東海大學資訊工程研究所

碩士論文

指導教授：朱正忠 教授

行動裝置的軟體定義與效能改善分析

Software Defined Performance Improvement for Mobile Device

研究生：王冠智

中華民國 一〇八 年 一 月

東海大學碩士學位論文考試審定書

東海大學資訊工程學系 研究所

研究生 王 冠 智 所提之論文

行動裝置的軟體定義與效能改善分析

經本委員會審查，符合碩士學位論文標準。

學位考試委員會

召 集 人 吳坤熹 簽章

委 員 蔡清揆

熊博安

張志宏

指 導 教 授 朱正忠 簽章

中華民國 108 年 1 月 8 日

摘要

本研究宗旨在於透過結合軟體定義的概念來配置手機資源。在現今資訊化時代下，資訊服務必須具備良好的服務品質。傳統管理手機資源配置多為系統以固定演算法控制，在手機資源配置上無論是在正確度、耗時等方面都還有改進的空間，而軟體定義是一個可以嘗試的方式，軟體定義能使傳統系統具有依照運算環境與情況彈性且聰明的反應調整的特性。透過軟體控制使系統具有及時性且正確度也能大幅提升，若能以軟體定義的概念協助手機效能的配置，則可以改善以往傳統系統控制造成所帶來的資源浪費以及降低操作上的不便。

在研究中蒐集使用者在不同時間使用手機的習慣與情況，並分析出適合使用者的資源配置方法。再由軟體定義規則事先調整每個時段的資源配置，讓使用者有更好的使用體驗。實驗中依據使用案例進行測試，結果表明本研究提出的方法可以有效的符合使用者需求。

關鍵字：行動裝置、軟體定義、智慧化、服務品質、軟體工程

Abstract

The purpose of this research is to configure mobile phone resources by combining the concept of software-defined. In today's information services must have good service quality. The traditional management of mobile phone resource allocation is mostly control by fixed algorithm. There is still room for improvement in terms of accuracy and time-consuming in the allocation of mobile phone resources. Software-defined enables the traditional system to have the characteristics of flexible, intelligent response adjustment according to the computing environment and situation. Through the software control, the immediacy and the accuracy can be greatly improved. The concept of software-defined can be used to assist in the configuration of mobile performance, the inconvenience caused by the fixed control can be improved.

In the study, the users' habits and conditions of using the mobile phone at different times were collected, and the resource allocation methods suitable for the users were analyzed. Then the software-defined rules to adjust the resource configuration in each period in advance so that the user has a better experience. This topic uses Android Studio as the client-side App design, and Web server-side MySQL as the space for storing data. It also uses PHP as the medium for the client to communicate with the Web server, thereby verifying the effectiveness of the software definition on the phone's performance configuration. .

Keywords: Mobile device, Software-defined, Intelligence, QoS, Software Engineering

致謝

首先，感謝我的指導教授 朱正忠老師長期以來的教導與照顧，老師以自由但不放縱的風格使我們學習如何獨立自主，並培養我們具有思考、發掘並解決問題的能力，這不只是在專業技術領域的能力，而是學習如何做一個自主負責的人，謝謝老師。同時感謝志宏學長這些年的指導，學長百忙之中仍抽空提供我們各種建議與幫助，使我們能順利克服各種挑戰，謝謝學長。還有感謝口試委員們的寶貴意見，使我的論文更加更臻完美。

非常感謝實驗室的許多同伴，彥華學長在畢業後仍經常關照我們並提供技術指導；沅隴學長、子維學長在求學期間，在修課與學習上從旁協助與引導，減少迷茫摸索的過程；昀陽、豐傑、哲維、晏宸、行健、容聿、佳雯、盈汝、育盈、嘉玟在我專心投入研究時，協助處理實驗室的大小事務；最感謝的是黃建華學長，不只提供了我許多指導與理念，學長隱藏的付出是實驗室最大的支柱；也感謝系桌的學弟妹們，為我的碩士生涯添增了許多歡樂。謝謝你們。

最後我要感謝我的家人，給予我莫大的支持與鼓勵，讓我能順利的完成碩士的課業與研究。

本論文能完成包含了指導教授朱正忠老師團隊眾多成員的努力，包含朱正忠老師、張志宏老師、許哲銓老師與實驗室成員們，謝謝各位。

目錄

圖目錄

表目錄

第 1 章　導論

1.1 前言

現今所使用的 Android 智慧型手機作業系統其實和一般使用的電腦系統相同，都是採用「多工處理」的作業系統，「多工鍵」就是為了開啟「多工選單」的專門按鍵，啟動「多工選單」後，可以查看目前運作中的應用程式（僅限「主動啟動」的程式，若程式為背景執行，則需到「應用程式管理員」查看、關閉），因此我們在使用手機時，可以同時執行多個不同的應用程式，也能在程式之間自由切換，或在多工選單將程序關閉。但因為「多工處理」的架構，在使用時若是跳離、切換程式，其實程式本身並未關閉，而是暫存在手機的記憶體中，提供使用者下次開啟應用時可以較快的啟動，所以當開啟的應用程式過多或占用的記憶體滿載後使用者有記憶體需求，Android 作業系統將會在記憶體低於標準值時啟動記憶體管理機制由最近最少使用的(Least Recently Used)算法，挑出記憶體中較久沒有被使用者啟用的應用程式以釋放足夠的記憶體空間給即將開啟的程式，可見手機資源不全然由作業系統管理，也能依使用者的行為做關閉。在 Android 4.0 之後的手機所設置的，除了點選切換其他程式與透過 android 作業系統做資源上的管理，市面上已經有許多 App（如獵豹清理大師、Clean Master、…）能在作業系統介入記憶體管理控制資源釋放以前先將系統級別以外的程序全部關閉、釋放記憶體，以達到省電、優化的作用。這麼做的確能夠方便快速的整理手機，但是使用者所需的應用程式如果被頻繁的關閉，將會導致該程序須不斷地重新載入記憶體，進而浪費更多時間[1]。

在資訊領域中，雲端計算、工業物聯網、大數據、人工智慧等領域蓬勃發展，軟體的重要性明顯大於硬體設備，為了應付瞬息萬變的環境，「軟體定義」的導入可以虛擬化所有元件、抽象化功能、將資源池化和自動化流程，動態的、可擴

充的、彈性的進行資源管理，由軟體管理以及時修改、快速正確調整及降低成本[2]。

現今有許多 App 能在 Android OS 之前先將系統級別以外的程序關閉、釋放記憶體，以達到省電、提升效能的作用。這麼做的確能夠方便快速的整理手機，但常用的 App 如果被系統頻繁的關閉，將會導致該程序要不斷地被重新啟動，進而花費更多等待時間。故認為較佳的應用程式管理，應是導入軟體定義概念，透過軟體管理，將不常使用的程序關閉，將經常使用的程序保存在背景裡，供使用者能迅速地開啟所需的程式，才能有效管理資源並減少時間與效能的耗損。

1.2 研究動機

智慧型手機日漸普及，已成為現代人生活中不可或缺且隨身攜帶的設備，且每人持有數量可能不只一隻，在使用習慣上也頻繁的檢視或使用手機。智慧型手機無論是普及率、持有率、使用率都非常高，但由於有些應用程式占用資源比較大，當同時執行這些應用程式時，背景程式往往會被關閉，導致下次要使用時必須重新開啟，這些應用程式可能是需要頻繁使用的，在頻繁的開關下對手機的消耗反而更大，同時背景程式可能有其他許多占用資源低或使用率低的應用程式，若是將它們關閉則可以空出足夠的資源分配給那些占用資源較高且被頻繁使用的應用程式。可以發現普遍而言智慧型手機在使用上的資源分配不佳、電量消耗過快，導致必須隨時充電和整理應用程式，這表示有大量的資源因未被正確的使用而浪費。就使用層面而言，使用者執行應用程式時，可能該應用程式會自動開啟另外的應用程式，但是新啟動的應用程式是不被使用者需要的，因此使用者會立即將其關閉。或是使用者正在使用某應用程式時習慣開啟其他的應用程式輔助，但因兩應用程式皆需大量記憶體資源，手機只允許其中一個應用程式執行，而不允許暫不使用的應用程式在背景待機，讓使用者在每次切換應用程式時都必須重新開啟。也因為手機資源無法有效的被分配或是無法配合使用者的喜好，無論是

對於手機或是手機的使用者而言都造成一大困擾，因此希望透過結合軟體定義來調整資源配置，達到減少使用者的負擔，增加使用者體驗。

例如：透過軟體定義使軟體自動調整，讓手機記憶使用者的習慣來調整資源配置，透過記錄結果進行分析來避免仍使用的應用程式被關閉以及減少開啟不必要的應用程式。藉由這個方式有效的進行資源配置提供使用者能更有效、更便利的進行應用程式的操作體驗。

1.3 研究目的

在本論文中，研究軟體定義技術，探討將此技術應用於行動裝置系統控制的可能性及方法。軟體定義是一個在資訊技術領域新興的軟體技術集合，但過往大多主題所探討的對象皆為基礎設施，如軟體定義網路，軟體定義網路的核心概念是將資料層(Data Plane) 與決定傳輸路徑的控制層(Control Plane)分隔開來，支援此一概念的網路設備將具有設計統一的資料層，由授權的使用者或預先定義的控制程式觸發，去動態地修改控制層(Control Plane)政策，提供高彈性化的調整。軟體定義技術有效的提升網路服務的品質，若將該技術應用於更多領域如作業系統、應用程式等等，將能建立具備高彈性與智慧化資訊服務，因此我們研究軟體定義網路中，藉由軟體去做定義與規範，應用至傳統行動裝置系統中，使系統服務能夠具備高彈性及智慧化的特性，有效提升使用者在操作上的體驗並降低資源的浪費。

為了將軟體定義技術應用於更多領域，首先我們必須了解其運作機制與特徵，在本文中比較並分析非軟體定義與軟體定義的系統的結構，了解其差異，從中找出是哪些元素使得系統能具備軟體定義的特徵；換句話說，只要系統具備這些元素，即是軟體定義的系統。

隨著對軟體定義研究的探討，我們注意到軟體定義的概念不只能對底層資源進行調控(如 SDN、SDDC 等)，對於作業系統層、應用程序層等等，理論上只要具備軟體定義的元素，即可具備軟體定義的特性以及優勢。軟體定義的理論已有許多討論，然而實際的建構與運用於行動裝置的方法卻缺乏研究，因此本文將設計並探討可能實現具備軟體定義機制行動裝置控制方法，並對傳統系統的控制機制進行改善，自動化的找出符合使用者需求的控制規範。

在本研究中，收集使用者於行動裝置上應用程式加入軟體定義的特性，以設計出具備軟體定義特性的行動裝置應用程式管理機制與雲端分析架構。透過監測方式收集個人化的使用情形，交由雲端服務平台行儲存與分析，找出應用程式之間的相關性，並依據使用習慣制定特製化的軟體定義控制規則。將軟體定義的元素加入系統。讓該系統具備智慧特性，如 QoS 保證、可靠性、高適應性、低功耗、情境感知...等。研究將包含 SD(軟體定義) Framework 設計、SD 系統運作流程、SD 規則與應用程式相關性的規範、行動裝置系統監視/儲存/控制。每位使用者皆有不同的使用習慣，如本次使用案例為使用者在固定的時段中須同時進行遊戲與查看遊戲攻略；也可以是一名上班族每天開始上班前與晚上睡覺前會長時間使用信箱、通訊軟體的需求。因此在實驗階段將以特定的環境與情境作為實驗對象，結合本文實作的 SD 配置機制針對行動裝置記憶體進行智慧化的管理，以展示和分析我們的研究的可行性與有效性，同時驗證研究出的驗證機制將可以應用在不同的使用者。

1.4 章節安排

本論文之架構分為五章，包含「導論」、「背景知識與相關研究」、「研究方法」、「實驗與分析」以及「結論與未來工作」等部分，其架構流程與大綱如下：

- 第一章 導論：說明本論文之研究背景、研究動機與目的
- 第二章 背景知識與相關研究：針對軟體定義網路、軟體定義、行動裝置架構以及相關文獻進行研究
- 第三章 研究方法：說明本論文提出之方法、架構模型與實作方法
- 第四章 實驗與分析：將本論文提出的方法進行實驗與結果分析
- 第五章 結論與未來工作：說明本論文的結論、貢獻與未來可能的研究方向

第 2 章　背景知識與相關研究

2.1 軟體定義網路(SDN)

軟體定義網路 (Software-Defined Networking, SDN)，又稱網路虛擬化 (Network Virtualization)[3]是一種新的網路建置概念或技術。透過這樣的改變，軟體定義網路具有自動化的特性，並提供更靈活、更易操作與管理、可擴充性、隨環境不斷變化與成本控制等優勢。SDN 整體架構[4]如圖 2-1 說明：

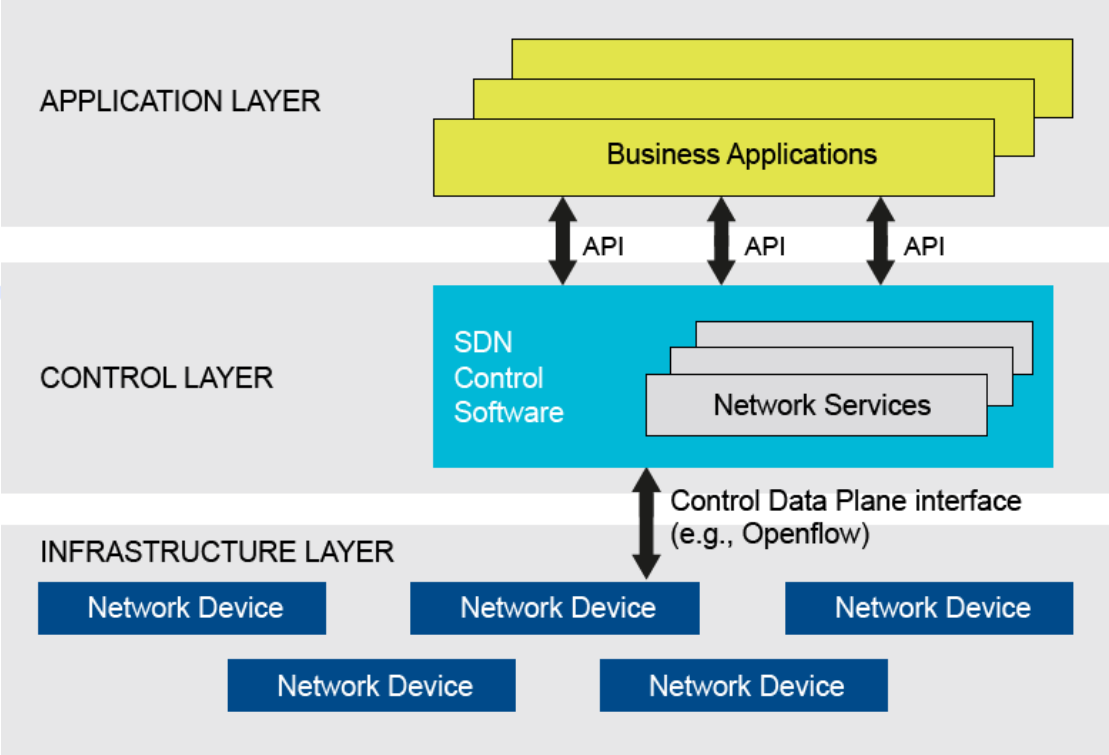


圖 2-1 SDN 架構

SDN 的網路模型總共分為 Application Layer、Control Layer 與 Infrastructure Layer 三層。與傳統網路架構最主要的差異在於它改變了網路的控制模式，將原本網路管理的功能交給控制層(Control Layer)的控制器負責，經由控制器下達指令給基礎設施層(Infrastructure Layer)的設備，網路設備則負責封包的傳送。開發者在 Application Layer 可以開發應用軟體並透過 Rest API 或 Java API 等介面

將規則部署於控制器內。控制層與基礎設施層之間的資料交換，目前最常見的是 Open Network Foundation (ONF) 這個組織所定義的 OpenFlow 協議。SDN 技術藉由配合自律 QoS 未來可達成自我管理解決方案的網路系統。

SDN 藉由軟體來改變網路中的動作與機能，讓網路可以自動且更加自由的控制與管理，提升網路的服務效率與品質，受規模逐漸擴大化、複雜化的通訊業者或企業重視[6]，透過 SDN 的技術能更有效率的管理大量的網路設施，並且在人力與時間等方面節省公司或企業的費用，並減少資源浪費。同時隨著 SDN 和 SDDC 等 SDE 技術問世，能實際支援多雲的雲端基礎架構管理功能陸續到位，軟體定義將成為企業實現雲端策略或 IT 即服務（IT as a Service）最主要的目標[4]。

2.1.1 SDN 的實施：

目前常見的 SDN 實施方法可分為三大類，包含 OpenFlow、Virtual Networking 與 API 實施[4]。

OpenFlow：該技術由開放式網路基金會 (Open Networking Foundation, ONF) 所推展，使用該技術的 SDN 建置可稱為 SDN/OpenFlow。採用 ONF 的 SDN 架構最主要優點，在於它有行業標準之通信協定 OpenFlow，以及眾多的供應商支援。OpenFlow 技術建立的 SDN 架構與上一章節說明的架構相同，並且克服控制層與基礎設施層的聯結問題。OpenFlow 技術將控制層移動到一個單獨的設備稱為控制器 (controller)，負責計算網路的最佳路徑，並控制基礎設施層的交換器進行對應的資料流轉發。並使用 OpenFlow 為 controller 的架構標準，以標準協議規範網路交換器(switch)和控制器(controller)之間的溝通協議。[4]

Virtual Networking: 採用分離的控制平面和轉發平面，由特定廠商所推出控制器與虛擬交換機組成，通常與現有網路設備採共存(overlay)的機制。[4]

API: 直接通過可程式化介面 (API) 對網路設備進行控制，廣義定義上它可控制包括 OSI 標準堆疊從 layer2 到 layer7 的所有設備。這種方式並沒有將控制平面和轉發平面分離，控制平面也不是集中的。目前由既有的網路設備大廠因應 SDN 趨勢而提供這樣的方式。以 API 方式提供其網路設備可被用戶程式直接控制。[4]

2.1.2 SDN 的優勢

根據 InformationWeek 2012 年對軟體定義網路的調查與研究，IT 組織認為 SDN 可以幫助他們克服一系列的挑戰，像是提高網絡的利用率和效率、提高日常任務的自動化、提高安全性。SDN 還可以降低成本。[4]

根據開放式網路基金會 ONF 的定義，SDN 包含以下優勢[8]：

- 可直接撰寫程式：藉由分離控制與轉發(forwarding)的耦合關係，網路的控制方法變得可以直接撰寫程式。
- 敏捷：將轉發的控制動作抽象分離，使得管理員可以動態調整網路流量，以滿足不斷變化的需求。
- 集中管理：除了網路的控制被集中在抽象的 SDN 控制器，控制器同時也能瞭解網路的全域狀態，進而提供後續的應用。
- 透過程式配置：SDN 使得網管人員能通過自定義的程式，動態、自動化的進行快速的網路配置與管理，包含優化網路和安全，並且自定義的程式不依賴於特定的軟體，因此可以編寫自己的 SDN 程式。
- 開放的協定與中立供應商：SDN 的實施採用開放的協定，提倡軟體程序介面的使用，允許第三方應用程序的導入來改善網絡服務或甚至成為新的網路服務。SDN 同時簡化了網路設計和操作，因為 SDN 控制器提供了指令，不再限制於特定的供應商或設備。

2.2 軟體定義一切(Software Defined Everything, SDE)

從技術的發展來看，全球資訊科技正朝著軟體定義(Software Defined)前進，終極目標是將所有裝置虛擬化，未來基礎架構能隨軟體需求隨選(Software on Demand)一樣的交付服務[9]。這個軟體定義方式將成為世界每個領先資料中心的典範，因為它代表企業具備能因應快速改變的強大核心引擎，能賦予資訊科技在瞬息萬變的商業環境，快速提供使用者所需的 Apps 和服務。「軟體定義一切」的論點穩固且緊密結合：虛擬化所有元件、抽象化功能、將資源池化和自動化流程。在這個模式中，將逐漸看不見個別的硬體設備[10][11]，只要透過動態、可擴充的軟體層就可以管理底層運算資源。近期較成熟技術有軟體定義網路、軟體定義安全、軟體定義儲存、軟體定義架構。

在 SDE 的環境下，靈活、彈性與即時是最基本也重要的因素，因此透過一些 Monitoring 與 Operations Automation 的工具及整合一些 Traffic Engineering 等相關的機制，就可以完全即時掌握網路的現況，並提供最符合應用軟體或服務的網路需求。目前此議題相關研究大多以探討標準化網路系統服務整合資源管理結構為主，而在資訊系統開發部份，較少相關研究，因應雲端化服務環境中，我們需要更多彈性化軟體系統支援管理環境，因此於本研究中，將探討關於如何進行定義軟體系統開發應用於軟體定義結構規則。

2.3 Android 系統架構

Android 是一個手機的完整系統，大致上可分為上下兩層，上層採用 Java 撰寫應用程式，而下層則採用 C 語言撰寫系統程式。如果我們將上下兩層再度細分，則可將上面的 Java 應用層分為應用程式框架（Application Framework）與應用程式（Applications），然後將下面的 C 語言系統層分為系統函式庫（Libraries）與 Linux 作業系統層，因而形成如下圖所示的四層式架構。

應用程式：Android 內已提供了一些核心應用程式，例如瀏覽器、日曆、連絡人等。在早期的 Android 應用程式開發中，通常通過在 Android SDK（Android 軟體開發包）中使用 Java 作為程式語言來開發應用程式。

應用程式框架：Android 應用程式框架層是開發人員在發佈核心應用時所使用的 API 框架，該應用程式架構用來簡化元件軟體的重用；並且任何其他的應用程式都可以使用其所發佈的功能塊（不過得遵循框架的安全性限制）。該應用程式重用機制使得組建可以被用戶替換。

函式庫：Android 包含了許多以 C 或 C++語言所寫的函式庫，開發者並不能夠直接使用，而必須透過上一小節提到的應用程式框架。

Android 執行環境：在 Android 執行環境裡最重要的就是 Dalvik 虛擬機器，Java 之所以能夠具有跨平台的特點，就是因為 Java 程式是在 Java 虛擬機器（Java Virtual Machine, JVM）上執行的。Android 上則是以 Dalvik 虛擬機器來執行 Java 應用程式。在 Android 上，Java 程式被編譯成 Java 類別檔(.class)之後，還需透過工具轉換成 Dalvik Executable(.dex)格式，才能在 Dalvik 虛擬機器上執行。Dalvik 虛擬機依賴於 Linux 的一些功能，比如執行緒機制和底層記憶體管理機制。

Linux 核心：Android 的核心系統服務依賴於 Linux 2.6 核心，如安全性、記憶體與進程管理、網路協定堆疊和驅動模型。Linux 核心也同時作為硬體和軟體堆疊之間的硬體抽象層。除了增加 Android 私有的驅動程式外，Android 核心是一個標準 Linux 核心。

2.4 Android 記憶體管理機制

Linux 記憶體管理中無論實體記憶體有多大，Linux 都將其充份利用，將一些程式調用過的硬碟資料讀入記憶體，利用記憶體讀寫的高速特性來提高 Linux 系統的資料訪問性能。而 Windows 是只在需要記憶體時，才為應用程式分配記

憶體，並不能充分利用大容量的記憶體空間。換句話說，每增加一些實體記憶體，Linux 都將能充分利用起來，發揮了硬體投資帶來的好處，而 Windows 只將其做為擺設，即使增加 8GB 甚至更大。

Android 上的應用是 Java 需要虛擬機器，而 Android 上的應用即是每開一個應用就會打開一個獨立的虛擬機器。其實和 Java 的垃圾回收機制類似，系統有一個規則來回收記憶體。進行記憶體調度有個閾值，只有低於這個值系統才會按一個清單來關閉使用者不需要的東西。當然這個值默認設置得很小，所以會看到記憶體經常於存量很少的數值徘徊。事實上並不影響速度。相反加快了下次啟動應用的速度，這是 Android 標榜的優勢之一。而記憶體少的時候運行大型程式會慢的原因是：在記憶體剩餘不多時打開大型程式時會觸發系統自身的調進程調度策略，這是十分消耗系統資源的操作，特別是在一個程式頻繁向系統申請記憶體的時候。這種情況下系統並不會關閉所有打開的進程，而是選擇性關閉頻繁的調度自然會拖慢系統。

LMK (Low Memory Killer)[12]

剩餘記憶體小於應用定義的 APP_MEM 值，開始查看 adj 值清單，kill 相應程式。以 other_file 紀錄系統的空閒記憶體數，根據邏輯判斷出，low memory killer 需要對 adj 高於多少（min_adj）的進程進行分析是否釋放。經過判斷，系統當前的狀態是否需要進行 low memory killer。對每個 sig->oom_adj 大於 min_adj 的進程，找到佔用記憶體最大的進程存放在 selected 中，發送 SIGKILL 資訊，殺掉該進程。

其機制和原理與標準的 Linux OOM 機制類似，只是實現方式稍有不同。標準 Linux 的 OOM Killer 機制在 mm/oom_kill.c 中實現，且會被 __alloc_pages_may_oom 調用(在分配記憶體時，即 mm/page_alloc.c 中)。oom_kill.c 最主要的一個函數是 out_of_memory，它選擇一個 bad 進程 Kill，Kill 的方法同

樣是透過發送 SIGKILL 信號。在 out_of_memory 中通過調用 select_bad_process 來選擇一個進程 Kill，選擇的依據在 badness 函數中實現，基於多個標準來給每個進程評分，評分最高的被選中並 Kill。一般而言，佔用記憶體越多，oom_adj 就越大，也就越有可能被選中。

2.5 開發流程與其他背景知識

RUP (Rational Unified Process)

統一軟體開發過程（RUP）[13]又稱為統一軟體過程，是一個面向對象且基於網路的程式開發方法論。根據 Rational(Rational Rose 和統一建模語言的開發者)的說法，如一個線上的指導者，它可以為所有方面和層次的程式開發提供指導方針、樣版以及使用案例。可以說是使用一個通用的框架(Framework)，用來描述特定的開發流程，任何的企業都可以根據自己的需求，適當的修改統一流程，以符合組織團體需要的開發流程。

它讓開發人員可以將統一流程適當的導入開發過程中，在遵循統一流程所定義之原則的同時，統一流程也指導開發團隊的方向。而從設計與實作統一流程的觀點而言，它是一個流程產品(Process Product)，跟其他的軟體產品具有相同的特性，而不同的是它提供了一組 Rational 的軟體開發工具的發展準則。統一流程具有定義明確且標準化的結構，這個結構主要是以物件導向的方式來形成。

RUP 本身是一種軟體工程流程(Software Engineering Process)。它提供研發單位一個嚴謹的方法，分配軟體開發工作和責任；目的是確保在可預期的時程和預算內，開發出符合使用者需求的高品質軟體。

QoS (Quality of Service)[15]

是指在整個網路連接上應用的各種通訊或程式類型優先技術。QoS 技術的存在是為了獲得更好的聯網服務品質，QoS 是一種控制機制，提供了針對不同用戶

或者不同資料串流才用相應不同的優先順序，或者是根據應用程序的要求，保證資料串流的性能達到一定的水準。

SOA (Service Oriented Architecture) [16][17]

SOA 服務導向架構是一種新興的系統架構模型，主要概念是針對學校或企業需求組合而成的一組軟體元件。組合的元素通常包括：軟體元件、服務及流程三個部份。當學校或企業面對外部要求時，流程負責定義外部要求的處理步驟，服務包括特定步驟的所有程式元件，而軟體元件則負責執行工作的程式。

SOA 已成為現今軟體發展的重要技術，透過 SOA 讓異質系統整合變得容易，程式再使用度也提高。開發者不必自行開發或擁有所有程式元件，發展者可以視其需要組合網路上最好的服務。不再受限於特定廠商的產品功能或是平台，達到真正的開放性(Openness)。

以目標為導向以樣式為基礎之軟體模型(Goal-Driven Pattern-Based Model) [18][19]

近年來，以模型轉換為導向的程式設計觀點引起了相當大的注意，其中最關鍵的在於如何透過模型轉換的方式將軟體模型細化。然而，在這些方法上缺乏對於非功能性需求的轉換探討，尤其是如何結合目標以及設計樣板來達成非功能性需求對品質的研究更是缺乏。為了進一步探討目標、設計樣板、XML 軟體模型轉換之間的關係，因此提出一個軟體模型以目標為導向、以樣板為基礎的模型轉換方法，以漸進式的方式，來分析及建構軟體分析模型，並且進一步的轉換成為設計模型以處理使用者需求中的非功能性需求。

物件導向(Object-Oriented, OO)：

現代的軟體開發觀點，許多都是透過物件導向的觀念來進行分析、設計。利用物件導向所建製的軟體系統，它們的主要構成元素，就是物件以及類別。Coad

與 Yourdon 在 1990 年時[21]，將物件做以下的定義：「物件—問題領域中某些事物的抽象化，可以反應保留系統資訊的能力、與系統互動的能力以及兩者兼顧。」

模型驅動架構(Model Driven Architecture, MDA)[18] [22]

MDA 是 2001 年 OMG(Object Management Group)所採用作為軟體發展的一種標準框架(framework)，MDA 可以支援 OMG 所建立的各種模型標準：UML(Unified Modeling Language)、MOF(Meta-Object Facility)、XML(Extensible Meta Language)、XMI(XML Meta-Data Interchange)、CWN(Common Warehouse Meta-Model)及 COBRA(Common Object Request Broker Architecture)等。

軟體資訊系統開發的特性是它們有著眾多不同的限制，這些限制包含了一個系統可能會使用的資源(計算資源、儲存運算、網路頻寬、服務操作流暢度等)，假設一個執行中的系統環境(包含到達速度、交錯的動作)，系統應提供服務的需求(時間限制、服務品質、可取得性、容錯度等等)，現有的 Model-Driven 開發架構(MDA)在處理上述這些量化限制方面的能力是較為不足的，所以若將 MDA 在軟體定義上結合軟體工程方法，建構相關系統框架，系統中處理量化能力大幅提升，就能展現出 MDA 在開發軟體資訊系統開發的潛力。

Chain of Responsibility：

Chain of Responsibility 的流程為當有多個物件都能處理某種請求，但處理的能力範圍(權限)不同，當這個物件沒有處理的權限時，能夠將這個請求，傳遞給下一個物件繼續處理[23]。物件的多寡視其情況或是環境需求進行增減，系統可以根據不同的需求或是判斷對其結構中的物件或是節點進行定義，此結構在軟體定義當中是個有效的、明確分工、逐層檢驗的運算方式。下圖圖 2-2 以網路功能請求為例：

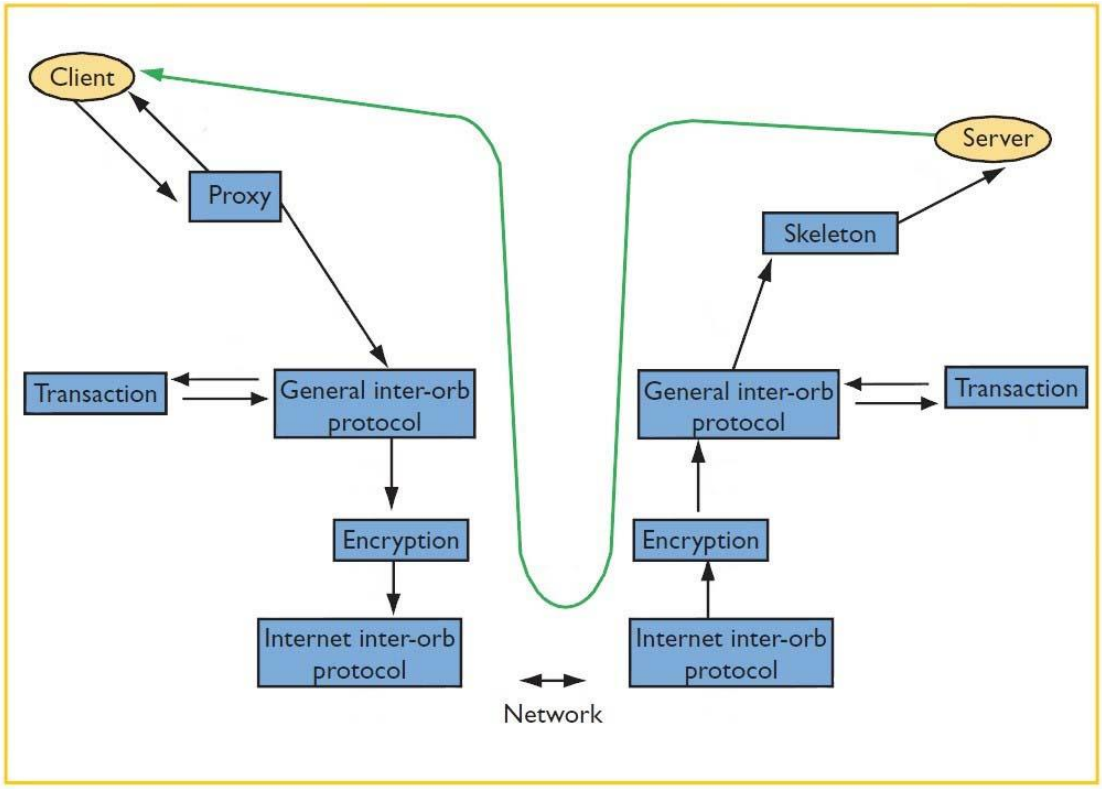


圖 2-2 Chain of Responsibility Architecture Discussion[23]

第 3 章　研究方法

本研究的主要目標為研究一套讓行動裝置資源配置符合使用者需求的方法，包含更彈性、更智慧、低功耗 QoS 保證等功能性或非功能性需求。藉由這個方法，現有的系統經過簡單的調整後能獲得部分的改善；而未來開發的系統，以此方法為基礎將能展現出更加靈活且聰明的特性，提供更佳的服務體驗。

欲使傳統系統具備有軟體定義的機制，本研究設計出一套流程，透過軟體自動化的收集行動裝置的效能資訊、應用程式開啟與關閉的情形，並上傳至服務平台進行使用習慣的分析與軟體定義資源分配的規則規範，找出最適合使用者的資源分配規則，大幅提升了使用者在操作上的體驗，同時降低了反覆開啟應用造成的資源浪費。本研究將以軟體定義為基礎，在現有行動裝置系統上導入軟體定義的技術使系統具備自我改善機制以及更有效率的資源運用。

在本章節中將依序說明我們的研究步驟，首先為了解軟體定義架構與原理導入行動裝置，必需了解並分析傳統網路與軟體定義網路技術，找出其中的差異與使其具備靈活性的主要元素。第二步設計出一套流程來蒐集、分析使用應用程式的相關資訊，並以服務平台進行儲存與後續分析，並自動化的定義出軟體定義控制規範。接著將軟體定義建構至行動裝置中，依照服務平台所訂定出的控制規則，隨著應用程式的優先權重不同進行自動化的控制。最後將重複蒐集、分析、規則定義的流程，透過優化軟體定義機制，為使用者定義出更加貼近使用者需求的控制規則。在下一個實驗的章節，將以此方法於常見的操作情境中進行實驗，作為驗證我們方法可行以及有效的依據。

3.1 軟體定義架構分析

3.1.1 傳統軟體定義架構

本研究以 SDN、SDDC 等技術為基礎，比較並分析非軟體定義與軟體定義的系統，研究將軟體定義技術應用在各種系統抽象層的方法，歸納出具備軟體定義特性的系統 Architecture，並設計符合軟體定義系統的 Model。

首先我們要設計 Software-defined 的系統架構，為了達成這個目標，我們將傳統非 Software-defined 系統與 Software-defined 系統兩者進行比較，分析彼此之間差異，探索傳統的系統受到哪些限制，並從中找出改善的架構及方法。

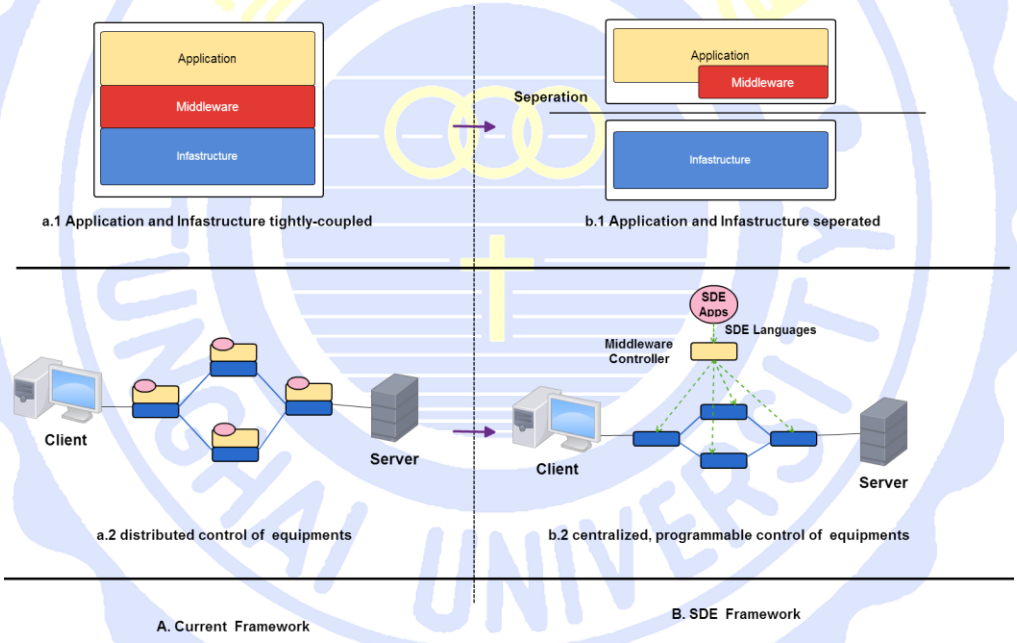


圖 3-1 Difference between Current & Software-Defined Architecture[24]

圖 3-1，列出了傳統系統與 SD 系統的架構。在傳統的架構中，軟體、作業系統與硬體三者是相互綁定的，系統建立之後即僵化不可變動；而 SD 系統則增加一個控制機制，並將軟硬體分離，藉以讓整體架構變得靈活。例如圖中的 a.2 假設是一個傳統網路，每個路由器都與控制機制綁定，在建置後封包將固定的路線前往目標，很容易發生雍塞的情形；而 b.2 假設是一個 SDN 的網路環境，控

制機制與實體路由器分離，當網路環境發生異狀，控制機制能整合資訊並選出更適當的路徑來傳送封包，即可有效降低雍塞的發生率。

從傳統應用程式的角度來看，一個傳統軟體在設計，並不會去考慮可用的資源是否會有變動，因為他們假設環境是固定的，例如圖中 a.1，一旦設計及實作完成後就變得不可變動，即使擁有更多的運算資源，也會因軟體程式不支援而無用武之地。而 SD 的系統架構，則把中介層與底層硬體分離(b.2 所示)，如此一來不論底層如何變化，皆可由中介控制層進行調整，讓上層軟體提供更符合現況環境的服務。

綜上所述，如果要設計一套智慧化的 SD 系統框架，讓系統包含安全性、可靠性、QoS 保證、高適應性、低功耗等等功能，系統必須將軟硬體以抽象的方式分離，並具備情境感知能力，最重要的是有一套能正確控制、調度資源的控制機制。控制機制是智慧化的重點所在，必須能同時準確的調度底層硬體資源，以及控制上層應用程式提供恰當的服務，當我們建立出正確的控制機制，即可讓使用者感受到程式變得更有彈性、更聰明，也就是智慧化的系統。

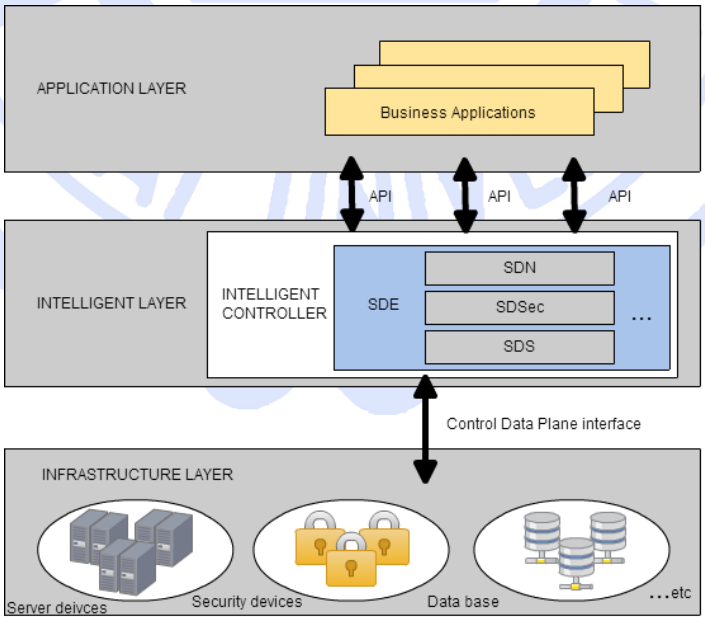


圖 3-2 Software Defined Everything Architecture [24]

前段說明了，要具備 SD 所必須具有的各種特性。以這些特性為依據來觀察現有的各種 SD 環境，可以清楚的發現它們皆屬於同一結構，如圖 3-2 所示，Software Defined Everything Architecture 圖分成三個層級，依序為 Application Layer(應用層)、Intelligent Layer(智慧層)和 Infrastructure Layer(基礎設施層)，藉由 Intelligent Layer 建立智慧化與軟硬體分離的結構，使所有底層資源虛擬化，讓使用者獲得隨選隨用、動態自動且智慧調整的基礎資源服務，這也就是現在 SDE 的最終目標，當然不論 SDN、SDSec...等等都被包含在 SDE 這個概念之下。隨著 Intelligent Layer 的完整度與智慧程度提升，其中所涵蓋的底層基礎設施環境和資源、安全加密保護...等，都能藉由 Intelligent Controller 架構加以調整與控制，漸漸使資源的使用調配達到最佳化，讓使用者感受到更佳的服務。

3.1.2 軟體定義概念進階應用

本文提出更進一步的概念，回歸資訊系統的最終目的，即為提供使用者想要的服務。當 SD 的概念應用於此，不再局限於 SDE 僅基礎資源軟硬體抽象分離的應用，而是將服務抽象化，將原始、不聰明的服務透過 SD 的概念，轉型為自動智慧化調配的服務。如圖 3-3 所示，SDN 技術將傳統不具靈活性的網路，透過將控制與硬體分離的方式達到靈活控制的目的。我們延伸此概念，將服務與控制分離達到靈活控制與自動控制的目的。這些服務不僅限於 Infrastructure Layer 或某些層級的服務，而是將所有資訊服務皆視為服務，包含各種硬體基礎設施提供的運算儲存資源、作業系統提供的平台服務、各種應用軟體、資訊系統所提供的服務，甚至更小範圍的例如一個功能、一個提供使用者輸入的動作等等皆為服務。

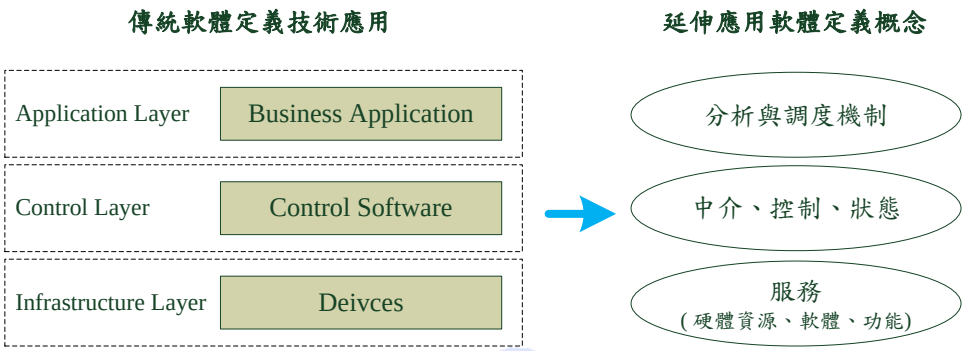


圖 3-3 軟體定義概念延伸示意圖

上述提到的這些服務，雖然牽涉到硬體、韌體、軟體、平台等領域，然而最終目的皆為提供某些服務給使用者，但在傳統的開發流程中，大部分服務的執行方法、流程、動作模式等皆互相綁定，無法依據環境或使用者的需求而改變。透過軟體定義的概念，服務與該服務的控制分離，並以更高層的軟體集中控制，達到靈活與動態調整的目的。

軟體定義技術除了使服務更加靈活、更容易調整之外，更重要的優勢是它將原本需要透過人力手動調整的動作，藉由軟體自動進行調整，使得服務能隨時依照環境的變化而自動動態的進行改變，提供更符合環境需求的服務。以網路環境為例，網路上資料的傳送數量無時無刻都在改變，即使架設一套中央控制系統可以對網路流量與封包傳送路徑進行調整，也必須消耗大量的人力與時間，才能不斷改善網路的傳輸狀況，一旦停止了人力的介入，網路即回到傳統的狀態，無法自動進行優化。從這個例子中可以發現，軟體定義網路的技術並不單只有將軟硬體分離的概念，它還包含情境感知、資料分析以及自動化控制的技術。最終使得服務的執行規則不再是完全由人工的方式去設置或調整，而是由軟體去定義其他軟體服務的規則。

本研究經過分析軟體定義技術之後，發現只要符合軟體定義技術所需的架構與特性，任何資訊服務都可以利用軟體定義技術的原理來提升服務品質，從抽象

的軟體結構來看(levels of abstraction)，不論是 Infrastructure Level、OS Level、Platform Level 或 Application Level，都可以應用軟體定義的技術。

本研究以延伸應用的軟體定義概念與軟體定義主要元素為基礎，設計出新的結構如圖 3-4。要讓服務具備 SD 與智慧化能力的種種必備功能我們合稱為軟體定義功能模組(Software-Defined Function Module, SDFM)，它包含了情境監控、調控、分析與規範等等模組。它可以應用於任何抽象軟體結構層從 Infrastructure Level 到 Application Level，並使之能依照使用者的習慣、需求，進行調控、資料收集分析、重新定義的能力，這些動作將不斷循環進行，使系統自我學習並改善，進而不斷提供更佳更適當的使用服務，也就是 SD 的服務。詳細的模組功能與執行流程將在下個章節進行說明。

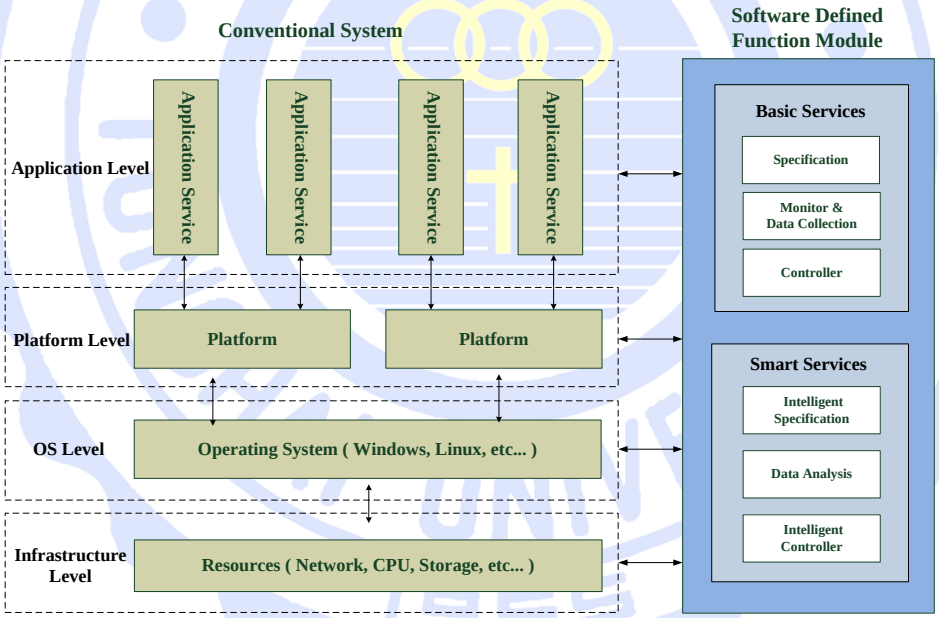


圖 3-4 Software Defined System External Load Architecture [24]

由於自動調控、智慧化的服務，其正確度主要受我們所訂定的規則、邏輯與規範影響，如果設計出的規則不夠完善，不但無法提供更佳的服務，更可能降低其服務品質，因此我們將 SD 功能模組分為基礎服務與智慧服務兩大區塊。基礎服務將依照原本的 SLA 規範，提供小幅度的調整與監控能力，如此將能確保提

供的服務能符合任何使用者的基本需求，以及自動修正嚴重的錯誤行為，同時系統也開始進行資料收集，供後續的分析與智慧化服務使用。智慧服務模組則包含了資料分析、智慧規範設定、智慧化資源調控、優先權重設定等等自我改進的智慧化服務機制。

3.2 軟體定義方法

在上個章節，說明了本研究中所提出的軟體定義技術的延伸應用與概念，並且提出以外部附加軟體定義功能模組的方式，將軟體定義的特性加入傳統資訊服務系統使之轉型為軟體定義系統(Software-Defined System)的方案，在本章節中將說明軟體定義功能模組的運作模式與架構。

軟體定義功能模組的內部運行模式主要可以分為三大模組。各模組間關聯性與互動如下圖 3-5 所示。

控制模組：包含所需的監測軟硬體元件、控制元件、通訊協定以及智慧調整機制，此控制模組基本功能為使用 Object-oriented 訊息傳遞機制（Message Passing）對傳統的系統或是程序進行操作與調整。此模組在建構時依據被掛載系統之 Non-Functional requirement 進行修訂。

資訊處理模組：收集與分析被附掛之系統或程序其所需、執行中或者執行完畢之使用經驗與紀錄。此模組針對 SD 所規範的需求進行資料分析後，分析完之資料將做為 SDS 新的規範基礎，提供 SDS 系統自我改善與學習的數據依據。

軟體定義規範模組：此模組為 SDS 的核心模組，包含了被附掛之系統所要求的各項服務數值，例如：系統服務品質要求、資源使用效率計算、SD 資源規劃與定義、SD 資源分配方法以及各種優先權重的評判等等。此模組採用資料處理模組分析後的結果進行運算與刪減權重值，透過控制模組對系統執行操作前、操作中所需要的軟體定義控制命令。此模組的運算流程多寡依照 Functional

requirement、Non-Functional requirement 進行增減，針對不同的傳統系統制定出所需要的軟體定義規範項目。

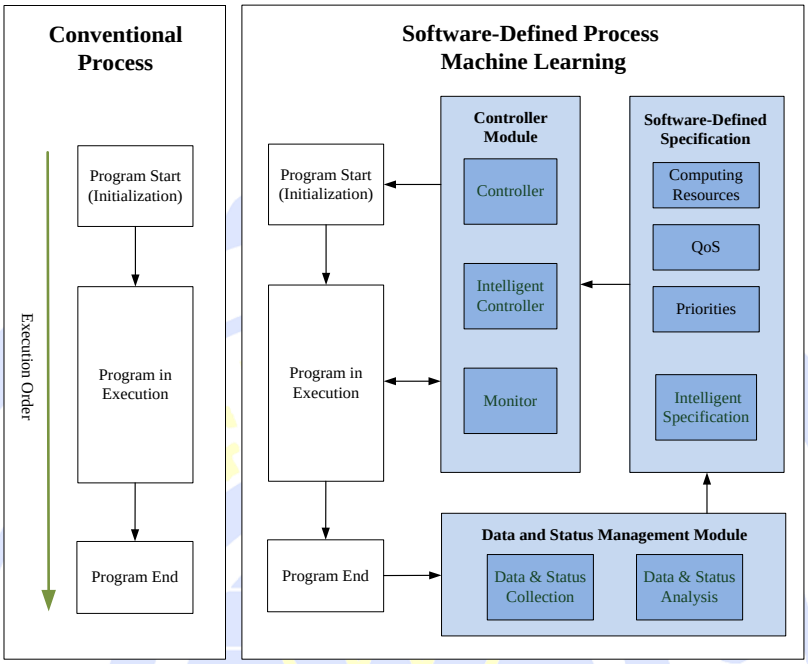


圖 3-5 Software Defined System Process Model [24]

本論文中，採用 Chain of Responsibility(CR)的概念與流程進行 SD 之情境判斷，在 SDS 當中主要區分成兩前進流程，如圖 3-6 說明，橫向與縱向 Chain of Responsibility 的概念，橫軸主要表示一個系統或是軟體從接收到運行指令與結束指令的生命週期，縱軸為說明 SDS 之資訊處理模組及軟體定義規範模組 Chain of Responsibility 的運行流程。

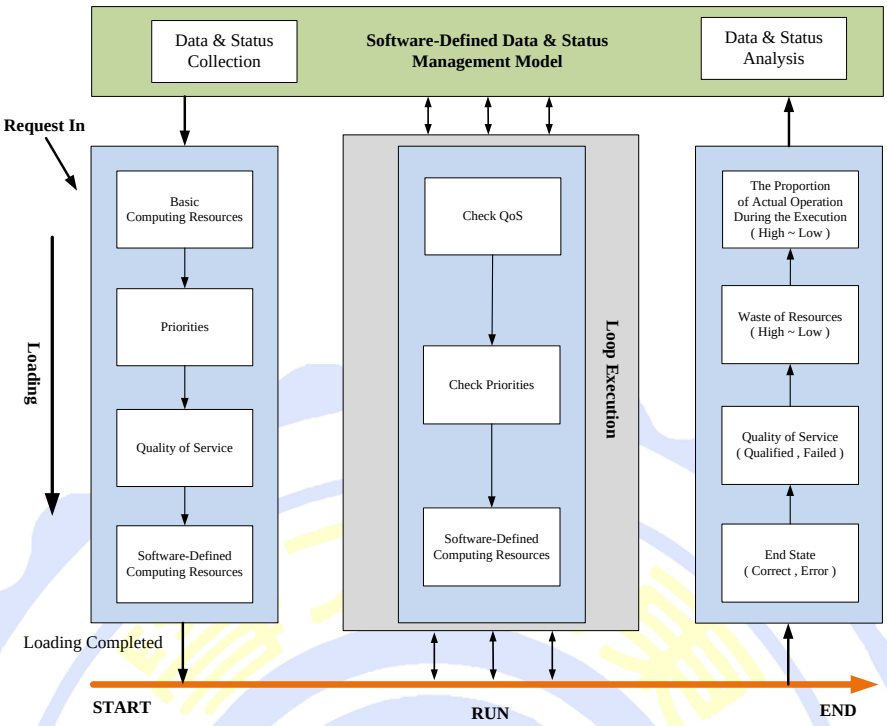


圖 3-6 Software Defined Requirement Process Model [24]

第一部分：SDS 接收到系統或軟體程序之執行需求直到最後接收到結束條件之後，將分為三個階段，分別針對不同階段再分別執行所屬的 Chain of Responsibility 分支，階段 I：收到啟動指令後且尚未載入軟體程序時，進行一次性的 SD 動作，階段 II：軟體程序運行當中執行多次 SD 動作，階段 III：程序運行結束需求滿足後，執行最後一次的 SD 運行。

第二部分：分別針對上述三個階段進行個別從屬的 SDS Chain of Responsibility 流程與規範探討。

- 階段 I：軟體定義載入資源或相關規範的 CR，例如：獲得原始程序所需運算資源，比對 SDS 定義之運算資源，再檢驗 SDS 系統整體資源狀況後，SDS 給出最適當運算資源。

- 階段 II：此階段 SDS 將進行多次監控，CR 流程須確保 SDS 依照當時系統環境給出最佳之運算資源與確保 QoS。軟體運行當中資源需求呈現

波動狀態，主要流程為：監控運算流量、比對 QoS、分配資源調整以及使用經驗紀錄。

- 階段 III：服務或是軟體程序結束之後，執行一套有效的資料收集 Chain of Responsibility 流程，以確保 SDS 的 QoS 以及資料正確性。

3.3 軟體定義需求分析

任何系統要達到軟體定義特性必須滿足以下三項基本需求：資訊蒐集(Information collection)、控制(Control)與規範(Specification)。若要使系統具有自我改善能力則須滿足另外一項進階需求-資訊分析(Analysis)。

◆ 資訊蒐集：

包含任何對欲滿足的服務所需要的關聯訊息，而不只是針對該服務直接影響的資訊。蒐集的資訊越完整，則越有可能進行正確的控制，達成更佳的服務品質。資訊包含服務的改變、基礎資源的變化、情境感知、使用者需求改變等。以 SDN 進行舉例，假設從路由器 A 發送至路由器 B 的封包數過多，除了修改該路徑的往返流量配置之外，也可透過尋找其他路徑的方式改善流量，在只具有該路徑流量資訊的情況下，所能採取的策略有限，反之若具有其他路徑的流量相關資訊，則能分析出更多解決方案，採用更佳的辦法。

◆ 控制：

包含任何對欲滿足的服務可能產生直接或間接影響之控制行為，所有的控制行為皆須受到規範的制約。與資訊蒐集相同，系統所能進行的控制範圍或權限越高，則越有可能調整出滿足使用者需求的結果。控制的範圍包含基礎資源的調度、優先執行順序的改變、改變現有的 rule、資料自動修正等。與資料蒐集不同之處，大量的資訊並不會產生即時的困境，也不易對服務造成直接的影響，但在控制機

制中，授予越高的權限，除了可能產生越佳的服務，也可能產生完全錯誤的服務，在設計時需要謹慎處理。

◆ 規範：

規範分為基礎規範與智能規範，基礎規範由客戶與開發者共同協商後訂定，而智能規範則以原始規範為基礎訂定，所有的智能規範必須滿足基礎規範的約束，且所有軟體定義系統中的控制行為，皆需以規範的約束為控制基準。以一個範例說明，假設一路徑總頻寬為 10，且基礎規範規定上傳下載至少需保留 2 的頻寬，則當下載頻寬不足時，系統可依照智能規範將頻寬資源分配給下載頻寬使用，但至少需保留 2 的頻寬供上傳功能使用。

◆ 分析：

此需求為進階需求，系統在開發及維護時，開發人員會依照規範，制訂大量的規則讓系統運行，然而除人力有其極限之外，不同的使用者對系統的需求也不盡相同，同一套規則要滿足所有使用者的需求是不可能達成的。藉由軟體定義系統的運作必然會蒐集到大量資訊的特性，透過分析這些訊息，可了解系統的運作，也可了解每一個不同使用者的操作習慣，進而讓系統自行制訂出新的智能規範，以滿足每個不同使用者的需求，當然這些智能規範仍需滿足基礎規範的約束。

3.3.1 軟體定義成本

本文提出的概念中，SD 技術的使用以擴充的方式進行，藉由這些額外的功能，使原本的系統或服務具有 SD 的特性。在基礎需求中，資料蒐集必須長期消耗額外的運算資源，且需要額外的資料庫以儲存相關資訊。控制與規範的需求，則需花費額外的開發成本。進階需求的分析，除需要開發相關演算法之外，執行過程也可能會花費大量的運算資源。

3.3.2 軟體定義需求總表

表 3-1 需求類別總表

需求類別	
Function Level	輸入(input)、輸出(output)、狀態、操作分布、使用率、
Application Level	基礎資源調度(僅限軟體內部)、功能調控(單一、多項、前台、背景)、功能優先權調控、功能載入調度
OS Level	基礎資源調度(RAM、CPU、HD、GPU、NETWORK)、服務調控(單一、多項、前台、背景)、服務優先權調控、服務載入調度

表 3-1 將所有可以附加軟體定義特性的需求，依照類型進行分類，並且依照服務的規模進行排列。在設計下層的服務時，可以對自己以及上層的所有需求類別進行設計，例如設計一個 OS Level 的服務時，可以對 Function Level 內的需求進行設計；反之上層的服務在設計時，無法對下層的需求類型進行設計。

表 3-2 通用需求與動作

通用需求	Rule 更新、SD 系統開關、SD 專用資料庫、資料分析，使用者 SD 設定
動作	紀錄、監視、確認、通知、終止、自動調控、智慧調控、加入分析佇列

在表 3-2 中，通用需求是特殊的需求，Rule 更新與 SD 系統開關這兩類型需求不論何種 SD 系統皆必須存在。當任何類別的需求有執行紀錄的動作時，則必定有 SD 專用資料庫的需求。資料分析需求則為進階需求，它影響整體 SD 系統的智能規範制訂，進而對所有自動調控動作產生影響，可以循序的方式加入資料分析的需求，逐步改善系統運行。

在表 3-1 中，所有類別的需求皆需對應表 3-2 中的動作，組合成完整的需求，動作包含：紀錄、監視、確認、通知、終止、自動調控、加入分析佇列，共七項。

例如一個電動門開關的服務，它具有開、關的狀態，對於這個狀態類別，我們要附加 SD 的需求，我們可以選擇要附加多少程度的 SD 屬性，包含狀態是否紀錄、即時狀態監視，特定條件發生時是否執行確認、通知相關人員的動作，或

者自動執行相對應的處理行為，例如現在自動門不斷再重複開關的動作，表示有許多人正在經過，系統可自動調控使門暫時不關閉。最後分析則是收集的資料是否有進行分析的需求，透過長時間的資料統計，尋找人潮密集的時段，制訂新的智慧規範，在人潮密集的時段發生前即可操作自動門採取不關閉的動作。

另外需注意的是，表 3-1 中描述的所有需求，皆包含兩種使用對象設定，分別為整體系統與個人化。舉例來說，若想對期末考成績輸入功能進行 SD 設計，讓它包含紀錄的動作，以整體系統的需求進行設計時，獲得的資料可了解全體學生的期末成績狀況；以個人化的需求設計時，由於輸入成績者為教師，因此教師可了解自己所有學生的成績狀況。使用對象的設定是不衝突的，可以同時使用，但應視為兩種需求處理。

3.4 軟體定義行動裝置

本研究欲使傳統的系統具備為 Software-Defined 的機制，為了達成這個目標，將軟體定義與手機配置進行合併。需要將傳統的手機資源配置呈現，再與加入軟體定義後兩者進行比較，了解相互之間差異，分析傳統的手機資源配置受到哪些限制，並從中找出改善的方法。傳統上手機資源無法依照使用者真實需求自由配置，且手機資源有限，因此資源吃緊時難免會關閉部分使用者仍需要的應用程式，造成使用困擾。而軟體定義可以依據分析數據的結果進行調整，可以提供更適合使用者的服務。

為了達到該目的，本研究設計應用程式並導入軟體定義的概念管理、監控手機資源。而 Android Studio 具有智能化的程式碼編輯功能，會自動提供相關的語法提示，並進一步協助開發者重組、完整化與分析程式碼。提供視覺化的監控工具 Memory Monitor，為開發者追蹤連結裝置的記憶體使用量，並監控 App 的記憶體使用情形。故本研究選擇 Android Studio 設計管理程式，蒐集大量的使用者

習慣作為配置手機效能的依據，透過 PHP 分析資料得知使用者在不同時間使用手機的規律與切換應用程式的習慣，來建立軟體定義規則。

本研究設計與實作之應用程式根據軟體定義規則分配記憶體並由 Android Studio 之資源管理、監控之 API 控制記憶體中的行程來達到抽離控制平面的效果。藉由軟體服務及集中管理分析的方式，找出詳細的使用行為，並預測使用者將來的行為模式，讓使用者會需要的應用程式維持開啟狀態，降低重啟應用程式所花費的時間及資源浪費，進而提升使用者的體驗。

以 Android Studio 作為開發程式的工具，編寫獲取前景程式的資訊及相關需要的服務。並採用 VMware 環境作為建置伺服器端儲存資料的方案，在本地端建立連線環境，透過在此虛擬機的系統上安裝 LAMP server (Linux + Apache + MySQL + PHP)，將研究中所需要用到的 PHP 檔案及資料庫都存放於此雲端虛擬機上，如此一來可以依需要調整虛擬機的配置環境，並達到實驗環境一致的目的。

3.5 蒐集手機使用效能資訊

透過本研究設計之應用程式記錄手機使用效能，並記錄每個應用程式開始與結束的執行時間，以利分析數據後整理出使用者在不同時間的各種使用習慣。傳統人為開啟或關閉皆由使用者根據當時使用狀況而定，因此為了結合軟體定義達到自動調整資源分配，蒐集使用者的習慣建立用以分析判斷的資料庫就是不可或缺的一環。先將資料匯入到 MySQL 資料庫中並建置數個資料表，存放每個應用程式開始與結束執行時間、RAM 的佔有率等資料。

本研究透過 Android Studio 設計的應用程式中，具有 Service 類別常駐於背景中蒐集使用者使用應用程式以及手機狀態的資訊，透過偵測到某應用程式使用完畢（完成一次開關或手機經過休眠）後將所記錄之資料透過網頁伺服器上傳(如圖 3-7、圖 3-8)。PHP 是一個應用範圍很廣的語言，且 PHP 可以在許多的不同

的伺服器、作業系統、平臺上執行，也可以和資料庫系統結合，非常適合於伺服器端執行，故本研究選擇 PHP 作為開發工具之一。本研究為求系統穩定及服務品質等層面後決定以虛擬機方式，建置一個簡易網路伺服器，並安裝 LAMP server 裡面搭載 Apache、PHP、MySQL、phpMyAdmin，且支援 TLS、SSL、HTTPS 的協定。透過 PHP 的程式作為 Client 端 Android Studio 程式傳遞資料的媒介，將資料寫入 MySQL 中，把使用者在各時間點切換的程式，依照程式名稱、開啟時間、關閉時間、剩餘的 RAM 填入資料表中。

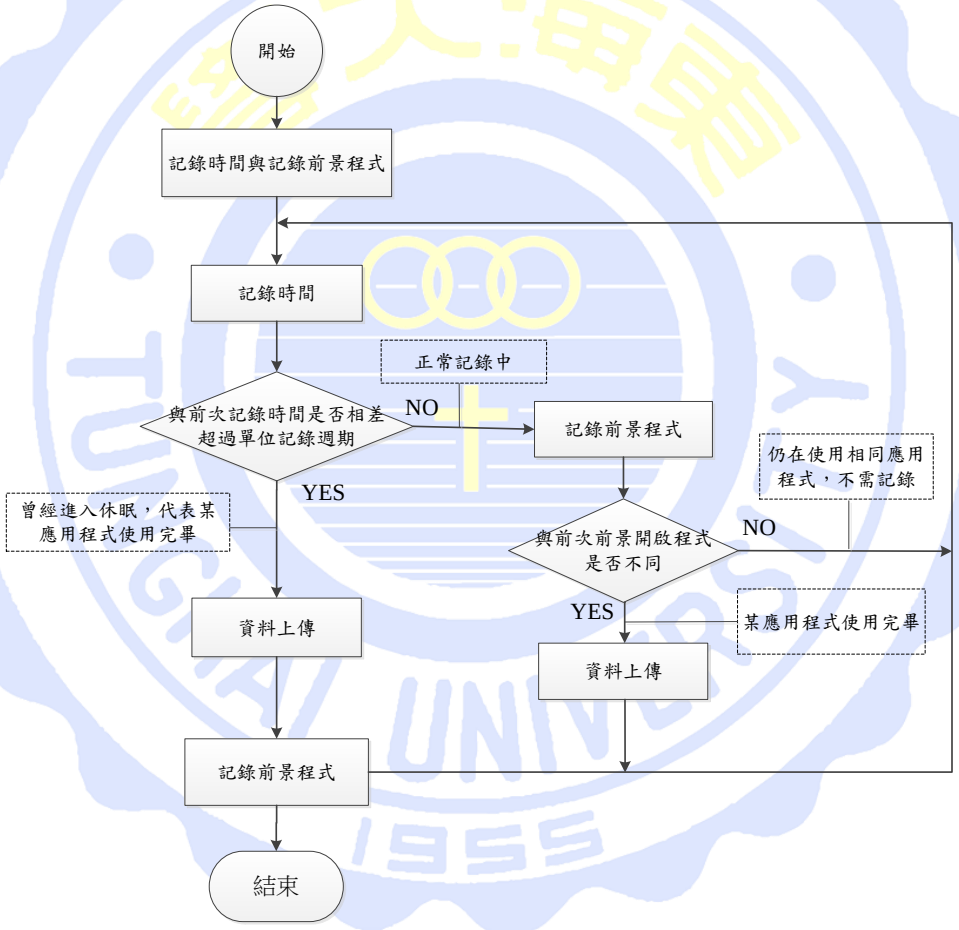


圖 3-7 蒐集使用者習慣演算法流程圖

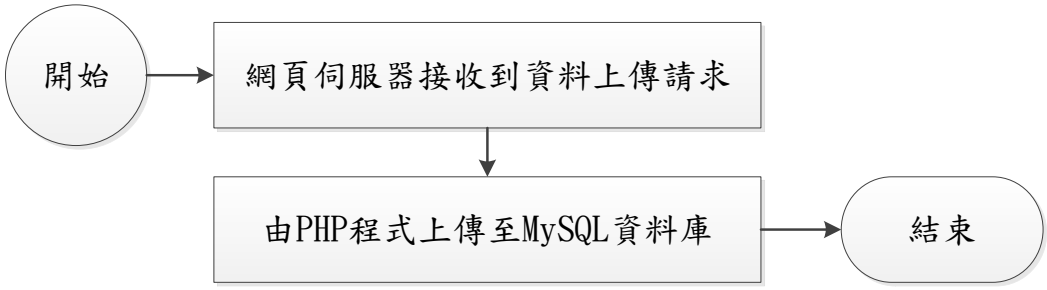


圖 3-8 資料上傳流程圖

紀錄當前被切換的前景程式，將它的包名（Package name)、該程式被開啟的時間、該程式被切換的時間、手機記憶體資源所剩餘的空間以及剩餘電量，透過程式(如圖 3-9、圖 3-10)存入資料庫中的使用者習慣資料表中，作為後續手機效能配置規則的資料分析來源，並建立 offline 資料表進行備份(如圖 3-11)，根據所蒐集的資料，考量到工作日、假日等因素會影響使用者使用情形，因此針對日期進行切割，區分出不同天數，再將三個禮拜每週中相同的日子（如三個禮拜的每個星期一等）根據使用時段將相同包名做次數統計當作它在該時段的權重(如圖 3-12)。

```
private void needUpload(){

    //取得剩餘記憶體
    ActivityManager.MemoryInfo mi = new ActivityManager.MemoryInfo();
    am = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
    am.getMemoryInfo(mi);
    memUse=""+ mi.availMem / 0x100000L;//剩餘記憶體大小(MB)
    //memUse=""+mi.availMem / (double)mi.totalMem;//記憶體剩餘率(MB)
    //取得剩餘記憶體

    //endDateTime=registerEndDateTime;
    mClock.setEndDateTime(mClock.getRegisterEndDateTime());
    upload(preForegroundTaskPackageName,mClock.getStartDateTime(),mClock.getEndDateTime(),memUse,""+Electricity);
    //上傳 pkgName APP開啟時間 APP結束時間 當時記憶體剩餘 當時剩餘電量
    preForegroundTaskPackageName=nowForegroundTaskPackageName;
    //startDateTime=nowDateTime;
    mClock.setStartDateTime(mClock.getNowDateTime());
}
```

圖 3-9 使用數據上傳的程式碼

```php
<?php
$host='127.0.0.1';//location
$name='root';//account
$pwd='st314';//pwd
$db='software_defined_mobile_phone_performance_configuration';//DB name之後要由程式選擇 $_POST['db_name']
//table 也是之後要由程式選擇 $_POST['table_name']
$con=mysql_connect($host,$name,$pwd) or die("connection failed");//連線
mysql_select_db($db,$con) or die("db selection failed");//db選擇
//insert data
$pack_name=$_POST['PkgName'];
$start_dtime=$_POST['StartTime'];
$end_dtime=$_POST['EndTime'];
$memF=$_POST['FreeMemory'];
$Electricity=$_POST['Electricity'];

mysql_query("SET NAMES utf8");
$sql="INSERT INTO user_behavior(textPackageName,textStartTime,textEndTime,textMemoryFreeMB
,textElectricity)VALUES('$pack_name','$start_dtime','$end_dtime','$memF','$Electricity')";
```

圖 3-10 PHP Query 資料庫的程式碼

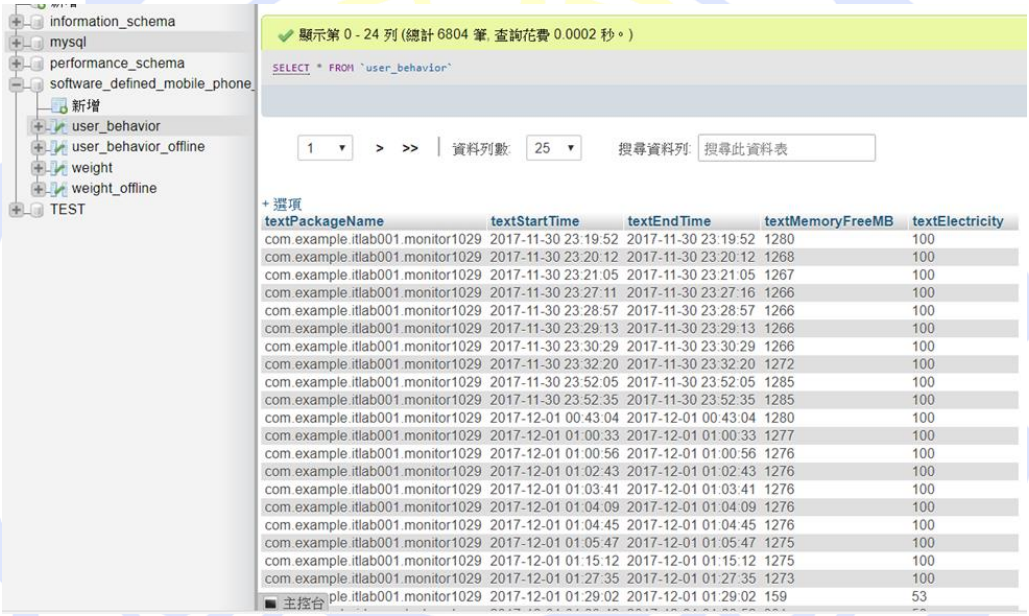


圖 3-11 資料庫儲存蒐集之數據

packageName	weight	startTime	exeDate
com.example.itlab001.monitor1029	6	23:00:00	2017-12-08
com.android.chrome	2	23:00:00	2017-12-08
com.example.itlab001.monitor1029	4	23:30:00	2017-12-08
com.android.chrome	4	23:30:00	2017-12-08
jp.naver.line.android	14	23:30:00	2017-12-08
com.rdevlab.freebitcoin	2	23:30:00	2017-12-08
com.android.systemui	2	23:30:00	2017-12-08
com.example.itlab001.monitor1029	1	00:30:00	2017-12-09
com.example.itlab001.monitor1029	10	01:00:00	2017-12-09
com.sec.android.app.clockpackage	1	01:30:00	2017-12-09
com.example.itlab001.monitor1029	1	01:30:00	2017-12-09
com.android.chrome	2	01:30:00	2017-12-09
com.rdevlab.freebitcoin	2	01:30:00	2017-12-09
com.example.itlab001.monitor1029	3	02:00:00	2017-12-09
com.android.chrome	1	02:00:00	2017-12-09
com.example.itlab001.monitor1029	5	02:30:00	2017-12-09
com.example.itlab001.monitor1029	1	03:00:00	2017-12-09
com.example.itlab001.monitor1029	1	06:00:00	2017-12-09
com.example.itlab001.monitor1029	2	06:30:00	2017-12-09
com.example.itlab001.monitor1029	1	21:30:00	2017-12-09
com.google.android.youtube	24	21:30:00	2017-12-09
com.google.android.youtube	25	22:00:00	2017-12-09
com.google.android.youtube	5	22:30:00	2017-12-09

圖 3-12 應用程式優先權重

使用 PHP 與 MySQL 查詢語言設計演算法統計分析數據再整理歸納軟體定義手機效能配置規則。將一天從 0 時 0 分到 24 點整以每三十分鐘作為基準，切割成四十八個時段，對這每三十分鐘內所有記錄在使用者資料表中的包名進行權重統計，以過去剛好位在今天、七天前、十四天前、二十一天前的時間點將次數乘以七、其餘乘以一，加總當作權重，藉此預測使用者七天後的使用情形。

目的是要區分出當天應用程式的重要性，以一個禮拜七天為例：假設當天為星期二，則將過去三週星期二的比重設定為七，其餘六天比重則為一，藉此將權重比更加區別開來，同時提升該特定時段的重要性。權重越大表示在該區間內被開啟次數頻繁；權重越小表示在該區間內被開啟次數較少，以此區分應用程式的重要性，在了解同一日中的對使用者的重要程度也參考其他六天同一時段使用該應用程式的可能性。

3.6 使用者需求分析與軟體定義控制規範

每個使用者的使用習慣都不相同，因此必須分析蒐集來的數據，藉由本研究定義的規則歸納出使用者的使用模式與習慣。分析哪些程式習慣被同時執行或互

斥執行，藉此得知使用者每段時間對於手機資源配置的具體需求和每個應用程式真正需要開啟及關閉的時段。此階段重點在於將蒐集而來的資料進行整理，分析出使用者的需求，模擬出使用者在不同條件下的資源配置方式，並根據不斷蒐集的資料進行更新並即時調整。

由於本研究會記錄大量的使用者在使用手機上的數據，並將其上傳至 MySQL 資料庫，因此將資料整理成資訊是非常重要的一環，透過 PHP 以及 SQL 查詢語言分析整理後歸納出不同時間與應用程式之間彼此的關係，找出使用者在使用手機上的習慣，建立或更新手機效能配置規則資料表。

在伺服端運算結束後，將包名、權重、執行日期、開始區間的時間點存入規則資料表中，之後依這個資料表中的資料判斷使用者在某個時間區段，可能會頻繁使用哪些應用程式，抓取資料表中的權重進行優先權排序，作為配置手機效能的依據，在手機記憶體吃緊時透過超級使用者權限陸續關閉一些在此區段權重較小的應用程式在記憶體中的行程（Process），釋放手機記憶體資源空間。

在程式碼編寫上，將上傳資料的方式與下載資料的方式區分開：上傳的部分，在 Android studio 專案中使用 volley 將參數傳到 PHP 程式中將資料寫入資料庫，藉此減少手機端負擔；下載的資料使用 JDBC 驅動程式負責，這樣區分是為了在下載資料的時候，能直接 SELECT 到需要的字串，不需要再經過 PHP 檔案的傳輸，直接以 Client 端－資料庫－Client 端較為簡單，也更加快速。

3.7 軟體定義資源分配

依據即時運算結果調整手機資源分配，讓手機關閉不必要的應用程式。隨著分析結果的不同，將會有不同的配置規則。將對應的資源配置方法於手機端執行，由於是非人為控制，所以能更正確更快速地調整與判斷，隨時依據不同的條件做出最符合需求的資源配置，能夠大幅提升資源配置的正確度，減少人為判斷錯誤

與手動調整造成的不便，進而提升手機資源配置上的管理與避免資源的閒置或浪費，增加使用手機的便利性，完成以軟體定義之方式配置手機資源，優化手機效能配置。

由於使用者習慣已完成分析，本研究於手機端的管理程式會定時向資料庫查詢手機效能的配置規則，並依據這些規則控制手機前景、背景之應用程式以及後臺服務對手機的記憶體資源進行調整(如圖 3-13)。

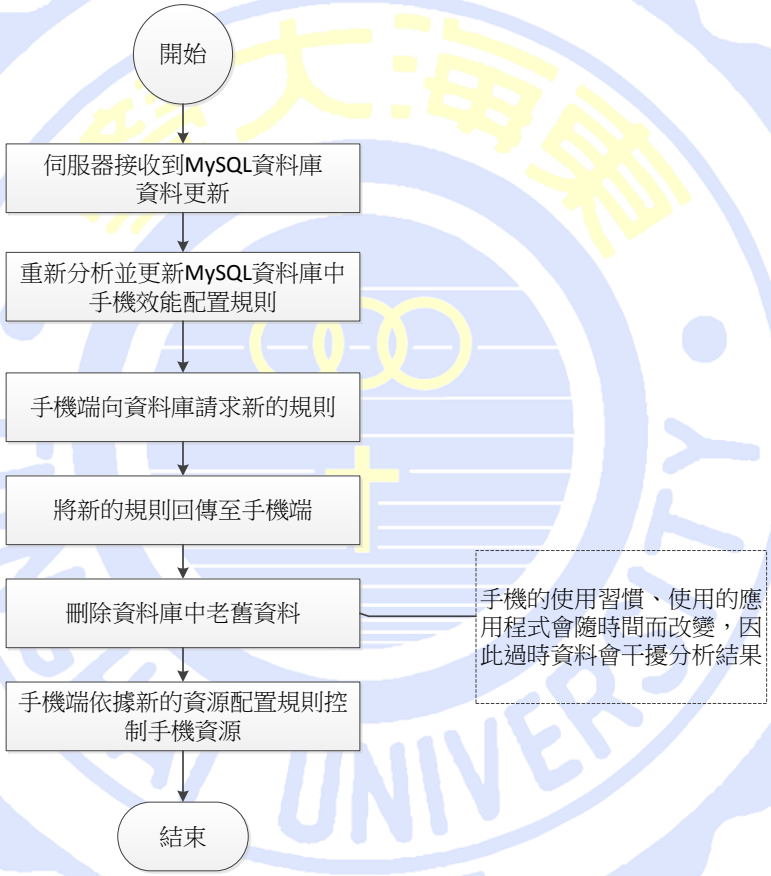


圖 3-13 手機效能配置流程圖

此階段由於需要配置手機資源，為了快速測試選擇使用夜神安卓模擬器作為測試工具減少下載 APK 等成本，具有同類模擬器中最快的運行速度和最穩定的性能。除了模擬 Android 系統運作之外，還可設定是否啟用 Root，性能部分還可設定低、中、高、甚至自定義 CPU 核心數量與記憶體，就看電腦的硬體配置來自行調整，因此可以模擬各種設定的手機，具有非常強大的優勢。

在伺服器產出權重表後，本論文設計之應用程式會讓手機端分別在 0 分與 30 分時下載權重表(如圖 3-14、圖 3-15)，週期性的將已產出的權重表下載至手機端，這樣設計是為了避免網路不穩、手機關機或者種種因素導致權重表遺失。如此一來，手機端就盡可能穩定的在每一個時段依照權重對各個應用程式做記憶體的分配。擁有權重表的手機端，隨時依照手機剩餘的記憶體空間，根據權重由小至大依序結束程序、釋放記憶體 (如圖 3-16)。

```
super.run();
mRules=new ArrayList<Rule>();
while (running) {
    isSorted=false;
    mRules.clear();
    Connection con = null;
    try {
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection
                ("jdbc:mysql://192.168.1.49:3306/software_defined_mobile_phone_performance_configuration", "gordon", "itlabitlab");
        try {
            String sql;
            sql = "SELECT * FROM weight WHERE ( (exeDate ='"+serviceCollectAndUpload.mClock.getDate()+"' )AND " +
                    "(startTime BETWEEN '"+serviceCollectAndUpload.mClock.getExeTime()+"' AND '"+serviceCollectAndUpload.mClock.getExeTimeAdd30()
            PreparedStatement prest = con.prepareStatement(sql);System.out.println(sql);
            ResultSet rs = prest.executeQuery();
            while (rs.next()) {
                mRules.add(CreateRule(rs.getInt(2) ,rs.getString(1)));
                System.out.println(rs.getString(1) + "/" + rs.getString(2));
            }
            Collections.sort(mRules, new Comparator<Rule>(){//權重由低至高排序
                @Override
                public int compare(Rule o1, Rule o2) {
                    return o1.getWeight() - o2.getWeight();
                }
            });
            isSorted=true;
            System.out.println("CONNNNNNNNNNNNNNNNNNn");
            //for(int i=0;i<mRules.size();++i){System.out.println(mRules.get(i).getPackageName() + "/" + mRules.get(i).getWeight());}
            prest.close();
            con.close();
            sleep(serviceCollectAndUpload.mClock.getSleep());//到00分或30分時醒來重抓資料
        } catch (SQLException s) {
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
```

圖 3-14 下載權重資料表程式碼（局部）

```
public int getSleep(){
    int SLEEP=0;

    minute=this.nowCalendar.get(Calendar.MINUTE);
    SLEEP=60-minute;

    if(minute>30){
        SLEEP-=30;
    }
    return SLEEP*60000;
}
```

圖 3-15 間隔 30 分鐘的時間校準程式碼（局部）

```
new Thread(){//偵測記憶體 清空應用程式
    @Override
    public void run() {
        super.run();
        while ( myManager.getAvailMem(myContext) < 50 ){//記憶體少於500MB
            while (!isSorted){//還在排序
                //等待
            }
            myManager.KillUselessApp(myContext);//刪除不用的應用程式
            while( myManager.getAvailMem(myContext) < 50 && mRules.size()>0){//記憶體還是少於500MB
                myManager.Killfunction(mRules.get(0).getPackageName());
                mRules.remove(mRules.get(0));
            }
        }
    }
}.start();
```

圖 3-16 記憶體配置範例程式碼（局部）

圖 3-17 應用程式運行監視介面

在專案裡，將先前的顯示介面再細分出：正在運行的應用程式、安裝在 SDcard 的應用程式、第三方應用程式、系統程序以及所有應用程式，由模擬器與手機安裝本研究設計之應用程式，透過操作模擬器模擬使用者使用手機的情形讓模擬器做到更細節的操作，也讓使用者能自由關閉一些不需要的應用程式或服務，

版面顯示的是所有在執行中的應用程式包含的名稱及所佔的大小空間,版面下方的則是目前手機記憶體使用狀況,目前單純模擬手機的應用程式管理員介面,一切有關上傳資料庫的動作會當作背景服務執行(如圖 3-17)。此介面的設計表現給使用者如應用程式管理員分別顯示所有存在手機上的 APP,並且在介面下方顯示 RAM 使用狀況。在夜神模擬器中,可以調整模擬手機環境的配置亦或是開放超級使用者權限,克服實體手機出廠時被開發商綁定的系統限制,可以在電腦模擬本研究適合的手機環境,且支援多開,方便比較是否導入軟體定義手機效能配置的差異。

第 4 章　實驗與分析

本研究中提出以軟體定義的方式來進行行動裝置的資源配置，說明如何讓使用者操作行動裝置的過程　具備軟體定義特性與優勢。在研究方法的第 3.4 節說明了一套如何將軟體定義導入至行動裝置中的方法，透過蒐集使用數據與分析結果規範出軟體定義規則進行資源配置的依據，使傳統行動裝置轉型為具軟體定義的行動裝置。

為了驗證提出的方法有效性，在本章節中將透過使用案例進行傳統行動裝置與具軟體定義控制的行動裝置，來證實提出的方法可以確實且智慧化的學習、分配運算資源，並確實達到使用者的需求。

4.1 實驗設計簡介

依據國家發展委員會發表的「106 年手機民眾數位機會調查報告」中 106 年調查首次以 APP 類型分類，詢問手機行動上網族是否應用包含社群、資訊、影音等各類應用服務。可複選的前提下，調查顯示，手機行動上網族目前透過手機會從事的活動依序是社群應用（如 Line、臉書，每百人次 98 人次）、資訊應用（如搜尋引擎、網路新聞、e-mail、地圖，每百人次 89 人次）、影音應用(如拍照、修圖、影片剪輯，每百人次 82 人次)、娛樂應用(如遊戲、影片、直播、音樂、唱歌，每百人次 78 人次)、工具服務（雲端硬碟、QR Code 掃描、文書、掃毒，每百人次 68 人次）、政府服務(如郵局、公車、高速公路路況、中央氣象局，每百人次 66 人次)、購物應用(如購物、點餐、租車、訂房、叫車，每百人次 52 人次)、金融應用(如網路銀行、股票交易、付款，每百人次 36 人次)、學習應用(如線上課程、翻譯、幼兒教育，每百人次 32 人次)及健康應用(如醫院相關、健康自主管理，每百人次 25 人次)。

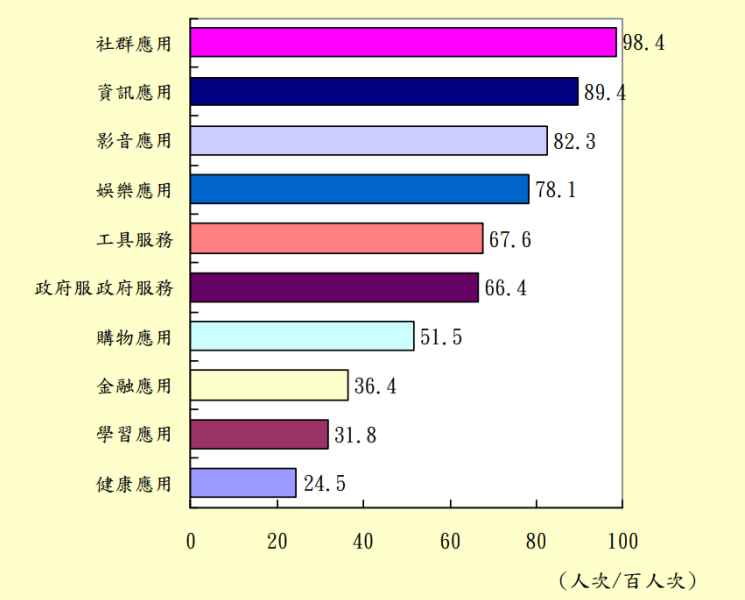


圖 4-1 手機行動上網族手機活動類型(國家發展委員會)

手機功能使用頻率，使用率居冠的應用程式為社群應用，也是十大服務中最頻繁被使用者 87.7%一天使用好幾次，使用率居次的資訊應用 51.6%一天使用多次，第三則是有 48.8%的民眾是一天使用多次娛樂應用。

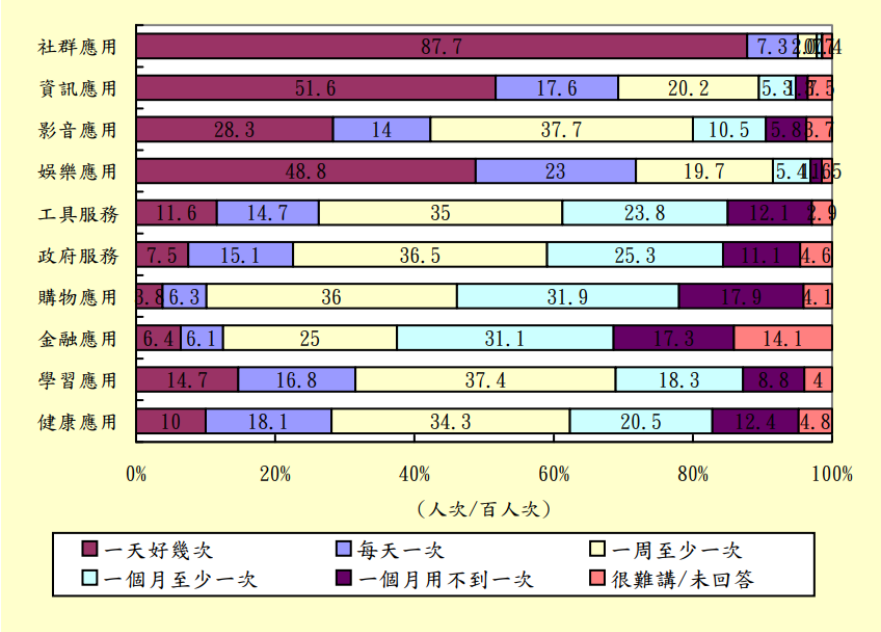


圖 4-2 手機功能使用頻率(國家發展委員會)

因此將以固定的流程驗證本論文提出的方法是否能在應用程式切換的情境中依據使用者的習慣正確的保留下與使用者最相關的程序，有效避免系統關閉錯誤應用的狀況，來減少重複開啟應用程式所需的時間與效能浪費，並改善使用者使用體驗。

4.2 實驗系統架構與環境

為求實驗的一致性，將以本研究開發之管理程式應用於固定、可操控的環境中。本實驗中使用上述提到的夜神模擬器，在可控制硬體效能固定的情況下，進行實驗數據收集。

實驗執行環境方面，實體機使用 Intel(R) Core i7-4799 @3.60GHz 運算處理器，配有 16GB RAM。實驗使用者數據系統架設於 VM 中，並配置四核 CPU 與 4GB RAM。作業系統使用 Ubuntu 12.04 版本，資料庫類型使用關聯式資料庫並採用 MySQL。程式編寫採用 Android studio，JDK 版本為 1.8.0。

4.3 實驗情境與參數

實驗主要目的為驗證軟體定義方法的有效性，因此我們將以原始的非軟體定義行動裝置與轉型後的軟體定義行動裝置功能進行比較，以證明軟體定義系統的服務具有依照環境變化自動動態調整的特性。實驗將以以下案例進行比較。

案例說明:

如何控制於背景執行的程式，主要關鍵於記憶體剩餘量的多寡。軟體定義提供動態調整的特性，透過以往使用的應用程式的狀況，依照使用者當時的狀態設置不同的應用程式優先序，可以更精準的保留下使用者需要的數項應用程式。

進程管理機制最主要的困難在於，如何設定啟動的時機以及如何確定使用噁所需的應用程序不會被優先關閉。使用者在操作行動裝置的過程中都有一定的習慣，若沒有一個良好的機制，在使用的過程中會有反覆分配資源無法有效反映使用著需求的情況。軟體定義前後的控制差異如下：

原始系統控制：記憶體不足時管理機制，會將執行中的近程分為空進程->後台進程->可見進程->服務進程->前台進程，由左至右依次關閉。

軟體定義系統控制：若系統中無使用者過去使用資料則依原始系統所規範進行管理，若系統中已有過去統計資料則在記憶體不足時，依照優先權重清單，先將權重值較低的應用關閉。

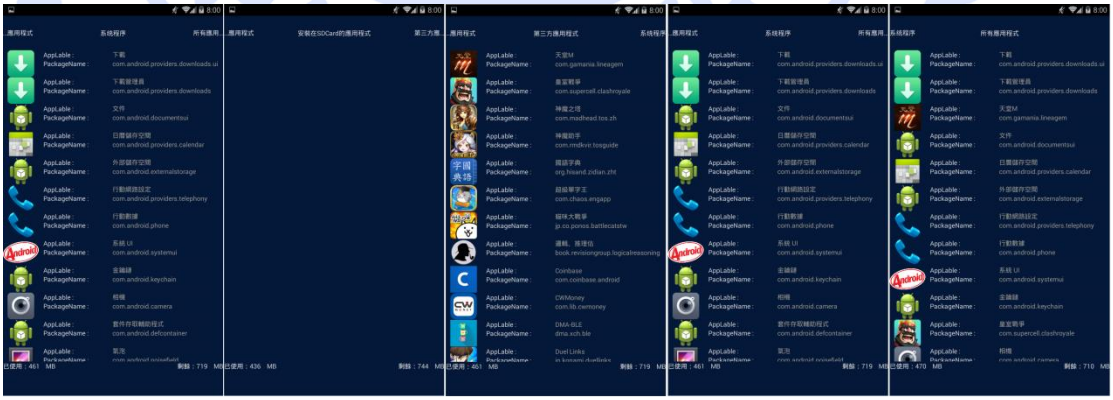
實驗中我們將模擬使用者會切換至不同的應用程式中(在本研究中將以多個遊戲類型的應用程式代表使用者可能頻繁切換於社群應用、資訊應用、影音應用與娛樂應用的情境，加強顯示不同應用在佔用記憶體資源的情況下，系統將會自動的關閉應用程式，並觀察軟體定義系統是否能依照行動裝置當下的記憶體剩餘容量的變化而自動依系統中使用者使用 app 的權重進行控制。而使用本研究控制機制的系統，將於系統判斷記憶體不足時依據使用者的習慣判斷當下可能會進行的應用程式為何，並選擇關閉的應用程式。

4.4 實驗結果

本研究設計一個應用程式，圖 4-3 為安裝前的提示畫面，可以看見其功能有開機時自動啟動、擷取執行中的應用程式與結束背景程序等；本研究將透過這些權限以背景服務(如圖 4-3)與伺服器完成蒐集使用者資訊、分析資訊與手機記憶體配置，實作軟體定義手機效能配置。

圖 4-3 應用程式安裝提示畫面、應用程式背景服務

在應用程式介面中有五個畫面，分別為(a)正在運行的應用程式、(b)安裝在 SDCard 的應用程式、(c)第三方應用程式、(d)系統程序及(e)所有應用程式，彼此間可以透過左右滑動切換（如圖 4-4）在點擊應用程式後會出現詢問是否要清除該應用程式行程的提示（如圖 4-5 左），若確認要清除則會出現超級使用者授權提示並完成清除行程（如圖 4-5 右）。



(a) (b) (c) (d) (e)

圖 4-4 應用程式執行畫面

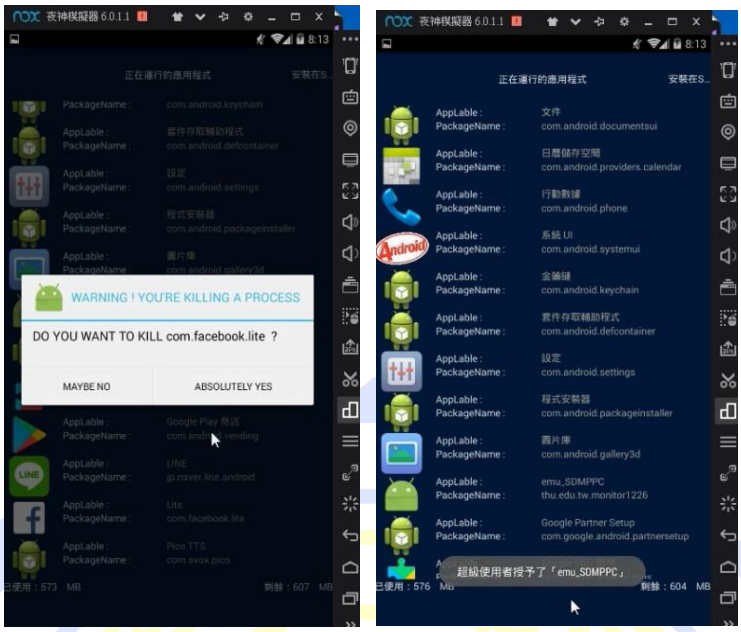


圖 4-5 詢問清除行程提示與獲得超級使用者權限清除行程

圖 4-6 與圖 4-7 顯示原本資料庫中的資料，在應用程式切換（如圖 4-8 與圖 4-9）以及手機進入休眠皆視為完成一次執行（該應用程式的工作告一段落），將相關訊息上傳至雲端資料庫，也因為有新的數據進入，代表有更多資訊可以分析，因此依據最新的數據更新資料庫（如圖 4-10 與圖 4-11），確保配置的規則的正確性與準確度。

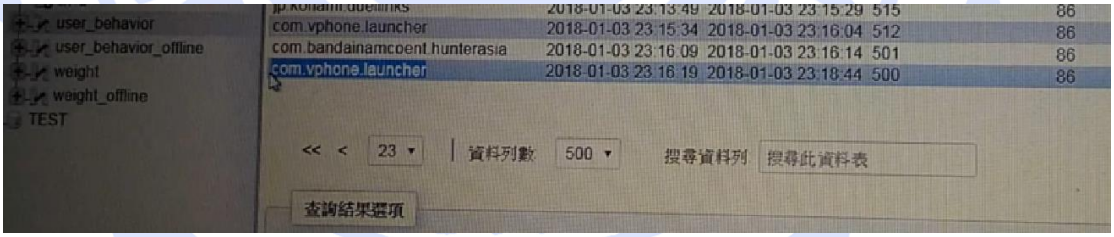


圖 4-6 使用者行為資料表更新前內容

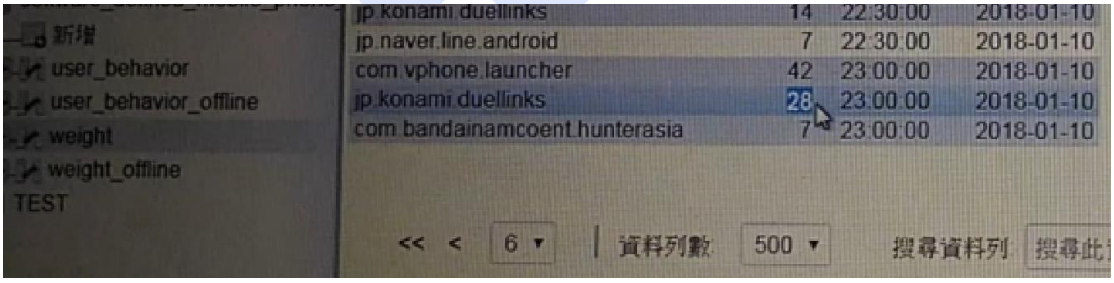


圖 4-7 權重資料表更新前內容

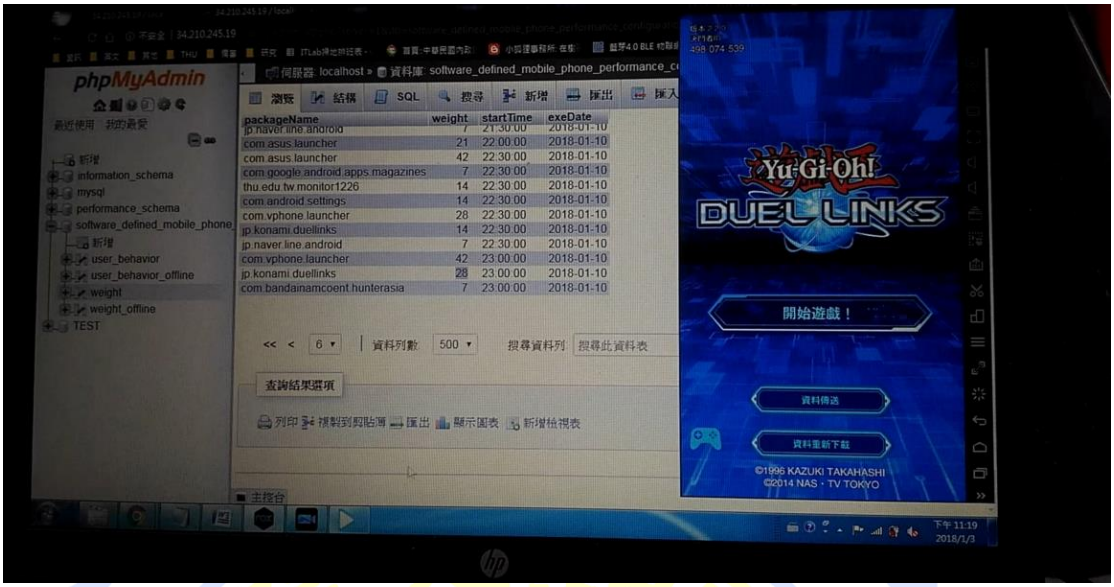


圖 4-8 開啟應用程式

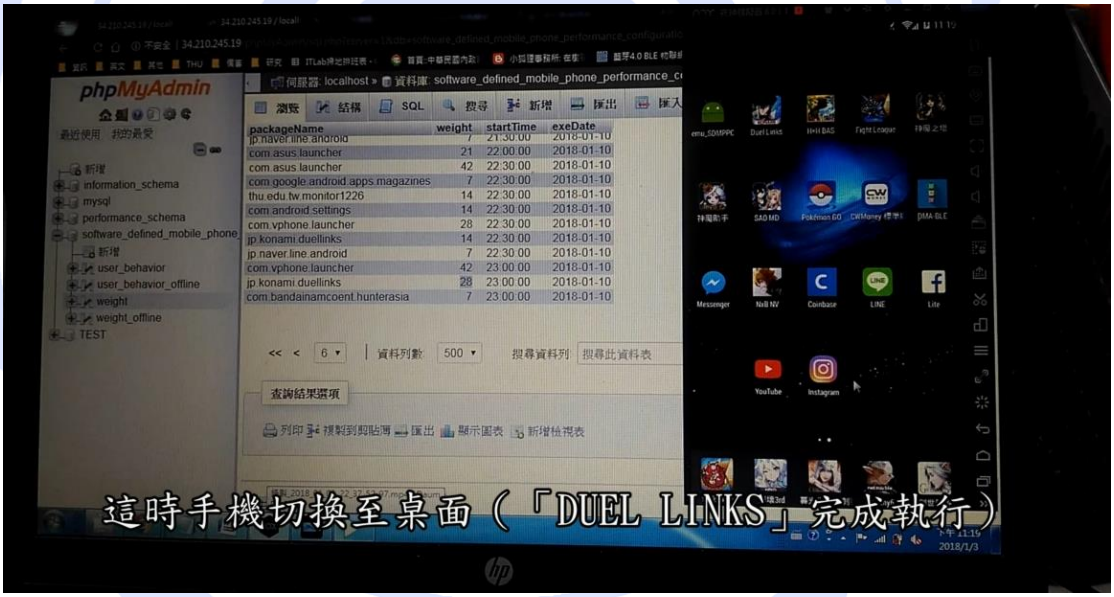


圖 4-9 關閉應用程式

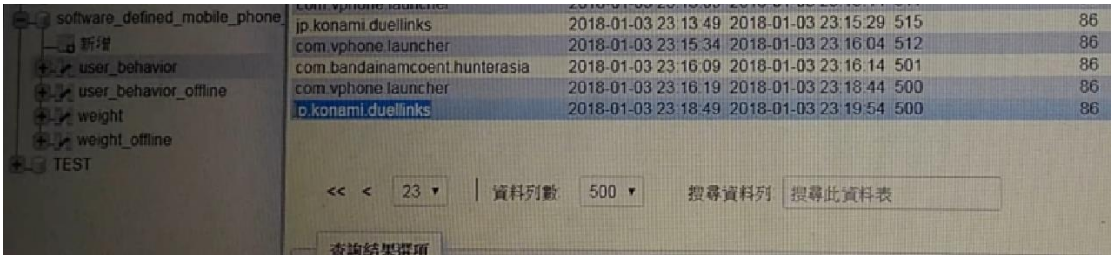


圖 4-10 使用者行為資料表更新後內容

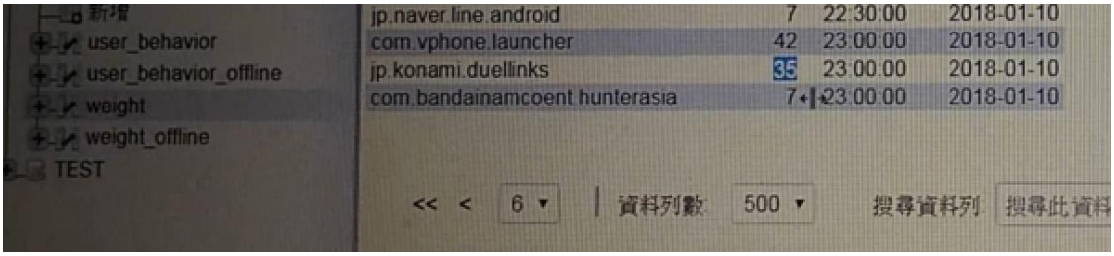


圖 4-11 權重資料表更新後內容

當偵測到記憶體資源不足時，本研究開發的應用程式會先將系統以外程序級別較高但不被需要的程序（重要性極低）刪除，再將不在權重表的背景程序刪除（誤觸或重要性低），最後將權重表中的應用程式按照重要性由低至高依序清除，藉此陸續釋放記憶體，讓記憶體只被花在使用者需要的應用程式上(如圖 4-12)，確保不被需要的應用程式不會佔據記憶體（如圖 4-13）。

packageName	weight	startTime	exeDate
com.instagram.android	9	01:30:00	2018-01-03
jp.konami.duellinks	8	01:30:00	2018-01-03
com.android.browser	7	00:30:00	2018-01-10
com.facebook.lite	14	00:30:00	2018-01-10
jp.konami.duellinks	14	00:30:00	2018-01-10
com.madhead.tos.zh	98	02:00:00	2018-01-03
com.rmdkvir.tosguide	91	02:00:00	2018-01-03
com.bandainamcoent.hunterasia	14	02:00:00	2018-01-03
com.facebook.orca	10	02:00:00	2018-01-03
com.bandainamcoent.saomdas	7	02:00:00	2018-01-03
dma.xch.ble	1	01:30:00	2018-01-03
org.hisand.zidian.zht	6	02:00:00	2018-01-03
com.instagram.android	9	02:00:00	2018-01-03
jp.konami.duellinks	8	02:00:00	2018-01-03
com.madhead.tos.zh	98	02:30:00	2018-01-03

圖 4-12 模擬使用者需求的權重資料表

圖 4-13 系統配置（左）與本研究配置（右）記憶體之差異

本研究中可以由所定義的規則分析使用者的習慣，針對由軟體定義產出的資料表，可以依照日期做到資源配置的預先規劃，藉由存在資料庫中的應用程式的包名與權重，回傳到應用程式中作為控制記憶體的依據，目的是希望將人為管理與系統管理的干涉縮減，能全由軟體定義做到最佳的配置規劃。將暫時用不到的系統程式所佔用的空間釋放出來，讓手機或模擬器依軟體定義的配置規劃，控制部分系統程式開關，藉此更加提升手機的效能，並反映到使用者體驗上。

使用案例:

由圖 4-12 中可以看出統計過去三周的使用情況使用者於下午兩點所需要的應用程式為神魔之塔與神魔助手，為了模擬情境實驗中先開啟神魔之塔與神魔助手並接著開啟其他遊戲來代表使用者可能會在使用的過程中，開啟其他的應用程

式(實際可能為電話、通訊軟體、社交軟體等)。同時以本研究所開發的應用程式紀錄正在使用中的每項程序、服務以及目前使用了多少記憶體。再開啟了三個應用占用記憶體後，傳統系統配置將較久沒開起的神魔助手關閉以供應給新的應用。而軟體定義配置則是依照前面所述，從使用者不需要的系統服務以及權重表中排序較低的應用進行刪除。

圖 4-14 中詳細敘述操作過程中記憶體以及運作中應用程式的變化，在時間軸 2 之處系統配置的經過配置立即被新的應用程式佔滿維持在高負載的狀況，而導入軟體定義的軟體配置方式則意識到記憶體資源即將不足，進而開始清理不必要之應用程式與系統服務(相片庫、行事曆等)，時間軸 3 之處原始系統配置開始釋出記憶體，卻也在此時將使用者所需要的應用程式(此案例為神魔助手)關閉，而軟體定義配置方式則因為先進行過管理，依權重表進行記憶體配置，讓使用者需要的應用程式可以保存於記憶體中而不必重新載入。

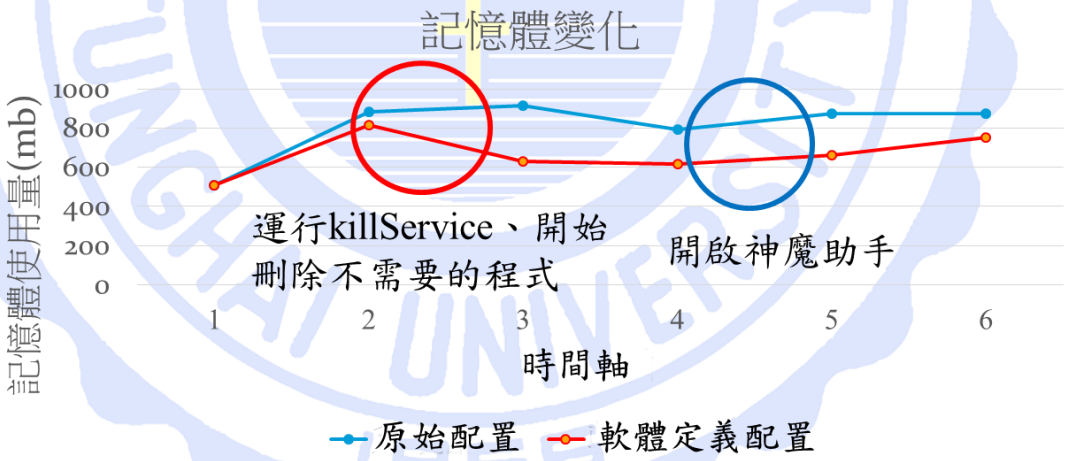


圖 4-14 系統配置與軟體配置之記憶體使用量折線圖

軟體定義配置還有一好處，在需要釋放記憶體時也會關閉使用者當下不需要的服務如行事曆、天氣、相片庫等，再釋放記憶體的同時也降低 CPU 需要隨時分出時間給相關應用的情況達到省電的效果將資源留給使用者。

第 5 章　結論與未來展望

5.1 結論與研究貢獻

現今資訊化的時代之下，使用者不斷的在追求更快好的使用體驗，資訊服務必須具備良好的服務品質，且能提供智慧的、準確的、高適應性與有效率等特性以滿足使用者的需求。在行動寬頻與行動裝置普及的情況下，運算資源提供更快速的服務與體驗，但大量的資源並不能保證提供的服務是正確的，且現存的系統控制時運算邏輯與規則皆以固定，不能依照運算環境狀況與使用者的習慣，彈性且聰明的反應調整。

軟體定義技術的實現，能使硬體基礎設施能自動觀察情況改變，並正確聰明的調整系統的運作方式，進而使服務更加符合使用者的需求。因此如何將軟體定義技術應用於更廣泛的資訊系統領域，使傳統的資訊服務具備軟體定義智慧化的特性成為重要的議題。其中的挑戰包含研究可應於一般資訊系統的軟體定義方法與導入軟體定義至環境使用中。

本研究以軟體定義網路技術(Software Defined network)為基礎，比較並分析非軟體定義與軟體定義的系統，找出具備軟體定義特性的系統結構，進而歸納出將軟體定義技術應用在各種系統抽象層的方法，將軟體定義技術的應用擴展至更廣泛的資訊服務系統。研究中提出軟體定義系統的運作原理、必備的核心元素、執行流程、系統架構與系統轉換方法。透過我們提出的方法，配合物件導向的概念，遷移這些傳統的非軟體定義系統至軟體定義系統將變得可行，實驗中以一個非軟體定義行動裝置系統為對象，結合本研究提出的軟體定義模組，結果表明系統具備了依環境自動動態調整功能，證明了方法的可行性與有效性。以學生使用遊戲與攻略兩項應用程式作為驗證，也可以帶入到一般上班族對於收信、通訊軟

體等應用程式於特定時段有大量使用需求的情形，可以看出本研究應用能涵蓋的使用者不單只限於遊戲與休閒娛樂而是可以應用於廣泛通用的用途。

此本文的成果將能依據不同的使用者的使用習慣遊戲統大幅改善原始作業系統的資源管理方式來保證符合使用者需要。透過將軟體定義概念應用於行動裝置中，替傳統的作業系統帶來智慧化、個人化的運算資源管理，未來若能普遍應用至行動裝置中，使用者將可以不必再購買到高規格的手機即能符合其使用上的需求。同時本研究將該方法的原理與架構等相關研究，制訂為一套標準的模型，提供後續研究者與系統開發人員作為研究智慧化系統控制的基礎。

5.2 未來研究方向

軟體定義技術應用於硬體設施以外的資訊系統是一系列等待研究的議題，並且牽涉眾多資訊領域的應用，因此要制訂完整的軟體定義系統方法理論還需要各方面的專家學者共同努力。 開發流程的角度來分別說明，並期望透過這些後續研究能完成一套適用於軟體定義系統的工程方法。

在規範階段(Software Specification)，本研究提出了軟體定義需求雛形，但還需要更多的案例來驗證與改進，以制訂出正確且完整的需求表。在需求與設計文件中，需求的描述方式目前最通用的統一塑模語言(Unified Modeling Language)並無法完整精確的描述軟體定義系統智慧化的特徵(例如系統要多聰明，聰明程度量化方法)，因此需要制訂一套適用於軟體定義系統的規範語言(Software Defined Definition Language)或者建立相關的模型化(Modeling)方法。

在設計與實作中(Design and Implementation)，本研究提出了一種高適應性的架構來實現軟體定義的系統，但軟體定義系統的實現還有更多的可能方法可以進行研究，例如在作業系統(OS Level)與應用服務(Application Level)之間建立軟體定義轉換平台。除了開發更多建立或轉換現有系統至軟體定義系統的架構之外，

軟體定義系統設計樣板(Design Pattern)的製作也是重要的工作之一，它能作為各種同類型系統開發的基礎依據。軟體定義系統的轉換工具、相關支援套件與完整框架(Framework)，甚至重整工程(Reengineering)等實現(Implementation)方面的工作也有許多研究可以進行。

驗證(Validation)與演化(Evolution)，軟體定義系統其運作結果能否滿足使用者的需求，受到其智慧化程度與分析演算法極大影響，不正確的智慧化機制甚至會降低系統的服務品質，因此建立相關的驗證機制來對建置完成的軟體定義系統進行驗證是非常重要工作。在資訊快速發展的時代，現有系統的演化速度越來越快，軟體定義系統如何配合傳統系統的改變進行演化也需要進行研究。

本研究提出的軟體定義概念，理論上適用於各種抽象軟體結構層(Application Level、Platform Level、OS Level、Infrastructure Level)，這代表著每一種抽象軟體結構層皆需要制訂一套適用的軟體開發流程。目前資訊領域正朝向軟體定義一切(Software Defined Everything)的時代發展，這也代表目前在 Infrastructure Level 上的軟體定義技術正不斷的進行研究與發展。透過將軟體定義概念應用於行動裝置中，替傳統的行動裝置作業系統帶來智慧化、個人化的運算資源管理，找出不同使用者的習慣做有效的資源配置，未來若能普遍應用至行動裝置中，使用者將可以在操作上獲得更好的使用流程與體驗。

軟體定義概念正開始發展，因此近期研究主要以設計與開發為主，可預期的是隨著軟體定義的發展，系統必將隨著演化而更加龐大與複雜，因此對於軟體定義系統本身的資源消耗、複雜結構的改善等相關問題將成為未來的挑戰。

參考文獻

[1] 秦庭祥,"如何快速關閉運作中的 App？其實比你想像更簡單！", https://axiang.cc/archives/17095 ,2017

[2] 由軟體定義的未來 IT：http://www.netadmin.com.tw/article_content.aspx?sn=1408280002, 2017

[3] MyungKi Shin, "Network Virtualization and Service Awareness Properties of FNs in ITU-T", *Workshop on Future Networks Standardization*, JUN 2012.

[4] "軟體定義網路(SDN)", *XIN guard*, http://www.xinguard.com/content.aspx?id=34. ,2017

[5] WenDong Wang, QingLei Qi, XiangYang Gong, YanNan Hu, and XiRong Que, "Autonomic QoS management mechanism in Software Defined Network", *Communications*, vol. 11, pp. 13-23, JUL 2014.

[6] "Software-Defined Networking", *NEC*, http://tw.nec.com/zh_TW/products/itpf/SDN/index.html. ,2017

[7] G. Kandiraju, H. Franke, M.D. Williams, M. Steinder, and S.M. Black, "Software defined infrastructures", *IBM Journal of Research and Development*, vol. 58, pp. 1-13, May 2014.

[8] Specifications, *Open Networking Foundation* https://www.opennetworking.org/about/onf-overview. ,2017

[9] Diego Kreutz, Fernando M.V. Ramos, Paulo E. Verissimo, Christian E. Rothenberg, Siamak Azodolmolky, and Steve Uhlig, "Software-Defined Networking: A Comprehensive Survey", *Proceedings of the IEEE*, vol. 103, pp. 14-76, JAN 2015.

[10] 許鴻基,"SDN 網路技術及應用服務發展趨勢", *工研院資通所通訊產業發展小組報告*, http://www.sharecourse.net/sharecourse/upload/course/80/d2acdae4d33620457924c11aa599edc3.pdf. , 2017

[11] 陳學智, "Software Defined Future of IT", *NetAdmin*, www.netadmin.com.tw/article_content.aspx?sn=1408280002, SEP 2014.

[12] 梁華松, "Android 內存管理機制詳解" https://blog.csdn.net/chaihuasong/article/details/8289367, 2018

[13] "Rational Unified Process", *Rational Software White Paper*, https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf. ,2017

[14] ShupPing Liu and Ling Pang, "The Research of V Model in Testing Embedded Software", *ICCSIT '08. International Conference on Computer Science and Information Technology*, pp. 463-466, AUG 2008.

[15] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut, "Quality of service for workflows and web service processes", *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, pp. 281-308, 2004.

[16] Haresh Luthria, Fethi Rabhi, and Michael Briers “Investigating the Potential of Service Oriented Architectures to Realize Dynamic Capabilities” IEEE Asia-Pacific Services Computing Conference. 2007

[17] Jiehan Zhou, Daniel Pakkala1, Juho Perälä, Eila Niemelä, and Jukka Riekki, Mika Ylianttila, “Dependency-aware Service Oriented Architecture and Service Composition” IEEE International Conference on Web Services (ICWS 2007).

[18] Heather J. Goldsby, Sascha Konrad, and Betty H.C. Cheng, "Goal-Oriented Patterns for UML-Based Modeling of Embedded Systems Requirements", *High Assurance Systems Engineering Symposium 10th*, pp. 7-14, NOV 2007.

[19] YongYi Fanjiang, NienLin Hsueh, and Jonathan Lee, "Towards a Pattern-based Model Transformation Approach for Design Quality Enhancement", *Journal of Software Enginnering Studies*, vol. 2, no. 3, pp. 120-133, 2007.

[20] Amen B. Hadj Ali, Faiyzu Boufares, and Abdekazuz Abdellatif, "Checking Constraints Consistency in UML class diagrams", *Information and Communication Technologies*, vol. 2, pp. 3599-3604, 2006.

[21] Grady Booch, "Object-oriented analysis and design with applications (2nd ed.)", *Benjamin-Cummings Publishing, ISBN:0-8053-5340-2*, 1994.

[22] Richard Soley and OMG Staff Strategy Group, "Model Driven Architecture", *Object Management Group White Paper Draft 3.2*, NOV 2000.

[23] S. Vinoski, "Chain of responsibility", *Internet Computing*, vol. 6, pp. 80-83, DEC 2002.

[24] ChengChung Chu, YenHua Huang, ChihHung Chang, and David Hsu, "Applying Software-defined Methodologies to Software Development", *Third International Conference on Trustworthy Systems and Their Applications*, 2016.

[25] 黃彥華,"軟體定義方法論對軟體開發與服務之影響", 東海大學資訊工程學系, 2017