擷取出 array 的 dimension，計算出所需產生的資料量，再產生對應型態的 test data。

Step 2-3：若為 object type，則會以遞迴的方式來找出在 object 中所有的 primitive type，再產生對應型態的 test data。

```
1.  Algorithm : test input data generation function
2.
3.  Function-List:FL
4.  Object-List:OL
5.  Parameter-List:PL
6.
7.  Procedure TestInputDataGenerationMainFunction()
8.  begin
9.    for_each method in FL
10.     get Parameter-List PL from FL
11.     get Return-List PL from FL
12.     call TestInputDataGenerationFunction(PL)
13.   end_for each method
14. end
15.
16. Procedure TestInputDataGenerator(Parameter-List PL)
17. begin
18.   for_each parameter p in PL
19.     switch (type of p)
20.       case int_type:
21.         if p = single element
22.           then generate int test data
23.         else // p= array type
24.           get dimension form p
25.           generate input data with for loop
26.         end_if
27.       break;
28.       case float_type :
29.       if p = single element
30.         then generate float test data
31.       else // p= array type
32.         get dimension form p
33.         generate input data with for loop
34.       end_if
35.       break;
36.
37.       //case char，  short, double, …
38.
39.       case object_type :
40.         get object attribute set OAS from Object-List
41.         let Parameter-List OPL = OAS;
42.         call TestInputDataGenerationFunction(OPL)
43.       break;
44.       default:
45.     end_switch
46.   end_for
47. End
```

圖 99 自動產生 test data 演算法

當 test data 產生之後，需再產生所有待測 function 的 test case，而每一組 test case 皆包含一組 test data，再根據 test data 的資料型態產生 test case，所產生的 test case 將會在測試執行時驗證待測程式是否有會例外錯誤或是否輸出符合使用者所輸

入的期望值，圖 100 為自動產生 test case 的演算法，其演算法中相關名詞說明如下：

Function-List

儲存待測程式中所有 function 的資訊，包含名稱、參數型態及回傳型態。

Object-List

儲存待測程式中所有 object type 的屬性。

Parameter-List

儲存所有 function 的參數型態。

TestCaseGenerationFunction

產生與 function 參數型態及回傳值型態的 test data。

演算法執行的步驟：

Step 1：從 function list 中取得 method name、parameter type 及 return type。

Step 2：產生 test case 的 function signature 程式碼。

Step 3：判斷 parameter type 是否為 primitive type、structure type、object type。

Step 4：依據 parameter type instrument 一個變數(variable)宣告的程式碼。

Step 4-1：若 parameter type 為 primitive type，則 instrument 讀取一筆 test data 給此變數的程式碼。

Step 4-2：若 parameter type 為 object type 或 structure type，則取出所有的 primitive type attribute，並且 instrument 讓所有 attribute 可讀取 test data 的程式碼。

Step 4-3：若 parameter type 為 array type 則計算出所需讀入的 test data 數目，並產生利用 loop 將所有 test data 讀入的程式碼。

Step 5：判斷此 method 的回傳型態(return type)

Step 5-1：若為 primitive type、structure type、object type 則 instrument 讀取 test data 的程式碼、呼叫待測程式 function 的程式碼，及 assertion 的程式碼。assertion 的程式碼可用來判斷待測程式的 method 的回傳值是否符合期望值。

Step 5-2：判斷此 method 的回傳型態(return type)，若為 void 則產生呼叫待測程式 function 的程式碼。

Step 5-3：若都不是則產生以 null 代入待測程式 method 的程式碼或以人工指定。

```
1.  Algorithm : test case generation function
2.
3.  Function-List:FL
4.  Object-List:OL
5.  Parameter-List:PL
6.
7.  Procedure TestCaseGenerationFunction()
8.  begin
9.     read test input data file
10.
11.    while has one test data set
12.       get function name from FL
13.       generate test-case-function-name code
14.       get Parameter-List PL from FL
15.
16.       for_each parameter p in PL
17.        switch (type of p)
18.        case int_type :
19.           if p = single element
20.              then generate int variable declaration to get input data
21.           else // p= array type
22.           generate int array variable declaration
23.           get int array diamesion form p
24.           generate for loop to get input data
25.           end_if
26.        break;
27.
28. //    case char_type，short_type, double_type ...
29.
30.        end_switch
31.      end_for
32.    end_while
33. end
```

圖 100 自動產生 test case 演算法

接著我們將說明，自動產生以 cppunit 為基礎的 test case 程式範例，如圖 101 所示，binsearch()為待測 function，其參數型態分別為 structure array type、整數、整數，回傳值型態為整數，此 test case 會去讀取與參數型態對應的 test data，並透過 CPPUNIT_ASSERT function 測試輸出是否符合使用者所輸入的預期值，若不為預期值，此 test case 將會發出測試未通過之訊息。

```
void TestDriver::test_0_binsearch()
{
  cout<<"function name="<<testCaseParser.getToken()<<endl;
   element a[289];
    for(int m0=0;m0<289;m0++)
    {
    a[m0].key  =atoi(testCaseParser.getToken());
    a[m0].link  =atoi(testCaseParser.getToken());

    }
    int b =atoi(testCaseParser.getToken());
    int c =atoi(testCaseParser.getToken());
    int result = atoi(testCaseParser.getToken());
    CPPUNIT_ASSERT( result == _TESTEDFILE.binsearch(a,b,c) );

}
```

圖 101 Test case 產生範列

最後，我們需再產生 test driver，圖 102 為自動產生的以 cppunit 為基礎的 test driver 範例，藉由 test driver 可以控制所有 test case 執行並擷取測試時待測程式所發生的錯誤，當測試完成並將測試結果寫入 XML 檔案中，但目前對於自動產生 test data、test case 仍有些限制，例如 C/C++的 template 型態或是指標。

```
CPPUNIT_NS::OStringStream stream;
 TestCaseParser testCaseParser;
testCaseParser.parserTestCase();

CPPUNIT_NS::TestResult controller;

CPPUNIT_NS::TestResultCollector result;
controller.addListener( &result );

CPPUNIT_NS::BriefTestProgressListener progress;
controller.addListener( &progress );

CPPUNIT_NS::TestRunner runner;
runner.addTest( CPPUNIT_NS::TestFactoryRegistry::getRegistry().makeTest() );
runner.run( controller );

CPPUNIT_NS::CompilerOutputter outputter1( &result, CPPUNIT_NS::stdCOut() );
outputter1.write();
//Write file to Result.xml.
CPPUNIT_NS::XmlOutputter outputter2( &result, stream );
outputter2.write();
std::string actualXml = stream.str();
ofstream fout("Result.xml");
fout<<actualXml;
fout.close();
```

圖 102 Test driver 產生範列

b. Multi-round coverage testing

本研究所發展的多回合覆蓋率測試技術主要是以 GNOME gcov 為基礎，結合多回合 cross-testing 機制，藉由多回合的機制，來提高測試的覆蓋率。並且在執行測試時，測試工程師可在 Host-Side 端以圖形化介面動態觀察每一回合測試的 line coverage 及 branch coverage。多回合覆蓋率測試的流程如圖 103 所示，以下為**執行步驟：**

Step 1：讀取待測程式

Step 2：剖析待測程式，擷取 function 資訊，其中包含參數型態。

Step 3：自動產生 test data、test case 及 test driver。

Step 4：將其待測程式與 test driver 編譯為執行檔，其編譯時額外加入 gcov 編譯參數。

Step 5:將測試程式、test case 及 gcov 產生的*.gcon 檔上傳至 Target-Side 上執行。

Step 6：測試執行完成，下載 gcov 所產生的 test log。

Step 7：分析 test log 判斷是否符合 coverage 的門檻值。

Step 7-1：若達到所設定的 coverage 門檻值則停止測試，並顯示每回合的視覺化測試報告給工程師。

Step 7-2：若未達到所設定的 coverage 門檻值則自動產生 test case，並回到 step 5 繼續下一回合測試，並且顯示每回合的視覺化測試報告給工程師。

接著我們將說明，如何實作多回合覆蓋率測試，圖 104 為多回合覆蓋率測試演算法，此演算法需要 2 個輸入，分別為 gcov 所產生的 test log 及我們自己新增的 totalCoverage log，在此我們將使用 flag 集合來記錄程式每一行或每一分支執行過或未執行過，當檢查至程式的第 i 行或分支在這一回合是否有被執行過，若是則將 flag[i]設定為 1，若否則檢查 totalCoverage log 是否在之前的回合已被執行過，若是則將 flag[i]設定為 1，否則設定為 0，當所有 flag 設定完成，則開始計算 line coverage 或 branch coverage，flag 集合大小為總行數或總分支數，所以 coverage 為(flag 集合裡為 1 的數目)/(flag 集合的大小)，最後 flag 集合寫入 totalCoverage log 中。

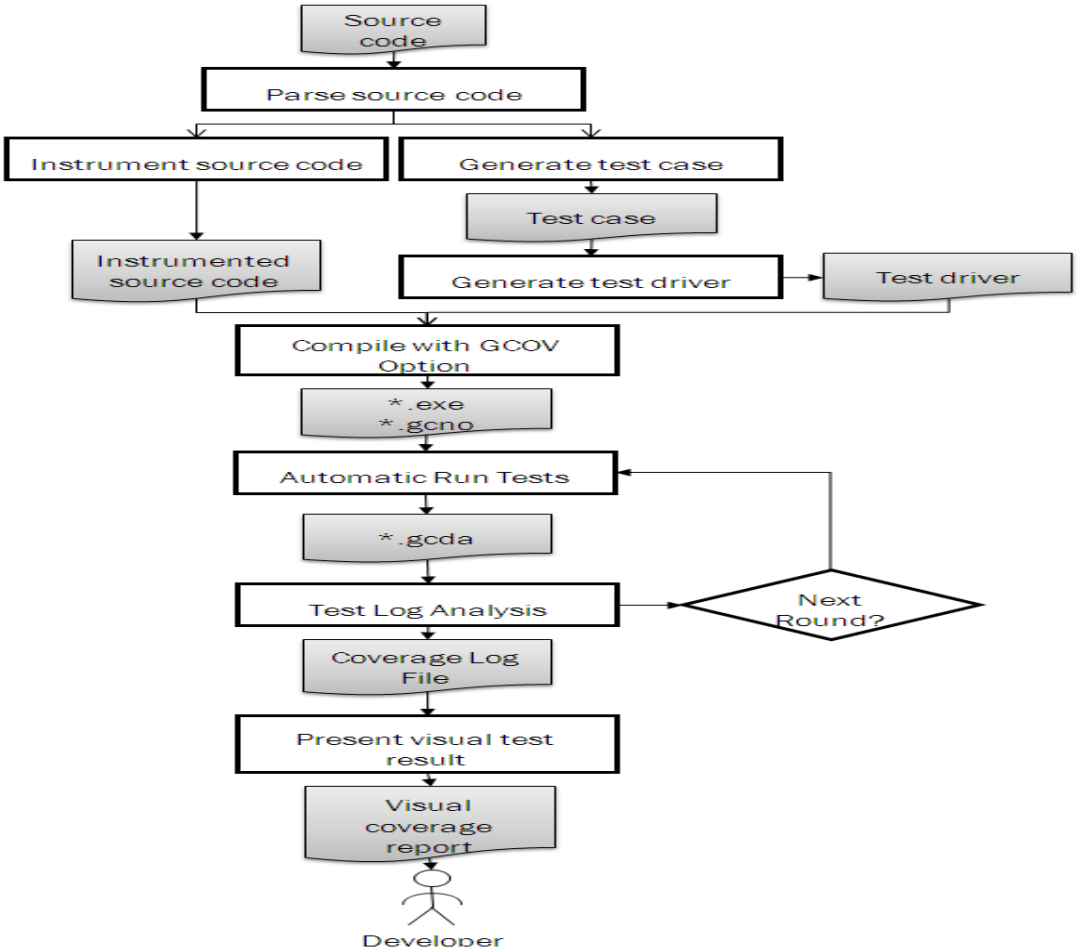


圖 103 多回合 coverage test 流程

```
1. input: String s, where s read form GCOV log file
2. input: String t, where t read form totalCoverge log file
3. output: coverage
4. set flag={}
5. set s'=s tokenize with line
6. set t'=t tokenize with line
7. while s' not null
8.     if s' equal to 1
9.         add 1 to flag
10.    else
11.        if t' equal to 1
12.            add 1 to flag
13.        else
14.            add 0 to flag
15.        end_if
16.    end_if
17. set s'=s tokenize with line
18. set t'=t tokenize with line
19. end_while
20. Write flag to totalCoverage log file
21. coverage=count(flag equl to 1)/size of flag
22. Exit
```

圖 104 多回合 coverage test 流程

c. TBB Pipeline 自動化效能量測

本章節將說明 Thread Building Block(TBB)平行程式 pipeline 效能量測方法，TBB pipeline 可以讓使用者去決定平行化的程度，此為 pipeline token 數，對於選擇一個正確的 pipeline token 數需要經驗的累積，若所設定的 token 數太小則會限制平行的程度，而 token 數太大則會耗費不必要的資源(例如需要更多的 buffer)。一般若要找出最佳的 pipeline token 數目，測試工程師需要多次手動修改 pipeline token 的參數，對此我們提出自動建議一個合適的 pipeline token 數目，節省使用者花費額外的時間在測試 pipeline 平行化的效率，並能更精準的提升平行程式的效能。

測試工程師需先選取包含 TBB pipeline 函式 source code，本研究將自動 instrument 測試程式片段來擷取效能資訊，如圖 105 所示，包含了 TBB 所提供的 timer 來擷取 pipeline 程式執行時間及取代原程式 pipeline 執行的參數(ppline.run(int))使其能代入自動產生的 pipeline token number test case，最後再藉由 Target-Host 機制來實現自動化多回合測試。當程式執行完，在 Host-Side 會顯示出執行時間的分布圖，及 CPU 每個核心的使用率，最後我們將會自動分析所收集的測試資料，提供測試工程師一個建議平行化門檻值範圍，使 pipeline 平行程式達到較好的效能。

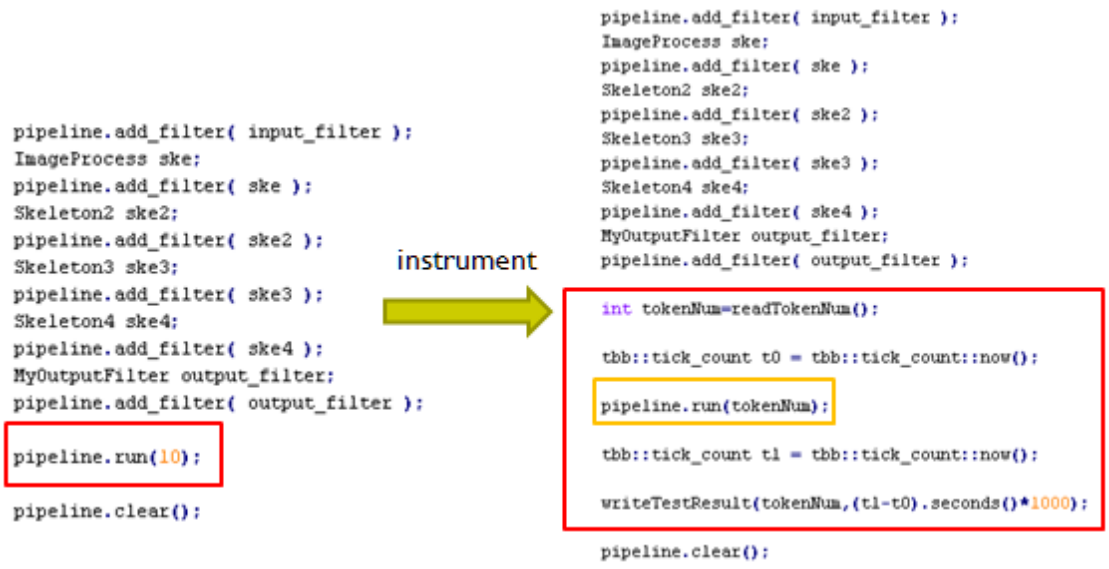


圖 105 Instrument code 至 TBB 平行程式

經過我們多次的分析，我們發現量測 token 數目對於程式執行時間的分布大都如圖 106 所示，因此我們以五個 token 數為單位，計算其平均值，並找出一個建議的 token 數範圍。以圖 107 所示，量測的 token 數為 1~10，我們取 token 數 1~5 執行時間的平均值為 17.6 毫秒，token 數 2~6 執行時間的平均值為 12.4 毫秒，以此類推，當我們發現下一次的平均值大於目前的平均值(Token No. 4~8 的執行時間大於 Token No. 3~7 的執行時間)，因此我們建議的 token 數為目前取平均數的範圍，以圖 9 為例，我們所建議的 token 數為 3~7。

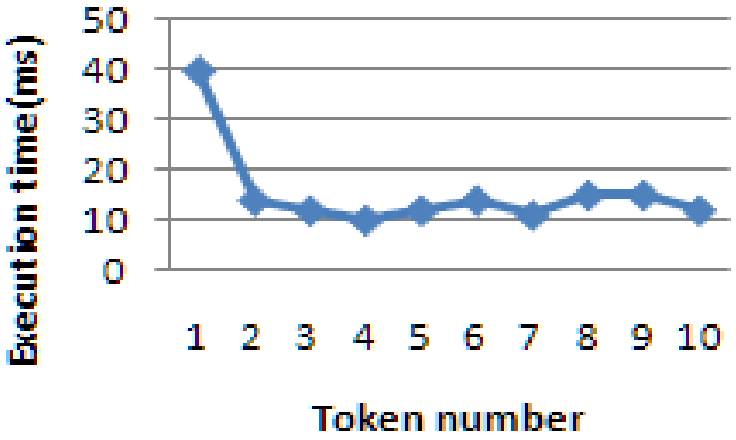


圖 106 TBB 效能量測時間分布範例

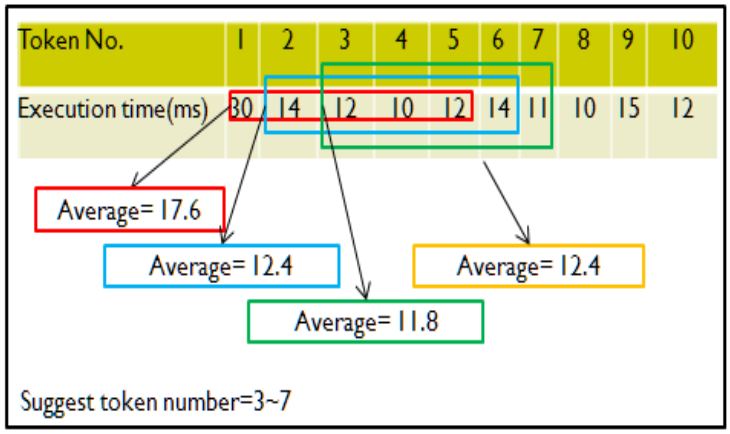


圖 107 TBB pipeline 效能量測方法

2. 系統架構

本章節我們介紹我們所發展的嵌入式軟體自動化測試環境(ATEMES，automatic testing environment for multi-core embedded software)，本系統(ATEMES)主要分成四個部分，分別為前置處理模組(Pre-processing Module, PRPM)、主機端自動測試模組(Host-Side Auto Testing Module, HSATM)、目的端自動測試模組(Target-Side Auto Testing Module, TSATM)、後置處理模組(Post-processing Module, POPM)，透過這四

個模組來實現遠端自動化測試(cross-testing)嵌入式軟體，系統提供的測試功能包含了單元測試、覆蓋率測試，平行程式效能量測；系統架構如圖 108 所示。

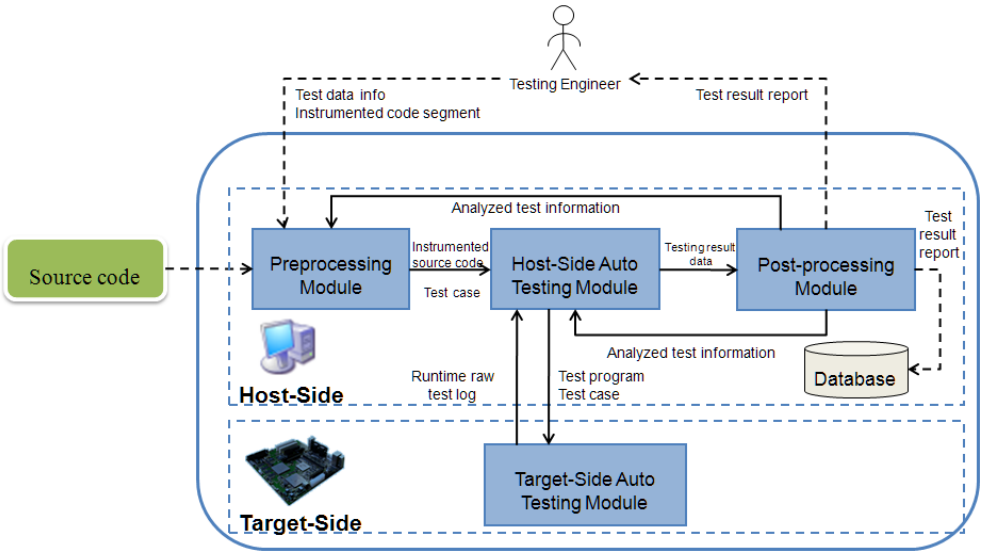


圖 108 ATEMES 系統架構

系統模組說明

本系統 ATEMES 共分為五層，如圖 109 所示，最底層為硬體平台，在 Host-side 我們使用 X86 平台，Target-Side 為 ARM11 MPCore 平台；第二層為 linux 作業系統；第三層為我們所使用的 open source library，包含了，GCOV、Cppunit、Jfreechart、Pin tool、TBB；第四層、第五層為我們所發展的模組，前者為 ATEMES 系統的模組，後者為 Host-side 在 JAVA 平台上執行的圖型化的操作介面。

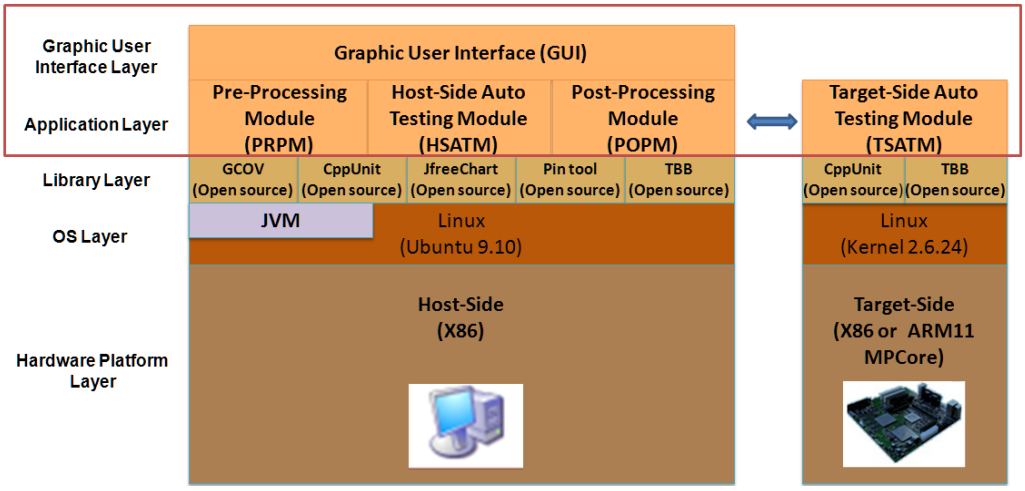


圖 109 ATEMES 系統階層式架構

a. 前置處理模組 (Pre-Processing Module, PRPM)

PRPM 為測試的前置處理，首先讀取待測程式，透過 Code Analyzer 來剖析(parse)及切割(tokenize)原始程式碼，擷取出程式內部資訊(例如 function 名稱、參數、Key-Word)，而 Test Case Generation Module 會依據測試需求(unit testing、coverage testing、performance monitor)自動產生 test case 及 test case 所需的 test data。Code Analyzer 會找出需插入程式片段(instrument code)的位置使 Code Instrumentation Module 能產生測試所需的 instrumented program，此外測試工程師也可自行輸入 test case、test data 及 instrument code 所需的資訊。此模組架構如圖 110 所示。

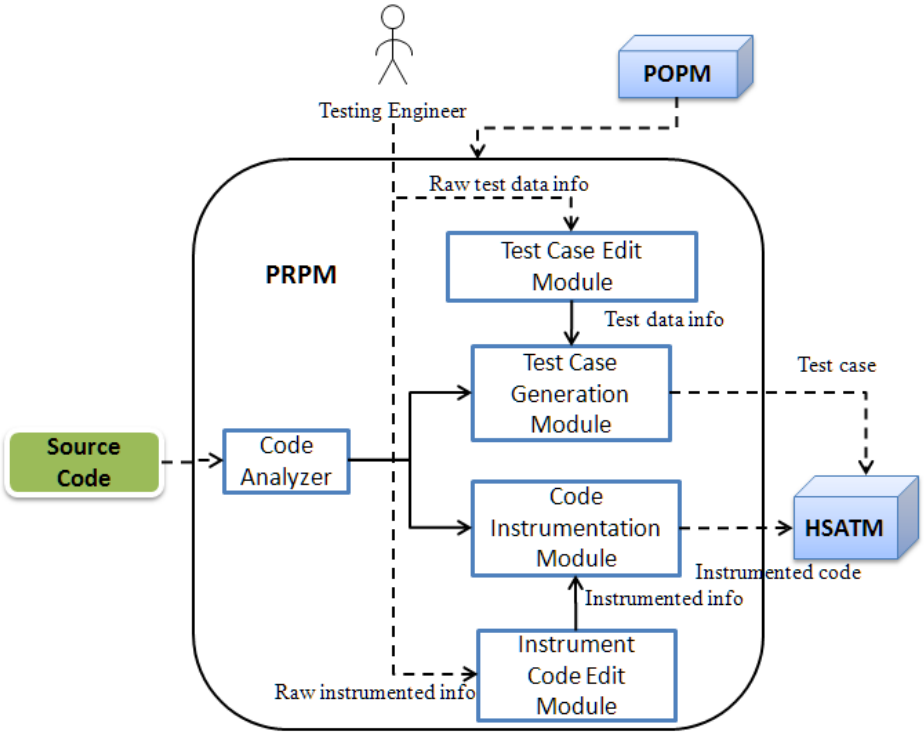


圖 110 PRPM 模組架構

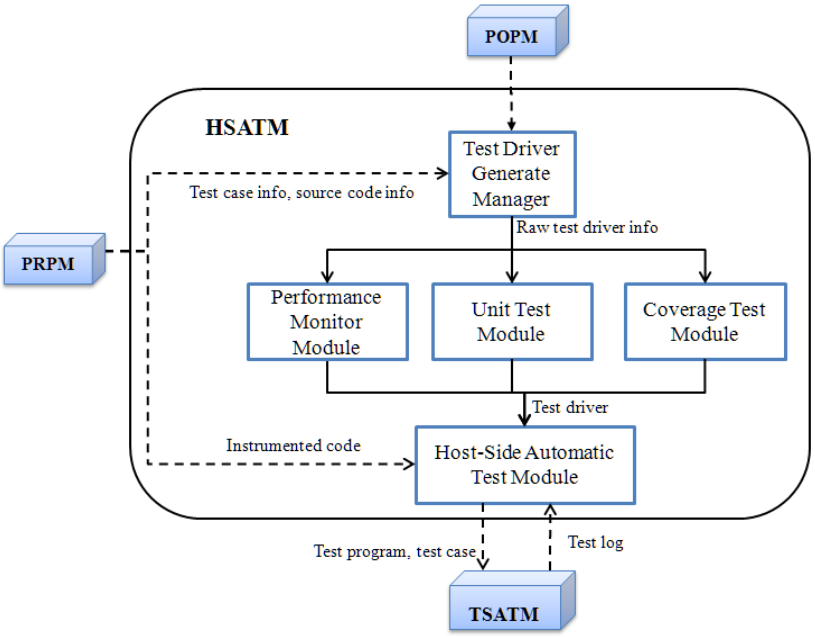


圖 111 HSATM 模組架構

b. 主機端自動化測試模組 (Host-Side Auto Test Module, HSATM)

HSATM 為實現自動化測試主要的模組，Test Driver Generate Manage 接收 PRPM 模組所產生的 test case 及 instrumented code，並依據測試需求執行所需的測試模組來產生 test driver，包含 Unit Test Module、Coverage Test Module 及 Performance Monitor Module，當 test driver 產生，Host-Side Automatic Test Controller 會將 PRPM 所產生的 instrumented code 及 test driver 編譯為測試用的執行檔(test program)，最後再將 test program 及 test case 傳送給 TSATM 模組，本模組會動態通知 PRPM 模組，以產生新的 test case，來達到動態測試；此模組架構如圖 13 所示。HSATM 為實現自動化測試主要的模組，Test Driver Generate Manage 接收 PRPM 模組所產生的 test case 及 instrumented code，並依據測試需求執行所需的測試模組來產生 test driver，包含 Unit Test Module、Coverage Test Module 及 Performance Monitor Module，當 test driver 產生，Host-Side Automatic Test Controller 會將 PRPM 所產生的 instrumented code 及 test driver 編譯為測試用的執行檔(test program)，最後再將 test program 及 test case 傳送給 TSATM 模組，本模組會動態通知 PRPM 模組，以產生新的 test case，來達到動態測試；此模組架構如圖 111 所示。

c. 後置處理模組 (Post-Processing Module, POPM)

本模組主要是在分析及處理測試的結果，將其測試結果分類成各個項目，並產生文字及圖形的報表，Test Log Manager 會依據測試需求，執行不同的剖析器(Parser)即時分析 Test Log 的資訊，包含 Coverage Test Log Parser、Unit Test Log Parser、Performance Monitor Log Parser，並且將分析後的結果動態的回饋給 PRPM 及 HSATM，做為下一步測試的參考。此外，測試工程師可在測試執行時即時地觀察 coverage test 結果或 performance monitor 結果，Test Result Presentation Module 會將測試結果以文字描述及圖形化介面的方式來呈現結果。模組架構如圖 112 所示。

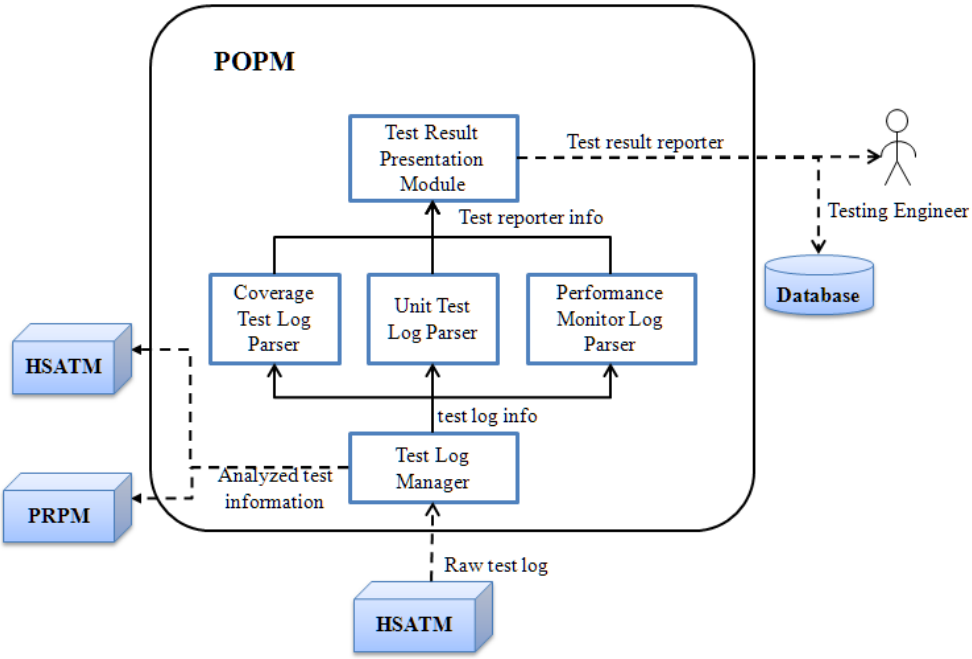


圖 112 POPM 模組架構

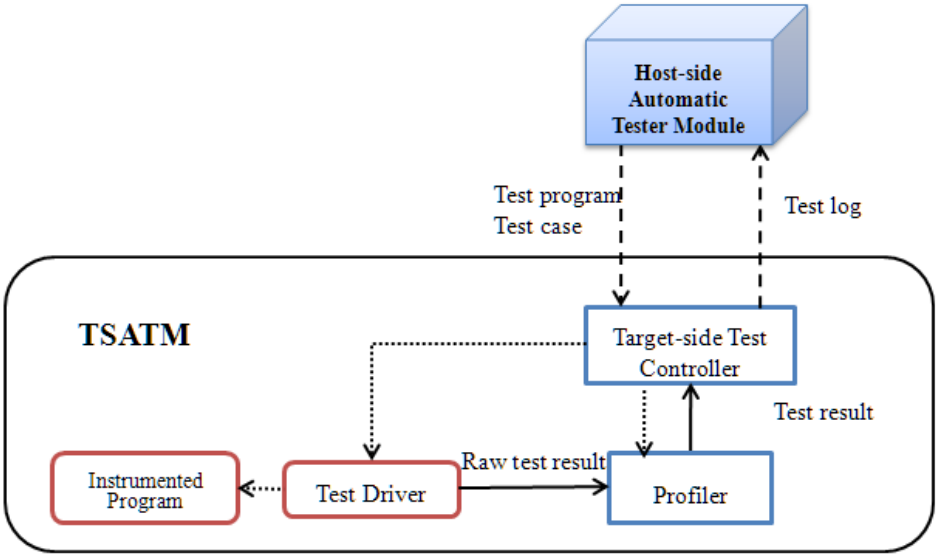


圖 113 TSATM 模組架構

d. 目的端自動化測試模組 (Target-Side Test Controller Module, TSATM)

本模組為一個測試執行環境，透過 Target-side Test Controller 來觸發 test driver 及 Profiler 執行，並可即時與 HSATM 互相傳遞訊息，包含 test case、test driver 等等。在執行測試的程式時，Profiler 會依據測試需求即時收集資料，並將測試過程及結果相關的資料記錄在 test log 上，test log 可能存在記憶體中或是一個檔案，最後將 test log 傳回至 POPM 執行結果的分析及測試記錄的輸出。模組架構如圖 113 所示。

3. 實驗設計與實驗結果分析

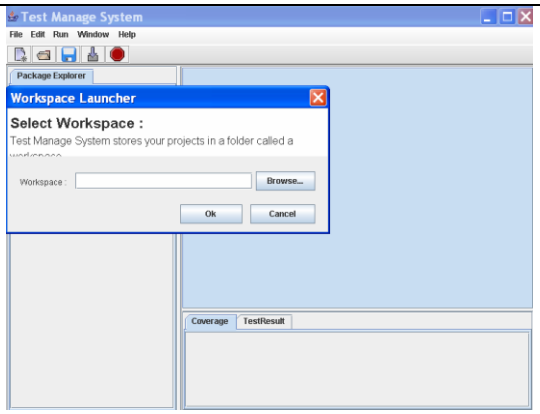
實驗環境

我們將 ATEMES 系統分為 Host-Side 及 Target-Side，Host-Side 以 JAVA 程式開發，而 Target-Side 則以 C++開發，兩端以網路 socket 傳遞訊息。嵌入式系統平台為 ARM 11 MPCore baseboard，四核心的 CPU，我們在嵌入式開發板上執行單元測試、覆蓋率測試，在效能量測部分，可監控每個核心的使用率，使測試工程師可以了解軟體在嵌入式平台上執行效能為何，並且可量測 TBB pipeline 平行化的效率，以下將列出我們所使用的軟硬體平台：

- 主機端硬體平台: IntelR Core? 2 Duo CPU P8400
- 2. 主機端作業系統: Linux Ubuntu 9.10
- 目標端硬體平台: The platform baseboard for ARM 11 MPCore with 4 ARM11MPCore CPUs
- 目標端作業系統: Linux Kernel 2.6.24

系統使用介面展示

在本節中，我們展示 ATEMES system 實際使用的過程，步驟說明如下：

Step 1: 在 Host-side 執行 ATEMES system，選擇專案執行路徑，如圖 114 所示。	 圖 114 開啟測試專案介面

Step 2: 開啟測試程式編輯介面，左邊可列出所有待測程式的檔案，右邊可列出所選擇的待測程式的 source code，使用者可在此編輯、新增、刪除待測程式的 source code，如圖 115 所示。

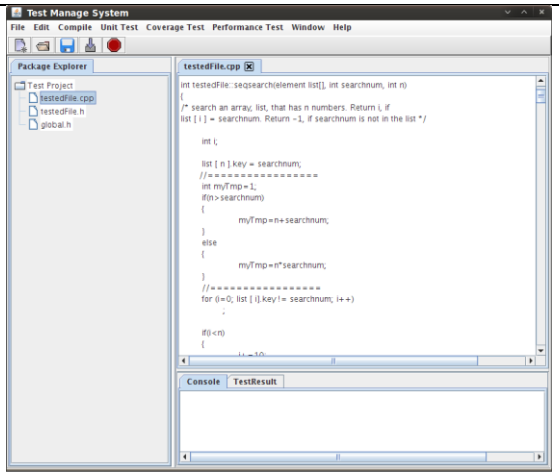


圖 115 待測程式編輯介面

Step 3: 測試工程師可以使用"Test Case and Driver Generator"功能，自動產生 test case 及 test driver 的 source code。ATEMES 在執行"Test Case and Driver Generator" 功能時，"Code Analyzer"會去 parse 待測程式的所有 method 的 signature，包含 method name、parameter type、return type 等資訊，並自動產生 test case and test driver；作為呼叫待測程式及讀 test data 用。此外，使用者也可以再編輯待測程式中每個 function 的 test case。如圖 116 所示，左邊可列出所有待測程式的檔案，右邊為系統自動產生的 test case 的 source code。

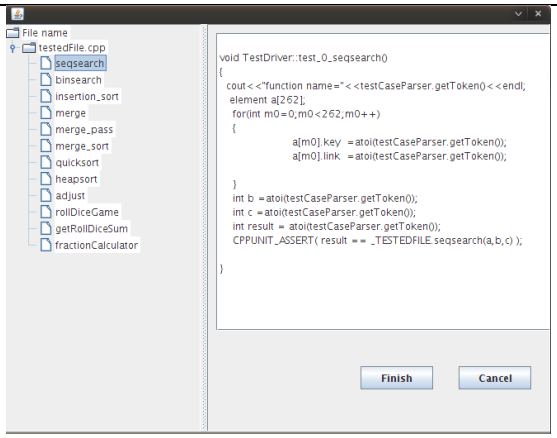


圖 116 Test case 和 test driver 產生器介面

Step 4: 測試工程師可以使用"Generate Random Test Case"功能，來自動產生隨機測試案例，且使用者可自行決定測試案例產生的數目，如

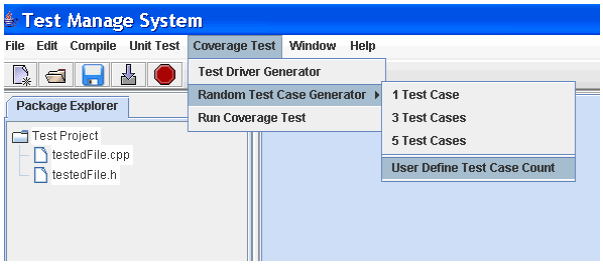


圖 117 所示

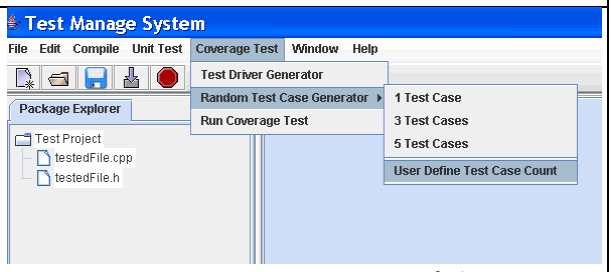


圖 117 產生 test case 選單介面

Step 5: 測試工程師可以使用"Run Coverage Test"功能，執行 Coverage Test，系統會自動將待測試程式、test case and test driver 做 cross-compile，並將執行檔及 test data file 傳送至 target-side (embedded system)執行測試工作，測試工作可以是 single-round，也可做 multi-round 的測試，在測試時，測試工程師可在 host-side 即時的觀察及接收測試結果，如圖 118 所示。

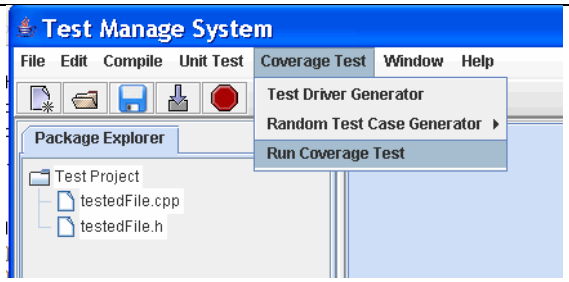
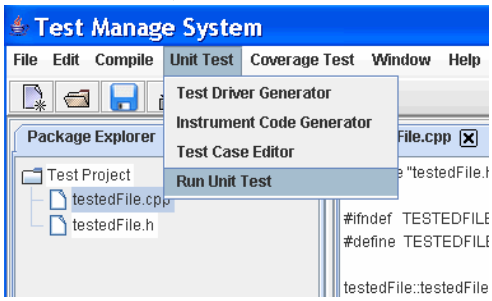


圖 118 執行 coverage test 選單介面

Step 6: 測試工程師可以使用"Run Unit Test"功能，如圖 119 所示，執行 Unit Test。測試工程師可根據待測程式裡的 function，新增、編輯 test driver 所要讀入的 test data and result，編輯完成即產生 test case and test driver 的 source code。系統會自動將待測試程式、test case and test driver 做 cross-compile，並將執行檔及 test data file 傳送至 target-side (embedded system)執行，並等待接收測試結果。



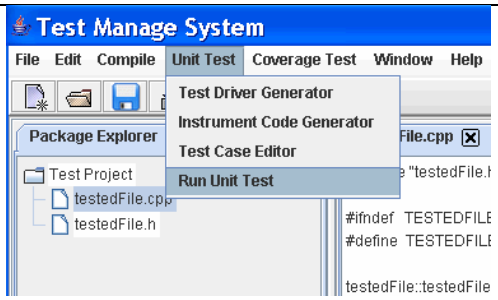
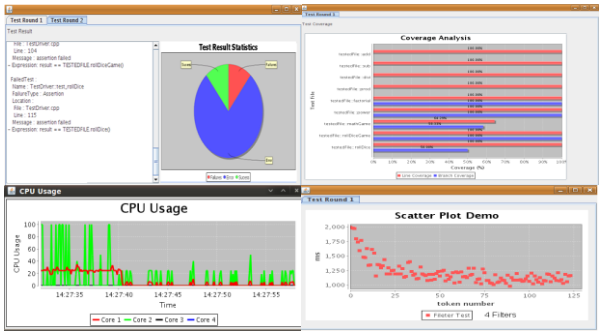


圖 119 執行單元測試選單介面

Step 7: ATEMES system 在執行測試時，target-side 的 Profiler 會在 runtime 時收集資訊並傳回 host-side，host-side 會即時的將 test result information 做分析，並做視覺化的呈現。如圖 120 所示，為執行 coverage testing，unit testing，performance monitor，pipeline performance monitor 的結果。



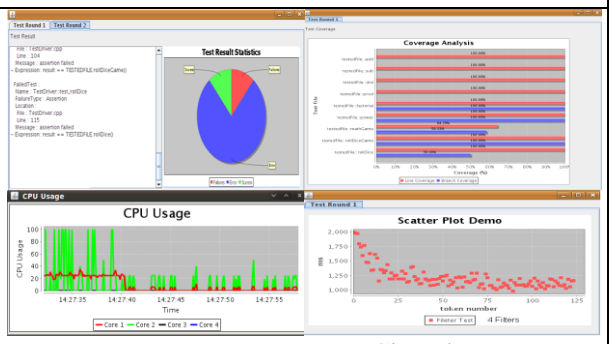


圖 120 測試結果視覺化介面

實驗及分析

本章節將做六組實驗，分別為以 C/C++資料結構範例執行自動化覆蓋率測試及單元測試，以矩陣相乘平行程式範例執行多核心效能監控、最後以 TBB pipeline 實作影像處理範例，針對不同 CPU core 數目的 TBB pipeline 效能量測，及針對不同的 stage 數目的 TBB pipeline 效能量測。

a. 實驗一 - 自動化覆蓋率測試

本實驗以 13 個 functions 當做待測程式，為挑自 C/C++資料結構參考書籍的範例程式碼，參數輸入的型態包含 primitive type 及 object type。測試環境採用 random

testing 的方式產生 test data，對每一個 function 而言，每一回合產生 3 組 test data，執行 20 回測試。在 runtime 時，使用者可觀察每一回合的 test coverage 圖 121 及最後的測試結果。系統並會記錄每一回合程式執行的資訊，包含 line coverage，branch coverage，executing time 及失敗的 test case 內容等。

實驗結果：

由圖 121、表 12、表 13 所示，就 insertion_sort()而言，當到第五回合，line coverage 及 branch coverage 皆可達到 100%，每一個 round 測試完成後，就其它 function 而言，對於 line coverage 及 branch coverage 皆有提升，但是就 fractionCalculator() 來看 line coverage 和 branch coverage 到了第四回合時，為 92%及 65%，且 coverage 很難再提升，會有這樣的瓶頸是因為我們所採用的演算法為 random testing，我們將會再利用其它的演算法來提升 coverage。

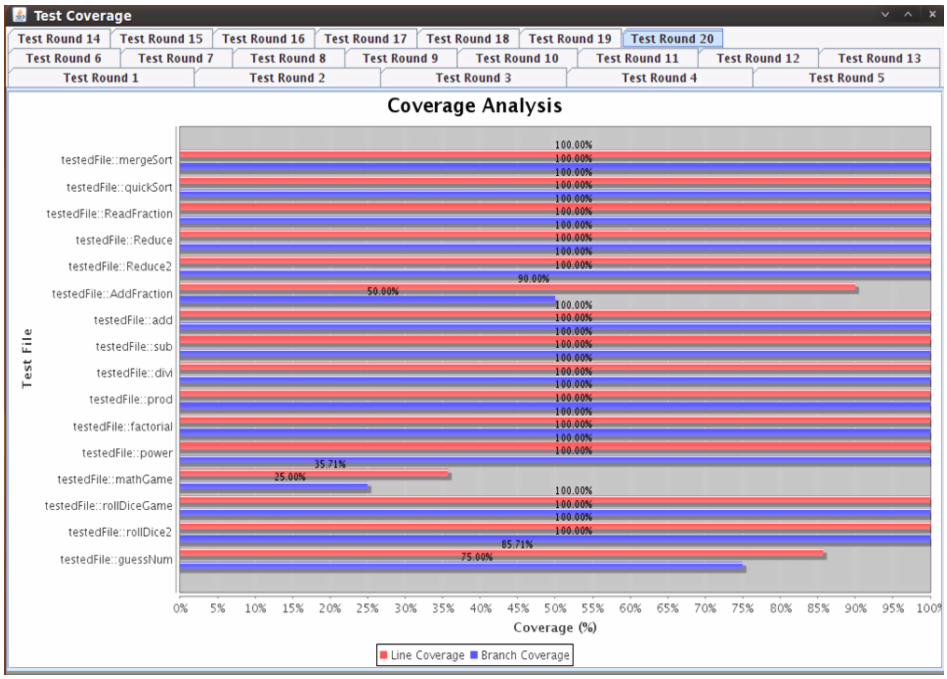


圖 121 Coverage test 測試結果介面

表 12Line Coverage 測試結果

Round / Function	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	20th
seqsearch	92%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
binsearch	79%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
insertion_sort	91%	91%	91%	91%	100%	100%	100%	100%	100%	100%	100%
merge	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
merge_pass	92%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
merge_sort	92%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
quicksort	26%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
heapsort	92%	92%	92%	92%	92%	92%	92%	92%	92%	100%	100%
adjust	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
rollDiceGame	72%	83%	83%	83%	83%	94%	94%	94%	100%	100%	100%

getRollDiceSum	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
fractionCalculator	55%	63%	80%	92%	92%	92%	92%	92%	92%	92%	92%
Execute Time (ms)	958	907	939	889	980	891	994	926	1006	977	909

表 13 Branch Coverage 測試結果

Round / Function	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	20th
seqsearch	83%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
binsearch	60%	90%	90%	90%	90%	90%	90%	90%	90%	90%	90%
insertion_sort	90%	90%	90%	90%	100%	100%	100%	100%	100%	100%	100%
merge	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
merge_pass	88%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
merge_sort	75%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
quicksort	17%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
heapsort	83%	83%	83%	83%	83%	83%	83%	83%	83%	100%	100%
adjust	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
rollDiceGame	64%	82%	82%	82%	82%	91%	91%	91%	100%	100%	100%
getRollDiceSum	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
fractionCalculator	29%	41%	53%	65%	65%	65%	65%	65%	65%	65%	65%
Execute Time(ms)	958	907	939	889	980	891	994	926	1006	977	909

b. 實驗二 - 單元測試實驗

本實驗同樣以實驗一的 functions 當做待測程式，經過我們測試環境分析原始程式碼後，自動找出要需要測試的 function，並可利用 test data 編輯介面，由使用者輸入所需的 test data，及預期結果值，接著，本系統會自動產生 test cases 及 test drivers，並自動做 cross-compile，上傳測試執行檔至 ARM 11 MPCore platform，當傳送成功將會自動開始測試工作。測試結果的資訊也是可在測試執行時傳回 host-site 做即時的分析，系統並會記錄每一回合程式執行的資訊，包含執行時間，成功的 test case 及失敗的 test case 內容等。

實驗結果：

如表 14 所示，以文字描述測試結果，包含了測試案例成功、失敗、錯誤的資訊，在介面的右邊則以圓餅圖總結測試結果，以本實驗為例，通過的測試案例總共佔 67% (20 test cases)，失敗的測試案例總共佔 16.5% (5 test cases)，最後為發生錯誤的 test case 總共佔 16.5% (5 test cases)。

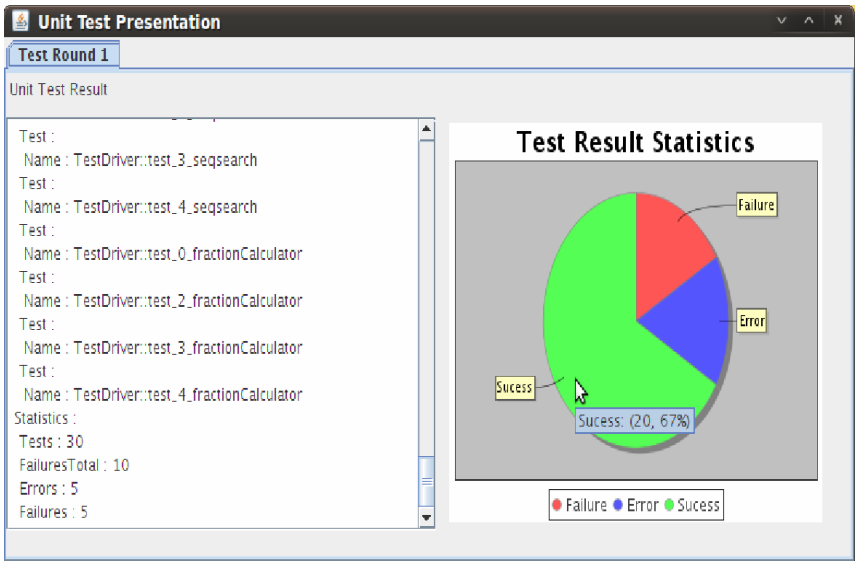


圖 122 Unit test 測試結果介面

表 14 單元測試結果

Function name	Test case				
	0	1	2	3	4
getSubString	success	exception out of range	success	success	exception out of range
electricityTraiff	success	assertion failed	success	assertion failed	success
factorial	success	success	assertion failed	assertion failed	success
insertion_sort	success	success	Segment fault	Segment fault	Segment fault
seqsearch	success	success	success	success	success
fractionCalculator	success	assertion failed	success	success	success

c. 實驗三 - 多核心效能監控

本實驗將測試 2 維矩陣乘法運算的範例程式效能，矩陣大小為 256*128 及 128*256，本系統將會自動分析待測程式碼，並 instrument 程式片段，使得能夠監控執行時間，並且自動將 cross-compile 後的測試執行檔上傳至 ARM 11MPCore 開發板上自動執行測試，在程式執行時，使用者能夠即時觀看 CPU 每個核心的使用率，且同時測試結果也會自動回傳至 Host-Side 做即時的分析。

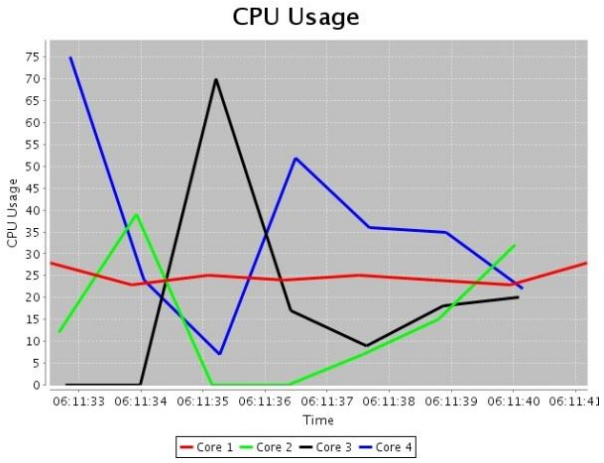
實驗結果：

程式執行的效能如圖 123 及表 15 所示，由圖 121 可以看出，core 1 的效能一直保持穩定狀態，而 core 2 使用率較低，由表 15 可以看出，對於 CPU 平均的使用率，core 3 和 core 4 相對於另外兩個核心還要高，再來為 core1，最後為 core4，

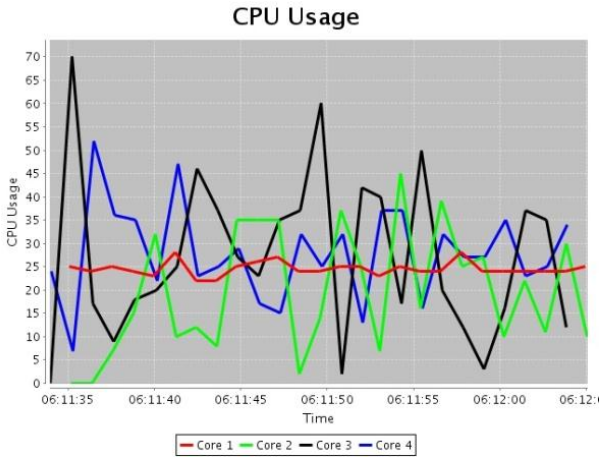
如此一來可以清楚看出，程式執行時，使用 core3 及 core4 為最頻繁的，且可發現對於 CPU 的使用率，還有可以加強的地方。

表 15 Multi-core CPU utilization

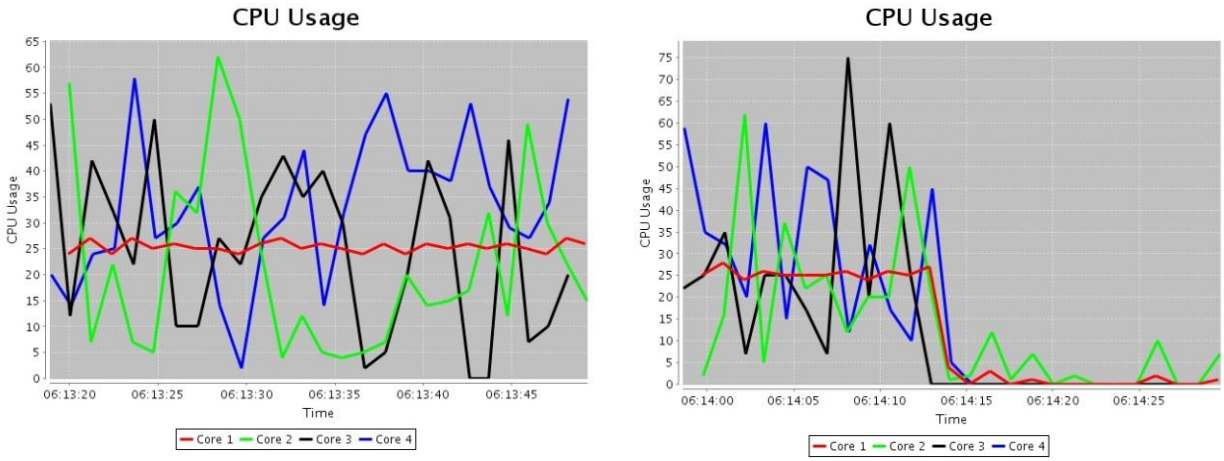
Core No.	Average (%)	Max (%)	Min (%)
core 1	25.05969	28	22
core 2	18.97810	62	0
core 3	29.93431	76	0
core 4	31.05839	75	0



(a)



(b)



(c) (d)

圖 123 多核心 CPU 使用率視覺化介面

d. 實驗四 - 不同 CPU core 數目在 TBB pipeline 技術下的效能量測

本實驗主要目的為運用我們所發展的工具來觀測 TBB pipeline 技術在不同的 CPU core 數目執行的效率，且不同的 token 數對效能的影響。以 TBB 使用影像處理範例，將不同階段的影像處理以 pipeline 方式處理，預處理的圖片大小為 1280*1024，對影像做邊緣化及去雜訊。在此我們將 pipeline 分為四個階段，分別為讀取影像、邊緣化、去雜訊、輸出影像；讀取影像、輸出影像我們設定以循序方式執行(serial)，而邊緣化、去雜訊為平行處理(parallel)。本系統將會自動產生 pipeline token number 的 test case，每個 test case 為 100 組資料集合，分別為 1、2、3…100 的 token 數目，自動執行多回合效能量測並找出一個建議的 pipeline 平行化門檻值及顯示各個 CPU 核心數的使用率。

實驗結果：

實驗結果如圖 39 所示，我們可以得知使用的 CPU core 數目越多則效率越高，以 pipeline token 數來看，當 token 數小於 5 時效率皆較差，當 token 數大於 5 時則效率明顯提升很多，因此我們可以發現 pipeline token 數越多則效率越佳，但對於 TBB pipeline，token 數到了某個數目則不會再有明顯的效率提升，反而會增加系統的負擔，以下(1)~(4)我們將分別說明以不同的 CPU core 數及不同的 token 數執行狀況及建議使用的 token 數目：

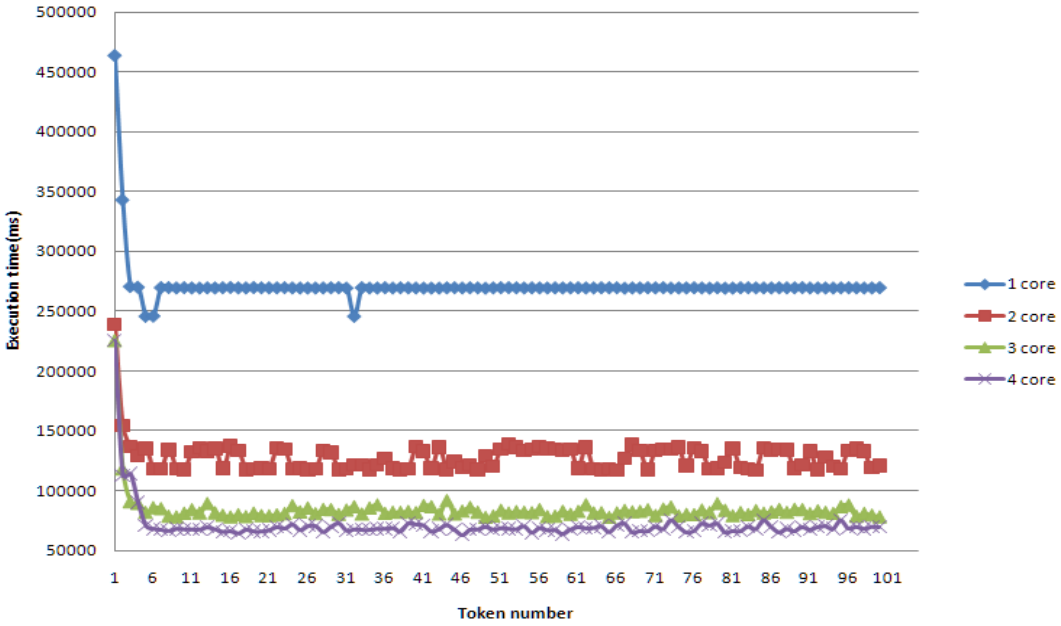


圖 124 TBB pipeline 以 1~4 CPU core 執行的效能量測

(1) TBB pipeline 效能量測 1 - 以 1 個 CPU core 執行

如圖 125 所示待測程式平均執行時間約為 271 秒，當 pipeline token 數為 1 效率最差，需 463 秒才執行完成，而 pipeline token 數大約為 5 至 9 時能達到較高的效率，pipeline token 數大於 21 皆到平穩狀態，很難再提升效率了，在 CPU 的使用率如表 16 所示，core 1、core2、core 4 的使用率偏不高，這是因為我們設定軟體執行時使用的 CPU core 數目為 1，core 2 的平均使用率最高。

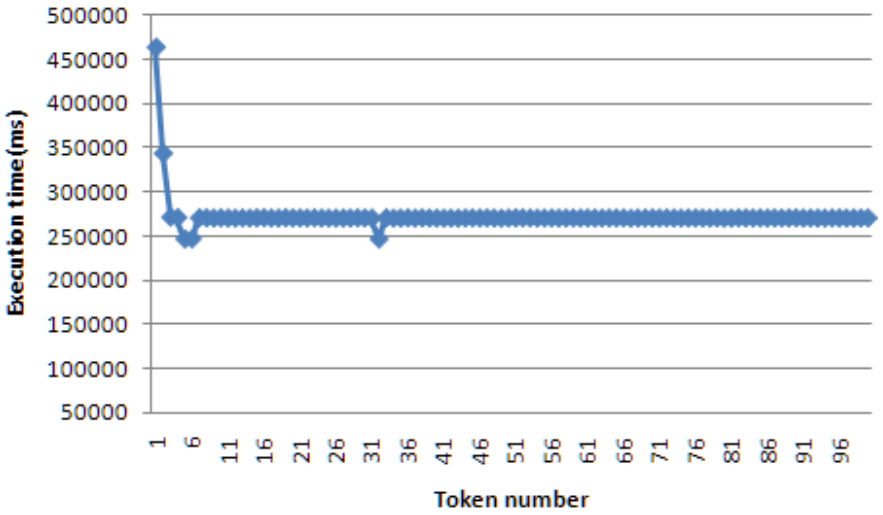


圖 125 TBB pipeline 以 1 個 CPU core 執行的效能量測

表 16 以 1 個 CPU core 執行 TBB pipeline 的 CPU utilization 效能量測

CPU utilization / Core No.	Min (%)	Max (%)	Average (%)
core 1	0	30	25
core 2	0	100	99

core 3	0	0	0
core 4	0	0	0

(2) TBB pipeline 效能量測 2 - 以 2 個 CPU core 執行

如圖 126 所示待測程式平均執行時間約為 127 秒，當 pipeline token 數為一時效率最差，需 239 秒才執行完成，而 pipeline token 數大約為 6 至 10 時能達到較高的效率，pipeline token 數大於 26 皆到平穩狀態，很難再提升效率了，在 CPU 的使用率如表 17 所示，core1 及 core2 及 core3 的使用率平穩，core4 的平均使用率最低。

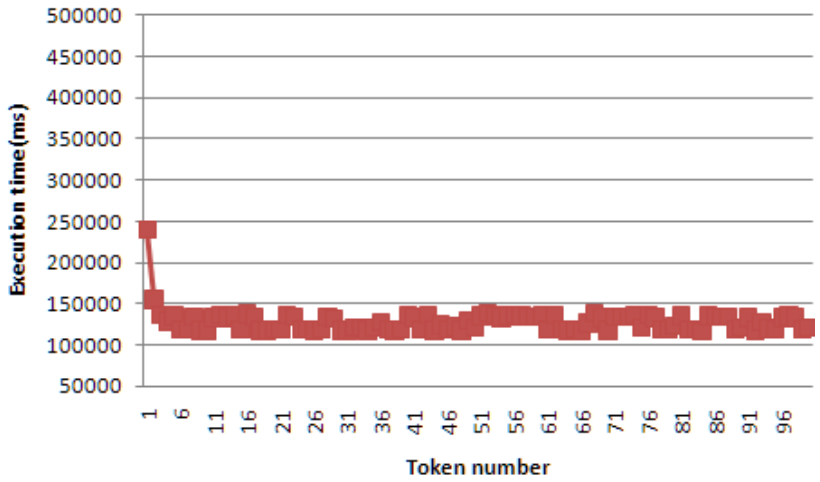


圖 126 TBB pipeline 以 2 個 CPU core 執行的效能量測

表 17 以 2 個 CPU core 執行 TBB pipeline 的 CPU utilization 效能量測

CPU utilization / Core No.	Min (%)	Max (%)	Average (%)
core 1	0	57	49
core 2	0	100	99
core 3	0	100	99
core 4	0	0	0

(3) TBB pipeline 效能量測 3 - 以 3 個 CPU core 執行

如圖 127 所示，我們可以看出效率有明顯的改進，待測程式平均執行時間約為 85 秒，當 pipeline token 數為 1 效率最差，需 225 秒才執行完成，而 pipeline token 數大約為 8 至 12 時能達到較高的效率，pipeline token 數大於 36 皆到平穩狀態，很難再提升效率了，在 CPU 的使用率如表 18 所示，core1、core、core3 及 core4 的 CPU 使用率平均皆算高，效率較好。

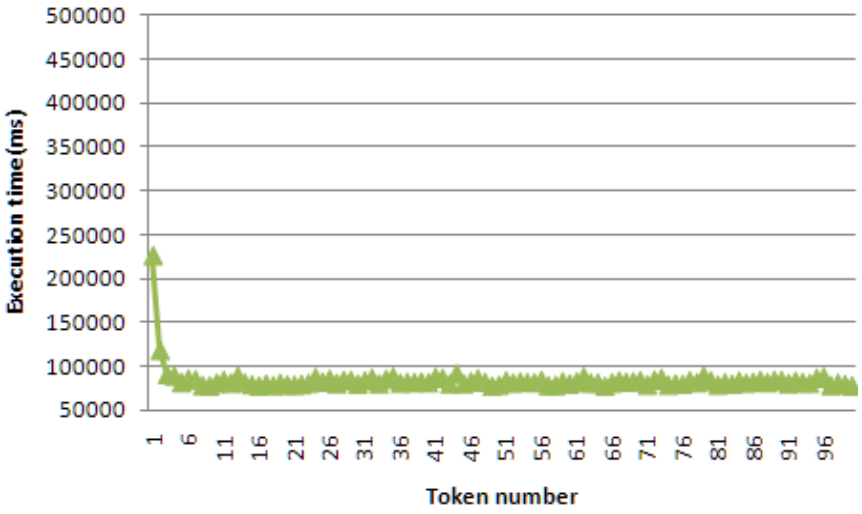


圖 127 TBB pipeline 以 3 個 CPU core 執行的效能量測

表 18 以 3 個 CPU core 執行 TBB pipeline 的 CPU utilization 效能量測

CPU utilization / Core No.	Min (%)	Max (%)	Average (%)
core 1	0	78	68
core 2	0	100	85
core 3	0	100	90
core 4	0	100	96

(4) TBB pipeline 效能量測 4 - 以 4 個 CPU core 執行

如圖 128 所示，我們可以看出效率有明顯的改進，待測程式平均執行時間約為 71 秒，當 pipeline token 數為 1 效率最差，需 200 秒才執行完成，而 pipeline token 數大約為 7 至 11 時能達到較高的效率，pipeline token 數大於 30 皆到平穩狀態，很難再提昇效率了，在 CPU 的使用率如表 19 所示，4 個 core 的使用率平均算高，執行的效率最好。

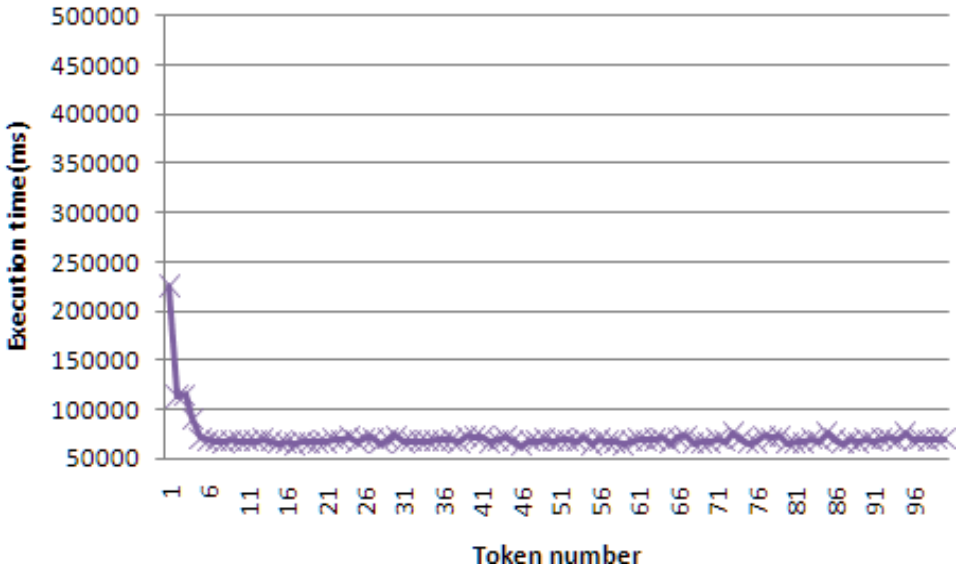


圖 128 TBB pipeline 以 4 個 CPU core 執行的效能量測

表 19 以 4 個 CPU core 執行 TBB pipeline 的 CPU utilization 效能量測

CPU utilization / Core No.	Min (%)	Max (%)	Average (%)
core 1	0	100	91
core 2	0	100	72
core 3	0	100	95
core 4	0	100	99

e. 實驗五 - TBB pipeline 效能分析

本實驗將以 TBB pipeline 實作影像處理程式，測試不同的 stage 數目的效能為何，並使用 4 個 CPU core 執行，預處理的影像大小為 128*128，共執行 12 種不同的影像處理主要工作為以不同的演算法達到影像的邊緣化(gradient、laplacian、Prewitt)、去雜訊(smooth、median)、去除影像不連續的接點(erosion/dilation)、細線化(thinning)、旋轉(rotation)及二值化(threshold)；在 pipeline 中需循序處理的包含讀取影像、輸出影像，可平行處理的為 gradient、laplacian、Prewitt、thinning、smooth、median、erosion、dilation、rotation、threshold，量測的 stage 數目為 1~12，在此我們去比較 1~12stage 的執行時間、CPU 使用率，並量測出一個建議的 pipeline token 數。

實驗結果：

圖 129、表 20 為 1-12 個 stage 分別輸入 1-36 個 token 數的執行時間，由此可以看出，當使用 1 個 stage 時效率最差，其次為使用 2 個 stage，且當 stage 的數目大於 3 時，程式所執行的時間都是差不多的，圖 45~圖 56 將個別列出使用不同 stage 及不同 token 數的執行時間分布。

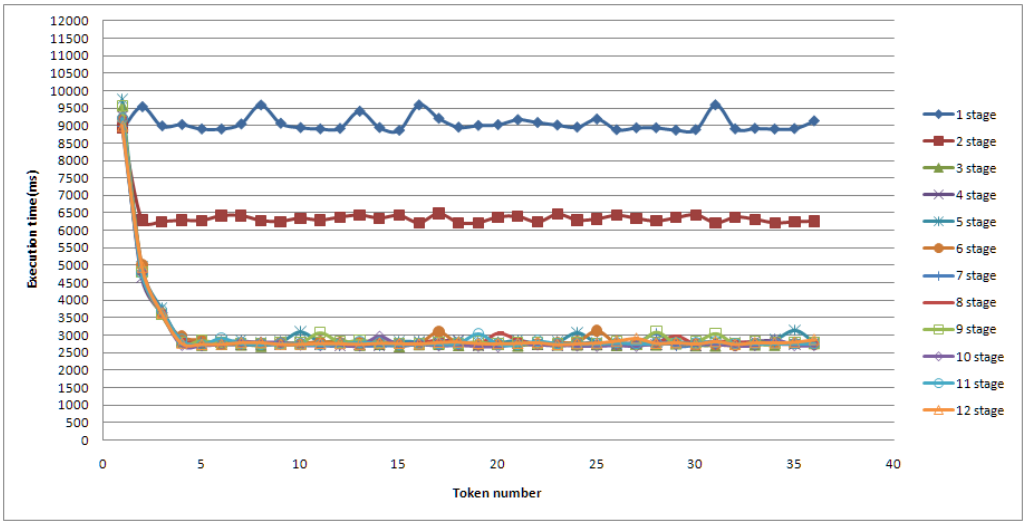


圖 129 TBB pipeline 1~12 stage 效能量測

表 20 TBB pipeline 1~12 stage 執行時間

Token No. / Stage No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	20	25	30	35	36
1	8878	9546	8986	9027	8897	8897	9042	9597	9063	8938	8901	8913	9415	8940	8857	9022	9187	8877	8909	9134
2	8936	6292	6253	6288	6277	6416	6423	6281	6253	6348	6290	6377	6436	6349	6427	6381	6326	6443	6245	6261
3	9565	4872	3725	2840	2729	2751	2736	2695	2769	2756	2766	2849	2723	2750	2683	2818	2743	2707	2790	2772
4	9267	4647	3628	2803	2763	2830	2744	2748	2738	2772	2739	2701	2729	2713	2763	2730	2749	2707	2746	2762
5	9753	4952	3771	2876	2842	2805	2849	2777	2808	3101	2822	2788	2802	2799	2830	2787	2784	2798	3147	2793
6	9198	5011	3586	2966	2856	2751	2739	2754	2778	2759	2822	2785	2794	2820	2745	2722	3127	2743	2753	2736
7	8990	4856	3710	2782	2775	2743	2750	2761	2747	2783	2726	2697	2712	2702	2730	2768	2798	2727	2767	2731
8	9203	4832	3675	2871	2794	2768	2745	2791	2730	2723	2739	2738	2731	2761	2751	3054	2777	2738	2736	2767
9	9575	4814	3597	2784	2847	2790	2771	2754	2772	2772	3075	2813	2860	2800	2758	2740	2764	2805	2792	2773
10	9228	4788	3649	2723	2709	2744	2825	2810	2807	2728	2712	2735	2703	2976	2727	2682	2692	2760	2716	2718
11	9204	4766	3711	2871	2734	2920	2772	2743	2766	2741	2733	2754	2809	2774	2744	2750	2756	2773	2761	2769
12	8997	4954	3637	2772	2737	2739	2790	2792	2740	2741	2767	2758	2740	2773	2762	2758	2776	2750	2803	2878

單位：毫秒

由於本實驗其中包含影像的輸入及輸出，因此只使用 1 個 stage 及 2 個 stage 執行 pipeline，為循序執行，導致執行的時間較慢，如圖 130，圖 131，而其它 stage 數執行的時間的差距不大，表 22 為本實驗針對 stage1 至 stage12 所建議的 token 數。

表 21 TBB pipeline 1~12 stage 執行時間

Token No. / Stage No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	20	25	30	35	36
1	8878	9546	8986	9027	8897	8897	9042	9597	9063	8938	8901	8913	9415	8940	8857	9022	9187	8877	8909	9134
2	8936	6292	6253	6288	6277	6416	6423	6281	6253	6348	6290	6377	6436	6349	6427	6381	6326	6443	6245	6261
3	9565	4872	3725	2840	2729	2751	2736	2695	2769	2756	2766	2849	2723	2750	2683	2818	2743	2707	2790	2772
4	9267	4647	3628	2803	2763	2830	2744	2748	2738	2772	2739	2701	2729	2713	2763	2730	2749	2707	2746	2762
5	9753	4952	3771	2876	2842	2805	2849	2777	2808	3101	2822	2788	2802	2799	2830	2787	2784	2798	3147	2793
6	9198	5011	3586	2966	2856	2751	2739	2754	2778	2759	2822	2785	2794	2820	2745	2722	3127	2743	2753	2736
7	8990	4856	3710	2782	2775	2743	2750	2761	2747	2783	2726	2697	2712	2702	2730	2768	2798	2727	2767	2731
8	9203	4832	3675	2871	2794	2768	2745	2791	2730	2723	2739	2738	2731	2761	2751	3054	2777	2738	2736	2767
9	9575	4814	3597	2784	2847	2790	2771	2754	2772	2772	3075	2813	2860	2800	2758	2740	2764	2805	2792	2773
10	9228	4788	3649	2723	2709	2744	2825	2810	2807	2728	2712	2735	2703	2976	2727	2682	2692	2760	2716	2718
11	9204	4766	3711	2871	2734	2920	2772	2743	2766	2741	2733	2754	2809	2774	2744	2750	2756	2773	2761	2769
12	8997	4954	3637	2772	2737	2739	2790	2792	2740	2741	2767	2758	2740	2773	2762	2758	2776	2750	2803	2878

單位：毫秒

表 22 TBB pipeline 1~12 stage 建議的 token 數

Stage No.	1	2	3	4	5	6	7	8	9	10	11	12
Token No.	1~5	2~6	5~9	5~9	5~9	6~10	5~9	9~13	6~10	4~8	5~9	5~9

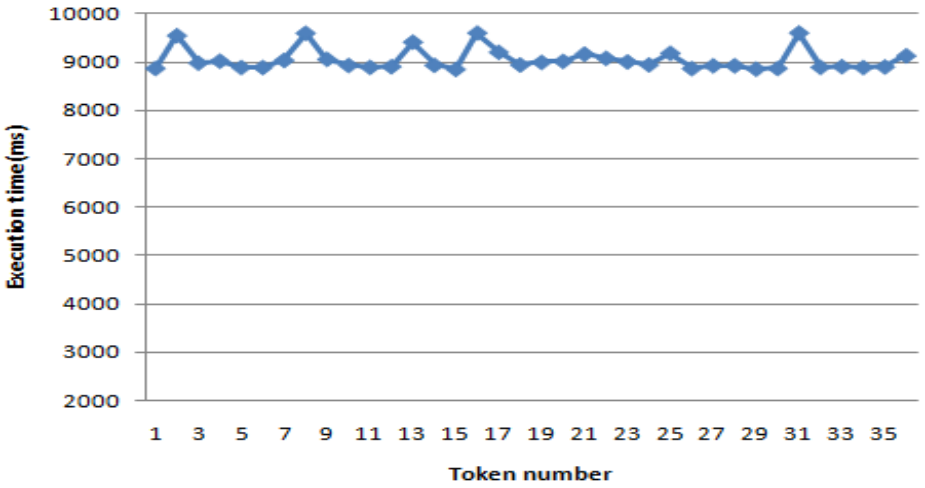


圖 130 使用 1 個 pipeline stage 執行的時間

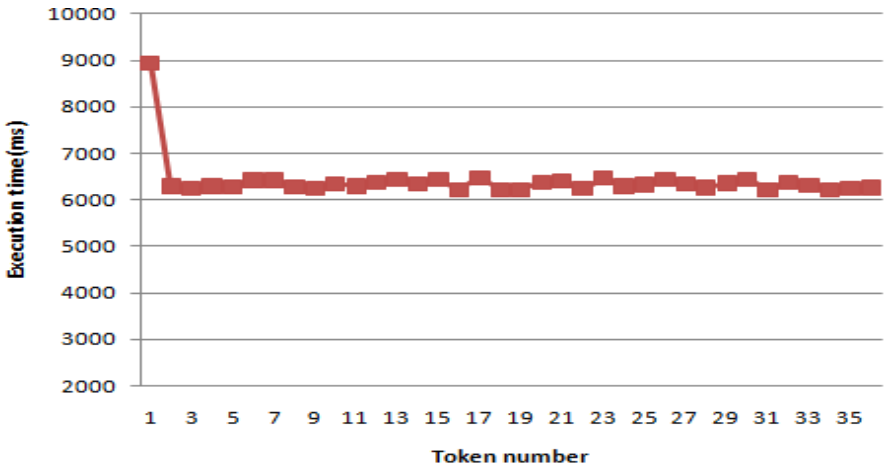


圖 131 使用 2 個 pipeline stage 執行的時間

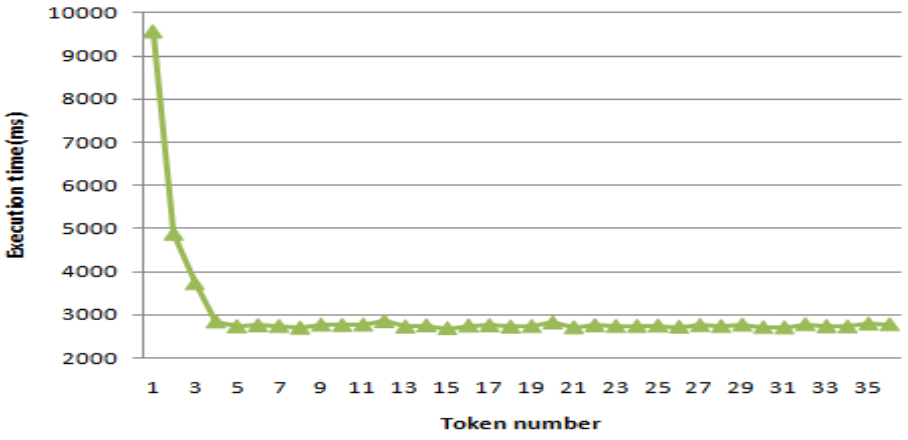


圖 132 使用 3 個 pipeline stage 執行的時間

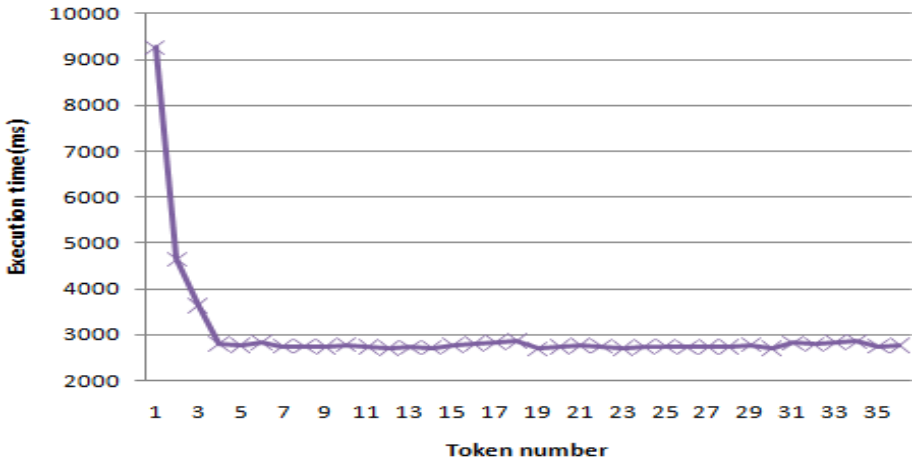


圖 133 使用 4 個 pipeline stage 執行的時間

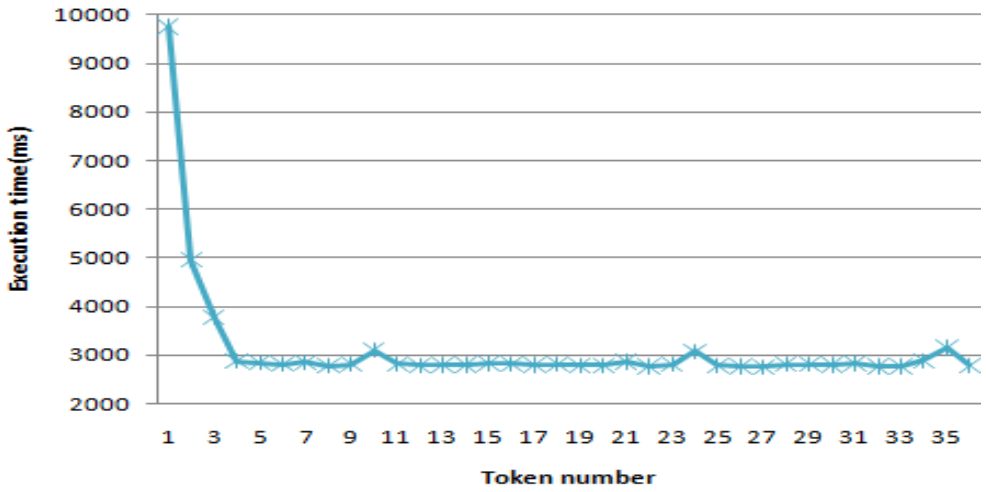


圖 134 使用 5 個 pipeline stage 執行的時間

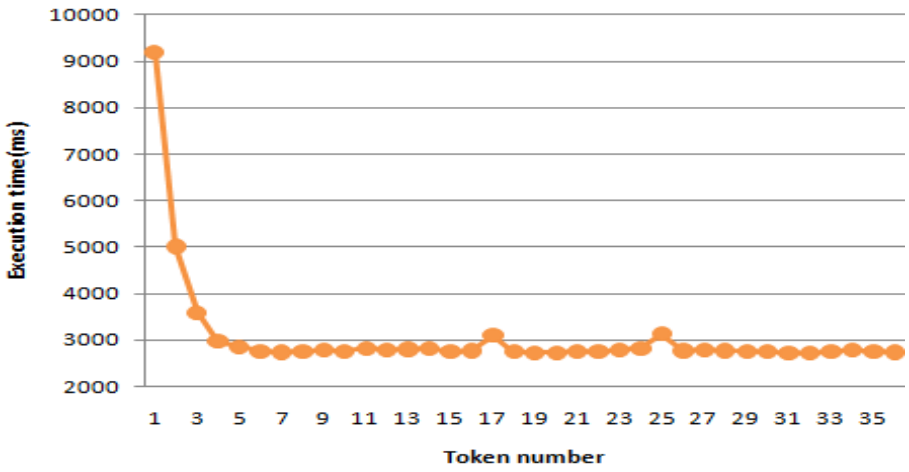


圖 135 使用 6 個 pipeline stage 執行的時間

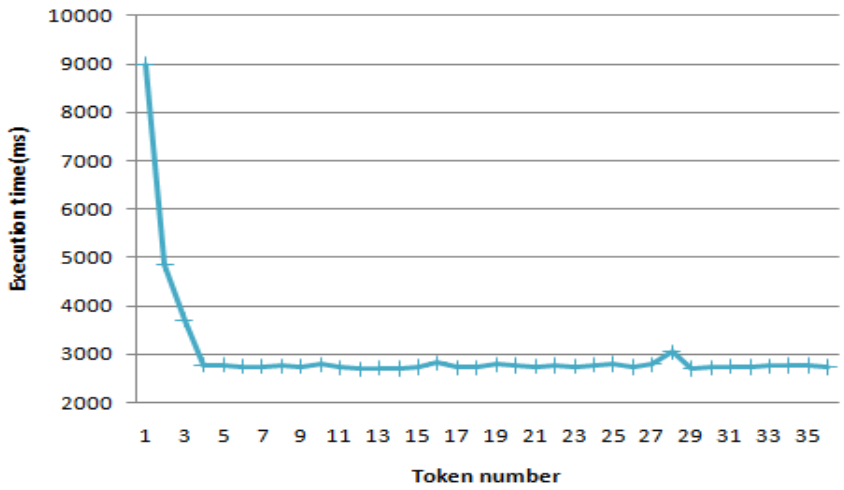


圖 136 使用 7 個 pipeline stage 執行的時間

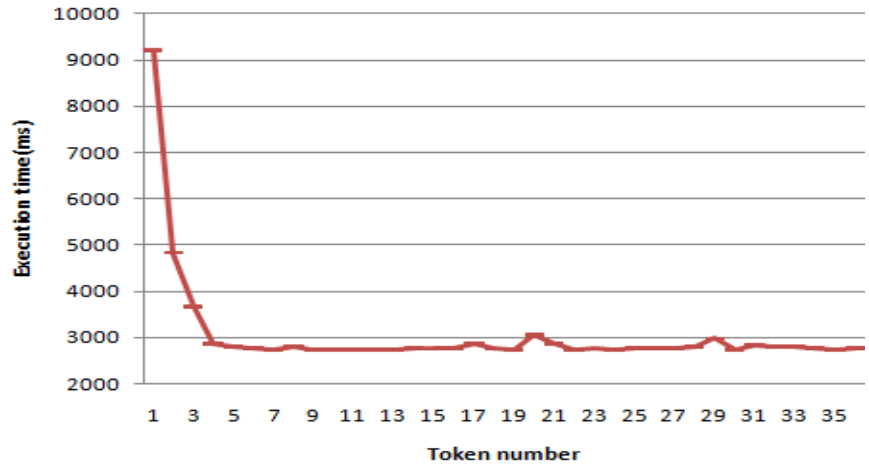


圖 137 使用 8 個 pipeline stage 執行的時間

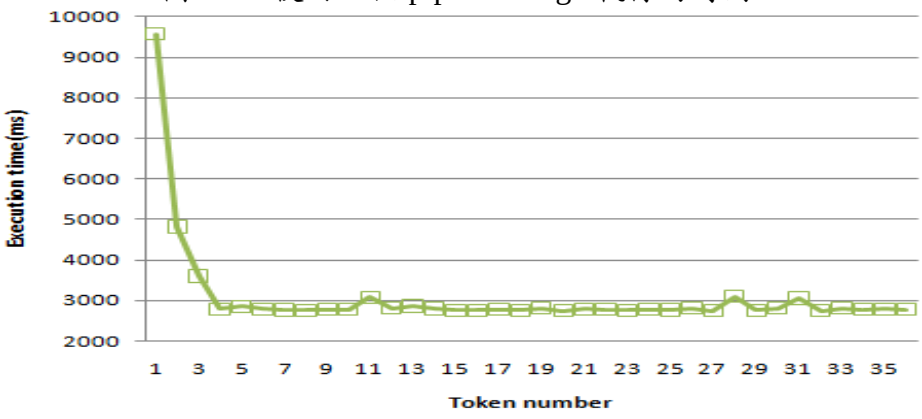


圖 138 使用 9 個 pipeline stage 執行的時間

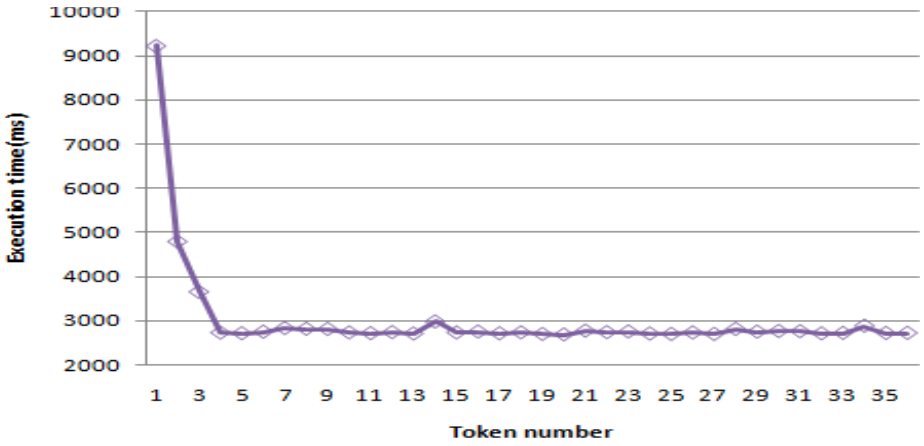


圖 139 使用 10 個 pipeline stage 執行的時間

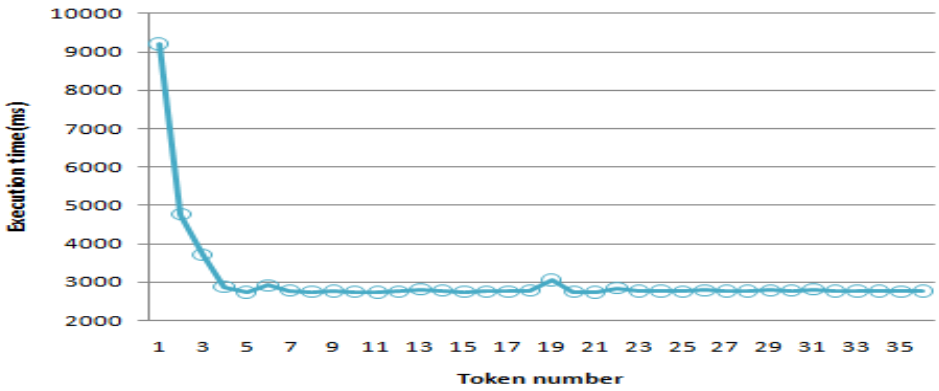


圖 140 使用 11 個 pipeline stage 執行的時間

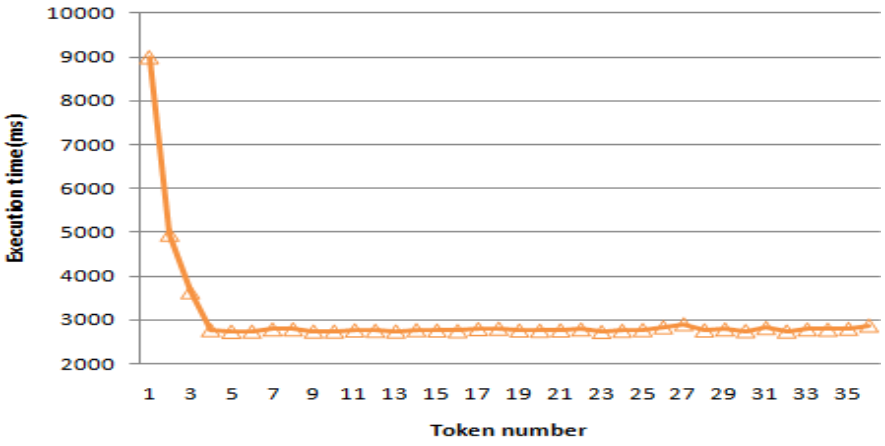


圖 141 使用 12 個 pipeline stage 執行的時間

實驗分析及討論

實驗一及實驗二的實驗結果說明了我們所開發的系統是可運作的，而且由於自動及半自動的產生 test data，自動的產生 test cases 及 test drivers 及多回合測試的機制，且測試皆以視覺化介面表現測試結果，能方便並減輕使用者的負擔。實驗三說明了在程式執行時可動態監控 CPU 使用率，能使得測試工程師容易得知軟體在嵌入式板上執行的效能，更容易去改進軟體效能，最後實驗四及五為測試 TBB

平行化的效率，讓測試工程師得知一個建議的 token 數，使得測試工程師不需花額外時間測試 TBB pipeline 平行化的效率。

C. 結論

本研究建構了一個支援嵌入式軟體開發的 cross-testing 的自動化測試環境系統，此系統能根據剖析原始碼後的資訊，能自動 instrument source code、自動產生 test case、test driver 及支援自動產生 primitive types，structure types and object types 的 test data，，可回合跨平台交叉測試，和，並以圖形化方式顯示測試結果。同時降低測試工程師的負擔，並提高的效率，嵌入式軟體測試過程中，這個系統開發的自動測試功能，包括單元測試，覆蓋測試，多核心平行參數的偵測。此外，ATEMES 可以直接測試執行在多核心嵌入式平台的嵌入式軟體，嵌入式軟體採用英特爾 TBB 庫，例如 parallel parameters such as pipeline tokens 測試。此工具也實際上使用在 ARM11 多核平台進行測試實驗，在實驗一及實驗二中，我們針對 C/C++資料結構範例程式進行測試，測試結果呈現我們所發展的系統是可運作的；實驗三為監控平行測試在嵌入式平台上執行的 CPU 使用率，使得測試工程師能夠清楚了解軟體執行狀況，實驗四及實驗五為 TBB pipeline 平行效率自動化測試，可使測試工程師不需花額外的時間手動輸入 test data 來測試 TBB pipeline 效率，利用我們所開發出來的系統，來減少測試的負擔並增加測試的效率。

三、 計畫成果(期刊、論文發表)

1. Chang, C.H.; Lu, C.W.; Chu, C.W.; Yang, C.T.; Hsiung, P.A.; Hsueh, N.L.; and Koong, C.S, "XML-based Reusable Component Repository for Embedded Software," Proceedings of the 35th IEEE Computer Software and Applications Conference (COMPSAC2011), 17-22 (2011.07). (EI)
2. Yu, K.Y.; Chen, T.C.; Lee, S.Y.; Chang, C.H.; and Chu, C.W.,"A SysML-based Requirement Environment for Embedded System Software Engineering," Proceedings of the 2011 Join Conference on Object-Oriented Technology Applications (OOTA) and Software Engineering (TCSE), pp. 437-441, 8-9 (2011.07)
3. P.-Y. Tsai, H.-Y. Yeh, T.-S. Hsu, S.-Y. Pan, L.-W. Sung, Bipin Kumar and P.-A. Hsiung, “A Multicore System Design for Basketball Detection and Tracking in Sports Competition Video Streaming,” In Proceedings of the Workshop on Consumer Electronics (WCE), Taiwan, 2011.
4. C.-Y. Shih, M.-C. Li, C.-S. Lin, P.-A. Hsiung, C.-H. Chang, W.C. Chu, N.-L. Hsueh,

C.-S. Shih, C.-T. Yang, C.-S. Koong, "Adaptive Performance Monitoring for Embedded Multicore Systems," In Proceedings of the International Conference on Parallel Processing Workshops on Embedded Multicore Systems (ICPPW-EMS), pp. 222–228, September 2011.

5. C.-S. Lin, C.-H. Lu, S-W. Lin, Y.-R. Chen, and P.-A. Hsiung, “VERTAF/MultiCore: A SysML-Based Application Framework for Multi-Core Embedded Software Development,” Journal of Computer Science and Technology (JCST), Vol. 26, No. 3, pp. 448-461, March/April 2011 (SCI)
6. Chang, C.H.; Lu, C.W.; Kao, K.F.; Chu, C.W.; Yang, C.T.; Hsueh, N.L.; Hsiung, P.A.; and Koong, C.S., "A SysML-based Requirement Supporting Tool for Embedded Software," Proceedings of the 5th International Conference on Secure Software Integration and Reliability Improvement (SSIRI 2011), 27-29 (2011.06). (EI)
7. 余冠穎, 陳廷肇, 張志宏, “以SysML為基礎之嵌入式系統軟體工程需求開發環境,” 2011年電子工程技術研討會 (ETS2011), 10 (2011.06).
8. C.W. Chu, “The application of Empirical Model-based Object-oriented Requirement Engineering(MORE) and its integration with OOA/OOD/OOP to real software projects”, Engineering Science & Technology Bulletin, NSC , Vol.104 , pp.28, FEB 2010 ,Vol. 110 ,pp.49, FEB 2011
9. Tung, C.C.; Yu, Y.Y.; Chang, C.H.; Chu, C.W.; and Lu, C.W., "XML-based Embedded Software Reusable Request Library System," Proceedings of the 2010 Join Conference on Object-Oriented Technology Applications (OOTA) and Software Engineering (TCSE) , pp. 134-139, 22-23 (2010.07).
10. Chao-Chin Wu*, Chao-Tung Yang, Kuan-Chou Li, and Po-Hsun Chiu, “Designing parallel loop self-scheduling schemes using the hybrid MPI and OpenMP programming model for multi-core grid systems”, Journal of Supercomputing, Springer Netherlands, 2010.
11. Chao-Tung Yang*, Chao-Chin Wu, and Jen-Hsiang Chang, “Performance-based Parallel Loop Self-Scheduling Using Hybrid OpenMP and MPI Programming on Multicore SMP Clusters”, Concurrency and Computation: Practice and Experience, Manuscript #CPE-09-0168R3, May 2010. (ISSN: 1532-0626, SCI JCR IF=1.791, EI).
12. Chao-Chin Wu*, Lien-Fu Lai, Chao-Tung Yang, and Po-Hsun Chiu, “Using Hybrid MPI and OpenMP Programming to Optimize Communications in Parallel Loop Self-Scheduling Schemes for Multicore PC Clusters”, Journal of Supercomputing, Springer Netherlands, 2010. (ISSN: 1573-0484, SCI JCR IF=0.615, EI).
13. Y.-H. Lin, S.-W. Lin, C.-S. Lin, C.-H. Lu, S.-Y. Tong, B.-H. Wang, C.-C. Ho, Y.-L.

Chang and P.-A. Hsiung, "Synthesis and Code Generation of Multi-Core Embedded Software," The 2010 Join Conference on Object-Oriented Technology Applications (OOTA) and Software Engineering (TCSE), 2010 (in Chinese).

14. Chao, T.Y.; Lin, B.Y.; Chang, C.H.; Chu, C.W.; and Chen, C.C., "XML-based Embedded Software Reusable Component Repository," Proceedings of the 2010 Join Conference on Object-Oriented Technology Applications (OOTA) and Software Engineering (TCSE), pp. 353-358, 22-23 (2010.07).
15. You, C.R.; Chao, T.Y.; Lin, B.Y.; Wu, C.W.; Chang, C.H.; and Chu, C.W., "A Supporting Tool for System Software in Requirements Phase," Proceedings of the 2010 Join Conference on Object-Oriented Technology Applications (OOTA) and Software Engineering (TCSE), pp. 377-382, 22-23 (2010.07)
16. Tung, H.Y.; Chang, C.H.; Chu, C.W.; Yang, H.J.; and Lu, C.W., "From Applications, to Models and to Embedded System Code: A Modeling Approach in Action," Proceedings of the 10th International Conference on Quality Software, pp. 488-494, 14-15 (2010.07). (EI)
17. You, C.R.; Xhao, T.Y.; Lin, B.Y.; Chang, C.H.; and Chu, C.W., "需求階段與設計階段整合追蹤性之探討," Proceedings of the 2010 the E-Learning and Information Technology Symposium, 31 (2010.03)
18. Chang, C.H.; Lu, C.W.; Chu, C.W.; Shih, C.H.; Hsiung, P.A.; Yang, C.T.; Hsueh, N.L.; and Koong, C.S.;" SysML-based Requirement Modeling Environment for Multicore Embedded System," Proceedings of the 25th Annual ACM Symposium on Applied Computing (SAC'10), Volume III, pp. 2224-2228, 22-26 (2010.03). (EI)
19. Chang, C.H.; Lu, C.W.; Chu, C.W.; Chen, W.C.; and Chen, T.Y., "Model-based Object-oriented Requirement Engineering for Supporting the Integration of Software Documents," Submitted to Journal of Software Engineering Studies.
20. P.A. Hsiung, S.W. Lin, Y.R. Chen, C.H. Huang, C.H. Shih and C.W. Chu, "Modeling and verification of real-time embedded systems with urgency", The Journal of Systems and Software, Vol. 82, pp. 1627-1641, October 2009 (EI)
21. C.S. Lin, P.A. Hsiung, S.W. Lin, Y.R. Chen, C.H. Lu, S.Y. Tong, W.T. Su, C.H. Shih, N.L. Hsueh, C.H. Chang, and C.S.Koong, "VERTAF/Multi-Core: A SysML-based Application Framework for Multi-Core Embedded Software Development," Journal of the Chinese Institute of Engineers, Vol. 32, No. 7, pp. 985-991
22. Tung, H.Y.; Chang, C.H.; Chu, C.W.; Yang, H.J.; and Lu, C.W., "From Applications, to Models and to Embedded System Code: A Modeling Approach in Action," to appear in the Proceedings of the 10th International Conference on Quality Software, 14-15

(2010.07)(EI).

23. You, C.R.; Xhao, T.Y.; Lin, B.Y.; Chang, C.H.; and Chu, C.W., "需求階段與設計階段整合追蹤性之探討," Proceedings of the 2010 the E-Learning and Information Technology Symposium, 31 (2010.03).
24. Chang, C.H.; Lu, C.W.; Chu, C.W.; Shih, C.H.; Hsiung, P.A.; Yang, C.T.; Hsueh, N.L.; and Koong, C.S.," SysML-based Requirement Modeling Environment for Multicore Embedded System," Proceedings of the 25th Annual ACM Symposium on Applied Computing (SAC'10), Volume III, 22-26 (2010.03), pp. 2224-2228. (EI)
25. C.-S. Lin, S.-W. Lin, C.-H. Lu, Y.-R. Chen, and P.-A. Hsiung, "Model-Driven Development of Multi-Core Embedded Software," Modern Software Engineering Concepts and Practices: Advanced Approaches, Editors: Ali H. Dogru and Veli Bicer, IGI Global, USA, 2010.
26. Tung, H.Y.; Chang, C.H.; and Chu, C.W., "Using UML to Implement Application on Different Embedded System," Proceedings of the 20th Workshop on Object-Oriented Technology and Applications, 20 (2009.11), pp. 1-6.
27. Chang, C.H.; Liao, W.B.; Chu, C.W.; Xu, C.Q.; and Tong, C.Q., "XML-based Embedded Software Reusable Component Library System," Proceedings of 20th Workshop on Object-Oriented Technology and Applications, 20 (2009.11), pp. 115-121.
28. Wei, J.Q.; Chang, C.H.; Huang, Y.R.; Chu, C.W., "再使用程式庫-以自動化倉管系統為例," Proceedings of the 20th Workshop on Object-Oriented Technology and Applications, 20 (2009.11), pp. 122-129.
29. Chang, C.H.; Lu, C.W.; Chu, C.W.; Chen, Y.W.; and Ger, R.E., "A SysML-based System Requirement Modeling Process," Proceedings of the 3rd Information Education and Technological Application Conference (IETAC2009), 6 (2009.11).
30. N.L. Hsueh, P.H. Chu, P.A. Hsiung, M.J. Chuang, C.W. Chu, C.H. Chang, C.S. Koong, and C.H. Shih, "Supporting Design Enhancement by Pattern-based Transformation," to appear in the Proceedings of the 34th IEEE Computer Software and Applications Conference (COMPSAC2010), July 19-23, 2010 (EI).
31. C.H. Chang, Y.W. Chen, J.J. Chen, William C. Chu, C.W. Lu and T.Y. Chao , "SysML-based Modeling Environment for Multicore Embedded System", The 20th Workshop on Object-Oriented Technology and Applications (OOTA 2009) , Taichung, Taiwan, pp.76-84, November 20, 2009
32. Chao-Tung Yang* and Kuan-Chou Lai, "A Directive-based MPI Code Generator on Linux PC Clusters," Journal of Supercomputing, Springer Netherlands, Volume 50, Number 2, pp. 177-207, Nov., 2009. (ISSN: 1573-0484, SCI JCR IF=0.615, EI)

33. M.U. Zhuang, N.L. Hsueh, P.H. Chu ,P.A. Hsiung, William C. Chu , S.C Hwang, C.H. Chang, C.H. Shih , C.S Ko , “Using MDA Approach in Design Pattern Specification and Transformation” The 20th Workshop on Object-Oriented Technology and Applications (OOTA 2009) , Taichung, Taiwan, pp.345-353, November 20, 2009
34. N.L. Hsueh, P.H. Chu, H.H. Shen and C.H. Chang, "A Semi-Automatic Approach for Test Case Traceability in a Test-Driven Development," Proceedings of the Workshop on Accountability and Traceability in Global Software Engineering (ATGSE 2008), pp. 21-22, December 1-5, 2008, Beijing.
35. L.C. Wen, N.L. Hsueh and P.H. Chu, "An Approach for Evaluating the Effectiveness of Design Patterns in Design Evolution," Object-Oriented Technology and Applications and Software Engineering, July 22-23, National Central University, Jhongli, 2010（審稿中）
36. P. A. Hsiung, C. S. Lin. “VERTAF/Multi-Core: A SysML-based Application Framework for Multi-Core Embedded Software Development,” The 9th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP’09).
37. P.A. Hsiung, S.W. Lin, Y.R. Chen, N.L. Hsueh, C.H. Chang, C.H. Shih, C.S. Koong, C.S. Lin, C.H. Lu, S.Y. Tong, W.T. Su, and W.C. Chu. “Model-driven development of multi-core embedded software,” Workshop on Multicore Software Engineering, 2009. IWMSE '09, PP. 9 – 16, July 12, 2009.
38. Chao-Tung Yang and Chih-Lin Huang, “Hybrid CUDA, OpenMP, and MPI Parallel Programming on Multicore GPU Clusters”, Conference on Computational Physics 2009 (CCP09), Kaohsiung, Taiwan, Dec. 15-19, 2009.
39. Chao-Tung Yang*, Chien-Hsiang Tseng, Keng-Yi Chou, and Shyh-Chang Tsaur, “A Virtualized HPC Cluster Computing Environment on Xen with Web-based User Interface”, The second International Conference on High Performance Computing and Applications, HPCA 2009, Lecture Notes in Computer Science, vol., 5938, pp. 503–508, Springer, Shanghai, China, August 10-12, 2009.
40. [3] Chao-Tung Yang*, Jen-Hsiang Chang, and Chao-Chin Wu, “Performance-based Parallel Loop Self-Scheduling on Heterogeneous Multicore PC Clusters”, The second International Conference on High Performance Computing and Applications, HPCA 2009, Lecture Notes in Computer Science, vol., 5938, pp. 509–514, Springer, Shanghai, China, August 10-12, 2009.

Competition:

1. P. A. Hsiung. Digital Video Recording on Embedded Multicore Systems. 九十七學年度全國大學校院嵌入式系統設計競賽. 教育部顧問室、SOC 總聯盟

2. P. A. Hsiung. Automatic Embedded System Code Generation for an Entrance Guard System with Mobile and Ubiquitous Control. 亞洲區 DSP 暨嵌入式系統應用競賽-台灣區分區競賽. 美商德州儀器股份有限公司

四、 國科會補助計畫衍生研發成果推廣資料表

國科會補助計畫	計畫名稱：多核心嵌入式軟體之模型驅動整合開發環境-VMC--子計畫六:多核心嵌入式軟體設計工具系統之平行程式優化支援實作-VMC_PPO(1/2) 計畫主持人：楊朝棟 計畫編號：NSC 98-2220-E-029-004　學門領域：資訊
技術/創作名稱	多核心嵌入式軟體設計工具系統之平行程式優化支援實作
發明人/創作人	楊朝棟
技術說明	中文： 本子計畫主要設計與實作多核心嵌入式軟體之平行程式優化支援。今日，處理器研發業者已不再依賴提高時脈速度方法來增進效能，因絕對的效能、增加的耗電量、以及攀高的成本使邊際效益呈現逐漸遞減的趨勢，目前，多數產業都已體認多重核心(Multi-core)是未來的發展方向，其主要設計挑戰已逐漸被成功克服，實際的建置行動更已經展開。多處理技術存在著三大挑戰，程式分割、程式平行和最佳化。目前最需要的是一個編譯模型，使其得以開發平行應用程式，將這些應用程式映射到平行硬體並進行最佳化，以及彙集資料以作出最佳化的決策。有效的平行迴圈切割與排程可以顯著地減少程式在多核心處理器系統環境中的整體回應時間，特別是針對具有許多迴圈的應用程式與針對新興的多核心嵌入式系統環境，其優化處理也更為困難。本子計畫第一年(2009/8~2010/7)，將使用編譯器技術如資料相依性分析與軟體管線化技術，完成指令階層平行化與平行迴圈切割與迴圈排程優化支援。本子計畫第二年(2010/8~2011/7)，將擴充 VMC 平行程式優化介面支援至更多的架構平台與應用，如支援多核心程式庫例如 OpenMP 及 Intel Threading Building Block (TBB)及強化與其他子計畫的介面。同時將建置一套在多核心嵌入式系統上平行程式優化支援模式適用於 OpenMP 及 Intel Threading Building Block 應用程式介面。

	英文： The main goal of this sub-project is to design and implement the parallel program optimization supporting for VERTAF/Multi-Core (VMC) embedded software. Today, the processor researcher is not dependent on the method of enhanced clock speed to improve performance. Because of absolute performance, the increase of power consumption, as well as the rising cost of marginal benefit to make a gradually descending trend. At present, most industries have been considered multi-core is the future of development. The main design challenge has succeeded gradually overcomes, the actual operations have already started to build. The multiprocessing and multi-core technology has three major challenges, program partition, and program parallelization and program optimization. The most important need is a compiler model, which enables to develop parallel applications. Using these applications mapped to the parallel hardware and run the action of optimization, as well as to collect information in order to make the best decision. Effective parallel loop splitting and scheduling can obviously reduce the program response time in the multiple processor environment, especially that optimize process is also more difficult when aims at a many circle application programs and emerging multi-core embedded system environment. First year, we will use compiler technology, such as the data dependency analysis and software pipelining that implement parallel instruction, parallel loop partitioning, and loop scheduling and optimization. Second year, we will extend the interface of VMC parallel program and support more architecture and applications, such as multi-core library like OpenMP and Intel Threading Building Block(TBB), and strengthen the interface of other sub-projects. At the same time, we will build a set of multi-core embedded systems which support parallel program optimization model for OpenMP and Intel Threading Building Block API.
可利用之產業 及 可開發之產品	本子計畫所開發，結合了平行技術與程式碼優化演算法，提供程式碼優化的基本功能和學習模式。本計畫建置一多核心嵌入式軟體之高效能平行編譯器 (PPO)，載入既有之程式碼，透過逐行逐段分析，輔以高效能平行演算法，基於 OpenMP 及 Intel Threading Building Block 多核心函式庫，產出平行程式碼，使其達到多核心同步執行，加速其運算速度，並達到最大之產出。有鑒於以往程式優化系統的全面性，我們採取了區塊優化，也就是經過學習模式就每個迴圈區塊做一次優化並記錄下來，以迴圈執行效率做為比較，去判定運算迴圈是否需要程式優化，因此本系統利用 Intel TBB 與 OpenMP 以及 ROSE 來達到程式碼平行與優化的效果。
技術特點	本系統的主要功能可區分為四個部份，分別敘述如下： ● 程式剖析子系統(Source Program Parsing Subsystem, SPPS) SPPS 接受使用者輸入之原始程式碼，逐行進行剖析，主要鎖定多數可平行運行之迴圈，以平行程式撰寫時常用的關鍵字與平行化之架構為基礎，針對輸入程式之上下文，以及各種區塊符號，判斷出各個程式區塊，最後交付後端之平行優化子系統。 ● 平行子系統(Parallelism Subsystem, PS) PS 接受前端程式剖析子系統之結果，針對剖析子系統所

	判斷出各個可平行化之標記部份，予以新增符合 TBB 或 OpenMP 之函式庫導引器，使其達到平行優化之目標。 ● 優化子系統(Optimization Subsystem, OS) OS 提供將平行子系統產生之程式碼進行最佳化，並將之輸出。 ● 紀錄子系統(Logging Subsystem, LS) LS 紀錄程式編譯優化過程中，所標記之程式碼行號以及所加入之程式導引器，並將之輸出以做檢驗或查詢用途。
推廣及運用的價值	本計畫針對 PPO 擬定了各種標準，以因應軟體開發上可能遭遇到的限制，各標準條例如下： ● 使用性：所選軟體是否易於操作開發 ● 可攜性：各種作業系統之移植可攜 ● 擴充性：系統後續擴充是否容易 ● 支援性：對於周邊設備是否具備足夠支援度 ● 維護性：系統是否易於維護與測試 隨著處理器從單核心進展至雙核心、四核心，甚至部分應用於伺服器的八核心之計算平臺，如何有效利用多核心的特性，並使應用程式於多執行緒的架構下加速運行，就成了開發人員極需研究的課題。關於嵌入式計算領域，也從單核心成長至多核心，為使原本效能備受限制之嵌入式裝置，透過多核心同步運行之機制，盡可能提高其運算效能，本計畫將建置一高效能平行編譯器，載入既有之程式碼，透過逐行逐段分析，輔以高效能平行演算法，基於 OpenMP 及 Intel Threading Building Block 多核心函式庫，產出平行程式碼，使其達到多核心同步執行，加速其運算速度，並達到最大之產出。

國科會補助計畫	計畫名稱：多核心嵌入式軟體之合成與程式碼生成 計畫主持人：熊博安 教授 計畫編號：NSC99－2220－E－194－013 領域：資訊、自由軟體嵌入式軟體		
研發成果名稱	（中文）多核心嵌入式系統之自動程式碼生成技術		
	（英文）synthesis and code generation technology of multi-core embedded software		
成果歸屬機構	中正大學	**發明人 (創作人)**	熊博安

技術說明	（中文） 設計者僅需要輸入工業標準的 SysML 模型，其中遵循我們 VMC 框架所訂的平行化之 stereotypes (標籤)，即可由 VMC 框架自動生成適合於多核心嵌入式系統執行的平行程式碼。目前，支援 C++ 程式語言、quantum platform (QP)狀態機程式庫、Intel Threading Building Blocks (TBB)多核心程式庫於 Intel 的多核心 CPU 以及 ARM 11 MPCore (4 核心)微處理機上執行。
	（英文） Designers have to only specify embedded software requirements via industry-standard SysML models, and use the VMC stereotypes to specify parallelism, then VMC can automatically generate parallel code. Currently, VMC supports the code generation of C++ code, quantum platform (QP) state machine library, Intel Threading Building Blocks (TBB) multicore library, that can run on Intel multicore CPU and ARM 11 MPCore CPU.
產業別	嵌入式系統設計之公司以及嵌入式軟體開發之公司。
技術/產品應用範圍	自動化多核心程式碼生成。
技術移轉可行性及預期效益	縮短多核心嵌入式軟體的開發時程。

國科會補助計畫	計畫名稱：多核心嵌入式軟體之模型驅動整合開發環境-VMC－子計畫七:多核心嵌入式軟體設計之測試支援系統(2/2) 計畫主持人：孔崇旭 計畫編號：NSC 99-2220-E-142 -001　　領域：軟體測試		
研發成果名稱	（中文）多核心嵌入式軟體自動測試工具		
	（英文）an automatic testing environment for multi-core embedded software (ATEMES)		
成果歸屬機構	國立台中教育大學	**發明人** **(創作人)**	孔崇旭

技術說明	（中文） 軟體測試在嵌入式軟體開發的過程不僅是複雜的，但也是品質控制的重要步聚；然而多核心嵌入式軟體測試則面臨著更多的挑戰。主要問題包括：（1）如何減少大量的人力及重複繁瑣的工作;（2）嵌入式系統平台資源的限制，如時間及記憶體容量的的限制;（3）如何讓嵌入式軟體能並行化，以發揮多核心 CPU 的計算能力;（4）如何分析，來提高嵌入式軟體的覆蓋測試;（5）如何做同步資料分析，例如分析在中斷導向的多核心嵌入式系統中的 race condition 問題 ;（ 6）高層次的可靠性測試，以確保客戶滿意度。為了解決這些問題，本研究開發一個多核嵌入式軟體（ATEMES）的自動測試環境。運用自動化機制的基礎上，系統可以解析原始程式碼，插入基於測試所需的的程式片段，自動產生測試案例和測試驅動程式，支援自動產生 primitive, structure and object types 測試輸入資料，可回合跨平台交叉測試，和，並以圖形化方式顯示測試結果。同時降低測試工程師的負擔，並提高的效率，嵌入式軟體測試過程中，這個系統開發的自動測試功能，包括單元測試，覆蓋測試，多核心平行參數的偵測。此外，ATEMES 可以直接測試執行在多核心嵌入式平台的嵌入式軟體，嵌入式軟體採用英特爾 TBB 庫，例如 parallel parameters such as pipeline tokens 測試。此工具也實際上使用在 ARM11 多核平台進行測試實驗，結果也加強了說明，我們建構的測試環境是有效的，並可以減少測試工程師的負擔，並能增強測試任務的效率。 （200-500 字） （英文） Software testing during the development process of embedded software is not only complex, but also the heart of quality control. Multi-core embedded software testing faces even more challenges. Major issues include: (1) how demanding efforts and repetitive tedious actions can be reduced; (2) how resource restraints of embedded system platform such as temporal and memory capacity can be tackled; (3) how embedded software parallelism degree can be controlled to empower multi-core CPU computing capacity; (4) how analysis is exercised to ensure sufficient coverage test of embedded software; (5) how to do data synchronization to address issues such as race conditions in the interrupt driven multi-core embedded system; (6) high level reliability testing to ensure customer satisfaction. To address these issues, this study develops an automatic testing environment for multi-core embedded software (ATEMES). Based on the automatic mechanism, the system can parse source code, instrument source code, generate testing programs for test case and test driver, support generating primitive, structure and object types of test input data, multi-round cross-testing, and visualize testing results. To both reduce test engineer’s burden and enhance his efficiency when embedded software testing is in process, this system developed

	automatic testing functions including unit testing, coverage testing, multi-core performance monitoring. Moreover, ATEMES can perform automatic multi-round cross-testing benchmark testing on multi-core embedded platform for parallel programs adopting Intel TBB library to recommend optimized parallel parameters such as pipeline tokens. Using ATEMES on the ARM11 multi-core platform to conduct testing experiments, the results show that our constructed testing environment is effective, and can reduce burdens of test engineer, and can enhance efficiency of testing task.
產業別	軟體業 嵌入式軟體開發
技術/產品應用範圍	嵌入式軟體系統開發 軟體自動化測試工具 平行程式之平行度效能測試
技術移轉可行性及預期效益	可提昇嵌入式軟體的品質 減少軟體測試的人力 加速嵌入式軟體的開發

出席國際學術會議心得報告

出國人員姓名 服務機關及職稱	資訊工程學系 朱正忠教授
會議時間地點	July 18-21, 2011 Munich, Germany
會議名稱	The 35th Annual IEEE Computer Software and Applications Conference (COMPSAC 2011)

一、參加會議經過

此會議 Computer Software and Applications Conference 是在電腦軟體及應用科學具領導地位的國際會議，討論的領域以軟體技術與應用為主也包括程式開發、分析、改進、測試與需求等。今年會議舉辦地點是德國慕尼黑，本次主要擔任會議之 Steering Committee, Program Committee, Session Chair，與會的學者都是世界上頂尖的專家，主要與會學者如 Bruce McMillin, Klaus Beetz, (General Chair) Sheikh Iqbal Ahamed (Program Chairs)等知名學者。本次研討會的核心議題主要以軟體工程為主，但也包含熱門的嵌入式系統、雲端計算等熱門議題，在會議的第一天的 workshop 與第三天下午的 Panel 中，也探討未來軟體技術的發展與軟體重用的觀念，討論熱烈，從中獲取不少心得，整體來說此會議涵蓋的研究範圍已相當完整。

二、與會心得

這次除了與來自各國專家一起討論軟體開發的研究之外，並發表一篇論文 Requirements Recovery by Matching Domain Ontology and Program Ontology 以及主持一場關於即時嵌入式系統的論文發表會，與眾多學者一起討論、交換心得是這次會議最大的收穫，未來有機會也許可以與國外研究團隊合作；在會場有許多專家討論到個人最近在做的關於多核心嵌入式系統方面的研究，各國學者皆對於我們的研究非常有興趣，與這些專家學者做了一些深入的討論，這學者們也認同我們在此領域的研究是有其必要性，並且這也是目前正熱門的議題之一，在交流的過程中，也瞭解這些專家學者所做的相關研究，也對我們的研究上有莫大的幫助。

From: cs_compsac@iastate.edu [mailto:cs_compsac@iastate.edu]
Sent: 24 March 2011 13:03
To: fengchen@dmu.ac.uk
Subject: COMPSAC 2011 (TR) : Notification of Decision On Your Paper 10397

Dear Feng Chen, Hongji Yang, Hong Zhou and W. C. Chu,

We are happy to inform you that your paper entitled "Requirements Recovery by Matching Domain Ontology and Program Ontology", submitted to the IEEE Computer Software and Applications Conference (COMPSAC2011), has been accepted for inclusion as a short paper (up to 6 pages) in the proceedings. Below, you will find the reviews. Please carefully consider the reviewers' comments when preparing the final version of your paper. In your final version, please try to concentrate on the key results of your approach.

In order for your paper to be included in the proceedings, by April 30, 2011 at the latest you must:

1. Upload the camera-ready copy of your paper: You must access the http://rs.cs.iastate.edu/COMPSAC2011/SubmitPaper.php upload interface and enter your paper id: 10397 and password: 5e6b9c. Please visit the conference website http://www.compsac.org/ for instructions on registration and preparation of the camera-ready copy after April 10, 2011.

2. Register your paper: Please note that for each paper, even with all student authors, at least one author must complete the advance registration by paying the full registration fee (lower than the on-site registration fee) before the paper can be included in the proceedings. We anticipate offering student travel matching grants for students authors. Please check the conference website for status updates and eligibility.

3. Inform us (cs_compsac@iastate.edu) that you have accepted the Program Committee's decision and will present in person the paper at COMPSAC 2011 Conference as soon as possible or no later than April 8. We would like to bring to your attention that IEEE recently established a policy to remove conference papers from the permanent archive (i.e. IEEE Xplore) for "no show" cases. Thus, at least one author must attend the conference and present the paper in person in order to enable the inclusion of your paper in the IEEE permanent archive.

Please note that any delay may prevent the inclusion of your paper in the proceedings.

We look forward to your participation in COMPSAC 2011 in Munich, Germany. The compsac.org <http://compsac.org/> website also provides all necessary

information for conference registration, visa letter requests, travel, and hotel accommodation.

Best regards,

Sheikh Iqbal Ahamed

Wolfgang Minker

Zhi Jin

COMPSAC 2011 Program Committee Chairs
==========

Reviewer: 1

Originality : Weak Accept
Quality : Weak Accept
Relevance : Neutral
Presentation : Weak Accept
Recommendation : Weak Accept

Summary: Ontology is really a foundamental element for reverse engineering. Applying ontology to reverse a program for the maintenance is a good approach.

Details: Ontology is really a foundamental element for reverse engineering. The authors provide two methods to help the reverse work based on ontology and give an example to indicate the success of their methods. The paper writing is acceptable. However, the example does not show how the steps in each method are applied. Therefore, it validity is not good enough.

Candidate for the best paper award? : No
Have you checked whether there may be a plagiarism with this paper? (give specific information on the review form) : Yes
Have you noticed that there may be a conflict of interest? (explain using the review form) : Yes

==========

Reviewer: 2

Originality : Weak Reject

Quality : Weak Accept
Relevance : Neutral
Presentation : Weak Accept
Recommendation : Weak Reject

Summary: The paper proposes an ontology-based re-engineering approach to recover requirements from existing systems.

Details: The case study provided in the paper does not show the effectiveness of the proposed approach. One does not necessarily need to apply a re-engineering methodology to a POST to know that buy items, log in and refund purchased items will be use cases of the system. The language is good and that makes it easy to comprehend. There are few syntax errors with respect to omissions which the authors need to pay attention to and correct. The following examples are not intended to be the only issues.
Page 1, Paragraph 3
Line 14
It must be: point of sale terminal not sale terminal.

Page 2, Paragraph 3
Line 8
Paul Rayson et al. [21] propose instead of Paul Rayson et al. [21] proposes

Figures 2 and 3 are hard to read

Candidate for the best paper award? : No
Have you checked whether there may be a plagiarism with this paper? (give specific information on the review form) : Yes
Have you noticed that there may be a conflict of interest? (explain using the review form) : No

==========

Reviewer: 3

Originality : Weak Reject
Quality : Weak Reject
Relevance : Accept
Presentation : Weak Reject
Recommendation : Weak Reject

Summary: This paper proposes a method of recovering domain ontology and program ontology from a system and mapping between them.

Details: Favor
Recovering ontologies from legacy might be useful to understand existing system.
The authors try to demonstrate the proposed method with POS (Point Of Sale) terminal.

Against
Whole paper lacks rigorous discussions.
A case study of POS(T) system is rather shallow, and details are not presented. Therefore, it's hard to see the effectiveness of the proposed method against the methods of local company.

Candidate for the best paper award? : No
Have you checked whether there may be a plagiarism with this paper? (give specific information on the review form) : No
Have you noticed that there may be a conflict of interest? (explain using the review form) : No

Requirements Recovery by Matching Domain Ontology and Program Ontology

Feng Chen, Hong Zhou, Hongji Yang and Martin Ward

Software Technology Research Laboratory, De Montfort University, Leicester, UK

{fengchen, hongzhou, hyang} @dmu.ac.uk, martin@gkc.org.uk
William Cheng-Chung Chu

Department of Computer Science, Tunghai University, Taiwan

E-mail: cchu@thu.edu.tw

***Abstract*—Users and systems requirements are fundamental for software development and maintenance. However, for most of existing systems, you may only find design documents without requirement specification. This paper presents an ontology-based reengineering approach to recovering requirements from existing systems. The proposed approach consists of three main parts: ontology development, ontology mapping and derivation of the requirements. Domain ontology is used to model domain specific requirements and program ontology is used to present system structure and behaviour. The algorithm of ontology mapping is developed to match domain ontology and program ontology for requirement recovery. A case study of Point of Sale Terminal (POST) system is used to illustrate the approach. Conclusions are drawn and further research directions are advocated.**

Keywords: Ontology; Software Requirement Engineering; Software Reengineering

1. Introduction

Constant innovation of information technology and ever-changing market requirements relegate more and more existing software to legacy status. Most of existing legacy software should be maintained and the software requirements are fundamental for software maintenance. However, for most of these systems, you may only find design documents without requirement specification. If the software requirements can be recovered, it will greatly facilitate the maintenance of the legacy systems.

As an inherently knowledge intensive activity, software requirements recovery requires a great number of knowledge and knowledge-based approaches are always hired as solutions to automate such tedious processes [28]. Knowledge Representation was developed as a branch of Artificial Intelligence to enable computer systems to perform tasks that require human intelligence. Providing an effective high-level description of the world will be essential in knowledge based approach, ontology is selected as the underlying mechanism in this study, with

which computer system will be able to find implicit consequences of explicit knowledge.

In general, it is expected to reduce the development and maintenance costs and delays through a relative general domain-specific pattern or architecture [3]. Since ontology can provide a vocabulary of terms and relations to model such domains, it will therefore facilitate the construction of the domain-specific solutions by introducing ontology-based approach.

2. Related Work

Much work has been carried out on ontology based software engineering. Devedzic [6] proposes that ontologies are needed in all software systems. Bures et al. [2] propose how ontology can facilitate schema-based program synthesis. Zimmer and Rauschmayer [29] present a way of enhanced ontology-based software modelling. Furtado et al. [9] propose a universal user interface design. The conceptual level is to create the domain ontology. Wongthongtham et al. [20] propose a software engineering ontology for software engineering knowledge management in multi-site software development environment.

Several studies have been carried out to utilise ontology and other knowledge representation techniques to facilitate program comprehension and reverse engineering. Yang et al. [24] suggest that ontology has a great potential for legacy software understanding and re-engineering. Li et al. [16] introduce an innovative approach to recovering domain knowledge with enhanced reliability from source code. Zhang et al. [26] propose an approach to identifying security flaws and reasoning security concerns. Zhou et al. [28] present an Ontology-based PlaTform-specIfic software Migration Approach (OPTIMA). Jin and Cordy [11] utilise ontology-based approach to facilitating software analysis and reengineering tools integration via Ontology Adaptive Service-Sharing Integration System (OASIS). Ambrosio et al. [1] use ontologies to help the combination of application domain information and software reengineering knowledge, producing up-to-date documentation that evolves along time. ONTODM [10] is an ontology-based tool supporting specification of domain models in Multi-Agent Domain Engineering.

In many cases, a requirement specification will prove to be inconsistent or inaccurate, or even worse, not available. Requirement reengineering is needed and the requirement of existing systems needs to be recovered. El-Ramly et al. [7, 8] propose a method to recover requirements from systems-user interaction trace and build use case models. Paul Rayson et al. [21] propose the probabilistic natural language processing techniques to assist requirements recovery from legacy documents. Liu [17] proposes a semiotic approach, Analysing and Modelling the Behaviour of Legacy Systems (AMBOLS), to requirement engineering. Melikhova et al. [18] show a practical approach for reengineering requirements for reuse along with examples of how it was applied in a cable TV environment. Colom [5] and Yu [25] propose an i* framework, which is a goal-oriented approach. The i* framework addresses requirements engineering and business process reengineering.

There are many challenges in Requirements Engineering research area [19] and it is very natural to apply ontology in this area. Kaiya et al. [12] propose a software requirements analysis method based on domain ontology technique to establish a mapping between a software requirements specification and the domain ontology that represents semantic components. Lee et al. [15] present an Ontology-based Active Requirements Engineering (Onto-ActRE) framework, which integrates various RE modelling techniques

with complementary semantics in a unifying ontological engineering process. Kossmann et al. [13] propose an Ontology-driven Requirements Engineering Methodology (OntoREM) that aims to improve the quality of requirements while also reducing the time and cost needed to develop, maintain and re-use requirements. OntoREM was applied to the aircraft operability (AO) domain. Based on these studies, the proposed approach in this paper is trying to recover the requirement by combining domain ontology with program ontology. Domain ontology and program ontology have been investigated in [4, 27, 28] and this work focuses on how to prepare and utilise ontologies for requirement recovery purpose.

3. Mapping of Domain Ontology and Program Ontology

In Sowa's book [22], it is indicated that knowledge representation is the application of logic and ontology to the task of constructing computable models for some domain. Ontology defines the kinds of things that exist in the application domain [22]. Following this idea, in this research, ontology forms the vocabulary used by both software systems and problem domain. Figure 1 shows the process of ontology mapping.

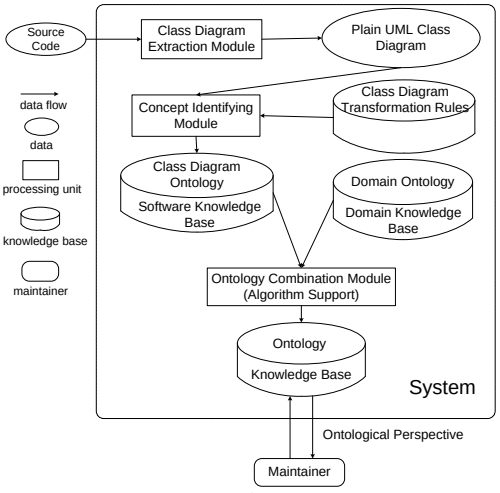


Figure 1. Process of ontology mapping

3.1. Domain Ontology

Domain ontology illustrates knowledge in specific problem domain, such as banking, shopping, and management, etc. The term domain ontology has the advantage of strongly emphasising a focus on domain concepts, not software entities. It consists of domain concepts and domain functions.

3.1.1. Domain Concept. The domain concepts are a set of concepts in the application domain that permit classification of different types of artefacts, which are related to the configuration of application domain. It is an extremely important part of the knowledge about any domain.

3.1.2. Domain Function. The domain functions are the functionality of the domain application and the relation among this functionality from a real world domain point of view. Domain functions are the general capabilities of an application domain, which represent all functions / operations related to domain activities.

3.1.3. Development of Domain Ontology. In building domain ontology, people should explore the rationality in expressing domain concepts and relations between the concepts. The development of the domain concept and functionality has the following steps: (1) determine the domain and scope of the ontology; (2) define ontology specification; (3) define the classes and their hierarchy; (4) define the properties; (5) define data properties; (6) create instances; (7) implement the ontology; (8) validate the domain ontology.

3.2. Program Ontology

Program ontology is summarised from the program comprehension point of view. It aims to describe the design of software, to represent software design patterns and related concepts. In this work, only Object-Oriented Programming will be investigated.

3.2.1. Development of Program Ontology. Program ontology comes from knowledge of code, database and application framework. Hence Program ontology describes programming elements and their relationships. Program ontology also includes artefacts at Schema level, such as conceptual database schemas. If there exists an application framework, it is normally domain specific, which can be linked to the domain ontology easily. Since Object-Oriented class has many features that make it easy to be transformed to the form of ontology, in this work, program ontology will be populated from a class diagram.

3.2.2. Class Diagram Transformation. A class generally includes three parts: class name, attributes and operations. Class itself will be transformed into concept in the ontology. The attributes of the class will be transformed into properties of that concept in ontology. For the generalisation and inheritance classes, the relationships SubClassOf and SuperClassOf will be preserved by the subclass and superclass concepts. Association will be transformed into ConnnectTo property, and it is a symmetric property. Dependency will be transformed into DependOn property and its inverse property Depend. Aggregation/composition will be transformed into HasA/PartOf property.

3.3. Ontology Mapping

Basically, ontology mapping is performed based on the understanding process. The mapping algorithm given below was proposed in [27]:

1. Load two ontologies Oc and Od, Oc is the class diagram ontology while Od is the domain ontology.
2. Scan the concepts names.
3. For each concept Cc in Oc, retrieve the initial list of suggestions. Find the best choice Cd from the suggestions in Od. Introduce the relation implement into these two ontologies: implement (Cc, Cd) and let the relation implement $\subseteq$ subsume, so $Cc \subseteq Cd$. As a result, the classes in class diagram are mapped to the concepts in Od.
4. Create a neighbourhood ontology On.
5. For each concept $Cc' \in Oc$, which does not have the initial list of suggestions, let R and R' be two binary relations, concept $Cc'' \in Oc$, and either $Cc' \subseteq \exists R..Cc''$ or $Cc'' \subseteq \exists R'.Cc'$. If Cc'' has best choice $Cd'' \in Od$, let $Cc'' \in On$ and $Cd'' \in On$.
6. Do 5 until $Cc'' \in \varnothing$.
7. Find out concept $Cd''' \in Od$, where $\forall Cd''(Cd'' \in Od$ and $Cd'' \in On)$ and $R \in Od \rightarrow Cd''' \subseteq \exists R.Cd''$. Put Cd''' into the initial list of suggestions for the concept Cc' in 5.
8. Do 3 until $Cd''' \in \varnothing$.
9. Generate application specific ontology Os.
10. Stop.

Based on this algorithm, the program ontology can be combined with domain ontology, and the application specific ontology is populated.

4. An Ontology Based Requirement Recovery Approach

Requirements artefacts include Use Case Model, Supplementary Specification, Glossary and System Sequence Diagram. The proposed approach is to recover requirements by rebuilding the use cases.

4.1. Use Case Diagram Recovery

The use case model is the model of system's functionality and environment. It may optionally include a UML use case diagram to show the names of use cases and actors, and their relationships. This gives a nice context diagram of a system and its environment. Use case model can be briefly (high level) or fully addressed. Generally, after many use cases have been identified and written in a brief format, a few (such as 10%) of the architecturally significant and high-value use cases should be written in detail.

The name of use case normally starts with a verb. It means that from ontology analysis point of view, to recover the use cases from the existing system, it is better to capture functions or operations from program

ontology first. These “actions” or “events” are included in some classes or concepts within program ontology. However no actors can be found in program ontology. When one concept in program ontology is matched to another one in domain ontology, a neighbourhood (all the filler concepts of binary relations) analysis can be performed from domain ontology to identify the actors in domain ontology. After all the functions in program ontology have been checked, the associated actions and processes will be identified and linked to the related actors, and hence use case diagram can be recovered.

4.2. Use Case Scenarios Recovery

The scenarios have been widely used for capturing requirements. However, only from domain and program ontology, it is impossible to obtain the use case scenarios. The associations between two concepts in program ontology can be specified by matching these two concepts to domain ontology and exploring the relationships between them in domain ontology. So what people can do for use case scenarios recovery is to list all the functions/operations in a use case as a template and to decide the sequence of these activities according to the domain knowledge.

5. Point of Sale Terminal (POST) – A Case Study

The example program used in this illustration was taken from an object-oriented design text book [14], which is a Point-Of-Sale Terminal (POST) system. The POST is a computerised system used to record sales and handle payments. The Java source code for POST system is given.

5.1. POST Domain Ontology

The domain ontology should represent meaningful (to the domain expert and software maintainer) concepts in POST system domain. Figure 2 depicts a small part of the domain ontology for POST system.

5.2. POST Program Ontology

The UML class diagram extracted from Java code can be read and analysed by UML parser. According to the class diagram to ontology transformation rules, the extracted POST system class diagram can be transformed into POST class diagram ontology. Figure 3 shows a part of this reverse engineered class diagram ontology for POST system. Through UML parser and Protege OWL-API [23], this generating process can be performed automatically.

5.3. Ontology Mapping

Following the combining algorithm proposed in Section 3.3, some analysis can be performed. Firstly, the concepts which have clear meanings (semantics) will be picked out from the program ontology. In this example, UML_POST is selected, and the concepts which directly relate (neighbouring) to UML_POST are UML_Sale, UML_Store, UML_ProductCatalog and UML_Payment. Then, back to the domain ontology, concept POST is chosen, with its neighbours Sale, Store, Cashier and Manager. So the pair (UML_Store, Store) and (UML_Sale, Sale) are matched. By doing neighbourhood analysis recursively, more concepts will be matched between two ontologies.

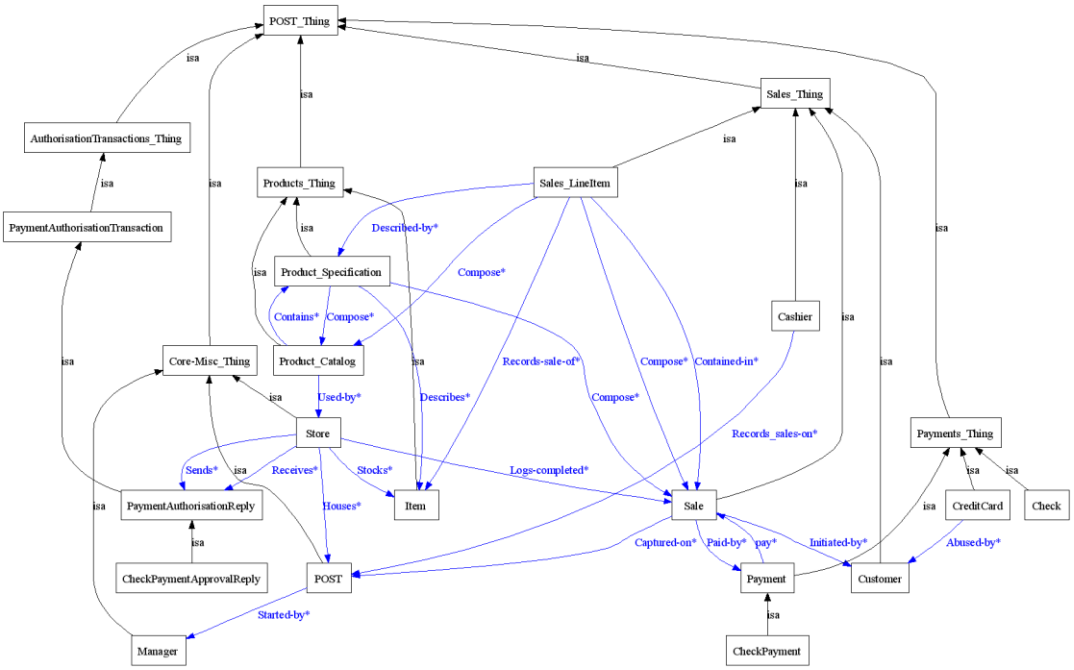


Figure 2. Domain ontology for POST system

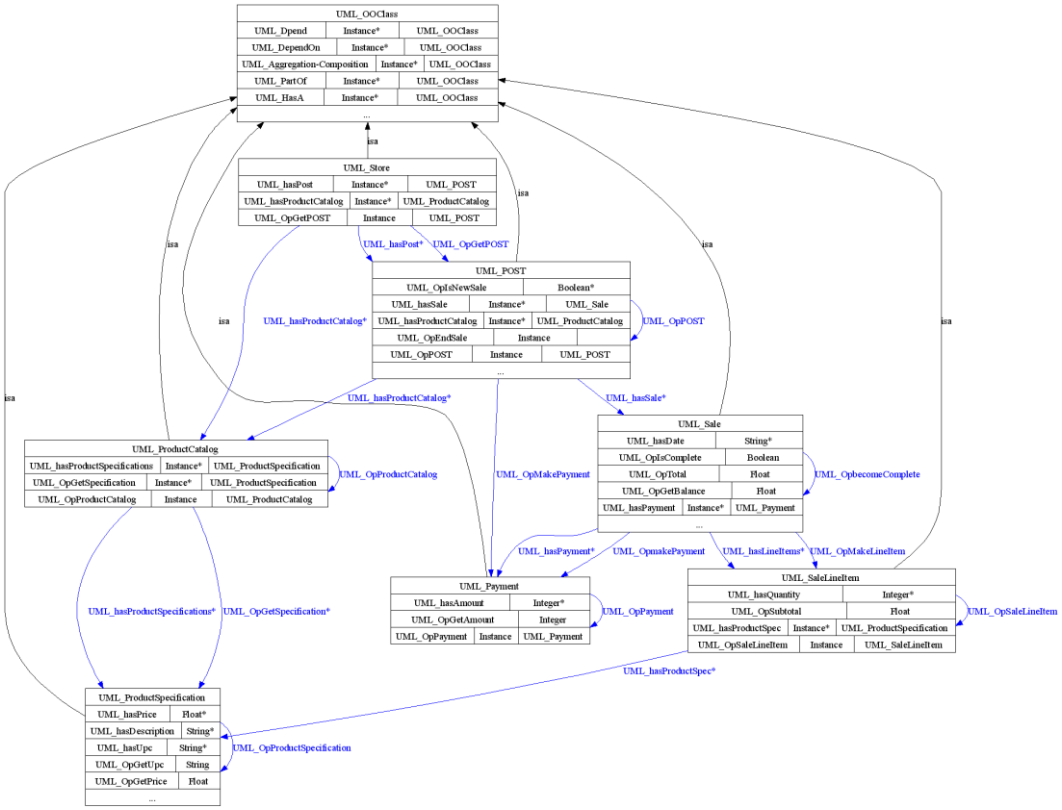


Figure 3. Program ontology for POST system

5.4. Requirement Recovery

5.4.1. Use Case Diagram Recovery. In POST case, the identification will start from concept of UML_Sale in program ontology. Since concept of UML_Sale in program ontology is matched to concept of SALE in domain ontology, a neighbourhood analysis can be performed from domain ontology to identify the actors in domain ontology. Here, Sale is linked to the Customer. There is also an association between Sale and POST, while POST is linked to the Cashier. A use case should be named starting with a verb in order to emphasise that it is a process. From customer viewpoint, "Sale" is "Buy" and "Buy items" is a recovered use case. The use case for Cashier can be obtained in the same way. Figure 4 shows a recovered use case diagram, in which "Log in" and "Refund purchased items" are added for illustration purpose.

5.4.2. Use Case Scenarios Recovery. In POST case, in order to capture requirement of "Buy items", use case scenarios should be described. By analysing domain ontology and exploring the relationships between Cashier and Sale via POST. Functions of enterItem(), endSale() and makePayment() can be listed and the sequence of these activities is obvious. It is not very hard to build the sequence diagram for this case.

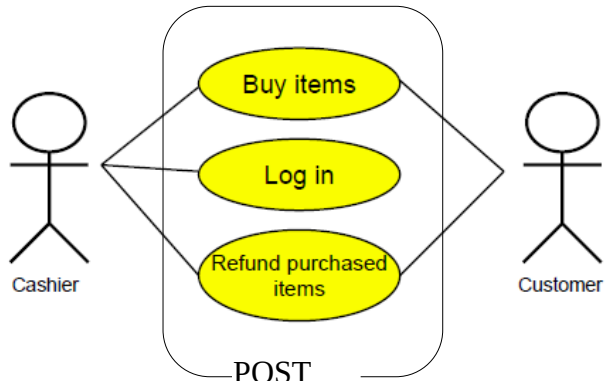


Figure 4. Recovered Use Case diagram

6. Conclusion and Future Work

In this paper, an ontology-based requirement recovery approach is proposed. The proposed approach is not a panacea but a promising starting point towards recovering high quality user and system requirement. The proposed approach focuses on functional aspect without discussing about non-functional issue. In the future, techniques of concept analysis will be introduced and compared. Furthermore, tools for supporting the proposed approach will be designed.

References

[1] A. Ambrosio, D. Santos, F. Lucena and J. Silva, "Software engineering documentation: an ontology-based approach," 10th Brazilian Symposium on Multimedia and the Web 2nd Latin American Web Congress, Oct. 2004.

[2] T. Bures, E. Denney, B. Fischer and E. Nistor, "The role of ontologies in schema-based program synthesis," Workshop on Ontologies as Software Engineering Artifacts, Vancouver, Canada, 2004.

[3] F. Chen, H. Guo, L. Dai and H. Yang, "An application framework for ontology-based data mining," *Journal of Dalian University of Technology*, vol. Suppl., 43(S1), China, pp. 143-145, Oct. 2003.
[4] F. Chen, Z. Zhang, J. Li and H. Yang, "Service identification via ontology mapping," *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '09)*. Seattle, WA 2009, pp. 486-491.
[5] G. G. Colom, "An i*-based reengineering framework for requirements engineering," Universitat Politècnica de Catalunya, 2008.
[6] V. Devedzic, "Understanding ontological engineering," *Communications of the ACM*, vol. 45(4), pp. 136-144, Apr. 2002.
[7] M. El-Ramly, E. Stroulia and P. Sorenson, "Mining system-user interaction traces for Use Case models," Proceedings of the 10th International Workshop on Program Comprehension (IWPC '02), 2002.
[8] M. El-Ramly, E. Stroulia and P. Sorenson, "Recovering software requirements from system-user interaction traces," *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE '02)*, 2002.
[9] E. Furtado, J. J. V. Furtado, W. B. Silva, et al., "An ontology based method for universal design of user interfaces," Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends, Lille, France, Sep. 2001.
[10] R. Girardi, C. G. d. Faria and L. Balby, "Ontology-based domain modeling of multi-agent systems," 3rd International Workshop on Agent-Oriented Methodologies at OOPSLA 2004, Vancouver, Canada, Oct. 2004.
[11] D. Jin and J. R. Cordy, "Ontology-based software analysis and reengineering tool integration: The OASIS Service-Sharing Methodology," 21st IEEE International Conference on Software Maintenance (ICSM'05), Budapest, Hungary, Sep. 2005.
[12] H. Kaiya and M. Saeki, "Ontology based requirements analysis: lightweight semantic processing approach," *Fifth International Conference on Quality Software (QSIC 2005)*. Sep. 2005, pp. 223-230.
[13] M. Kossmann and M. Odeh, "Ontology-driven requirements engineering – A case study of OntoREM in the aerospace context," *2010 INCOSE Conference*. Chicago, USA, 2010.
[14] C. Larman, *Applying UML and Patterns: an Introduction to Object-oriented Analysis and Design*: Prentice Hall PTR, 1997.
[15] S. W. Lee and R. A. Gandhi, "Ontology-based active requirements engineering framework," *12th Asia-Pacific Software Engineering Conference (APSEC '05)*, Dec. 2005, pp. 8-15.
[16] Y. Li, H. Yang and W. Chu, "A concept-oriented belief revision approach to domain knowledge recovery from source code," *Journal of Software Maintenance: Research and Practice*, vol. 13(1), pp. 31-52, Jan. 2001.
[17] K. Liu, "Requirements reengineering from legacy information systems using semiotic techniques," *Systems, Signs and Actions – The International Journal on Communication, Information Technology and Work*, vol. 1 (1), pp. 36–61, 2005.
[18] L. Melikhova, A. Elcock, A. A. Dovzhikov, G. Bulatov and D. Vavilov, "Reengineering for system requirements reuse: methodology and use-case," IEEE International Symposium on Consumer Electronics (ISCE 2007), Irving, TX, Jun. 2007.
[19] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," *Proceedings of the Conference on The Future of Software Engineering (ICSE '00)*, 2000, pp. 35-46.
[20] W. Pornpit, E. Chang and I. Sommerville, "Software engineering ontology for software engineering knowledge management in multi-site software development environment," 10th International Protege Conference 2007, Budapest, Hungary, Jul. 2007.
[21] P. Rayson, R. Garside and P. Sawyer, "Assisting requirements recovery from legacy documents," Technical Report CSEG/8/2000, Computing Department; Lancaster University 2000.
[22] J. F. Sowa, *Knowledge Representation*: Brooks/Cole, an Imprint of Thomson Learning, 2000.
[23] Stanford, "Protégé Programming Development Kit (PDK)," http://protege.stanford.edu/doc/dev.html.
[24] H. Yang, Z. Cui and P. O'Brien, "Extracting ontologies from legacy systems for understanding and re-engineering," 23rd Annual International Computer Software and Applications Conference (COMPSAC'99), Phoenix, AZ, Oct. 1999.
[25] E. Yu, "Towards modeling and reasoning support for early-phase requirements engineering," Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE '97) 1997.
[26] Y. Zhang, J. Rilling and V. Haarslev, "An ontology-based approach to software comprehension - reasoning about security concerns," 30th Annual International Computer Software and Applications Conference (COMPSAC'06), Chicago, USA, Sep. 2006.
[27] H. Zhou, F. Chen and H. Yang, "Developing application specific ontology for program comprehension by combining domain ontology with code ontology," *8th International Conference on Quality Software (QSIC'08)*. Oxford, UK, Aug. 2008, pp. 225-234.
[28] H. Zhou, J. Kang, F. Chen and H. Yang, "OPTIMA: an Ontology-based PlaTform-specIfic software Migration Approach," 7th International Conference on Quality Software (QSIC'07), Portland, Oregon, USA, Oct. 2007.
[29] C. Zimmer and A. Rauschmayer, "Tuna: ontology-based source code navigation and annotation," Workshop on Ontologies as Software Engineering Artifacts, Vancouver, Canada, 2004.

無研發成果推廣資料

99 年度專題研究計畫研究成果彙整表

計畫主持人：朱正忠	計畫編號：99-2220-E-029-001-
計畫名稱：多核心嵌入式軟體之模型驅動整合開發環境-VMC--總計畫(2/2)	

成果項目			量化			單位	備註（質化說明：如數個計畫共同成果、成果列為該期刊之封面故事...等）
			實際已達成數（被接受或已發表）	預期總達成數(含實際已達成數)	本計畫實際貢獻百分比		
國內	論文著作	期刊論文	0	0	100%	篇	
		研究報告/技術報告	0	0	100%		
		研討會論文	0	0	100%		
		專書	0	0	100%		
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（本國籍）	碩士生	0	0	100%	人次	
		博士生	0	0	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		
國外	論文著作	期刊論文	0	0	100%	篇	
		研究報告/技術報告	0	0	100%		
		研討會論文	0	0	100%		
		專書	0	0	100%	章/本	
	專利	申請中件數	0	0	100%	件	
		已獲得件數	0	0	100%		
	技術移轉	件數	0	0	100%	件	
		權利金	0	0	100%	千元	
	參與計畫人力（外國籍）	碩士生	0	0	100%	人次	
		博士生	0	0	100%		
		博士後研究員	0	0	100%		
		專任助理	0	0	100%		

其他成果 (無法以量化表達之成果如辦理學術活動、獲得獎項、重要國際合作、研究成果國際影響力及其他協助產業技術發展之具體效益事項等，請以文字敘述填列。)	1.P. A. Hsiung. Digital Video Recording on Embedded Multicore Systems. 九十七學年度全國大學校院嵌入式系統設計競賽. 教育部顧問室、SOC 總聯盟 2.P. A. Hsiung. Automatic Embedded System Code Generation for an Entrance Guard System with Mobile and Ubiquitous Control. 亞洲區 DSP 暨嵌入式系統應用競賽-台灣區分區競賽. 美商德州儀器股份有限公司

	成果項目	量化	名稱或內容性質簡述
科教處計畫加填項目	測驗工具(含質性與量性)	0	
	課程/模組	0	
	電腦及網路系統或工具	0	
	教材	0	
	舉辦之活動/競賽	0	
	研討會/工作坊	0	
	電子報、網站	0	
	計畫成果推廣之參與（閱聽）人數	0	

國科會補助專題研究計畫成果報告自評表

請就研究內容與原計畫相符程度、達成預期目標情況、研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）、是否適合在學術期刊發表或申請專利、主要發現或其他有關價值等，作一綜合評估。

1. 請就研究內容與原計畫相符程度、達成預期目標情況作一綜合評估

■達成目標

□未達成目標（請說明，以 100 字為限）

□實驗失敗

□因故實驗中斷

□其他原因

說明：

2. 研究成果在學術期刊發表或申請專利等情形：

論文：■已發表 □未發表之文稿 □撰寫中 □無

專利：□已獲得 □申請中 ■無

技轉：□已技轉 □洽談中 ■無

其他：(以 100 字為限)

本計畫共產出 40 篇國內外相關期刊論文。

3. 請依學術成就、技術創新、社會影響等方面，評估研究成果之學術或應用價值（簡要敘述成果所代表之意義、價值、影響或進一步發展之可能性）(以 500 字為限)

研究目的：本整合型計畫網羅了各方面的專家，包括了需求分析、軟體設計、軟體工程、嵌入式系統、平行化計算、軟體測試、系統整合等，共同開發設計出一套整合式多核心嵌入式軟體之模型驅動整合開發環境－VERTAF/Multi-Core (VMC)。VMC 對於目前國內嵌入式軟體產業缺乏支援自動化的軟體發展整合開發環境現象，提供一個符合開放原始碼(open source)規範的選擇，進而提升國內嵌入式產業軟體設計能力與品質。

2. 人才培育成果說明：計有三位博士、十七位碩士的畢業論文與本計畫密切相關。

3. 技術研發成果說明：本計畫研究集合在需求分析、軟體設計、嵌入式系統、軟體測試、系統整合等領域，開發設計出一套整合式多核心嵌入式軟體之模型驅動整合開發環境，提供給使用者一個最完整的多核心軟體開發環境。

4. 技術特點說明：軟體必須導入平行處理的概念才能有效利用多核心的優點。然而對於熟悉傳統循序式設計的軟體開發人員而言，實現平行運算的技術勢必遭遇到許多的困難。因此我們提出模型驅動及樣式支援的方法來加速多核心軟體的開發，並且降低開發人員設計多核心軟體的門檻。。

5. 可利用之產業及可開發之產品：多核心嵌入式軟體開發。

6. 推廣及運用的價值：多核心嵌入式軟體開發是目前的軟體趨勢，一個整合的多核心嵌入式軟體開發環境將成為不可或缺的利器。