私立東海大學資訊工程學系研究所

碩士論文

指導教授：朱正忠教授

利用 XML 與雙向機制增加軟體人工製品維護性

Using XML and Two-way Mechanism to Increase Software Artifact Maintainability

研究生：吳巧惠

中 華 民 國 100 年 6 月 27 日

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 吳 巧 惠 所提之論文

利用 XML 與雙向機制增加軟體人工製品維護性

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 王豐堅 簽章

委 員 焦惠津

熊博安

吳秋峰

指 導 教 授 朱正忠 簽章

中華民國 100 年 6 月 27 日

摘要

在系統開發生命週期的實作與測試階段中，當系統開發完成後，經由不同的測試階段或使用者需求不停的變更，開發人員需不斷的修改程式與更新部署程式至其它測試主機或開發主機，而為了避免開發人員反反覆覆的更新部署程式，或在更新程式的過程中更新程式被遺漏而導致系統錯誤等情況發生。故在本研究中，我們提出以物件導向程式為基礎所設計的軟體人工製品-SyncDeploymentSystem(簡稱 SDS)，以求來改善目前的現況。SDS 主要是利用物件導向程式設計與 XML 資料記錄的機制，節省開發人員部署程式至各主機上所需要往返的更新部署時間及減少部署的複雜度，更提高部署程式及文件的準確度、提升開發人員對軟體的生產力，並增加系統開發之維護性，以解決不完整的更新部署程式及文件所造成的系統問題。

關鍵字：軟體人工製品，部署機制，雙向機制，XML

Abstract

In the practicing and testing phase of SDLC(System Development Life Cycle), after the system development has been completed, developers have to keep modifying programs and updating deployment programs to other testing servers or development servers. In this research, to avoid system errors caused by developers repeatedly update the deployment program or the deployment program being missed during the updating procedure, we proposed an OOP (Object-oriented programming) based software artifact – SDS(SyncDeploymentSystem). SDS mainly takes advantage of data access mechanism of OOP and XML to save the time consumed in routine deployment program update procedure, as well as reducing the complexity of deployment. This can also increase the accuracy in deploying programs and documents, raise developers' software production power, increase maintainability of developed system and therefore it is predictable that this can be a fine solution to system problems caused by incomplete update of deployment programs or documents.

Keywords: Software artifacts, deployment mechanism, two-way mechanism, XML.

致謝

本論文之所以能夠順利完成，最感謝的就是指導教授朱正忠博士、張志宏老師和陳瑞男老師的指導，一直不斷的協助修正我論文的方向與提供論文修改建議，讓我受益良多。在口試過程中，也要感謝口試委員熊博安教授、焦惠津教授、王豐堅教授、吳毅成教授等人，對本論文內容不足之處一一給予建議指導。

並感謝資訊技術實驗室的同學、學弟妹們的協助幫助，以及學長姐、同學們的幫助，讓我得以順利完成學位考試。

最後感謝家人與朋友的支持，時時提醒與鼓勵我，讓我繼續努力，並順利完成碩士班的學業。

吳巧惠　謹誌

2011 年 7 月

目錄

圖目錄

表目錄

第一章　緒論

1.1 前言

軟體發展的生命週期就是將邏輯性的系統概念，發展規劃成可以實作的系統設計文件，並以設計與架構程式碼的方式去實現此系統概念，當系統完成時，將其產出交付部署、測試、運行，最後進入運行維護的發展步驟。

1.2 研究動機

軟體開發的生命週期可分成六個階段：需求分析階段(Requirements analysis Phase)，規格階段(Specification Phase)，設計階段(Design Phase)，建置階段(Implementation Phase)，測試階段(Testing Phase)，維護階段(Maintenances Phase)，其定義如下：

(1) 需求分析階段(Requirements analysis Phase)：需求分析指的是在建立一個新的或改變一個現存的系統或產品時，確定新系統的目的、範圍、定義和功能時所要做的所有工作。需求分析是軟體工程中的一個關鍵過程。在這個過程中，系統分析員和軟體工程師確定顧客的需要。只有在確定了這些需要後他們才能夠分析和尋求新系統的解決方法。

(2) 規格階段(Specification Phase)：客戶同意在需求階段的了解後，規格小組(Specification team)將畫出規格文件(Specification document)。

(3) 設計階段(Design Phase)：設計過程將需求轉變為軟體的表示，以便在程式碼產生前了解其品質。

(4) 建置階段(Implementation Phase)：設計必須被轉變為機器可讀取的形式。這項工作即進行程式碼產生的步驟。

(5) 測試階段(Testing Phase)：程式碼產生後，即開始進行程式測試。設計必須被轉變為機器可讀取的形式。這項工作即進行程式碼產生的步驟。一旦程式碼產生後，即開始進行程式測試。測試過程將焦點放在軟體內部的邏輯、確保所有敘述都被測試，以及外部的機能上。進行測試以找出錯誤，並確保輸入將會產生和所需結果相同的正確結果。

(6) 維護階段(Maintenances Phase)：軟體如何確保順利的上線，或軟體在交給客戶後，毫無疑問的一定會有改變的需求。改變會因為遇到錯誤而發生，因為軟體必須能適應外部環境改變的需求，例如因為新的作業系統或週邊裝置改變所產生需求，或因為客戶要求功能或效能的增進。

當系統開發到一個程度後，開發人員經常會遇到的問題是如何快速的部署並更新系統開發完成後的程式碼及文件，當其當下的開發環境是需要部署至多台主機以提供測試、執行的情況下時，開發人員如

何用最快速、最方便的方法將程式碼或文件更新至每一台主機上，或當系統分析人員完成需求分析規劃書，以及測試人員完成之測試報告等文件，又該如何自動化分送給各開發人員或需求者，其文件的備份又該如何進行管控，程式碼及文件的備份管理機制，如何用更有效率且快速的進行程式碼及文件部署與備份，此為本論文的研究動機。

1.3 研究目的

故本研究中，我們提出以物件導向程式設計之軟體人工製品-SyncDeploymentSystem(簡稱 SDS)，用來解決上述的問題，SDS 是利用物件導向程式設計 IDE 操作介面與 XML 存取機制，以一對多或多對一的雙向部署機制，使用者將程式碼及文件部署到指定的路徑目錄下，本實作可指定不同的機器設備，如：開發機、測試機、正式機或其它個人電腦等設備，以進行部署管控。

1.4 論文架構與研究

本研究論文撰寫的架構共分成六章，以下分別對各章節做簡述。第一章，緒論，包括描述本研究的背景及動機、研究目的及研究架構與流程。

第二章，文獻回顧，將會介紹本論文所實作的系統利用的各軟體與技術，包括物件導向技術、XML、版本控制等。

第三章，系統架構與設計，針對本論文之基礎概念與及理論做概念性的說明。

第四章，系統分析，描述本論文如何應用前章所述作出一合理系統的流程。

第五章，實作展示，將會展示所實作的系統，包含 IDE 操作介面與程式碼的介紹。

第六章，實作評估，展示實作系統之實驗結果與優劣分析。

第七章，結論與未來展望，總結論文成果，並提出未來可後續研究的方向。

第二章　文獻回顧

2.1 物件導向技術

物件導向程式語言是目前於業界、學界都主推的核心基礎技術，維基百科對於物件導向程式設計的定義如下：物件導向程式設計（英語：Object Oriented Programming，縮寫：OOP），指一種程式設計典範，同時也是一種程式開發的方法論。它將物件作為程式的基本單元，將程式和資料封裝其中，以提高軟體的重用性、靈活性和擴充功能性。當我們提到物件導向的時候，它不僅指一種程式設計方法。它更多意義上是一種程式開發方式。在這一方面，我們必須了解更多關於物件導向系統分析和物件導向設計（Object Oriented Design，簡稱 OOD）方面的知識[16]。

何謂物件導向技術/方法[9]：

(1) 將真實世界視為由各種物件所構成；以人類直覺的方式將問題模擬成各種物件。

(2) 真實世界的運作是由各個物件之間的互動來產生。

(3) 將真實世界的問題對應(抽象化)成分析設計的模型，再轉換成相對應的程式碼。

(4) 將實際生活中所遇見的物件應用於軟體設計裡，進而演化而成的一種軟體發展的模式。

物件導向技術有三種重要觀念：物件、訊息和類別。簡單描述物件導向的三種重要觀念：

(1) 物件(Object)：提供資料和處理資料程序（Java 語言是方法）的封裝。

(2) 訊息(Message)：在物件之間的溝通方式，可以建立互動和支援多型。

(3) 類別(Class)：物件的分類，可以實作類別架構的繼承。

物件是物件導向技術的關鍵，以程式的角度來說，它是電腦用來模擬現實生活的東西或事件，也是組成整個程式的元件。物件是資料與相關處理資料的程序和函數結合在一起的組合體，資料就是變數，程序和函數在 Java 語言稱為方法，如圖 1 所示：

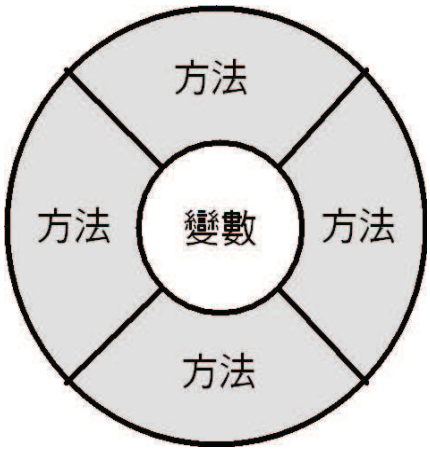

圖 1 封裝

物件的方法是對外的使用介面，資料和相關方法的實作程式碼都包裹隱藏起來，稱為「封裝」(Encapsulation)。對於程式設計者來說，並不用考慮物件內部方法的程式碼是如何撰寫，只需要知道這個物件提供什麼介面和如何使用它即可。

物件對應到現實生活中的實體和事件，有下列三種特性：

(1) 狀態（State)：物件所有屬性（Attributes）目前的狀態值，屬性是用來儲存物件的狀態，可以是布林值變數，也可能是另一個物件。例如：車子的車型、排氣量、色彩和自排或手排等屬性。

(2) 行為（Behavior)：行為是物件可見部分提供的服務，也就是塑模所抽象化的操作，可以作什麼事。Java 語言是使用方法來實作行為。例如：車子可以發動、停車、加速和換擋等。

(3) 識別字（Identity)：識別字是用來識別不同的物件，每一個物件都擁有獨一無二的識別字，Java 語言是使用物件參考（Reference）作為物件的識別字。

本論文的系統實作，其軟體開發部份使用 Microsoft Visual Studio 2005 進行 Windows 應用程式的開發，而維基百科對於 Microsoft Visual Studio（簡稱 VS）的定義如下：是美國微軟公司的開發工具套件系列產品。VS 是一個基本完整的開發工具集，它包括了整個軟體生命周期中所需要的大部分工具，如 UML 工具、代碼管控工具、整合式開發

環境等等。所寫的目的碼適用於微軟支援的所有平台，包括 Microsoft Windows、Windows Mobile、Windows CE、.NET Framework、.NET Compact Framework 和 Microsoft Silverlight。而 Visual Studio .NET 是用於快速生成企業級 ASP.NET Web 應用程式和高性能桌面應用程式的工具。在 Visual Studio 的開發工具中，其包含了能用多語言來開發元件的工具（如 Visual C#、Visual J#、Visual Basic 和 Visual C++），以及許多用於簡化基於小組的解決方案的設計、開發和部署的其他技術，而本研究利用 Visual Studio .Net C# 工具進行系統程式的開發。

2.2 XML 機制

維基百科對於 XML 的定義如下：可延伸標示語言（eXtensible Markup Language，簡稱 XML），又稱可延伸標記語言，是一種置標語言。置標指電腦所能理解的資訊符號，透過此種標記，電腦之間可以處理包含各種資訊的文章等。如何定義這些標記，既可以選擇國際通用的標記語言，比如 HTML，也可以使用像 XML 這樣由相關人士自由決定的標記語言，這就是語言的可延伸性。XML 是從標準通用置標語言（SGML）中簡化修改出來的。它主要用到的有可延伸標示語言、可延伸樣式語言（XSL）、XBRL 和 XPath 等。

XML 是一種[中介標籤語言](meta-markup language)，可提供描述

結構化資料的格式，這將有助於文件內容的宣告，且由於其為純文字檔所組成，固能符合跨平台的搜尋作業，XML 也是新一代網路資料呈現與運作的關鍵技術，XML 被設計成使用簡單、具有彈性與開放的語言，以便讓不同的使用者製作各種 XML 文件[15]。XML 的優點有以下幾點[7]：

一、可擴充性：使用者可定義新的標籤。

二、結構化：可以描述複雜的階層式資料。

三、驗証：可檢查資料結構的正確性。

四、可自我描述。

五、簡單易讀的特性。

六、不受限電腦平台，程式設計。

XML 的成功是肇因於下列幾項簡單的因素[7]：

(1) 當電腦都連接在網路上時，這些電腦就經常需要互傳訊息以便完成工作。

(2) 這些資料必須轉換為一長串的形式，稱為串列化格式(serialized)，因為網路上的資料是以串列的方式傳輸的。

(3) 接收端的電腦必須要能解讀這些串列式資料的格式。

(4) 使用一個標準的格式，可以使系統發展者不用費心設計資料交換的格式，而把精力專注於更重要的問題上，例如：什麼是這個系統所

需的資料結構、資料的類別及其意義、系統如何運用這些資料。

而 XML 能夠成功的原因是，它是唯一能將資料串列化的標準格式。另一個嚴謹的解決方案是 ISO 的 ASN.1(Abstract Syntax Notation 1)；但是 ASN.1 是二進位的格式；而 XML 是文字形態的格式，因此可以使用每部電腦上都有的文字編輯程式建立及修改 XML 文件。

在本研究中，把 XML 視為一個儲存媒介，將使用者所需的相關資訊儲存在 XML 內。使用者透過專屬的工具介面，對 XML 檔案進行新增、刪除、修改等編輯動作，將其操作過程中所產生的相關資料儲存至 XML 檔案內，當系統欲進行部署的動作時，則由系統透過 XML 檔案做為媒介，來處理所需要的相關環境設定，固 XML 在本研究的實作上極為重要。

2.3 版本控制

版本控制系統提供了一個地方集中存放開發過程中的所有程式檔案及文件，以便達到集中控管的目的。版本控制系統通常由以下幾種元件組成[13]：

(1) 檔案庫(Repository)：

版本控制系統有一個集中存放檔案的地方，這個地方有個正式名稱叫做「檔案庫」。檔案庫裡面儲存了專案檔案的所有歷史版本(包括

目前開發中的版本)，有的版本控制系統是以資料庫的方式儲存，有的是以檔案的方式儲存，不論儲存的方式為何，對使用者來說，最重要的就是要把檔案庫放在一台穩定、安全的機器上，並且還要定期備份。

(2) 主從式架構(Client/Server)：

檔案庫既然是檔案的集中營，那麼一定是放在某台機器上，供所有開發人員存取，其作業方式如圖 2 所示，是一種主從式的架構：

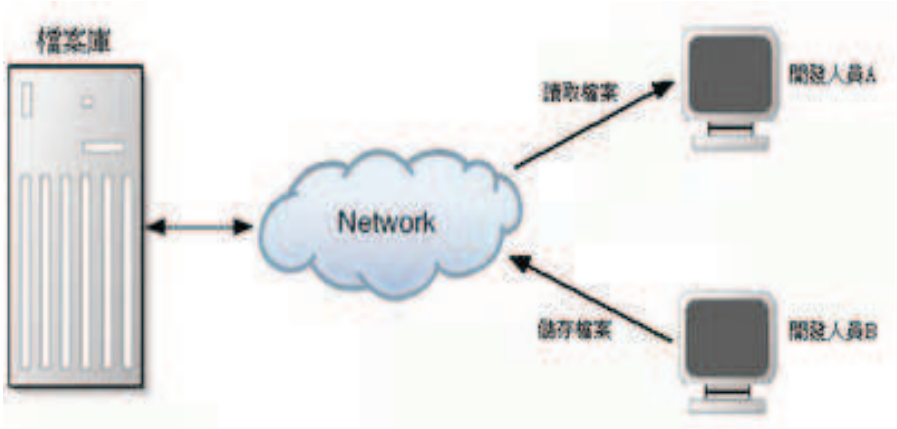

圖 2 主從式的架構

檔案庫所在的機器，必須要安裝版本控制系統，以便提供檔案存取的服務給各個用戶端；圖 2 中的「開發人員 A」和「開發人員 B」即代表用戶端，用戶端機器上必須安裝版本控制系統的用戶端工具，才能存取檔案庫。在連線方式上，用戶端可以透過多種網路協定來存取檔案庫，某些版本控制系統要求一定要隨時與檔案庫保持連線，才能修改檔案內容；某些版本控制系統(例如 Subversion)則採用比較寬鬆的方式，開發人員可以在家用筆記型電腦修改程式，等到回辦公室時

再將檔案同步。

(3) 檔案庫資料：

程式碼要存放在檔案庫中，以便進行版本控制。在開發一個專案的過程當中，需要用來建置開發內容的檔案，可能都要放到檔案庫裡面，例如：建置專案的組態檔或 makefile、測試資料等。

目前版本控制軟體有以下幾種：BitKeeper、CVS(Concurrent Versions System)、Microsoft Visual SourceSafe、Perforce、Rational ClearCase、RCS(GNU Revision Control System)、Serena Dimention、Subversion、SVK、Git、en:Monotone(software)、en:Bazaar(software)、Mercurial、en:SourceGear Vault，而在下一章節中針對 CVS 與 Subversion 二種版本控制軟體，再加以描述與介紹。

2.3.1 CSV

維基百科對於 CVS 的定義如下：CVS（Concurrent Versions System）[2][4][5]代表協作版本系統或者併發版本系統，是一種版本控制系統，方便軟體的開發和使用者協同工作。很多開放原始碼軟體或者自由軟體項目都使用 CVS 作為其程式設計師之間的中心點，以便能夠綜合各程式設計師的改進和更改。這些項目包括：Gnome、KDE、GIMP、Wine 等。CVS 的使用獲 GNU 通用公共許可證授權。這是一個將一組

文件放在層次目錄樹中以保持同步的系統。使用者可以從 CVS 伺服器上更新他們的本地層次樹副本，並將修改的結果或新文件發回；或者刪除舊文件。CVS 基於客戶端/伺服器結構（C/S）的行為使得其可容納多用戶，構成網路也很方便。這一特性使得 CVS 成為位於不同地點的人同時處理數據文件（特別是程序的原始碼）時的首選。CVS 版本控制系統(Concurrent Version System)，本控制是對於所寫的 source code 做有效率的管理工具，軟體開發時都有版本，有時發現新版本會產生某種狀況是舊版本沒有的，或者是誰改了這一段程式碼的等等紀錄，都可以用 CVS 紀錄，取回這些舊資料，所以非常的重要。

現在 CVS 不僅是有效的管理工具，還是非常棒的展示工具，在做 code review 等場合非常好用，還可以減少紙張浪費。CVS 也是很好的 download 用工具，因為 CVS 每次只 download 跟上個版本的差異部分，也可以減少網路頻寬的使用。CVS 不僅如此，還可以管理一個多人共同開發的專案計劃，開放原始碼軟體裡面的專案透過全世界的網路將全世界的工程師一起來發展軟體，CVS 的功勞是最大的，其實很多大公司內部除了商用的 ClearCase 外，都用這個做版本控制。

CVS 是現今 Open Source 成功發展的幕後功臣之一。CVS 解決多人合作開發時程式版本控管的問題，通常會再搭配郵件列表(Mailing List)做為開發團隊溝通的管道。這種組合，使開發團隊不受時間地域

限制，合作伙伴分散全世界，且團隊大小沒有上限，因此 Open Source 才能集合世界各地高手，不斷地薪火相傳、不斷地推出高品質的自由軟體。CVS 基本說明如下所示[12]：

(1) Version(版本)、Revision(修訂版)、Release(發行版)：開發期間的版本稱為 Revision(校訂版)，釋放給客戶使用的版本稱為 Release(發行版)。CVS 的版次編號(revision number)只做為 CVS 內部控管之用，和將來發行的軟體版本(version)無關。也就是說：若某一支程式在 CVS 中版次編號為 1.5，而發行的軟體版本『把它稱為』SFS3.0 版，那麼，這個 1.5 內部版次和 3.0 軟體版本，是完全不相關的。CVS 的版次成長的過程如下：第一次將專案匯入 CVS Server 時，所有檔案的版次編號皆為 1.1.1.1。若修改存入檔案庫之後，版次編號就由 1.2 起跳，之後每存入一次，版次編號就增加 1。可以放心儘情地修改，不同的程式檔之間的版次編號不必一致，也就是說：一個專案中，A 這支程式到了 1.83 版，B 這支程式只到了 1.3 版，並不會影響將來發行軟體的版本，可以利用標記(tag)等方式，來達到進一步控管的功能。

(2) 取出過去的專案版本：CVS 提供二種方式取出過去的專案版本：依時間點取出(by date)、依標記取出(by tag)。專案會一直往前發展，終於有一天，專案管理者評估之後，認為該專案目前已經穩定成熟，可以推出正式版本了。專案管理者通常會給專案的程式碼加上一個標

記，比如：r2002_10_20，然後將專案打包，並為該軟體命名一個發行版本(如 SFS3.0 版)。在正式版推出一段時間之後，可能會有使用者發現軟體在某些地方功能不太正常，因此將問題回報給專案成員。而這段期間，專案仍持續不斷地往前發展，在發展中的這種最新版本，通常不穩定，無法立即可用，實在不適合正式推出。為了修正程式碼中的臭蟲，有必要取出過去的程式碼。此時，當初設定的標記就十分重要了。可以根據標記，取出當時專案中的所有程式碼，以進行修正的動作。至於目前持續發展中的版本，則不會受到任何影響。當然，也可以依時間點來取出當時的專案，一般而言，還是以使用標記的方式居多。當取出過去的專案時，目前的工作版本會暫時變成舊專案，若修改它，是無法把它存入 CVS 檔案庫中。因為它是過去的歷史，CVS 不容許您修改歷史，此時必須在原先的發展路線上，開闢另外一個分支(branch)，所有修正臭蟲的程式碼，全部在這條分支上去進行。

(3) 分支(branch)：CVS 可以提供回到某一穩定版本，由該處去修正程式的錯誤，修正的結果循另一條路線發展，以提供更可靠的程式版本。這就是分支的概念。原來的主要發展版本，就稱之為主幹(HEAD)。

(4) 合併分支及主幹：分支的目的之一是為了修正某些程式的錯誤，開發中的版本，極可能也有這個 bug 存在，因此通常分支修改完之後，會和主幹做合併的動作。假設 r2002_10_20 推出正式版之後，發現有

bug，為了修正臭蟲，於是做了分支 r2002_10_20_branch。以 index.aspx 為例，設若 bug 是在 index.aspx 中，且已被修正，今想把它和主幹合併，以修正主幹的錯誤。

(5) 取出專案，推出(release)軟體版本：當 checkout 出庫存專案時，在工作目錄下每一個子目錄中皆有一個 CVS 資訊目錄，但要推出正式版本的軟體，卻不希望把這些 CVS 目錄也打包進去。CVS 有提供一個動作命令，用來取出不含控制訊息的整份專案。

2.3.2 Subversion

維基百科對於 Subversion 的定義如下：Subversion[5][6]，簡稱 SVN，是一個開放原始碼的版本控制系統，相對於的 RCS、CVS，採用了分支管理系統，它的設計目標就是取代 CVS。網際網路上越來越多的控制服務從 CVS 轉移到 Subversion。

Subversion 是一個自由/開放原始碼軟體版本控制系統，它管理文件和目錄可以超越時間。一組文件存放在中心版本庫，這個版本庫很像一個普通的文件伺服器，只是它可以記錄每一次文件和目錄的修改，這便可以取得資料以前的版本，從而可以檢查所作的更改。從這個方面看，許多人把版本控制系統當作一種「時間機器」。

Subversion 可以通過網路訪問它的版本庫，從而使用戶可以在不同

的電腦上使用。一定程度上可以說，允許用戶在各自的地方修改同一份資料是促進協作。進展可能非常的迅速，並沒有一個所有的改變都會取得效果的通道，由於所有的工作都有歷史版本，不必擔心由於失去某個通道而影響質量，如果存在不正確的改變，只要取消改變。

Subversion 具有以下的特性[14]：

(1) 版本化的目錄：CVS 只能追蹤單個文件的變更歷史，但是 Subversion 實現的「虛擬」版本化文件系統則可以追蹤目錄樹的變更。在 Subversion 中，文件和目錄都是版本化的。

(2) 真實的版本歷史：由於只能追蹤單個文件的變更，CVS 無法支持如文件拷貝和改名這些常見的操作—這些操作改變了目錄的內容。同樣，在 CVS 中，一個目錄下的文件只要名字相同即擁有相同的歷史，即使這些同名文件在歷史上毫無關係。而在 Subversion 中，可以對文件或目錄進行增加、拷貝和改名操作，也解決了同名而無關的文件之間的歷史聯繫問題。

(3) 原子提交：一系列相關的更改，要就全部提交到版本庫，不然的話，一個也不提交。這樣用戶就可以將相關的更改組成一個邏輯整體，防止出現只有部分修改提交到版本庫的情況。

(4) 版本化的 Meta 資料：每一個文件和目錄都有自己的一組屬性-鍵和它們的值。可以根據需要建立並儲存任何鍵/值。和文件本身的內容一

樣，屬性也在版本控制之下。

(5) 可選的網路層：Subversion 在版本庫訪問的實現上具有較高的抽象程度，利於人們實現新的網路訪問機制。Subversion 可以作為一個擴展模塊嵌入到 Apache 之中。這種方式在穩定性和交互性方面有很大的優勢，可以直接使用伺服器的成熟技術—認證、授權和傳輸壓縮等。此外，Subversion 自身也實現了一個輕型的，可獨立運行的伺服器軟體。這個伺服器使用了一個自行定義協議，可以輕鬆的用 SSH 封裝。

(6) 一致的資料操作：Subversion 用一個二進制差異算法描述文件的變化，對於文字(可讀)和二進制(不可讀)文件其操作方式是一致的。這兩種類型的文件壓縮儲存在版本庫中，而差異訊息則在網路上雙向傳遞。

(7) 高效的分支和標籤操作：在 Subversion 中，分支與標籤操作的開銷與工程的大小無關。Subversion 的分支和標籤操作用只是一種類似於硬鏈接的機制拷貝整個工程。因而這些操作通常只會花費很少且相對固定的時間。

(8) 可修改性：Subversion 沒有歷史負擔，它以一系列優質的共享 C 程序庫的方式實現，具有定義良好的 API。這使得 Subversion 非常容易維護，和其它語言的互操作性很強。

Subversion 是現代 CVS(Concurrent Versioning System)功能延伸軟體，是一個完整的檔案版本控管伺服器。它包含了 CVS 上提供的大部

份功能，如 commit、merge、branch、tag 等。解決了一些 CVS 上的缺點，如沒法對目錄作版本控制、使用網頁伺服器作外部連接、縮短處理 branch 所需的時間等。Subversion 比 CVS、VSS 好用，和各種 IDE 整合度也高，同時配合上 TRAC 這類軟體，更是能發揮戰力，每個 Check-in，都會被記錄在 Change-set 裡頭，可以任意比較任何版次的差異所在。

Subversion 是一個典型的主從式架構設計，如圖 3 所示，使用者安裝 Client 端應用程式，可透過指令模式或圖形介面模式進行資料的 Check In & Check Out 等動作，經由內建 SVN Server 或 Apache 網頁伺服器的 DAV 模組提供網路連結服務。最後進入 SVN 的檔案庫系統中，檔案庫本身預設提供 Berkeley DB 或 FSFS 檔案系統用以儲存資料，亦可與其他資料庫系統進行整合。

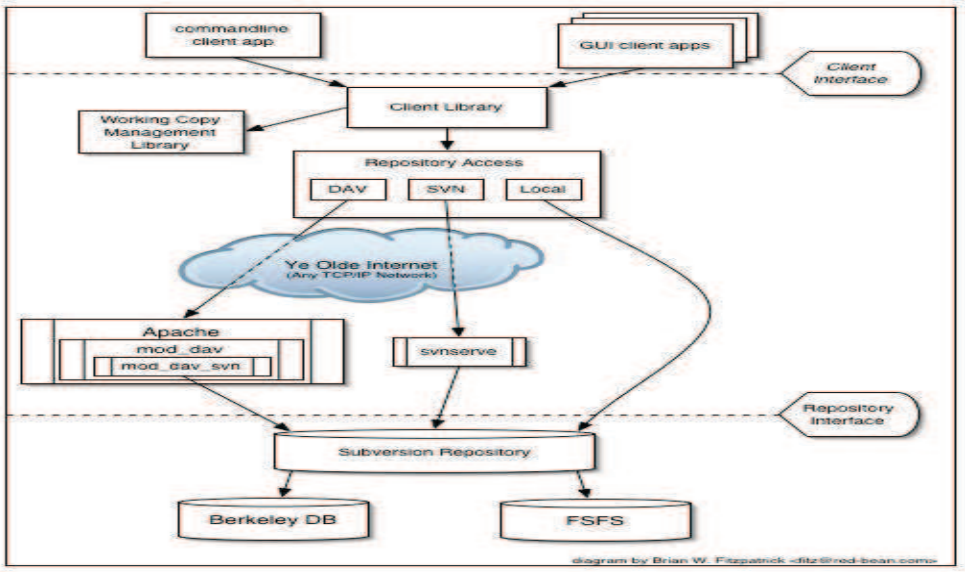

圖 3 Subversion 架構圖

第三章　系統架構與設計

3.1 系統開發源起

SyncDeploymentSystem(簡稱 SDS)專案開發，是源於版本控制的概念去開發實作此系統，版本控制對於已開發的原始碼(source code)而言，為一可以有效的提高產品正確性的管理工具，在軟體開發的過程中會產生不同版本，在測試時，如果發現新版本產生某種先前版本所沒有的問題或狀況，則開發人員可以透過版本控制將歷史紀錄取回。

Subversion 與 CVS(Concurrent Versions System)都是屬於版本控制系統，而版本控制系統，簡單的說，可以記錄程式碼及文件在開發修改的過程中，有一個完整的歷史記錄。在本研究實作的 SDS 專案，具備版本控制與部署更新的機制(如表 1)，它具有 CVS 的優點，除了可以做為原始碼(source code)的版本控制，也可以針對已經開發之原始碼及文件，部署更新至其它多台遠端主機上，並可以依使用者的需求，彈性地配置管理。其中可針對整體的部署去做比較與分析，可由多台 Local 更新至遠端主機，並由遠端主機當基礎，統一更新部署至其它主機，所以是採取 M:1 和 1:M 的雙向機制，前段 M:1 是類似版本控制的作法，而後段 1:M 則是屬於部署更新至各遠端主機的概念，提供使用

者(開發人員)選擇不同的作法，具有極大的彈性調配空間。在新增配置管理後，操作執行配置內容，也可以針對已經更新至各遠端主機的版本或備份資料，進行比對的動作，提供使用者查看部署更新後版本間的差異，以利於檔案文件間的管控。

表 1 各功能比較

軟體\功能	版本控管	部署機制
SDS	V	V
CSV	V	無
Subversion	V	無

3.2 軟體平台的考量

隨著物件導向的發展，愈來愈多人使用物件導向程式設計，而現今主流為微軟(Microsoft)的 Microsoft Visual Studio .NET，.NET 平台主導於 Microsoft, HP, Intel，主要是以跨平台程式開發為主，相對於 Sun 的 JAVA EE，它更提供了多種程式語言的開發程式互通，諸如：Visual Basic.NET，C++/CLI，C#(讀作 C Sharp)，JAVA，F#，COBOL，Python，Perl 等。目前以微軟開發的 Visual Studio.NET 為.NET 開發的主要平台。藉由.NET Framework 的橋接，使得不同的程式語言間能夠互通[8]。.NET 在分散式系統支援上，強調使用 XML 做為傳輸的標準資料格式，並且使用 Internet 上標準的 HTTP 與 SOAP(Simple Object Access

Protocol)通訊協定來傳輸 XML 資料，因此幾乎所有可以連上 Internet 的系統都支援，十分適合用在異質系統的整合上[10]。

.NET Framework 包含了兩個主要的核心技術，分別為 Common Language Runtime(簡稱 CLR)和.NET Framework 類別庫。說明如下[8]：

(1) Common Language Runtime(簡稱 CLR)：CLR 為.NET Framework 的基礎，是一種以物件導向為核心的程式執行環境，可以將 CLR 視為程式在執行時期的管理程式碼的代理者，提供如記憶體管理、執行緒與遠端處理功能等核心處理程序。

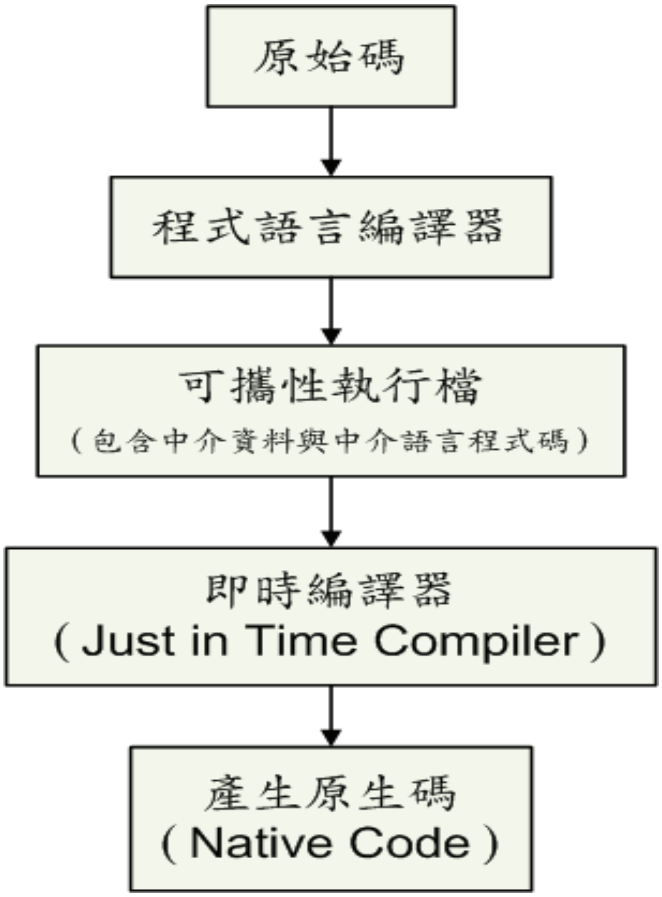

圖 4 CLR 在程式時期所扮演的角色

(2) NET Framework：維基百科對於.NET Framework 的定義如下：.NET 框架（.NET Framework）是由微軟開發，一個致力於敏捷軟體開發（Agile software development）、快速應用開發（Rapid application development）、平臺無關性和網路透明化的軟體開發平臺。.NET 是微軟為下一個十年對伺服器和桌上型軟體工程邁出的第一步。NET 包含許多有助於互聯網和內部網應用迅捷開發的技術。.NET 框架是微軟公司繼 Windows DNA 之後的新開發平臺。.NET 框架是以一種採用系統虛擬機運行的編程平臺，以通用語言運行庫（Common Language Runtime）為基礎，支援多種語言（C#、VB.NET、C++、Python 等）的開發。.NET 也為應用程序介面（API）提供了新功能和開發工具。這些革新使得程式設計員可以同時進行 Windows 應用軟體和網路應用軟體以及元件和服務（web 服務）的開發。.NET 提供了一個新的反射性的且物件導向程式設計編程介面。.NET 設計得足夠通用化從而使許多不同高階語言都得以被彙集。Sun 公司的 Java 編程語言和 Java 平臺，企業版技術是.NET 平臺的競爭對手之一，它們有很多概念也是互通的。

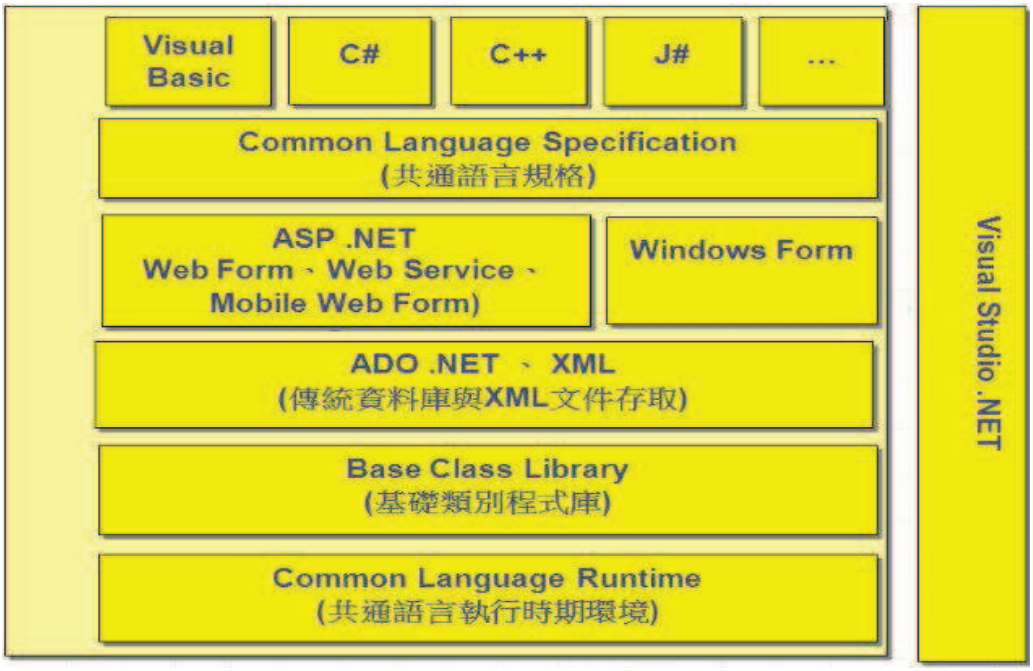

圖 5 .NET Framework 架構圖

在.NET 中支援多種程式語言，單就.NET Framework 中就預設至少支援 Visual Basic .NET 與 C#(#讀做 sharp)兩個程式語言。C#則是專門針對在.NET Framework 上開發應用程式所設計的新程式語言，擁有完整的物件導向支援，在程式語法的執行效率上也比 Visual Basic .NET 快一點。C#的類別宣告與物件使用像 Java；語法則與 C++雷同。維基百科的定義：C#是微軟推出的一種基於.NET 框架的、物件導向的高階程式語言。C#由 C 語言和 C++衍生而來，繼承了其強大的性能，同時又以.NET 框架類別庫作為基礎，擁有類似 Visual Basic 的快速開發能力。C#中發音為 C Sharp，其中「#」的創意來源於音樂中的升調符號「＃」，讀作「sharp」，表示技術進一步提升之意。微軟希望藉助這種語言來取代 Java。C#已經成為 Ecma 國際和國際標準組織的標準規

範。

C#並不被編譯成為能夠直接在電腦上執行的二進制原生代碼。與 Java 類似，它被編譯成為中間代碼(Microsoft Intermediate Language)，然後透過.NET Framework 的虛擬機器——被稱之為通用語言執行層（Common Language Runtime）——執行。

所有的.Net 程式語言都被編譯成這種被稱為 MSIL（Microsoft Intermediate Language）的中間代碼。因此雖然最終的程式在表面上仍然與傳統意義上的執行檔都具有「.exe」的副檔名。但是實際上，如果電腦上沒有安裝.Net Framework，那麼這些程式將不能夠被執行。

在程式執行時，.Net Framework 將中間代碼翻譯成為二進制機器碼，從而使它得到正確的執行。最終的二進制代碼被儲存在一個緩衝器(Buffer)中。所以一旦程式使用了相同的代碼，那麼將會呼叫緩衝器中的版本。這樣如果一個.Net 程式第二次被執行，那麼這種翻譯不需要進行第二次，速度明顯加快。

以現有的程式開發架構來說，不同的程式語言所要利用到的類別函式庫(Class Library)不盡相同，例如：以 VB 來開發須使用 Visual Basic Runtime Library；採用 C++則會運用 MFC Library 及 Active Template Library 等不同的函式庫，而以後的發展趨勢是將這些類別函式庫整合在一起，讓所有的程式語言能夠互通有無，達到跨平台、跨語言的境

界。.Net Framework 就是一個用來建立、開發、部署及執行應用程式的整合環境。它包含了多種的類別函式庫(Class Library)及API(Application Programming Interface)用來支援所有在 Win32 環境下已開發的應用程式及未來 Web Services 所需之功能，讓程式開發時不再因為不同類別函式庫而產生限制與困擾。

因為在.NET Framework 中的類別函式庫已可以共用許多不同程式語言的 Library，所以應用程式的開發不再受限於單一程式語言的限制。目前支援.NET Framework 的程式語言已有四十多種，更方便快速的在不同程式語言間進行類別的繼承、錯誤處理及程式除錯工作，讓程式開發人員可以自由選擇熟悉的程式語言，而不需顧慮多種程式語言之間的問題。相對於 Web Form 這種 Thin Client 應用開發程式架構，Win Form 指的是 Windows-based UI 的應用程式，程式設計人員可以直接繼承.NET Framework Class Library 中的 System.Windows.Forms 類別，建立視窗，使用者操作介面，開發出類似現行以 VB，C++…等語言所開發的桌上型應用程式。此類的應用程式較 Web-based 應用程式更具良好、豐富的使用者介面，又無 Web-based 應用程式 Session Timeout 的限制，操作較為方便，適合需要大量輸入資料的應用。但缺點是必須安裝在每一台要執行的電腦上，程式有修正時，安裝程序便需要重做一次，維護成本較高。在本研究實作上，採用的是 Win Form

的作法，簡單操作的 IDE 介面，不會增加使用者或開發人員的負擔。

而表 2、表 3 列出 SDS 專案實作的各主機的硬體設備與作業系統。

表 2 目的端、目標主機與來源端之硬體環境

	目的端	目標主機	來源端
CPU	Inter(R) Core(TM)2 CPU 6320 @ 1.86GHz 320@1.86GHz	Inter(R) Xeon(TM) CPU 3.66GHz	Inter(R) Xeon(TM) CPU 3.66GHz
記憶體	2G RAM	3G RAM	4G RAM
硬碟	250G	3G RAM	250G
顯示卡	NVIDIA 7100GS	3G RAM	Vmware SVGA 2
網路卡	Realtek 10/100/1000 Mbps	Inter(R)PRO/1000 MT Network Connection	Inter(R)PRO/1000 MT Network Connection

表 3 目的端、目標主機與來源端之軟體環境

	來源端	目標主機	目的端
作業系統	Windows XP Server Pack 3	Windows Server 2003 R2 Service Pack 2	Windows Server 2003 R2 Service Pack 2
IP 位置	172.23.162.134 172.23.162.135 172.23.162.136	172.23.172.20	172.23.172.20 172.23.172.77 172.23.172.41

	172.23.162.137 172.23.162.138		

3.3 系統架構

在本研究實作的 SDS 專案中，其主要架構流程包含以下兩個部分：

(1) 由其它多個來源端主機透過此 SDS 專案，將程式及文件同步部署更新至目標主機，即採用多對一的同步機制。

(2) 再由單一目標主機透過 SDS 專案，同步部署更新程式及文件至其它多台目的端主機上，此為一對多的同步機制。

3.4 架構功能說明

本研究實作 SDS 專案的系統架構功能包含以下三個主要功能：

(1) 配置管理(Configure)：建立欲部署之[來源端]與[目的端]的基本設定資料。

(2) 執行管理(Run)：執行目前已存在於系統內欲部署之[來源端]與[目的端]的程序。

(3) 比對記錄(Compare)：查看部署完成後，其程式及文件的比對結果記錄。

第四章　系統分析

4.1 資訊系統

對於本研究之各功能分析採用的是統一塑模語言（UML，Unified Modeling Language）[1][2][4]，維基百科對於 UML 的定義如下：統一塑模語言（UML，Unified Modeling Language）是非專利的第三代塑模和規約語言。UML 是一種開放的方法，用於說明、可視化、構建和編寫一個正在開發的、物件導向的、軟體密集系統的製品的開放方法。UML 展現了一系列最佳工程實踐，這些最佳實踐在對大規模，複雜系統進行塑模方面，特別是在軟體架構層次已經被驗證有效。

UML 集成了 Booch，OMT 和物件導向軟體工程的概念，將這些方法融合為單一的，通用的，並且可以廣泛使用的塑模語言。UML 打算成為可以對併發和分布式系統的標準塑模語言。

UML 並不是一個工業標準，但在 Object Management Group 的主持和資助下，UML 正在逐漸成為工業標準。OMG 之前曾經呼籲業界向其提供有關物件導向的理論及實現的方法，以便製作一個嚴謹的軟體塑模語言（Software Modeling Language）。有很多業界的領袖亦真誠地回應 OMG，幫助她建立一個業界標準。

在 UML 系統開發中有三個主要的模型：

(1) 功能模型：從用戶的角度展示系統的功能，包括使用個案圖。

(2) 物件模型：採用物件，屬性，操作，關聯等概念展示系統的結構和基礎，包括類圖。

(3) 動態模型：展現系統的內部行為。包括序列圖，活動圖，狀態圖。

UML2.2 中一共定義了 14 種圖示（diagrams）。為方便了解，可分類成圖 6 的結構。

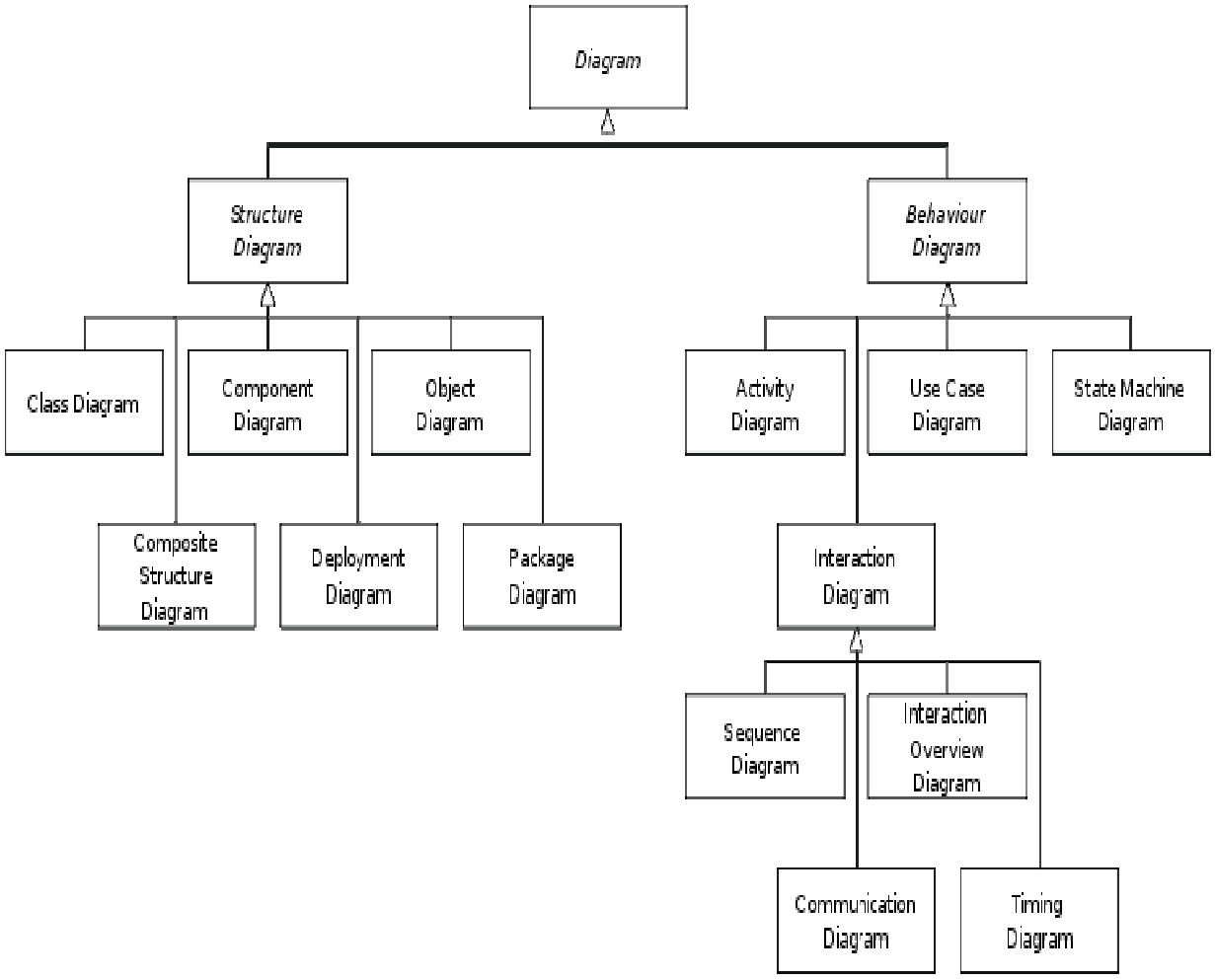

圖 6 UML 2.2 的圖示結構

其結構可分成三大類：

(1) 結構性圖形（Structure diagrams）強調的是系統式的塑模：

- 類別圖(Class Diagram)
- 元件圖(Component diagram)
- 複合結構圖(Composite structure diagram)
- 部署圖(Deployment diagram)
- 物件圖(Object diagram)
- 套件圖(Package diagram)
- 剖面圖(Profile diagram)

(2) 行為式圖形（Behavior diagrams）強調系統模型中觸發的事件：

- 活動圖(Activity diagram)
- 狀態機圖(State Machine diagram)
- 使用個案圖(Use Case Diagram)

(3) 溝通性圖形（Interaction diagrams），屬於行為圖形的子集合，強調系統模型中的資料流程：

- 通信圖(Communication diagram)
- 交互概述圖(Interaction overview diagram)(UML 2.0)
- 時序圖(Sequence diagram)
- 時間圖(UML Timing Diagram)(UML 2.0)

在本研究中使用到 UML 圖示有三種：類別圖、活動圖與使用者案例圖。

4.2 需求定義

在系統開發生命週期的實作與測試階段中，當系統開發完成後，經由不同的測試階段或使用者需求不停的變更，開發人員需不斷的修改程式與更新部署程式至其它測試主機或開發主機，而為了避免開發人員反反覆覆的更新部署程式，或在更新程式的過程中更新程式被遺漏而導致系統錯誤等情況發生。故在本研究中，我們提出以物件導向程式設計之軟體人工製品。而針對各項需求則整理如下：

(1) 簡單的配置管理功能列表。

(2) 簡單易懂的執行管理功能列表。

(3) 部署後之版本比對記錄列表。

表 4 系統中主要各功能需求定義

功能定義	需求定義
配置管理	1.來源端配置管理 2.目的端配置管理
執行管理	1.由來源端執行複製檔案至目標主機

	2.由目標主機執行複製檔案至目的端
比對管理	1.比對分析報表

4.3 整體架構圖

SDS 的整體架構圖如圖 7 所示，分成三個部份：

(1) 來源端：各個不同的來源主機。

(2) 中介端：中間層目標主機，將檔案或文件用以統一集中管理。

(3) 目的端：將最後檔案或文件的版本配置至各個不同的目的主機。

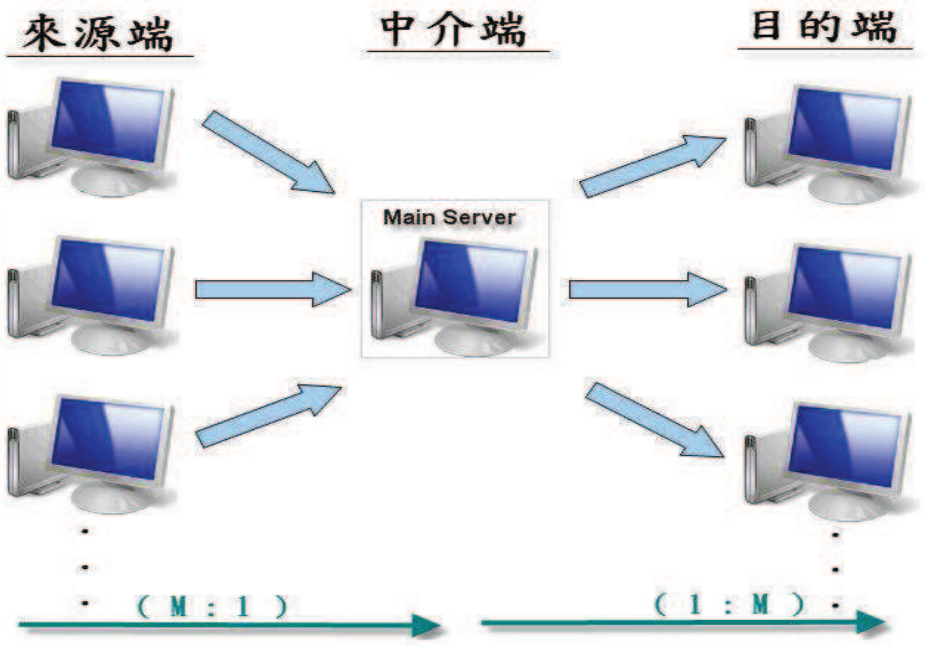

圖 7 整體架構圖

4.4 系統架構圖

SDS 的系統架構圖如圖 8 所示，其功能定義如下：

(1) 配置管理(Configure)：建立欲部署之[來源端]與[目的端]的基本設定資料。

(2) 執行管理(Run)：執行目前已存在在系統內欲部署之[來源端]與[目的端]的程序。

(3) 比對記錄(Compare)：查看部署之比對結果記錄之分析資料。

圖 7 為 SDS 的整體架構圖，而圖 8 為 SDS 之系統架構圖。

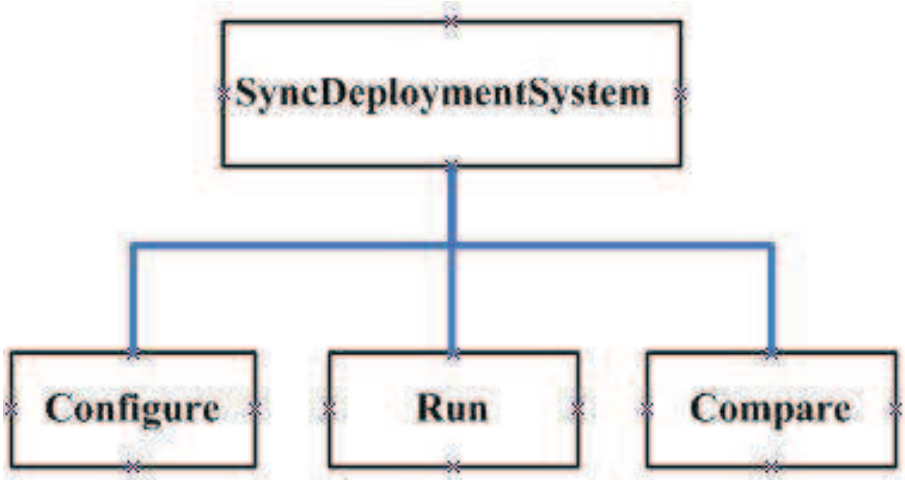

圖 8 系統架構圖

4.5 系統流程圖

SDS 的系統流程圖如圖 9 所示，使用者執行 SDS，操作步驟如下：

步驟一，使用者執行 SDS，先配置來源端與目的端之資料。

步驟二，儲存後，再執行版本控管與部署作業。

步驟三，成功後，可比對各版本之間的差異。

步驟四，產生出差異報表。

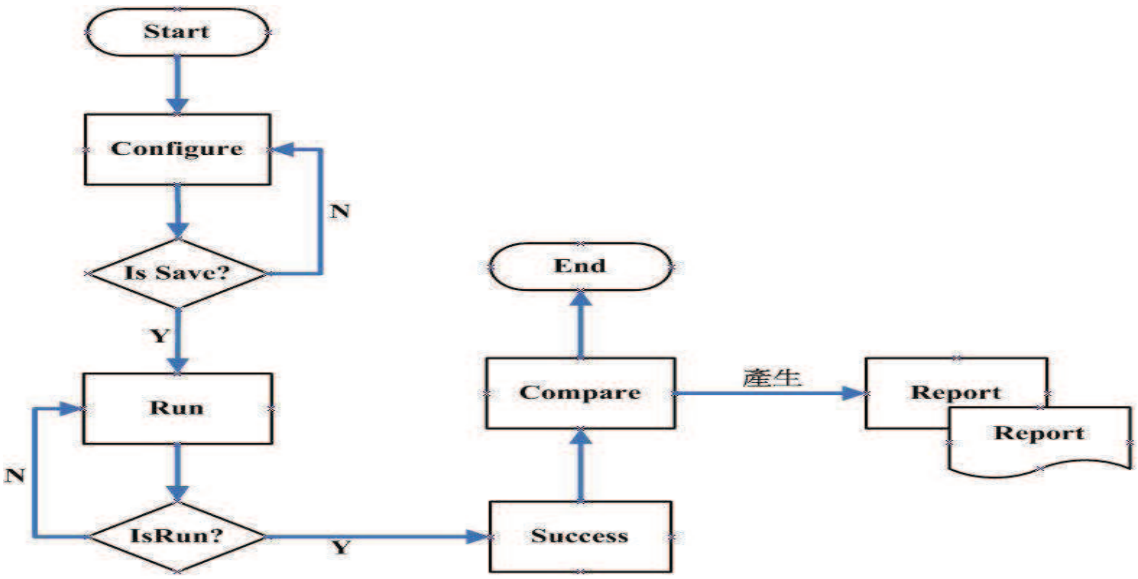

圖 9 系統流程圖

SDS 的 UML 類別圖如圖 10、圖 11 所示。

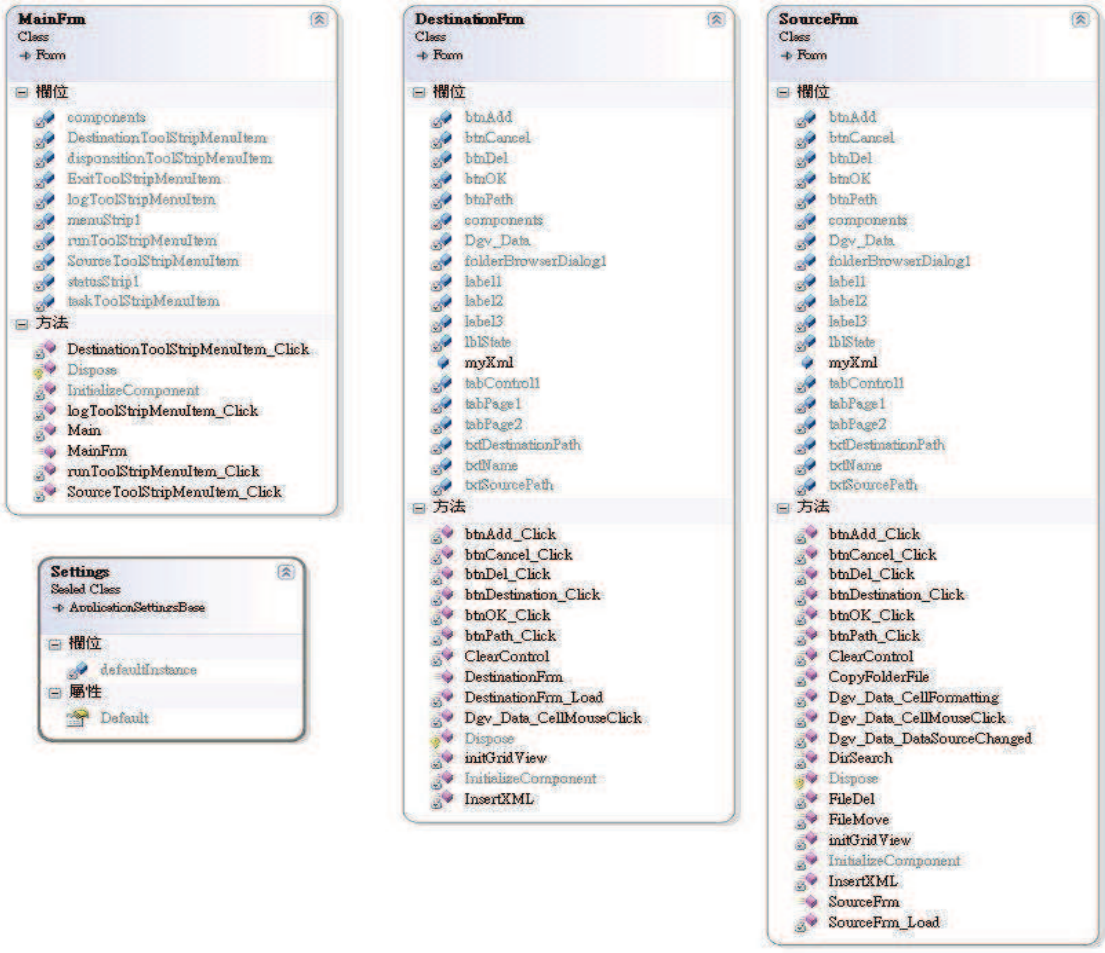

圖 10 UML 類別圖之畫面一

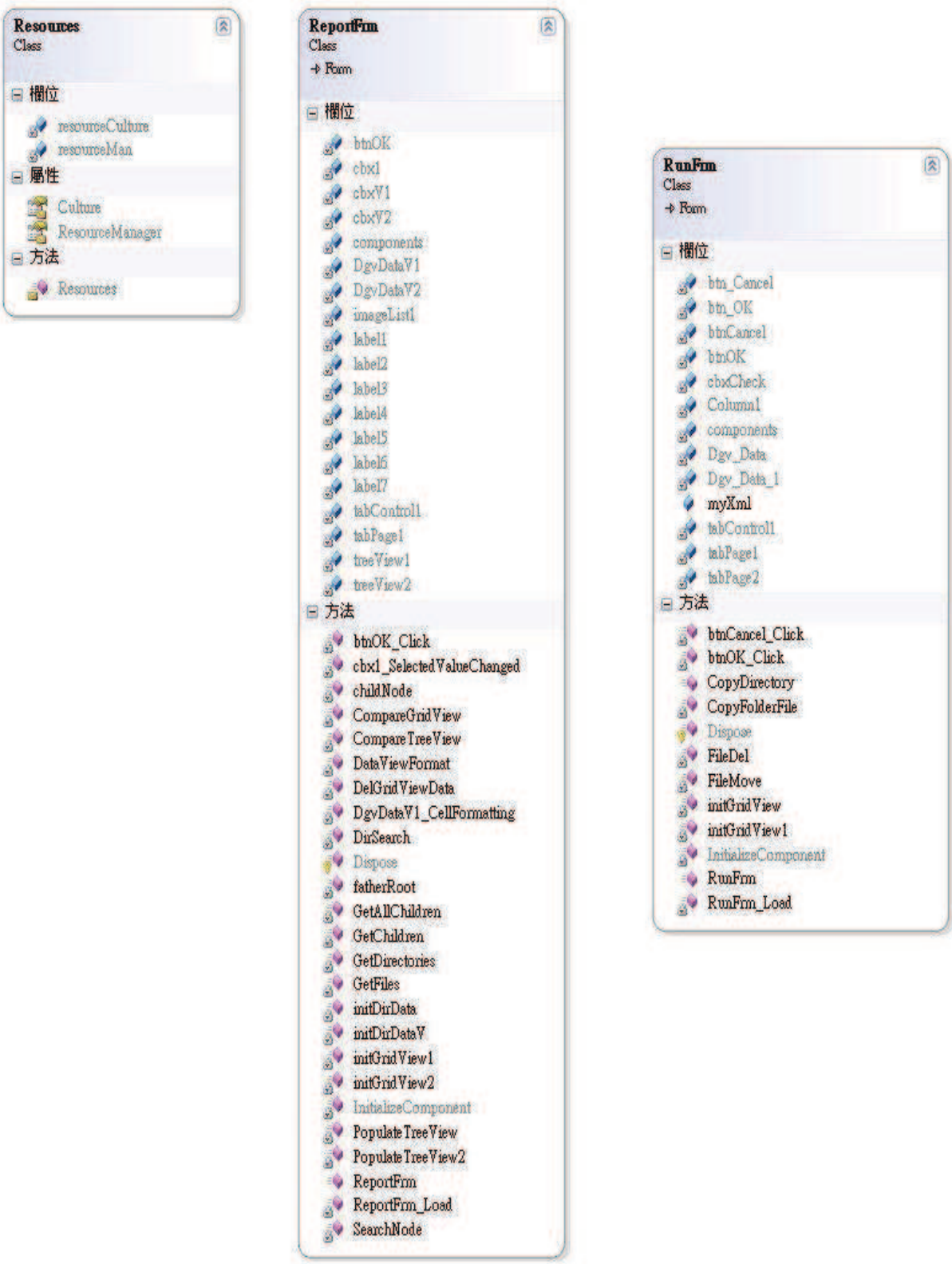

圖 11 UML 類別圖之畫面二

SDS 的 UML 活動圖如圖 12 所示。

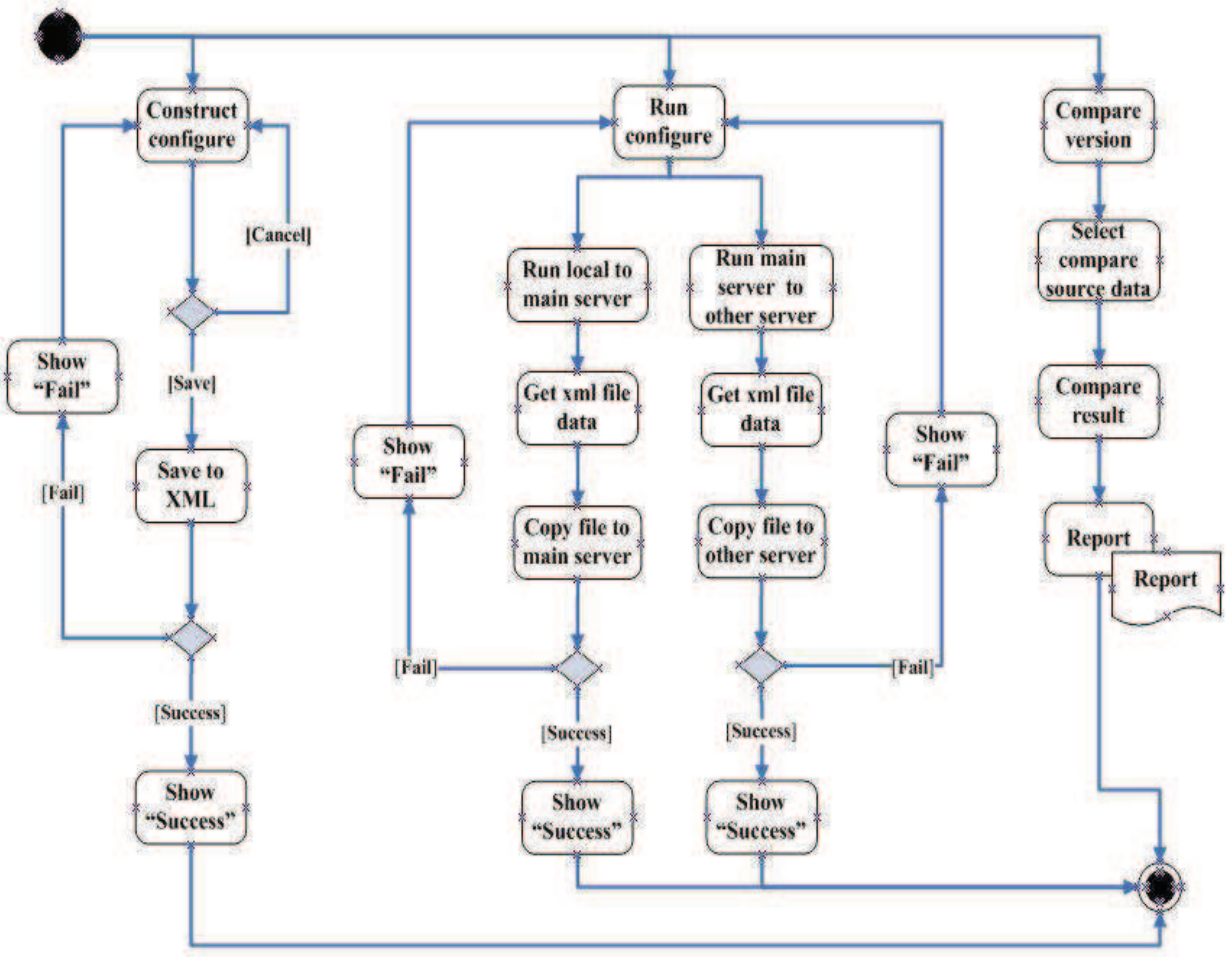

圖 12 UML 活動圖

4.6 使用者案例圖

使用者案例圖(User Case Diagram)主要表示使用者與電腦系統或系統與系統間兩者之間的互動情形，由圖形中可以了解到一個系統提供的功能，並確認系統或應用程式的外部或內部元素及系統的範圍。因採用圖形化方式呈現出來，因此讓使用者清楚了解此系統的整體需求與目標。

SDS 專案在整個設計上分成四個部份，分別為操作 SDS 專案的使

用者、來源主機、目標主機及各目的主機。系統使用者案例圖，如圖13(SDS 的基本使用者案例圖)、圖 14(SDS 的系統使用者案例圖)所示。

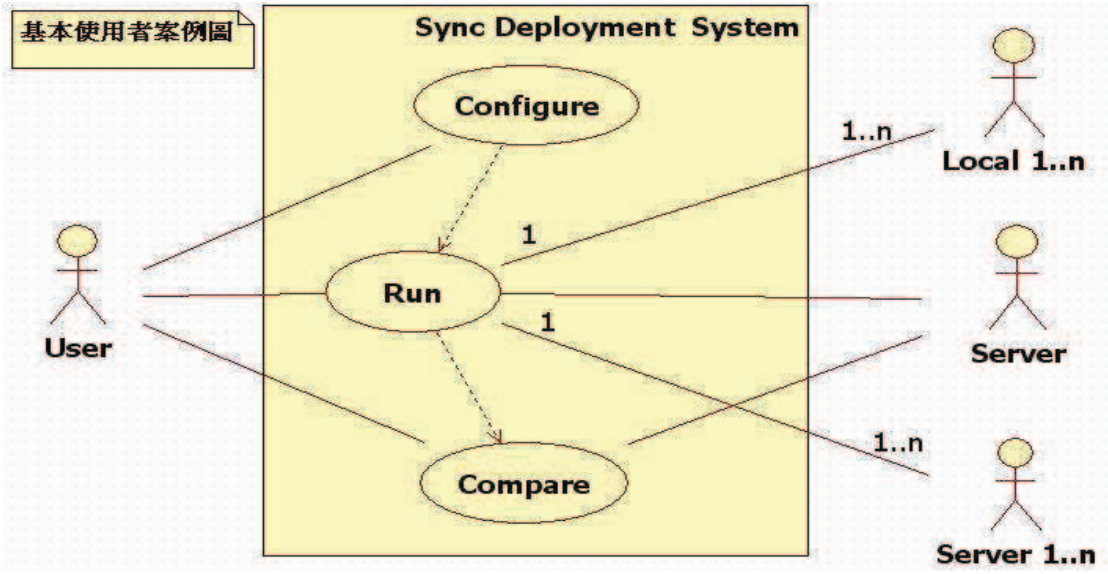

圖 13 基本使用者案例圖

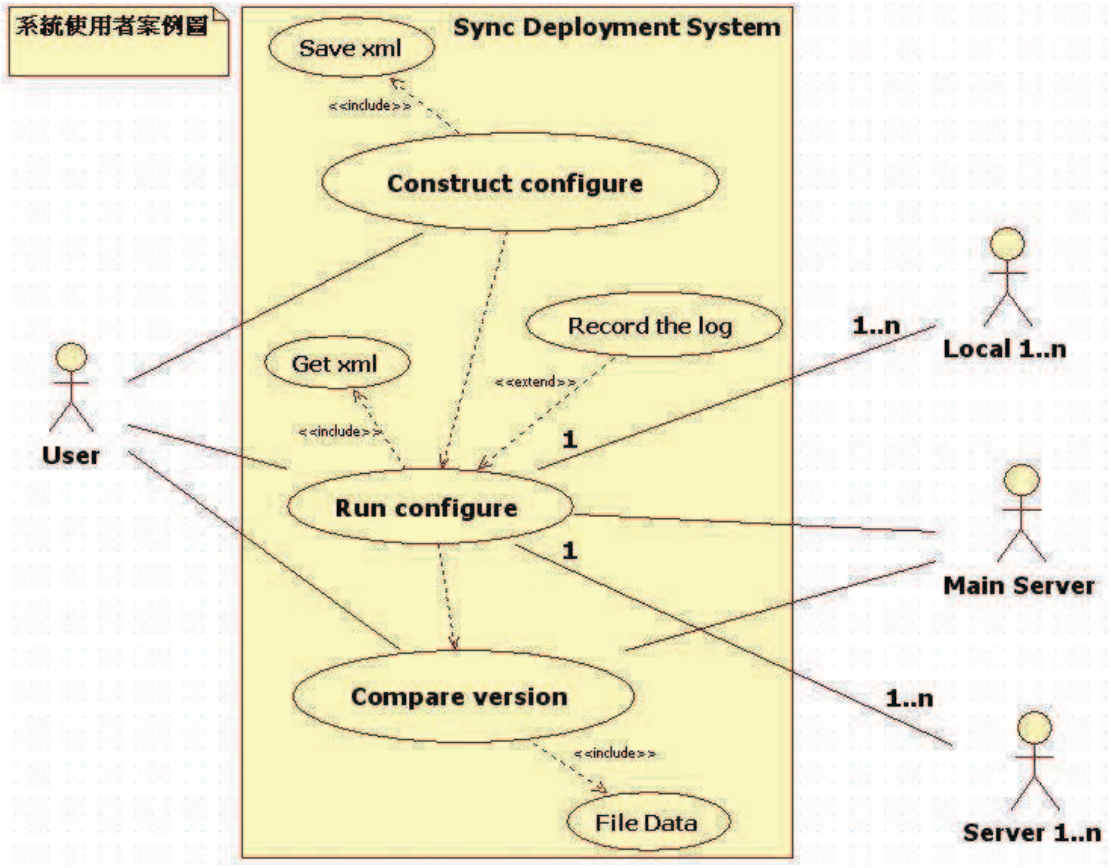

圖 14 系統使用者案例圖

表 5 Configure 基本功能說明，表 6 Compare 基本功能說明，表 7 Run 基本功能說明。

表 5 Configure 基本功能說明

項目	說明
使用者案例	Configure
目的	配置管理
使用者動作	系統反應與回應
1.使用者選擇來源端/目的端	2.將來源端/目的端資料儲存至 XML

表 6 Compare 基本功能說明

項目	說明
使用者案例	Compare
目的	比對管理
使用者動作	系統反應與回應
1.使用者選擇欲比對的版本	2.將差異內容呈現在畫面，供使用者參考

表 7 Run 基本功能說明

項目	說明
使用者案例	Run

目的	執行管理
使用者動作	系統反應與回應
1.使用者勾選欲部署的資料	2.將資料部署至所指定的主機上

第五章　實作展示

5.1 系統實作介面

SDS 具有雙重機制的概念：(1)單純版本控管備份機制、(2)程式及文件同步部署的機制，依使用者的需求或目的使用此實作系統。

本研究實作之軟體人工製品，在經由編譯產生之 EXE 檔，僅需存放在目標主機上，使用者可在目標主機上，直接執行，功能操作大致上可分成四個步驟，其步驟如下：

(1) 先建立配置及部署之[來源端]資料，欄位內容包含：名稱、來源、目的，可以針對目前已經存在於系統內的資料，進行新增、刪除、修改等動作。

(2) 建立配置及部署之[目的端]資料，欄位內容包含：名稱、來源與目的，可以針對目前已經存在於系統內的資料，進行新增、刪除、修改等動作。

(3) 執行[來源端]的程式部署，將來源端的程式碼及文件進行複製的動作，將來源端的程式碼及文件部署至中介端(目前目標主機的路徑)。

(4) 執行[目的端]的程式部署，將中介端(目標主機)的程式碼及文件部署至目的端。

SDS 之配置管理設定畫面，建立[來源端]的資料，如圖 15 所示，再將[來源端]的資料儲存至 XML 檔案，XML 格式內容如圖 16 所示。

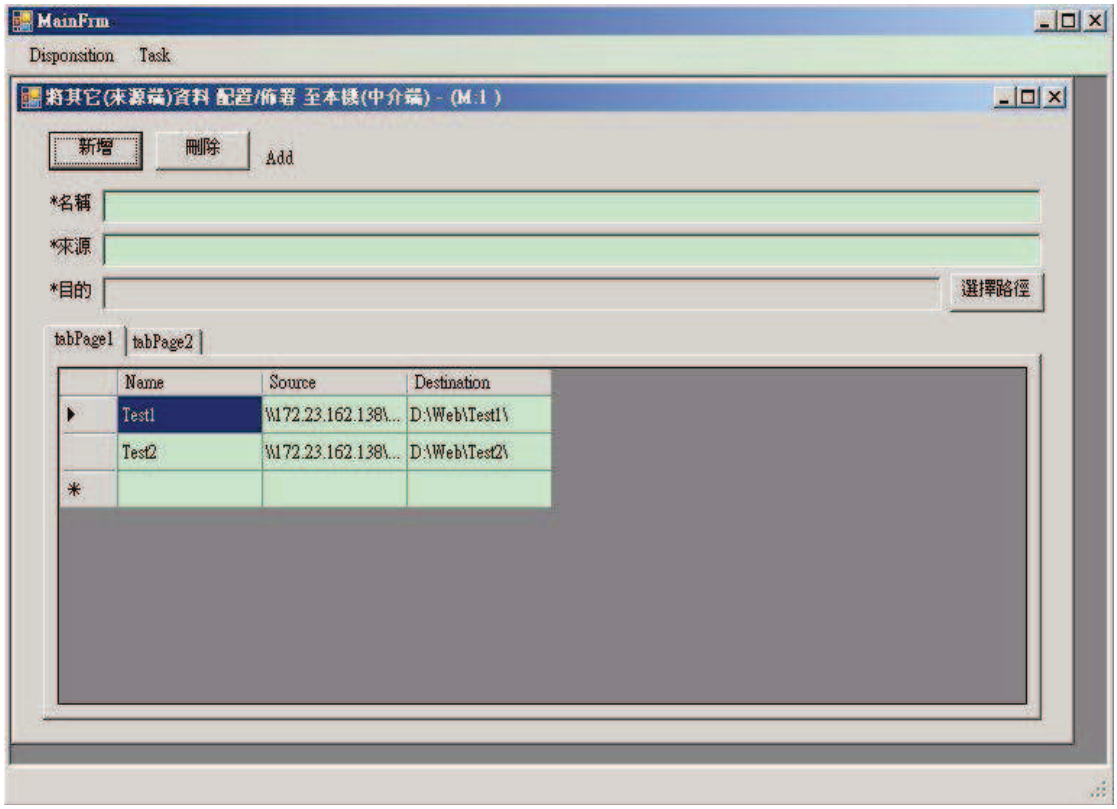

圖 15 來源端-設定畫面

```
<?xml version="1.0" encoding="utf-8"?>
<Main>
    <Path>
        <Name>AuWeb</Name>
        <Source>\\172.23.162.138\Web\AuWeb</Source>
        <Destination>D:\Test</Destination>
    </Path>
    <Path>
        <Name>BTSWeb</Name>
        <Source>\\172.23.162.139\Web\BTSWeb</Source>
        <Destination>D:\Test</Destination>
    </Path>
</Main>
```

圖 16 XML 來源端內容

SDS 之配置管理設定畫面，建立[目的端]的資料，如圖 17 所示，再將[目的端]的資料儲存至 XML 檔案，XML 格式內容如圖 18 所示。

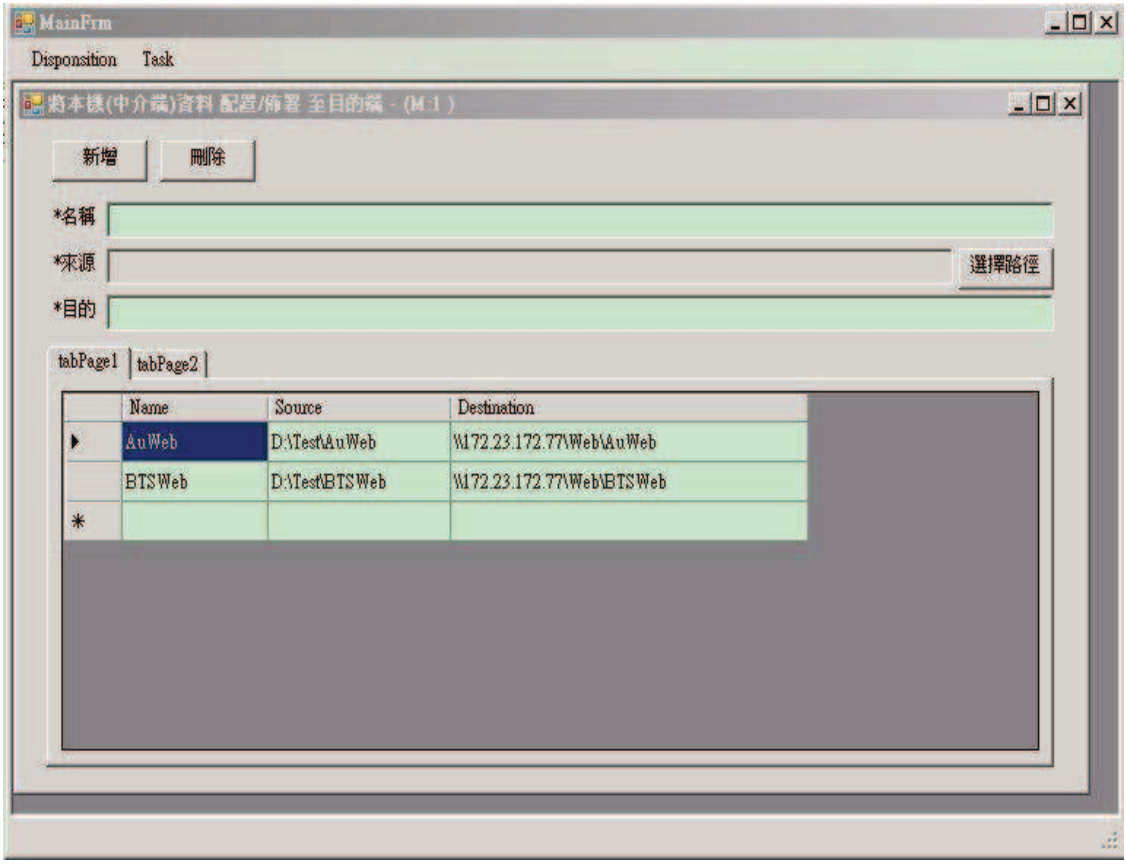

圖 17 目的端-設定畫面

```
<?xml version="1.0" encoding="utf-8"?>
<Main>
    <Path>
        <Name>AuWeb</Name>
        <Source>D:\Test\AuWeb</Source>
        <Destination>\\172.23.172.77\Web\AuWeb</Destination>
    </Path>
    <Path>
        <Name>BTSWeb</Name>
        <Source>D:\Test\BTSWeb</Source>
        <Destination>\\172.23.172.77\Web\BTSWeb</Destination>
    </Path>
</Main>
```

圖 18 XML 目的端內容

SDS 之執行畫面，使用者可以選擇欲執行的項目(單一或多筆:圖

19 及圖 20)。

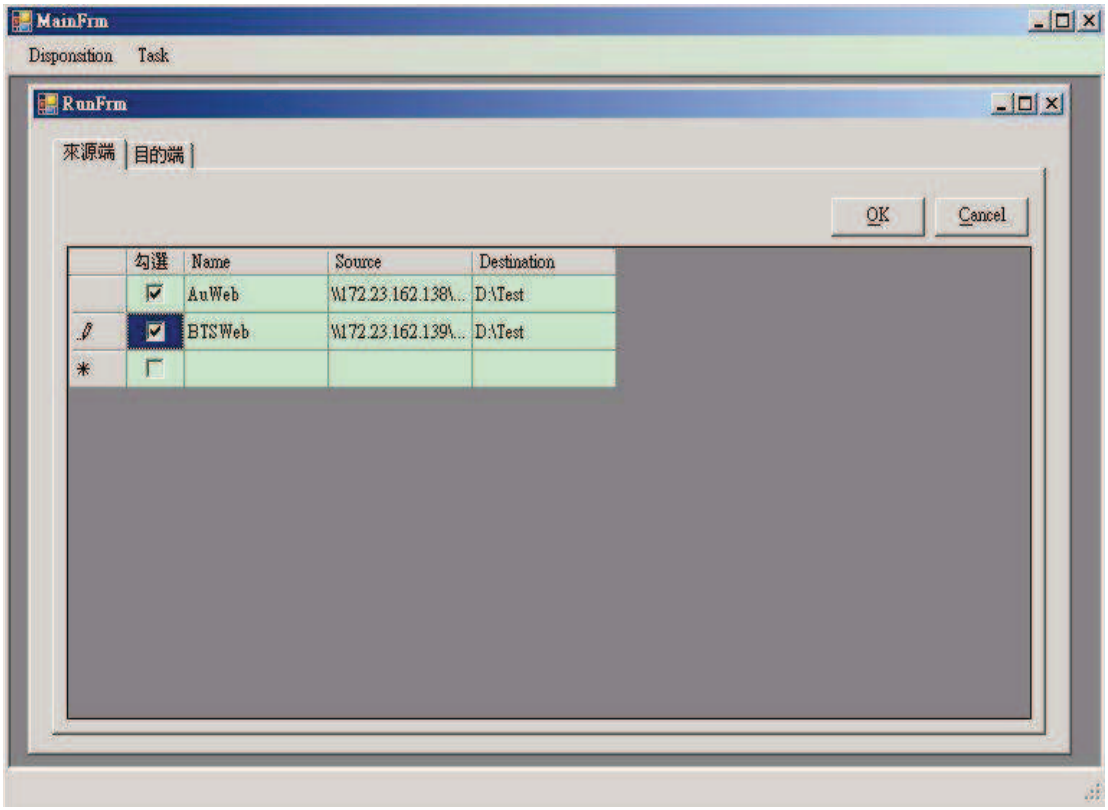

圖 19 SDS 執行畫面一

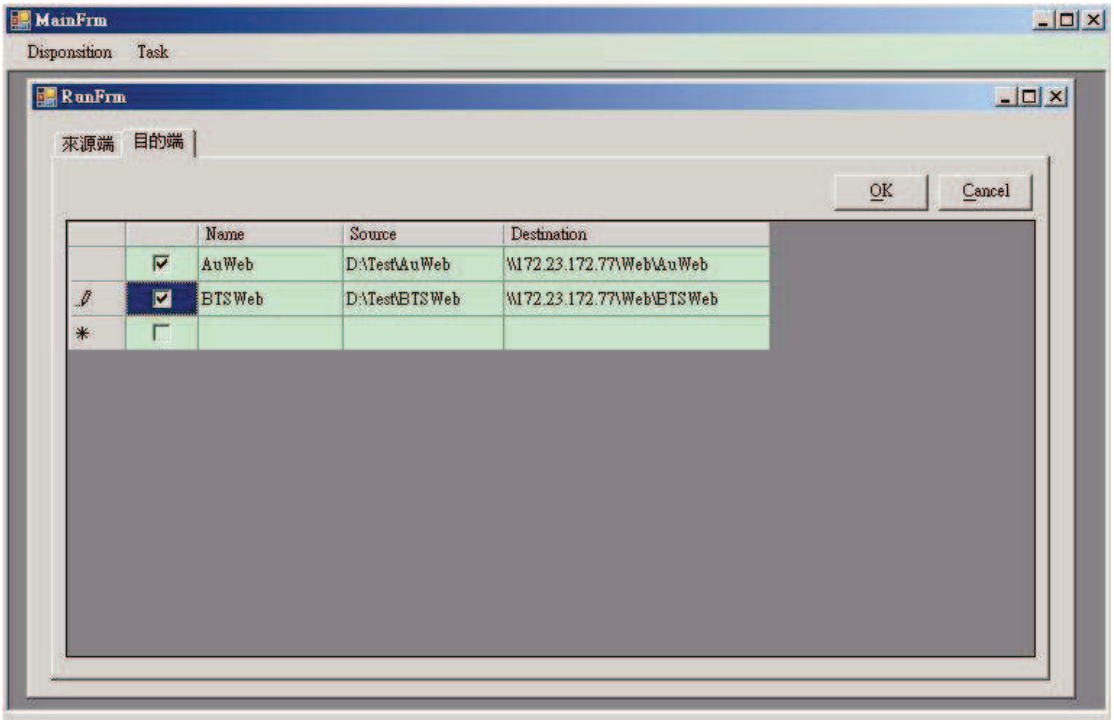

圖 20 SDS 執行畫面二

SDS 的主要比對的條件判斷，針對相同檔案文件，利用檔案文件大小與檔案文件的修改日期，進行版本間的比對。而 SDS 實作的結果，可透過 IDE 呈現在畫面上，透過顏色的劃分，提供簡單易懂的視覺化效果以區別之。圖範例中二個版本間比對的結果(V1 與 V2 比對)，粉紅色列表示檔案存在在 V1 的版本，但已經不存在在 V2 的版本，或只存在在 V2 的版本，但不存在在 V1 的版本(圖 21 及圖 22)；藍色列表示檔案都有存在在二邊的版本(V1 與 V2)，但檔案文件內容已經被異動(圖 22)。

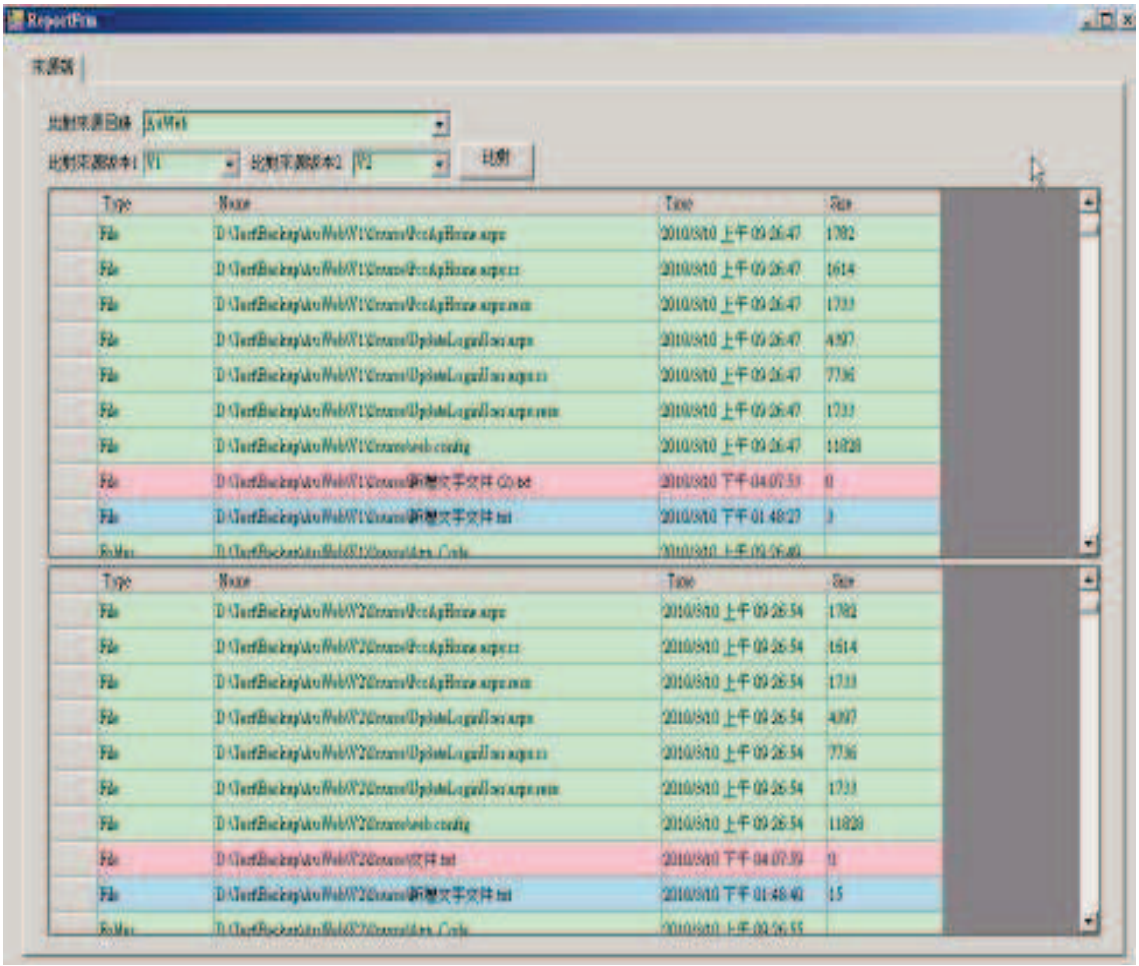

圖 21 SDS 版本比對畫面一

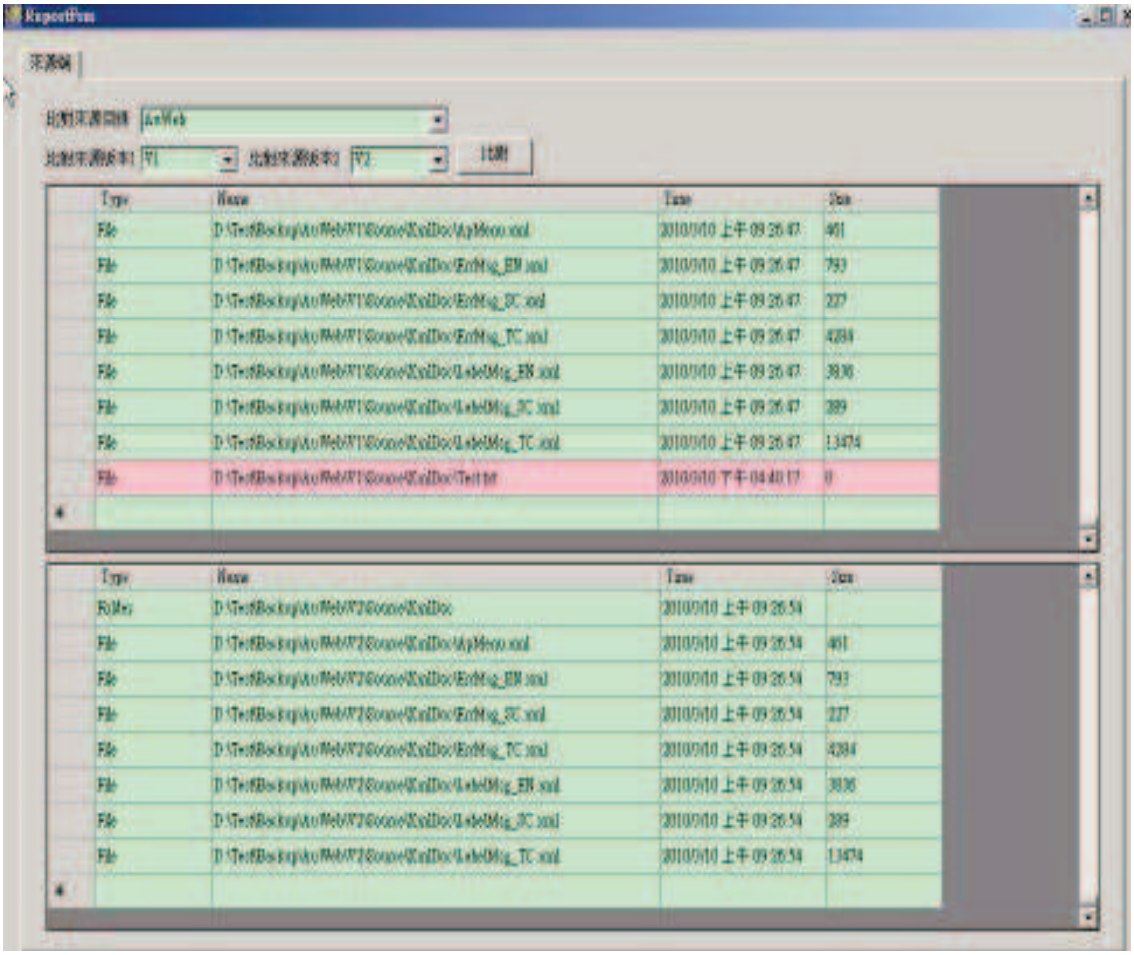

圖 22 SDS 版本比對畫面二

5.2 系統實作程式

SDS 之配置管理設定的程式碼：圖 23、圖 24。配置管理介面初始化時，載入[來源端]的 XML 檔案內容，並顯示在介面中，提供給使用者參考，使用者可以查看是否需要再增加或刪除設定來源端的資料，以進行部署更新的動作，請參考程式碼如圖 23。

```
//載入XML檔案內容，呈現在介面，給使用者參考
        private void initGridView()
        {
            myXml.Load("Source.xml");//讀取xml檔
            int intCount = myXml.ChildNodes.Item(1).ChildNodes.Count;//取得路徑數量

            DataTable DT_Data = new DataTable();
            DataRow DR_Data;
            DataView DVI_Data = new DataView();

            DT_Data.Columns.Add("Name");
            DT_Data.Columns.Add("Source");
            DT_Data.Columns.Add("Destination");

            for (int i = 0; i < intCount; i++)
            {
                DR_Data = DT_Data.NewRow();
                DR_Data["Name"] = myXml.ChildNodes.Item(1).ChildNodes.Item(i).ChildNodes.Item(0).InnerText;
                DR_Data["Source"] = myXml.ChildNodes.Item(1).ChildNodes.Item(i).ChildNodes.Item(1).InnerText;
                DR_Data["Destination"] = myXml.ChildNodes.Item(1).ChildNodes.Item(i).ChildNodes.Item(2).InnerText;
                DT_Data.Rows.Add(DR_Data);
            }
            Dgv_Data.DataSource = DT_Data;
        }
```

圖 23 載入 XML 檔案內容

配置管理介面，當使用者增加或刪除設定[來源端]的資料，將設定後的內容，儲存至 XML 檔案內，請參考程式碼如圖 24。

```
//複寫至XML檔
        private void InsertXML()
        {
            //將新增的資料,儲存在xml內
            XmlWriterSettings settings = new XmlWriterSettings();
            settings.Indent = true;
            settings.IndentChars = ("    ");

            using (XmlWriter writer = XmlWriter.Create("Source.xml", settings))
            {
                // Write XML data.
                writer.WriteStartElement("Main");
                //GridView內的資料,再重寫一次
                for (int i = 0; i < Dgv_Data.Rows.Count - 1; i++)
                {
                    if (lblState.Text == "Edit")
                    {
                        if (Dgv_Data.CurrentRow.Index == i)
                        {
                            writer.WriteStartElement("Path");
                            writer.WriteElementString("Name", txtName.Text.Trim());
                            writer.WriteElementString("Source", txtSourcePath.Text.Trim());
                            writer.WriteElementString("Source", txtSourcePath.Text.Trim());
                            writer.WriteElementString("Destination", txtDestinationPath.Text.Trim());
                            writer.WriteEndElement();
                        }
                        else
                        {
                            writer.WriteStartElement("Path");
                            writer.WriteElementString("Name", Dgv_Data.Rows[i].Cells[0].Value.ToString());
                            writer.WriteElementString("Source", Dgv_Data.Rows[i].Cells[1].Value.ToString());
                            writer.WriteElementString("Destination", Dgv_Data.Rows[i].Cells[2].Value.ToString());
                            writer.WriteEndElement();
                        }
                    }
                    else
                    {
                        writer.WriteStartElement("Path");
                        writer.WriteElementString("Name", Dgv_Data.Rows[i].Cells[0].Value.ToString());
                        writer.WriteElementString("Source", Dgv_Data.Rows[i].Cells[1].Value.ToString());
                        writer.WriteElementString("Destination", Dgv_Data.Rows[i].Cells[2].Value.ToString());
                        writer.WriteEndElement();
                    }
                }
                //
                if (lblState.Text == "Add")
                {
                    writer.WriteStartElement("Path");
                    writer.WriteElementString("Name", txtName.Text.Trim());
                    writer.WriteElementString("Source", txtSourcePath.Text.Trim());
                    writer.WriteElementString("Destination", txtDestinationPath.Text.Trim());
                    writer.WriteEndElement();
                }
                //writer.WriteEndElement();
                writer.Close();
                writer.Flush();
            }
        }
```

圖 24 複寫至 XML 檔案

SDS 之[來源端]執行管理的程式碼：圖 25、圖 26、圖 27。[來源端]執行管理的執行功能，可以依據使用者勾選預計要進行部署的項目，將來源主機的檔案與資料夾部署至目標主機，請參考程式碼如圖 25 所示。

```
//執行
        private void btnOK_Click(object sender, EventArgs e)
        {
            //CheckBox cbx;
            //GridView內的資料,再重寫一次
            string sSource = "";
            string sDestination = "";

            for (int i = 0; i < Dgv_Data.Rows.Count - 1; i++)
            {
                if (Dgv_Data.Rows[i].Cells[0].Value != null)
                {
                    //\\172.23.172.20\web\HR
                    sSource = Dgv_Data.Rows[i].Cells[2].Value.ToString().Replace("\\", @"\");
                    sDestination = Dgv_Data.Rows[i].Cells[3].Value.ToString().Replace("\\", @"\");
                    CopyFolderFile(@"" + sSource, @"" + sDestination);
                }
            }

        }
```

圖 25 執行程式碼

執行管理的[來源端]之執行功能，將來源主機的檔案與資料夾，複製至目標主機，並在目標主機將檔案與資料夾另外存檔備份，做為比對管理時之比對記錄，請參考程式碼如圖 26 所示。

```
private void CopyFolderFile(string strSource, string strDestination)
        {
            string sourcePath = strSource;
            string targetPath = strDestination;
            string sourceFile = "";
            string destFile = "";
            string fileName = "";

            if (System.IO.Directory.Exists(sourcePath))
            {
                DirectoryInfo source = new DirectoryInfo(strSource);
                DirectoryInfo target = new DirectoryInfo(strDestination);

                //Determine the source directory exists
                string sDir = strDestination + @"\" + source.Name.Trim();
                DirectoryInfo dDir = new DirectoryInfo(sDir);
                if (!dDir.Exists)
                    dDir.Create();

                string sbDir = strDestination + @"_Backup\" + source.Name.Trim();
                DirectoryInfo backDir = new DirectoryInfo(sbDir);
                if (!backDir.Exists)
                    backDir.Create();

                //Copy Directory
                CopyDirectory(strSource, strDestination + @"\" + source.Name.Trim());
                //DirSearch(@"" + sourcePath);

                //Copy Directory(備份檔)
                CopyDirectory(strSource, strDestination + @"_Backup\" + source.Name.Trim());
            }
            else
            {
                Console.WriteLine("Source path does not exist!");
            }
        }
```

圖 26 複製與備份檔案與資料夾

複製檔案與資料夾的程式碼，請參考程式碼如圖 27 所示。

```
public void CopyDirectory(string SourceDirectory, string TargetDirectory)
        {
            DirectoryInfo source = new DirectoryInfo(SourceDirectory);
            DirectoryInfo target = new DirectoryInfo(TargetDirectory);

            //Determine the source directory exists.
            if (!source.Exists)
                return;
            if (!target.Exists)
                target.Create();

            //Copy Files.
            FileInfo[] sourceFiles = source.GetFiles();
            for (int i = 0; i < sourceFiles.Length; ++i)
                File.Copy(sourceFiles[i].FullName, target.FullName + "\\" + sourceFiles[i].Name.Trim(), true);

            //Copy Directoryies.
            DirectoryInfo[] sourceDirectories = source.GetDirectories();
            for (int j = 0; j < sourceDirectories.Length; ++j)
                CopyDirectory(sourceDirectories[j].FullName, target.FullName + "\\" + sourceDirectories[j].Name.Trim());
        }
```

圖 27 複製檔案與資料夾

SDS 之比對管理的程式碼：圖 28、圖 29。將比對的版本進行檔案比對的動作，將版本一與版本二的檔案與資料夾進行比對的動作，在二個版本中，相同的檔案名稱，則比較檔案大小，若檔案大小不一致或比對版本中檔案不存在，則用顏色區隔標示出來，請參考程式碼如圖 28 與圖 29 所示。

```
//比較GridView1 與GridView2
        private void CompareGridView()
        {
            string strName1 = "";
            string strName2 = "";
            string strSize1 = "";
            string strSize2 = "";
            bool blnExist = true;
            int intExistR = 0;
            int intExistC = 0;

            //左邊與右邊比較
            for (int i = 0; i < DgvDataV1.RowCount-1; i++)
            {
                strName1 = DgvDataV1.Rows[i].Cells[1].Value.ToString().Replace(string.Format("\\{0}\\", cbxV1.SelectedItem.ToString()),
"\\Compare\\");
                strSize1 = DgvDataV1.Rows[i].Cells[3].Value.ToString();
                for (int j = 0; j < DgvDataV2.RowCount-1; j++)
                {
                    strName2 = DgvDataV2.Rows[j].Cells[1].Value.ToString().Replace(string.Format("\\{0}\\", cbxV2.SelectedItem.ToString()),
"\\Compare\\");
                    strSize2 = DgvDataV2.Rows[j].Cells[3].Value.ToString();
                    if (strName1 == strName2)
                    {
                        if (strSize1 != strSize2)
                        {
                            //DgvDataV1.Rows[i].Cells[0].Value = "Y";
                            DgvDataV1.Rows[i].DefaultCellStyle.BackColor = Color.LightBlue;
                            //DgvDataV2.Rows[j].Cells[0].Value = "Y";
                            DgvDataV2.Rows[i].DefaultCellStyle.BackColor = Color.LightBlue;
                        }
                        blnExist = true;
                        intExistR = i;
                        intExistC = j;
                        break;
                    }
                    else
                    {
                        blnExist = false;
                    }
                }
```

圖 28 版本比對(一)

```
                //若不存在,表示是新增/刪除的資料(表示左邊有,右邊沒有)
                if (blnExist == false)
                {
                    DgvDataV1.Rows[i].DefaultCellStyle.BackColor = Color.Pink;
                    blnExist = true;
                }
            }
            string strName11 = "";
            string strName22 = "";
            string strSize11 = "";
            string strSize22 = "";
            bool blnExist1 = true;
            int intExistR1 = 0;
            int intExistC1 = 0;
            //右邊與左邊比較
            for (int x = 0; x < DgvDataV2.RowCount - 1; x++)
            {
                strName11 = DgvDataV2.Rows[x].Cells[1].Value.ToString().Replace(string.Format("\\{0}\\", cbxV2.SelectedItem.ToString()),
"\\Compare\\");
                strSize11 = DgvDataV2.Rows[x].Cells[3].Value.ToString();
                for (int y = 0; y < DgvDataV1.RowCount - 1; y++)
                {
                    strName22 = DgvDataV1.Rows[y].Cells[1].Value.ToString().Replace(string.Format("\\{0}\\", cbxV1.SelectedItem.ToString()),
"\\Compare\\");
                    strSize22 = DgvDataV1.Rows[y].Cells[3].Value.ToString();
                    if (strName11 == strName22)
                    {
                        if (strSize11 != strSize22)
                        {
                            //DgvDataV1.Rows[i].Cells[0].Value = "Y";
                            DgvDataV1.Rows[x].DefaultCellStyle.BackColor = Color.LightBlue;
                            //DgvDataV2.Rows[j].Cells[0].Value = "Y";
                            DgvDataV2.Rows[x].DefaultCellStyle.BackColor = Color.LightBlue;
                        }
                        blnExist1 = true;
                        intExistR1 = x;
                        intExistC1 = y;
                        break;
                    }
                    else
                    {
                        blnExist1 = false;
                    }
                }
                //若不存在,表示是新增/刪除的資料(表示右邊有,左邊沒有)
                if (blnExist1 == false)
                {
                    DgvDataV2.Rows[x].DefaultCellStyle.BackColor = Color.Pink;
                    blnExist = true;
                }
            }
            DelGridViewData();
        }
```

圖 29 版本比對(二)

第六章　實作評估

6.1 模擬環境

在本節中，在 Intranet 內部網路模擬環境中，分別以 4 台來源主機、1 台目標主機與 3 台目的主機，進行本論文的實驗測試，其對應的 IP Address 如表 8 所示。

表 8 實驗主機的 IP Address

IP	來源主機	目標主機	目的主機
IP Address	172.23.162.135 172.23.162.136 172.23.162.137 172.23.162.138	172.23.172.20	172.23.172.77 172.23.172.41 172.23.172.90

6.2 實驗與結果(比較結果)

在本節中，以兩個例子做為實驗，一個來源路徑指向 AuWeb 資料夾，其參數 Name 設定為 AuWeb，另外一個來源路徑指向 BTSWeb 資料夾，其參數 Name 設定為 BTSWeb。首先提出說明 AuWeb 的細節，相同的來源檔案，測試在不同時間進行同步部署的動作(實驗測試檔案內容有異動的情況下)，而每次部署都會備份一份完整的資料記錄在目

標主機，圖 30 為目標主機 AuWeb 備份的記錄內容，圖 31 為目標主機 BTSWeb 備份的記錄內容。

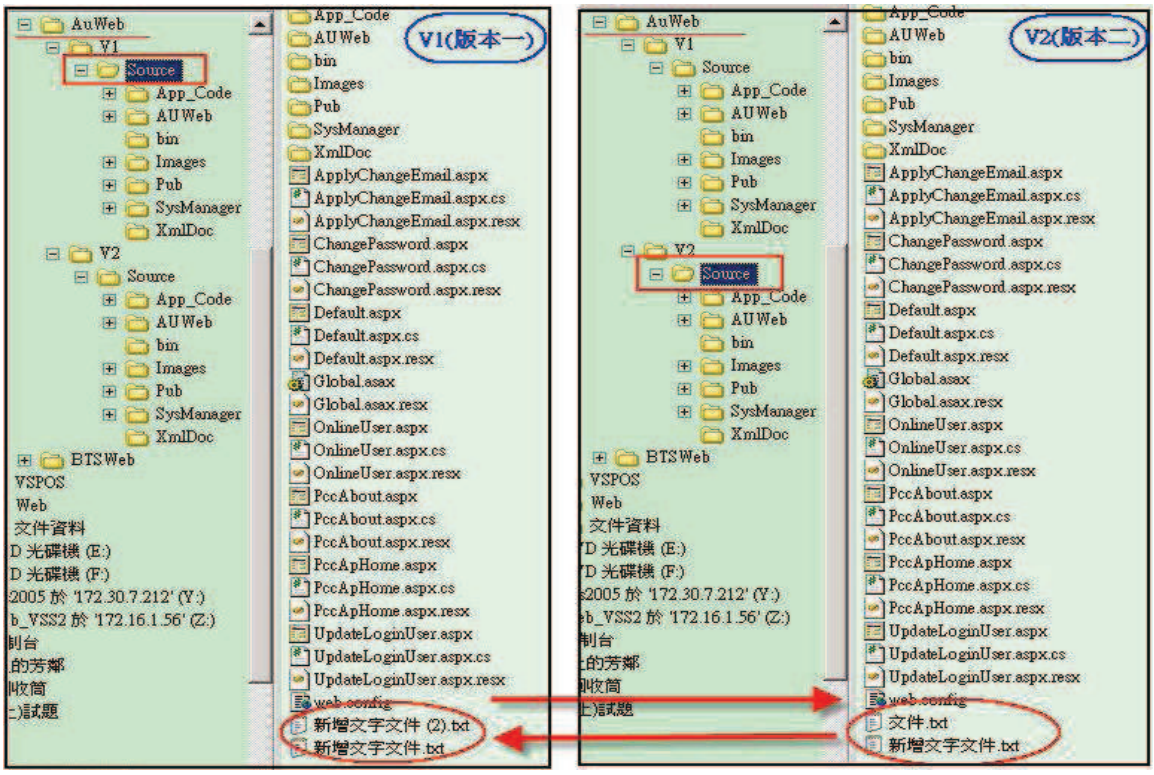

圖 30 實驗測試後備份結果(一)

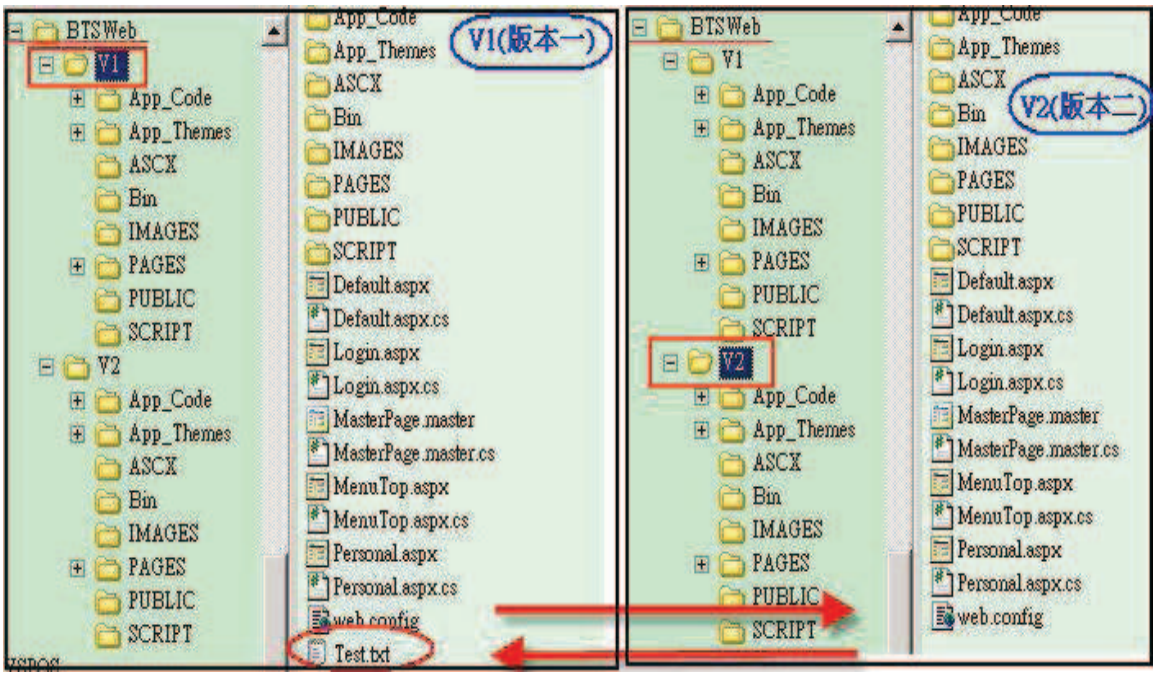

圖 31 實驗測試後備份結果(二)

部署至目的主機的實驗結果，圖 32 為部署至目的主機 172.23.172.41 後的結果，與目標主機的參數 Name=AuWeb 的 V2 版本二(圖 30)的檔案內容一致，故實驗結果顯示，當使用者進行執行的操作時，SDS 則會將來源主機的檔案或文件部署至所目標主機，或將目標主機的檔案或文件部署至所指定的目的主機上，故 SDS 的雙向部署機制是可行的。

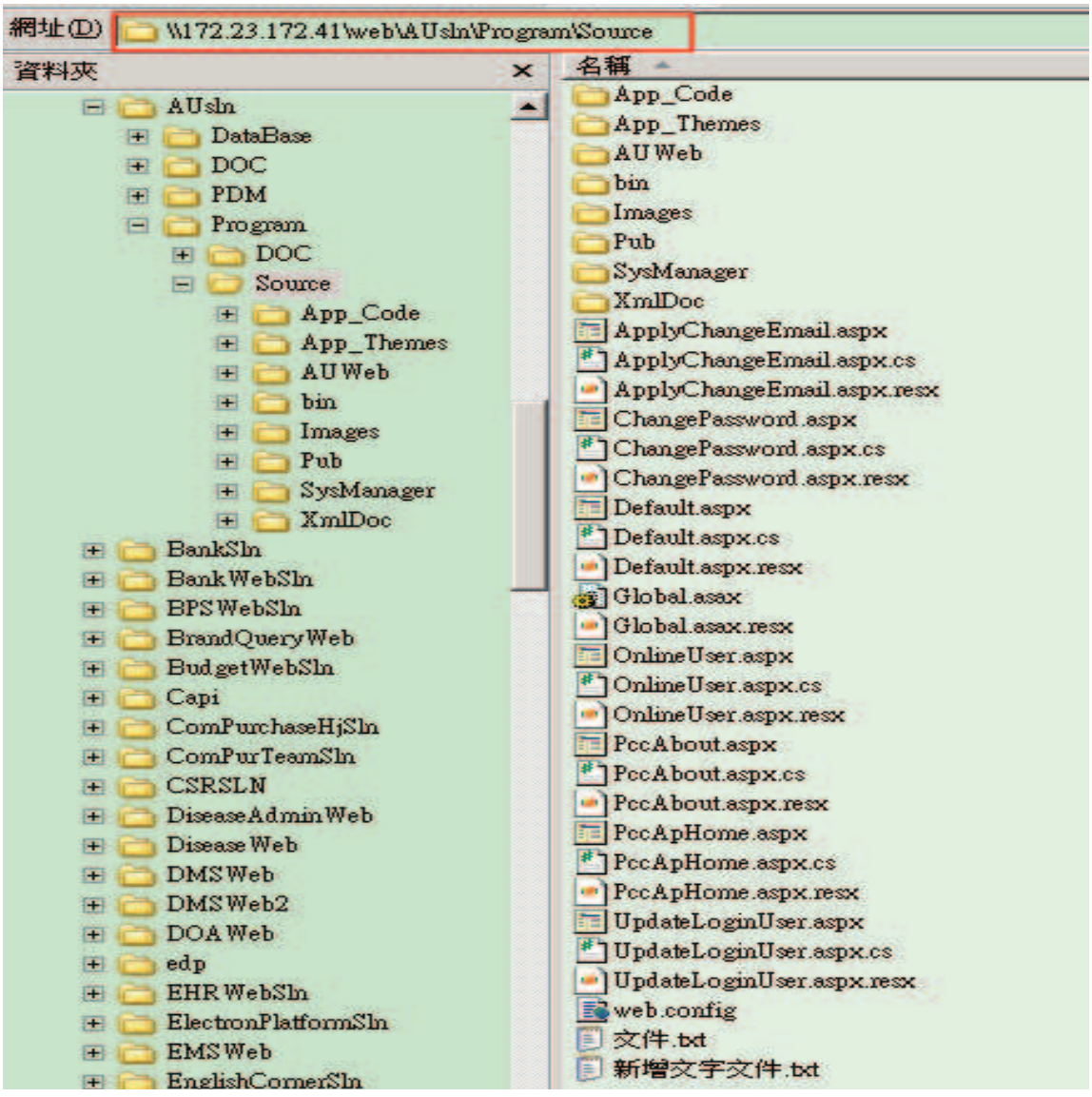

圖 32 實驗-部署至目的主機 172.23.172.41

6.3 優劣分析

SDS 系統優點方面，利用簡單易懂的物件導向程式設計的方式完成實作開發，使用介面容易上手，且使用端不需安裝任何程式，只需將程式安裝在目標主機，以統一集中管理檔案，達到版本控制與統一部署的功效。而系統缺失一，當系統進行版本比對時，因檔案檔案愈大或愈多的時候，其比對的時間就會愈慢，故需加強版本比對時的運算方式，以加快比對的速度。系統缺失二，沒有考慮到檔案愈大或愈多的時候，網路傳送檔案速度愈慢，可能造成網路塞車，導致網路癱瘓等問題，故要加強在檔案傳送時，採用分散處理機制以改進系統檔案傳送塞車問題，使系統更加完善。

第七章　結論與未來展望

7.1 結論

本研究提出的實作內容，主要是依據目前現有的需求而進行設計，利用物件導向的程式設計為基礎來設計 IDE 介面並引用 XML 的概念，建立具備簡單操作的 IDE 介面，不會增加使用者(開發人員)的負擔，對於在系統分析、規劃、開發、測試及維護階段皆有一定的影響力與便利性，且著重在程式碼及文件備份的重要性，降低部署的複雜度，降低部署的成本，提高部署的準確度，並提升開發人員對系統的生產力，增加系統開發之維護性，或解決不完整的更新部署程式碼或文件所造成的系統問題。

SDS 具有雙重機制的概念：(1)單純版本控管備份機制，(2)程式及文件同步部署的機制，依使用者的需求或目的使用此實作系統，其具有相當大的彈性空間與調配空間，並提供開發團隊更簡單實用且無負擔的系統。

7.2 未來展望

目前的設計只是一個初步的設計雛型，希望有更多的使用者給予

評價，讓此實作可以繼續延伸，期望未來朝系統自動化與人性化的方面繼續研究。

參考文獻

[1] Andrea De Lucia, Fausto Fasano, Giuseppe Scanniello, and Genoveffa Tortora, “Concurrent Fine-grained Versioning of UML Models,” 2009 European Conference on Software Maintenance and Reengineering © 2009 IEEE.

[2] Phanarut Srichetta Department of Computer Science Faculty of Science, Mahidol University, and Songsri Tangsripairoj Department of Computer Science Faculty of Science, Mahidol University, “Using Social Network Analysis to Better Understand the Dependencies in UML Use Cases Collection,” © 2009 IEEE.

[3] Tien N. Nguyen, Ethan V. Munson, John T. Boyland, and Cheng Thao Computer Science Department University of Wisconsin-Milwaukee , “Multi-level Configuration Management with Fine-grained Logical Units,” Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications,Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications,© 2005 IEEE.

[4] Steven P. Reiss, “Incremental Maintenance of Software Artifacts,” IEEE TRANSACTIONS ON SOFTWARE EGINEERING, VOL. 32, NO. 9, SEPTEMBER 2006 .© 2006 IEEE.

[5] Orla Greevy, Tudor Gˆırba1 and St´ephane Ducasse, “How Developers Develop Features,” 11th European Conference on Software Maintenance and Reengineering © 2007 IEEE.

[6] Marco D'Ambros," Supporting Software Evolution Analysis with Historical Dependencies and Defect Information," Authorized licensed use limited to: Feng Chia University. © 2008 IEEE.

[7] 張錦堂，XML 入門與 QAML 應用實例，中央研究院計算中心，2000 年 4 月。

[8] 成功大學機械工程研究所虛擬實境與多媒體研究室，Microsoft Visual Studio .NET 平台簡介。

[9] 陳會安，Java 2 與 UML 物件導向程式設計範例教本，2005。

[10] 曹祖聖等，Visual C# .net 程式設計經典，松崗。

[11] 南港 IC 設計育成中心,「認識版本控制系統- 建置 Subversion 為例」,2009 年 11 月。

[12] 臥龍小三，台南縣教育網路中心，CVS 入門說明。

[13] 蔡煥麟-版本控制系統的基礎觀念。

[14] Ben Collins-Sussman Brian、W.Fitzpatrick、C.Michael Pilato，Subversion 權威指南，2002。

[15] 林弘倫,「Web Service 於防救災工程之應用與探討（以基隆市為例）」，國立臺灣科技大學營建工程系碩士論文，2007 年 1 月 30 日。

[16] 吳仁和,「物件導向系統分析與設計 結合 MDA 與 XML」，智勝文化，2005 年。

[17]Rick Jelliffe 撰，張錦堂譯，XML 簡介，1999 年。

[18] 陳長念、陳勤意，「活用 XML：XML by Example」，知城數位，2001 年 6 月。

[19] 黃中杰、洪菁懌 著，「JAVA 與 XML 技術手冊」，2002 年。

[20] 林建宏，「Java、XML 與資料庫之技術探討與整合應用」，2006 年。

[21] 譚浩強，「C++物件導向程式設計」，清華大學出版社，2006 年 1 月第一版。

[22] 陳會安，XML 網頁製作徹底研究，旗標科技圖書股份有限公司，民國 92 年。

網站：

Object Oriented Programming，中科永聯高階技術培訓中心。

Microsoft Visual Studio .NET 平台簡介，http://vr.me.ncku.edu.tw/xms/content/show.php?id=448

維基百科，http://zh.wikipedia.org/zh-tw

微軟 MSDN，http://msdn.microsoft.com/zh-tw/default

Rick Jelliffe 撰，張錦堂 譯 XML 簡介，1999 年，http://xml.ascc.net/zh/big5/docs/xml-intro.txt

成功大學機械工程研究所虛擬實境與多媒體研究室，http://vr.me.ncku.edu.tw/xms/content/show.php?id=448

陳昭珍，「XML, Metadata 與檔案資料數位化」，檔案管理運用研討會，http://archives.sinica.edu.tw/main/seminar06.html，2001 年 7 月

簡西村，「服務導向架構應用」，http://www.microsoft.com/taiwan/msdn/columns/soa/SOA_overview_2004112901.htm，2004 年

Subversion 介紹，http://svndoc.minitw.com/svn.intro.whatis.html

臥龍小三 ols3@www.tnc.edu.tw 台南縣教育網路中心，CVS 入門說明，http://www.csie.nctu.edu.tw/~tsaiwn/course/introcs/history/linux/linu

x.tnc.edu.tw/techdoc/cvs/book1.html

Version control with Subversion，

http://svnbook.red-bean.com/

http://svn.stu.edu.tw/svnbook/book.html

Subversion 官方網站，

http://subversion.tigris.org/

蔡煥麟，「版本控制系統的基礎觀念」，

http://huan-lin.blogspot.com/2009/04/introduction-to-version-control-system.html

附錄 A 系統之函式：

表 A-1 本研究 SDS 系統提供之程式名稱

程式名稱	函式名稱
DestinationFrm.cs	將本機(中介端)資料配置/部署至目的端-(M:1)
MainFrm.cs	主選單畫面
ReportFrm.cs	報表比對
RunFrm.cs	執行管理
SourceFrm.cs	將其它(來源端)資料配置/部署至本機(中介端)-(M:1)

表 A-2 本研究 SDS 系統提供之功能函式

程式名稱	函式名稱	函式說明	輸入值	輸出值
DestinationFrm.cs	DestinationFrm	Main	-	-
	DestinationFrm_Load	Load 初始化	-	object sender EventArgs e
	btnPath_Click	選擇來源路徑	-	object sender EventArgs e
	btnDestination_Click	選擇目的路徑	-	object sender EventArgs e
	btnOK_Click	複製	-	object sender

				EventArgs e
	InsertXML	複寫至 XML 檔	-	-
	initGridView	初始化 GridView	-	-
	ClearControl	清除欄位資料	-	-
	Dgv_Data_CellMouseClick	點選 GridView 時的任一列記錄時	-	object sender DataGridViewCellMouseEventArgs e
	btnDel_Click	刪除 GridPath	-	object sender EventArgs e
	btnAdd_Click	新增	-	object sender EventArgs e
	btnCancel_Click	取消	-	object sender EventArgs e
MainFrm.cs	Main	Main	-	string[] args
	MainFrm	Load	-	-
	SourceToolStripMenuItem_Click	設定來源	-	object sender EventArgs e
	DestinationToolStripMenuItem_Click	設定目的	-	object sender EventArgs e

	logToolStripMenuItem_Click	查看記錄-版本比對	-	object sender EventArgs e
	runToolStripMenuItem_Click	執行	-	object sender EventArgs e
ReportFrm.cs	ReportFrm	Main	-	-
	ReportFrm_Load	Load 初始化	-	object sender EventArgs e
	initDirData	初始化目前的目錄內容	-	-
	initDirDataV	初始化目前的目錄內容(備份)	-	-
	btnOK_Click	進行比對	-	object sender EventArgs e
	initGridView1	初始化 GridView(版本一)	-	-
	initGridView2	初始化 GridView(版本二)	-	-
	DirSearch	遞迴搜尋	-	DataTable DT_Data string sDir
	CompareGridVie	比較 GridView1 與	-	-

	w	GridView2		
	DelGridViewData	將沒有變化的資料從 GridView 中移除	-	-
RunFrm.cs	RunFrm	Main	-	-
	RunFrm_Load	Load	-	object sender EventArgs e
	initGridView	初始化 GridView	-	-
	initGridView1	初始化 GridView	-	-
	CopyFolderFile	複製與與備份	-	string strSource string strDestination
	FileMove	搬移	-	-
	FileDel	刪除	-	string strSource string strDestination
	btnOK_Click	執行	-	object sender, EventArgs e
	CopyDirectory	複製檔案與資料夾		string SourceDirectory, string TargetDirectory
SourceFrm.cs	SourceFrm	Main	-	-

	SourceFrm_Load	Load	-	object sender EventArgs e
	btnPath_Click	選擇來源路徑	-	object sender EventArgs e
	btnDestination_Click	選擇目的路徑	-	object sender EventArgs e
	btnOK_Click	複製	-	object sender EventArgs e
	InsertXML	複寫至 XML 檔	-	-
	DirSearch	遞迴搜尋	-	string sDir
	CopyFolderFile	複製	-	-
	FileMove	搬移	-	-
	FileDel	刪除	-	string strSource string strDestination
	initGridView	初始化 GridView	-	-
	ClearControl	清除欄位資料	-	-
	Dgv_Data_CellMouseClick	點選 GridView 時的任一列記錄時	-	object sender DataGridViewCellMouseEventArgs e

	btnDel_Click	刪除 GridPath	-	object sender EventArgs e
	btnAdd_Click	新增	-	object sender EventArgs e