

SAWA: 支援單一登入及 MVC 樣式的校務行政系統應用程式框架

廖峻鋒 李蔡彥

國立政治大學電子計算機中心

{try, li}@nccu.edu.tw

摘要

各級學校資訊化後，校務行政系統漸成為行政業務中樞。隨著系統功能日漸擴增，若不建立適當的軟體架構，Butterfly 效應[1]將會日趨明顯，使得系統難以維護。我們利用設計樣式(Design Patterns)及框架(Framework)的概念，結合過去建置 Web 校務行政系統的經驗，發展出一個以 MVC(Model-View-Controller)為基礎，支援單一登入(Single Sign-on)的校務行政系統 Web 框架。目前政治大學以 Web 為界面的校務行政系統即建置在此框架之上。我們發現在這個框架基礎之上，開發人員可以輕易地在短時間內發展出具有功能完整、風格一致且容易維護的應用程式。

關鍵詞：框架，校務行政系統，MVC，單一登入

Abstract

As school administrative information systems become a helpful and critical in processing daily school affairs, the size and number of these systems also grow rapidly. If these systems are not built on top of an appropriate software architecture, they could easily become unmanagable due to the butterfly effect [1]. In this paper, we propose a software framework based on our development experiences as well as Object-Oriented techniques such as design patterns and frameworks. Most of web-based administrative systems in NCCU were developed based on this framework which natively supports the MVC pattern and Single Sign-on facility. We found that this framework can increase the productivity of software developers as well as the quality and maintainability of these systems.

1. 前言

國內多數大學院校的校務行政系統都是由電算中心人員自行建置或以委外方式開發。系統的需求則由相關行政業務負責人提出，不同的開發人員再依需求量身訂做。時間長久之後，整個校務行政系統變得小系統林立，造成系統維護上的嚴重問題。隨著校務決策的改變，分散在各系統相關的程式碼都要一一取出並加以修改，不但浪費寶貴人力，也容易造成錯誤。

大部份校務行政系統的行為都十分類似，以

Web 上的校務行政系統為例，都會有登入、登出動作，都需要防止使用者不當存取的資源管制(Access Control)，也都要做錯誤的處理(Error Handling)。同時整個校務行政系統通常會要求有一致的外觀及操作界面。應用程式林立的結果還會讓使用者每使用單獨一個系統，皆需重複做登出、登入的動作。因此，許多使用者也會要求校務系統只需登入一次便能使用所有應用程式。

近年來軟體的重用技術已成為軟體工程學門研究的重點[1]。我們利用設計樣式(Design Patterns)及框架(Framework)的觀念，以建置 Web 校務行政系統經驗為基礎，從中解析出抽象類別，並定義這些類別之間的互動，加上自訂伺服器端控制項(Custom Server Controls)及公用函式庫(Class Library)共同組成了一個支援單一登入(Single Sign-on)的校務行政系統 Web 應用程式框架。我們稱這個框架為 SAWA Framework(School Administrative-affair Web Application Framework)。

SAWA Framework 使用 J2EE[2]的前端技術(JSP 1.2/JSTL 1.0 與 Servlet 2.3)，以 Framework-Recipe 方法[3]為基礎，建置一個支援單一登入及 MVC(Model-View-Controller)樣式的校務行政系統 Web 框架。基於這個框架，開發人員上可以在短時間內開發出具有 MVC、單一登入、一致外觀、有效資源管制及結構化錯誤處理機制的應用程式。

校務系統開發人力通常來自學生、約聘人員或外包廠商，流動性較大，這也是造成各子系統間不相容、不易維護的原因之一。我們發現藉由框架的協助，不同程度的開發人員所產出之程式碼也能具有一致風格及品質，並有助於軟體穩定性及可維護性的提升。

2. 研究背景與文獻探討

2.1 框架(Frameworks)

2.1.1 框架的定義

框架是一種重用的技術，通常會由一組抽象類別及這群抽象類別的互動來描述[4]。框架本身是一個半成品，設計人員將可重用的設計製成框架，再由程式開發人員將這個半成品客製化成可用的應用程式。

框架通常會針對特定的領域開發，讓成功的設計可以被重覆使用。設計人員將一個領域的應用系統，切分成若干個抽象類別，並定義各個類別的責任(responsibility)與其相互間的合作關係來提供新系統架構設計的指引。開發者可繼承這些類別並建立類別之間的關連來建構新系統[5]。

框架與傳統函式庫(Library)皆屬於軟體重用技術。其最大不同在於「實作和流程控制角色的互換(inversion of control)」，這也是建立框架時最重要的技巧[1]。

2.1.2 建構框架的方法

建構框架方法論有很多種，一般將其分為由舊系統重構(Refactoring Approach)及重新建構(Priori Approach)二類。

由舊系統重構適用於原本就有即存系統正在運作的單位，由開發人員將可重用的部份抽象化再建立框架。以校務系統為例，建構框架前，大部份都會有許多即存的系統，故較適用於由舊系統重構方式。

若想將原本不存在的軟體建立框架，則較適合重新建構的方法。依一般開發流程，由使用案例(Use-case)開始，再將使用案例抽象化(Use-case assortment)，找出系統變異點(hot-spot)，這些變異點便成為建置抽象類別的候選對象[6]。

2.1.3 Framework Recipe

Framework Recipe 是建構框架的一種方法論，主要是以「問題-處方箋」(Problem-Recipe)為思考方式來建來框架。另外在 Framework Recipe 中提供了七個框架常出現的 Patterns 供框架設計者參考[3]。

首先開發人員針對要設計框架的領域(Domain)擬出關鍵的問題(Problem)。針對這些問題，善用設計樣式或其它方法加以解決(Recipe)。將這些 Recipe 加以整合後，建構成一個「框架的基礎結構」。我們還要針對實際情況來加強(refine)，才能成為一個完整的框架。

2.2 Web 應用程式之 MVC 樣式與相關實作

MVC 樣式在 1970 年代在 smalltalk 程式設計中曾被廣泛使用。適當地使用 MVC 樣式可以讓 Web 應用程式外觀和邏輯的程式碼分開，讓系統更容易維護。採用 MVC 樣式開發 Web 應用程式已成為 Web 開發社群的共識[7]。

Sun 官方的 J2EE Patterns Catalog 中，和前端(Front-end)有關的 Pattern 共有六種。其中 Front Controller 及 Services to Worker 是開發人員實作 MVC 的重要參考[8]。但 Front Controller 及 Services to Worker 類別分割較細，對規模一般為小至中型專

案的校務系統來說並不好用，有疊床架屋的感覺。

Apache 的 Jakarta 專案中發展了一個 Struts Framework，提供完整且相當具有彈性的 MVC 開發框架 [9]。但其學習曲線也相當高，對於人力流動性高的開發環境來說並不合適。另外 Struts 並非官方的標準框架，所以直接在 Struts 上開發的程式碼並不能保證將來規格改變時原有程式仍可正常運作。

HansBergsten 則提出一種較簡化的 MVC 實作方式[10]，他以一個 Controller Servlet 類別來實作 Front Controller 及 Service to Worker 中 Controller 及 Dispatcher 這二個類別的功能。其建議做法如圖 1。

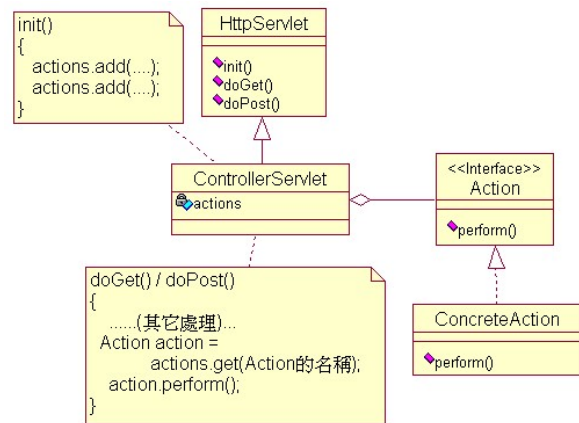


圖 1. Hans Bergsten's MVC 類別圖

在 Hans Bergsten 建議的模型中，所有 URI 結尾為.action 的要求都會交由 Controller Servlet 來處理。Controller Servlet 中維護了一個稱為「Action Table」的資料結構，記錄「Action 指定字串」與「Action 物件」的對應關係。這樣就完成了一個最精簡的 MVC 運作方式。

Hans Bergsten 提出的 MVC 實作方式，一方面容易學習，同時也符合 MVC 的精神。我們認為這個 Pattern 對於小型到中型的 Web 應用程式而言極為合適，而且也很容易被程式開發人員瞭解及採用。

3. 校務行政系統所面臨之問題與需求

根據過去的開發經驗，我們發現大部份校務行政系統都具有以下問題及需求：

- **風格一致的使用者界面(User Interface):**校務行政系統會不斷的成長及修改，如果每支系統都有各有一種界面及操作方法，不但造成使用者操作上的困擾，也造成整個系統看起來沒有整體感。若校務系統屬於外包形式，這種情況可能更加嚴重。因此校務系統需要有風格一致的使用者界面。新增的應用程式應以元件方式嵌入整個系統，而且其使用者界面應該要與既有系統一致，嵌入的過程不應造成程式開發人員

額外的限制及負擔。

- **如何協助開發人員以有效率地以 MVC 樣式來開發 Web 應用程式:** Web 的應用程式中實作 MVC 要處理的細節很多，若無特殊的機制協助，可能造成開發人員無法專注於真正要處理的邏輯上，對開發過程反而造成負面影響。我們希望能有一種容易學習的機制來協助開發人員處理 MVC 重覆的細節部份。
- **必須支援單一登入 (Single Sign-on):** 應用程式之間如缺乏單一登入的機制，會讓使用者每使用單獨一個系統，皆需重複做登出、登入的動作。因此，許多使用者也會要求校務系統必需登入一次就能使用所有應用程式。
- **登入機制的抽象化:** 幾乎進入所有的系統的第一步都是登入並取得使用者的資訊，應該將登入機制加以分析，將共同部份適當地抽象化，降低開發時的負擔。
- **管制資源的不當存取:** 許多設計不良的應用程式可能會忽略資源存取的管制。程式必須確保除了登入的合法使用者之外，系統資源沒有其它任何的進入點。但若在開發時期就加上管制，常造成除錯的困難。所以需要一種經由設定檔就能動態加上或解除管制的機制。
- **結構化的錯誤處理:** 當應用程式發生錯誤時，程式必須向使用者說明錯誤的原因及解決方法。如果沒有一定的規範，每個開發人員所寫的錯誤處理網頁都不同，容易造成使用者的困擾。

4. SAWA Framework 組成元件探討

4.1 系統概觀

針對校務行政系統所面臨的問題與需求，我們採用 Refactoring Approach[4]及 Framework Recipe[3]的方法來建構整個框架。對上節探討的六項問題，我們建議了六個 Recipe 來解決這些問題，這六個 Recipe 也構成了 SAWA Framework 的六個基礎元件，如表 1 所示：

表 1. SAWA 框架的 Problem-Recipe 列表

Problem	Recipe
風格一致的使用者界面	以 JSP Custom Tag Library 為基礎，建構 Server Controls 來覆寫 (override)部份 HTML 標籤之功能。
協助開發人員處理 MVC 細節	以 Huns MVC[9]為基礎，重構後得到 MVC Helper。
登入機制的抽象化	以 Template Method Pattern[10]，實作 LoginHandler。
管制資源的不當存取	使用 Interception Filter[7]，達到彈性的存取管理機制。
結構化的錯誤處理	開發人員以 xml 設定檔定義錯誤列表，錯誤發生時由框架負責轉換成結構化的錯誤訊息畫面，給予使

	用者具體的協助。
單一登入	建構在整個框架上，透過加密 cookies 及 ApplicationBridge 機制達成單一登入的功能。

在以下各節當中，我們將針對每個元件的功能及設計詳細說明。

4.2 核心樣式--MVC Helper

使用 HansBergsten 的 MVC 開發方式來開發 Web 應用程式時，雖然實作方式很容易了解，卻不容易做到程式碼的重用(reuse)。每次實作該模型，都會進行以下重覆的動作:寫作這個 Web 應用程式專用的 Controller，並在 Controller 的 init()方法中註冊所有 action 類別。

其實每一支 Web 應用程式的 Controller 大部份行為皆相同。不同的只有「註冊 Action 類別」部份。因此可採用 Template method [11]，將邏輯相同的程式碼抽象化，成立一個新的抽象類別 (Abstract Class)，稱為「AbstractController」。此類別使用一個 doRegister()的抽象方法，在 init()中被呼叫。ControllerServlet 初始化時，會透過 init()進行註冊動作(doRegister());至於如何註冊，則交給子類別來定義。

另外，Hans 只利用 Action 抽象介面定義每一個 Action 子類別應有的行為。在實際應用上，Action 的子類別有更多共同點(如偵測 web.xml 中所設定的初始化參數)，但這些共同點又不適合做為 Action 界面的一部份(如參數讀取功能本身和 Action 無關，不適合放在 Action 界面中)。利用 Framework Recipe 中提供的「Placeholder」pattern[3]，在 Action 界面及 ConcreteAction 之間加入了 AbstractAction 這個抽象類別，用它來存放與 Action 無關，但又是大部份 Action 會有的共同功能，成為一個 Placeholder。

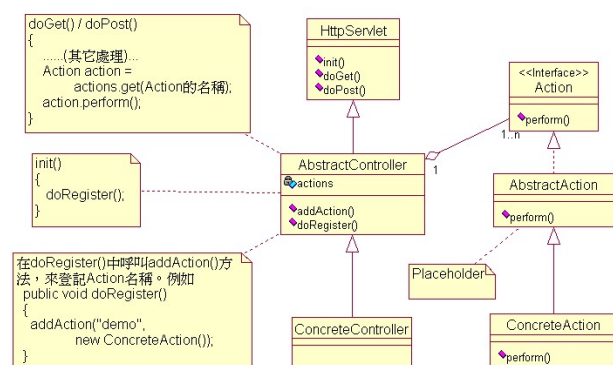


圖 2. MVC Helper 類別圖

經過二個階段重構後，可得到一個新 pattern 稱為「MVC Helper」，這也是 SAWA Framework 中的核心部份，其類別圖(Class Diagram)如圖 2 所

示。

4.3 提供一致性的外觀—Server Controls

提供一致性外觀的解決方案可歸納為六大類：剪貼、Server Side Include、Template Framework、JSP Include、自訂標籤 (Custom Tag Library) 及 Filter[12]。在 MVC 中，View 的部份(使用者界面)應以 JSTL 或自訂標籤方式取代 scriptlet。因此我們採用 Custom Tag Library 做為提供一致性外觀的解決方案。

SAWA 所提供的自訂標籤庫類似 ASP.NET[13] 中 Server Control。Server Control 標籤庫由框架提供，覆寫並擴充原有 html 標籤的功能。開發人員以 <nccu:html>、<nccu:body> 等自訂標籤取代原有之 <html>、<body> 標籤(如圖 3)。當此頁面在伺服器執行時，框架內的 Tag Library 會自動加上 Header、Footer、CSS 等一致性外觀。(如圖 4)。

```
<nccu:html>
  <nccu:header pageTitle="網頁標題" contentType="text/html"
  encoding="big5">
    <%-- html 標頭區，有其它標頭設定或java script可放在這 --%>
  </nccu:header>
  <nccu:body apname="行政系統"
  userName="${sessionScope.chname}"
  message="${sessionScope.apname}" >
    <%-- 在這裏開始寫JSP --%>
  </nccu:body>
</nccu:html>
```

圖 3. 使用自訂標配合 JSTL 寫作 JSP

透過 Server Controls，我們將系統的美工與程式部份清楚地切開，一旦校務系統的外觀需要修改，除 Server Control 外，其它程式完全不需要更動。

圖 4. Server Control 為應用程式提供一致的外觀

SAWA 提供的 Server Control 同時可完整支援 JSTL 1.0 的 Expression Language(如圖 3 中之 \${sessionScope.chname} 等敘述)。因此對開發人員來說，除了 namespace 不同，使用 Server Control

和使用 JSTL 的其它標籤感覺沒有差別，不必額外花時間學習。

4.4 登入機制抽象化—Login Handler

LoginHandler 主要功能是將登入功能抽象化，並整合至「單一登入」(詳見 4.7 節)及即存的「職務代理人機制」。

檢視所有 web 應用程式的登入邏輯，可發現登入程序其實大部份是一樣的，差別在於「登入成功了要做什麼?」「代理權限不符怎麼辦?」「登入失敗了要做什麼?」的處理(例如可能是轉到登入失敗的頁面)。所以我們使用 Template method [11]設計樣式，將這些不同的地方留給子類別，而其它則在抽象類別中處理。(如圖 5 所示)

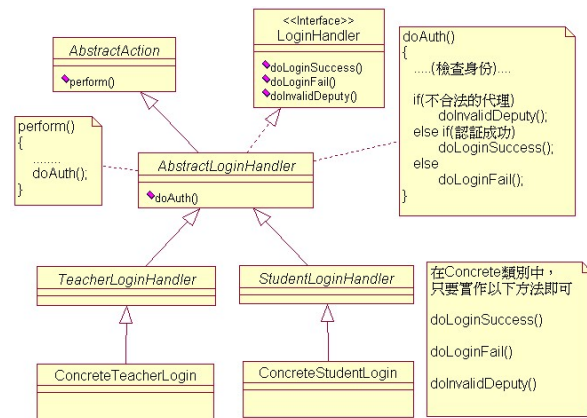


圖 5. LoginHandler 的類別圖

在實際的情況中，不同身分的使用者，帳號及密碼都存在不同的資料表中，所以登入邏輯也不同，因此我們另外還會為各種身份實作一個抽象類別。

4.5 管制資源的不當存取—Access Filter

傳統防止使用者跳過認證頁直接指到某頁的方法都是在處理 Http 要求時，一開始便檢查 session 中是否有認證資料。但只要檢查權限的邏輯一改，則每處都必須跟著修改。另外在程式在開發時期，有時候不想加這個限制，比較容易除錯。

J2EE Pattern 建議了我們使用 Filter 來做 Access Control[7]，藉由改變 web.xml 的設定，可以隨時將 filter 裝上或卸下。在 web.xml 中可以事先設定，那些要求會被 FilterServlet 攔截。攔截後加以驗證，如果不合法，就導到登入頁要求登入。

AccessFilter 依照這個原則實作，將以下二件事交由子類別設定：

- 資源不當存取時要重導到那個錯誤頁面。
- 那些頁面不需要管制。

LoginHandler 的類別圖如圖 6 所示。

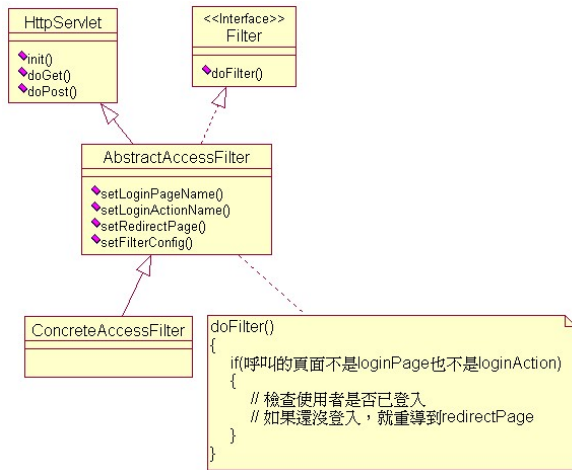


圖 6. AccessFilter 的類別圖

4.6 結構化的錯誤處理—ErrorPage Builder

依照過去開發的經驗，我們將錯誤資訊分成四個部份：

- **錯誤標題：**以一句話簡述錯誤情況。
- **錯誤 ID：**用來識別這個錯誤。
- **錯誤描述：**向使用者簡述錯誤發生的原因及復元的可能性。
- **解決方案：**以超連結方式引導使用者回報錯誤或重試。

```

<ErrorItem>
  <id>500</id>
  <title>系統內部目前發生錯誤，暫時無法提供服務</title>
  <detail>.....</detail>
  <recipes>
    <recipeItem>
      <solution>回前一頁</solution>
      <url>javascript:history.back()</url>
    </recipeItem>
    <recipeItem>
      <solution>登出</solution>
      <url>http://moltke.cc.nccu.edu.tw/SSO/doLogout</url>
    </recipeItem>
  </recipes>
</ErrorItem>

```

圖 7. 錯誤網頁的設定檔

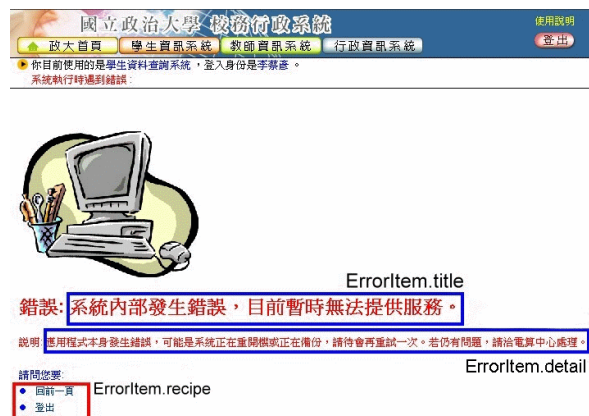


圖 8. 系統依據 xml 檔建立的錯誤網頁

SAWA 框架的開發人員在開發時期不需要寫作錯誤網頁，他們只要事先在 xml 檔案中設定(如圖 7)，給予 ErrorPage Builder 錯誤標題、錯誤描述及解決方案的資訊，系統會自行建立錯誤的網頁(如圖 8)。ErrorPage Builder 在 Web 應用程式初始化時會讀入 error.xml 設定檔，並使用 SAX API 解析此設定檔後，將之轉換成錯誤資訊的網頁。

4.7 單一登入的實作—Application Bridge

單一登入機制指的是使用者在某支 Web 應用程式登入後，便不需重覆登入即可使用他有權使用的應用程式。

實作單一登入面臨到最大的考驗是如何跨 Web 應用程式分享使用者的登入資訊。登入資訊一般利用 Session 儲存，但在 Servlet 2.3 的規格書中明訂 Session 不可跨 Web 應用程式分享[14]。另外，整個單一登入機制必須是可擴充的，以便新加入的應用程式隨都可以單一登入的功能。

我們以生命週期和 Session 一致的加密 cookie 來解決這個問題。使用者登入後，一個記錄登入資訊的加密 cookie 會送到 Client 端，其生命週期與 Session 一致；關閉瀏覽器時，cookie 也同時失效。另外，ApplicationBridge 的機制使單一登入系統具有可擴充的彈性。ApplicationBridge 是一個 Servlet，它會讀出 cookie 並加以解密，再將資訊橋接到框架中的 LoginHandler 中，自動為使用者完成登入程序。而單一登入系統本身就是一個建構在 SAWA 框架上的 Web 應用程式。

5. 實作與討論

目前政治大學 Web 上的校務行政系統均以 J2EE 技術為基礎，建構在單一的校務行政系統框架上。新增之專案在完成系統分析，進入設計階段後，開發人員藉由校務行政系統框架的幫助，以其提供之抽象類別為基礎，訂製出該系統的 Controller 及 Action 類別，建立整個系統的基礎結構。

使用者界面則以 JSTL 1.0(Java Standard Tag Library)為主，利用校務行政系統框架所提供的 Server Controls 提供一致的外觀。待整個系統完成之後，再修改設定檔，設定 AccessFilter 之管制範圍，將整支子系統嵌入校務系統中。

我們發現這種開發方式具有以下優點：

- 由於每支應用程式均基於相同的框架開發，所以程式碼整體的品質提高。
- 不同開發人員開發之應用程式網頁美工及錯誤處理風格均能保持一致。
- 框架中幫助開發人員處理許多細節，因此提高了系統開發的效率。
- 系統以 MVC 方式清楚地切割，所以程式除錯及維護較容易。

- 資源管制功能由框架統一處理，因此不用擔心開發人員寫程式時造成管制漏洞。
- 一致且清楚的錯誤及解決方案說明，幫助使用者了解系統失效的原因及初步解決方案。有效降低了使用者諮詢服務的負擔。
- 框架扮演了設計規範的角色，降低外包風險。

另外我們也發現有以下的困難及問題需要克服：

- 開發人員需具備一定程度的物件導向觀念及一段的時間訓練與學習。
- 所有的應用程式均基於框架中的類別或元件，因此框架的正確性及穩定性十分重要。框架程式碼的維護人員責任重大，流動性也不宜太高。

因此，我們建議建立框架的角色應由經驗較豐富、流動性較低的資深工程師擔任。在適當的文件及訓練輔助下，應用程式框架才能發揮最大效益。

6. 未來延展方向

未來我們會針對以下各點對 SAWA Framework 的功能加以擴展。

- **建置各項行政業務的框架：**校務行政雖然十分繁瑣，但大部份業務均重複性高且有一定的邏輯。我們希望針對各單位的業務加以抽象化。再以這些抽象化的類別擴充框架的功能。在應付新的需求時，開發效率才能更加提昇。
- **與目錄服務整合：**未來校內之人員及各項設備將建置目錄服務，因此也必須考量透過 JNDI 與目錄服務整合。
- **.NET 平台上的解決方案：**微軟所提出的 .NET Framework 是另一個重要的開發平台。我們正在 .NET 平台上開發類似的解決方案，目前 MVC Helper 已在 .NET 平台上成功實作[15]，成為 TANET2003 研討會系統的骨架。

7. 結論

針對校務行政系統所面臨的問題與需求，我們使用 Framework Recipe，在程式開發的經驗及設計樣式(Design Patterns)的輔助下，建立了一個以 MVC 為基礎，支援單一登入(Single Sign-on)的應用程式框架，並將此框架實際應用在政治大學的校務行政系統。這個框架的建立讓開發人員在短時間內可發展出具有完整功能、風格一致且容易維護的應用程式。未來我們希望在行政業務部份也都能加以抽象化，並加入支援目錄服務之功能，使 SAWA Framework 的功能更完整。

8. 參考文獻

- [1] D. Govoni., *Java Application Frameworks*, Wiley Computer Publishing, 1999.
- [2] Java 2 Enterprise Edition Technologies Home, <http://java.sun.com/j2ee/>.
- [3] S. R. Jones, "A Framework Recipe," *Building Application Frameworks - Object-Oriented Foundations of Framework Design*, Wiley Computer Publishing, Chap. 10, pp. 237-266, 1999.
- [4] M. E. Fayad, D. C. Schmidt and R. E. Johnson, "Application Frameworks," *Building Application Frameworks - Object-Oriented Foundations of Framework Design*, Wiley Computer Publishing, Chap. 1, pp. 1-28, 1999.
- [5] 陳泓志, 物件導向企業框架之研究, 國立政治大學資訊管理學系論文, 民國八十九年九月。
- [6] P. Wolfgang, "Hot-Spot-Driven Development," *Building Application Frameworks - Object-Oriented Foundations of Framework Design*, Wiley Computer Publishing, Chap. 16, pp. 379-393, 1999.
- [7] K. Duffey, et al., "Component-Driven Web Programming," *Professional JSP Site Design - Coding Core Web Applications*, Wrox Press Ltd. Chap. 1, pp. 7-28, 2001.
- [8] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns - Best Practices and Design Strategies*. Prentice Hall PTR, 2002.
- [9] Apache Jakarta Struts Framework Project Home. <http://jakarta.apache.org/struts/index.html>
- [10] H. Bergsten, "Combining JSP and Servlets," *JavaServer Pages - Help for Server-Side Java Developers, 2nd Edition*, pp. 338-376, O'reilly, 2002.
- [11] E. Gamma, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Longman, 1995.
- [12] K. Duffey, et al., "Page Layout," *Professional JSP Site Design - Coding Core Web Applications*, Wrox Press Ltd. Chap. 5, pp. 217-266, 2001.
- [13] Microsoft ASP.NET Technologies Home, on MSDN, <http://msdn.microsoft.com/asp.net/>.
- [14] D. Coward, *Java Servlet Specification version 2.3*, Sun Microsystems, 2001.
- [15] C.-F. Liao, *How to Implement MVC Helper Pattern on .NET Platform*, available on <http://java.cc.nccu.edu.tw/try/talks/webframework/MVCHelperDotNetImpl.doc>, NCCU Computer Center, 2003