以資料反向工程技術與 XML
整合傳統資訊系統與全球資訊網之工具

A Tool for Integrating Legacy File Systems with WWW
Based on Data Reverse Engineering and XML

鄭秀麗
屏東科技大學電算中心
showli@mail.npust.edu.tw

葉道明
高雄師範大學資訊教育所
dmyeh@nknucc.nknu.edu.tw

摘　要

隨著二十一世紀來臨及網際網路風行，企業不得不尋求 Internet/Intranet/Extranet **的解決方案，資訊管理人員因此面臨了許多的工作挑戰。因為資訊管理人員必須管理暨維護企業中舊有資訊系統，且同時必須進一步瞭解目前新的軟體技術趨勢對於企業的重要性和影響，及如何引進企業必要的資訊技術和架構，以整合新、舊系統。舊資訊系統軟體之穩定度跟成熟度，往往令資訊部門主管視重寫舊有程式碼為畏途，因此如何做到系統間資料互通的功能，成為不可忽視的議題。本研究運用資料反向工程的觀念與技術將** COBOL **的程式片斷進行分割，透過程式碼模組之分析處理，抽出有用** Physical Schema**，整合傳統資訊系統的資料，顯示在** WWW **的平台上，延伸傳統應用系統原始程式之生命週期，並且減少系統重新開發及資料移轉時所產生不可預期之風險。**

關鍵詞：資料反向工程、XML**、軟體工程、軟體工具、系統整合。**

In the Internet era, most legacy systems seem like burdens rather than assets. After the Y2k crisis, however, the assets in legacy systems are re-evaluated. One of the major reasons is that the universal data access brought by WWW makes data valued more than before, and many data still reside in legacy systems. There are two ways to integrate legacy systems with new systems, one from the component perspective, and the other from the data perspective. The latter excels the former in its ability to effectively handle data. Legacy systems are often huge and complex. To make integration possible, it is vital to gain sufficient knowledge about the legacy system through data reverse engineering. We develop a tool to ease the data-centric legacy integration, which apply data reverse engineering techniques to identify the data architecture in a legacy system, produce code for data retrieval and corresponding XML tags, and generate XML documents. Thus the legacy data assets could be integrated with WWW for broader access through browsers. Moreover, data exchange capability of legacy systems is achieved.

Keywords ： reverse engineering, software engineering, system integration, software tools, xml.

一、 緒 論

隨著網際網路盛行，軟體開發多著重於 Web 介面及網際網路語言之撰寫，但實質上許多企業仍以舊傳統資訊系統的程式碼與資料繼續維持運作著。對一個已穩定運作多年的傳統系統，程式開發暨維護人員多為資訊部門資深人員，為了因應潮流所趨及提昇企業競爭力，勢必面臨兩種選擇：系統重新開發並改寫、維持舊傳統資訊系統並結合新的程式語言技術系統。

選擇系統重新開發並改寫，對資訊部門人員而言，除了要了解舊系統、分析使用者單位作業的需求及新技術作業模式，並且要學習新的網際網路語言、並將舊系統的資料移轉。除了對一個資深資訊人員而言是一項夢魘外，對於企業而言也是一個極大的風險。企業 CEO 跟資訊部門主管應多針對組織內部資訊人員技術、人力評估及舊系統移轉風險性，持審慎評估的態度。

因此選擇維持舊傳統資訊系統並結合新的程式語言技術系統，變成資訊部門人員考量的焦點。雖是疊床架屋方法，但卻能保持舊傳統資訊系統穩定運作、減少風險性，又能因新技術的提昇，進而增加企業競爭優勢。

目前國內在大型傳統系統開發技術上仍普遍使用傳統之 COBOL 語言及集中管理模式，資料多以文數字型態來處理個人資料，應用方面效率偏低且缺乏彈性。隨著個人電腦發達，網際網路盛行，取而代之的是各種 WEB service 程式語言[6][7]，資料也已不限於文數字資料，聲音、影像、圖檔皆可透過網際網路來進行交換，及時提供資料即時分享及提升產品加值。傳統檔案結構資料不獨立，若更動檔案規格或欄位(增減欄位或修改欄位屬性與長度)，都將會造成所有與該檔案結構相關的程式必需也隨之異動，並且要進行重新編譯，對一個龐大繁雜的應用系統，逐一編譯相關程式，不僅太耗時，而且有時也會遺忘對某些相關程式重新編譯跟連結，因而造成應用程式執行上錯誤。因此 COBOL 在系統維護上不僅成本提高，而且在系統維護上缺乏彈性。

而在未來商品多樣化的趨勢下，必定會因 COBOL之維護開發成本及效益不能與目前科技方向密切結合而影響整個資訊系統之整合發展。而其產能不高及人才難覓也會造成企業組織成長瓶頸。但反觀企業既有的 COBOL 程式，經長時間考驗、系統穩定度及成熟度之考驗 可靠的 COBOL程式仍大多繼續支援公司的核心運算。所以為因應未來需求，替更成為網際網路語言雖勢在必行，但仍希望在系統轉換過程中保留系統舊有程式，才能保有公司的競爭優勢。

故本研究針對這個問題，開發一個簡易程式模組，將傳統系統視為一個可再使用的軟體元件，當執行全球資訊網服務程式時，可以透過該應用程式直接存取傳統資訊系統的資料，進而延續舊傳統資訊系統的生命週期。本研究著重在完全不影響原始的應用程式，讓資訊部門無須花很多人力及成本再重新開發新的系統或使用界面。且可以在充裕的時間開始進行問題評估、資料轉換、及程式修改、程式執行的測試，以減少不必要的風險、成本。

在本節中對本篇論文作簡要的概述，提出本研究的研究背景、動機及目的。第二節介紹資料反向工程與XML技術，我們將對傳統資訊系統(COBOL)、反向工程、XML作一介紹，並針對傳統資訊系統與XML 關聯做進一步的探討。第三節提出一個系統架構，將針對既有COBOL程式實施案例實作，第四節提出與其他整合COBOL系統與WWW的方案比較，第五節是結論。

二、 資料反向工程與XML技術

本研究乃針對使用 COBOL 程式語言所開發文字模式之傳統資訊系統為背景，討論如何結合資料反向工程及 XML 的技術來整合傳

統資訊系統及全球資訊網，故本節主要先針對傳統資訊系統傳統表達方式及以全球資訊網(WWW)網頁方式之間做關聯性的探討。

（一） XML 結構與 Schema 設計方法

XML 主要的目的就是提供文件資料內容的結構化[1][2]，每一個 XML 文件都包含了解邏輯結構與實體結構兩個部份，在邏輯結構中包含了文件的元素及其架構的階層與順序，而實體結構包含了文件的資料內容，如圖 2.1。

```
<?xml version="1.0" encoding="BIG5" standalone="yes" ?>
<!DOCTYPE STUDENT(View Source for full doctype...)>
<STUDENT>
 <STID>A8911001</STID>
 <NAME>張小強</NAME>
 <ADDR>屏東縣內埔鄉學府路一號</ADDR>
 <BIRTHDAY>1987/11/05</BIRTHDAY>
 <SEX>男</SEX>
 </STUDENT>
```

圖 2.1 XML 結構

邏輯結構是指各個不同元素的組織結構，大體上，我們將 XML 文件分成標頭跟主體兩個部份。標頭主要功能在於宣告 XML 規格的版本及定義文件中資料結構的規則。主體部份主要是存放資料內容的區域。而 XML 文件中可以包含樹狀結構或其他文件，也就是說一個元素包含在另一個元素中，而且每個子元素都是包含在它的父元素中。實體結構是由文件中的內容所組成，每個實體都有其名稱與內容，只要實體在文件標頭區有被宣告過的，實體就可以被使用。

XML 最初設計的理念是在文件上使用標籤(Tag)，但 XML 文件本身無法被定義成開放性的標籤語言，所以 就像過去的 HTML 一樣，XML 文件本身無法對文件中的資料進行嚴格的規範，所以必須借由 Schema 來輔助。Schema 是一種描述資料結構的模型，他使用資料庫中的一種描述相關表格內容的機制，規範了文件中的標籤和內容可能的組合形式，用來定義 XML 文件的結構、資料類型等 XML 文件規則(如圖 2.2)。

（二）資料反向工程

軟體再生工程(software-reengineering)的定義是當重新結構或者重寫全部(或部分)傳統系統時不用更改其功能性，使得再維護時更加容易[5]。其領域包括軟體系統的了解及其發展[3] ，其主要目的於瞭解程式，也就是說你所看到、所頗析出來的知識就是你希望得到的更多資訊。

圖 2.2 XML Schema 結構範例

反向工程(reverse engineering)只是軟體再生工程中的一環，透過現有程式碼，整理分析其程序邏輯及功能後，經過抽象化過程，發現其間不明確的關聯性再轉換成高階設計表示法，擷取出原有系統之設計理念，甚至需求規格。將所得之設計或需求加以修改，最後階段再根據這些修改後之設計或需求，重新產生程式碼或修改原有之程式碼。

資料反向工程(data reverse engineering)則是使用結構化的方法重新建構現有系統中的資料資產，其中包含了資料庫反向工程，其定義為辨識某一特定資料庫實作之可能規格。相關的研究多半由程式碼、程式控制語言以及資料描述語言等來源，進行資料庫反向工程，產生資料模式(data model)以進行資料庫轉換。

資料反向工程基本可分為三大主要的部份：

1. 準備工作
2. 萃取資料結構
3. 資料結構概念化

資料反向工程方法論第一階段是準備工作階

段。進行準備工作之初，必先了解恢復完整的資料結構描述。第二階段為萃取資料結構階段。每一個原始程式碼中的資料結構的描述，可能只是檔案或記錄中結構的一小部份。而檔案或記錄中提供極重要的資訊，因此程式和資料分析就成為整合技術上重要的一環。第三階段為資料結構概念化階段。這個階段是當萃取資結構階段完成時，將邏輯性 Schema 萃取成概念性的 Schema。

（三）傳統 COBOL 程式之轉換

傳統的 COBOL 程式語言撰寫主要強調程序性及結構化，程式主體部分主要分成四大部分：識別部(Identification Division)、環境部(Environment Division)、資料部(Data Division)及程序部(Procedure Division)。 識別部主要描述程式的一些基本相關資訊(如程式代碼及撰寫日期等)。環境部主要描述檔案組織(如索引檔、循序檔等)及其存取方式(如循序存取、隨機存取等)。資料部主要是檔案資料欄位的宣告(CRD：COBOL Record Definition)及其邏輯結構的關係描述。 程序部主要是針對商業規則(Business Rule)之邏輯處理。 程式片斷如圖 2.3。經分析後與研究議題較有相關的是 ENVIRONMENT DIVISION 及 DATA DIVISION，至於 PROCEDURE DIVISION 主要是處理不同的商業邏輯，其處理內容將依使用者不同需求而異動，因此並不將該處理內容納入本研究討論重點。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUM-OF-PRICES.
AUTHOR.
SOURCE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INP-DATA ASSIGN TO INPUT.
    SELECT RESULT-FILE ASSIGN TO OUTPUT.
DATA DIVISION.
FILE SECTION.
FD INP-DATA LABEL RECORD IS OMITTED.
01 SCORE.
   05 STID     PIC X(8).
   05 FNAME    PIC X(10).
05 SCOREF.
      10       CODE     PIC X(4).
      10       CNAME    PIC X(60).
      10       DETAIL   PIC 9(5)V99.
WORKING-STORAGE SECTION.
01 SUM-LINE.
   02 FILLER VALUE 'SUM =PIC    X(12).
   02 SUM-OUT    PIC   $$,$$$,$$9.99.
   02 FILLER VALUE 'NO. OF ITEMS ='PIC   X(21).
   02 COUNT-OUT    PIC   ZZZ9.99.
01 ITEM-LINE.
   02 ITEM-OUT   PIC    X(30).
   02 PRICE-OUT   PIC   ZZZ9.99.
PROCEDURE DIVISION.
START.
    OPEN INPUT INP-DATA AND OUTPUT RESULT-FILE.
READ-DATA.
    READ INP-DATA AT END GO TO PRINT-LINE.
    ADD PRICE TO TOT.
    ADD 1 TO COUNT.
MOVE PRICE TO PRICE-OUT.
    MOVE ITEM TO ITEM-OUT.
    WRITE RESULT-LINE FROM ITEM-LINE.
    GO TO READ-DATA.
PRINT-LINE.
    MOVE TOT TO SUM-OUT.
    MOVE COUNT TO COUNT-OUT.
    WRITE RESULT-LINE FROM SUM-LINE.
    CLOSE INP-DATA AND RESULT-FILE.
    STOP RUN.
```

圖 2.3 COBOL 程式範例片斷

在 ENVIRONMENT DIVISION(環境部)主要是討論檔案的邏輯檔案名稱、實體檔案名稱、檔案組織及檔案實際存取的方式。DATA DIVISION(資料部)主要是討論邏輯檔案中每一筆記錄存放的欄位宣告。分析以上兩個 DIVISION，經分析找出有用的資訊後，彙整出與本文相關的資訊為邏輯檔案名稱、實體檔案名稱、檔案組織及每一筆記錄存放的欄位宣告。因此本研究中將試著結合這兩個 DIVISION，萃取出以上所提的資訊，產生一個 Physical Schema。

因使用者需求不同，雖然每一個COBOL處理程式的商業邏輯不同，但有可能會同時存取到同一個實體資料檔案，更有可能數個程式在存取記錄時，其所存取的欄位名稱、欄位屬性及欄位長度會依其商業邏輯需要不同而不同。舉例說，程式一及程式二存取的每一筆記錄長度皆為101個bytes，但程式一及程式二可能因商業邏輯需求不同，而記錄欄位宣告方式隨之不同。如程式一中之住址，於程式中欄位宣告只細分成郵遞區號及住址，但程式二中之住址卻因程序部的邏輯處理需求需將住址細分成郵遞區號、鄰里及住址三個部份。換句話說，不同程式皆會依其不同的商業邏輯，對同一檔案的記錄欄位宣告方式會有所不同。

因此若只針對單一COBOL程式所抽出的 Physical Schema ，所得出的結果並不是完整的 Schema 。因此本研究就針對多個 COBOL 程式的模組進行完整 Physical Schema 抽出。舉例說，先由單一 COBOL 程式(程式 A)產生 Physical Schema ， 並 以 程 式 A 中 的 ENVIRONMENT DIVISION 的實體檔案名稱為 Physical Schema 的名稱；當另一個 COBOL 程式(程式 B)執行時，將檢視程式 B 中 ENVIRONMENT DIVISION 的實體檔案名稱 ，是否已經存在；若檢視存在，則表示該檔案邏輯檔案已經曾被(程式 A)分析並抽出，反之則表示尚未被分析並抽出。若檢視存在時，則抽出程式 B 的 Physical Schema 後，

兩組 Physical schema 進行相互比對，並相互調整成更細部的 Physical Schema。

三、系 統 實 作

本研究套用COBOL程式語言設計的基本架構、資料反向工程等知識，將傳統COBOL系統中所使用到的實體資料檔案及邏輯檔案結構分析出來，並套用轉換成XML技巧，把實體資料及邏輯檔案結構結合並轉換成網際網路上可用來交換的XML。系統實作之基本原理歷經分析及推演後所得系統架構圖(如圖3.1)，分成兩個主要模組。模組A專司如何針對傳統COBOL程式抽出有效的資訊，模組B專司如何將所得結果轉換成XML顯示的方式。實作的工具乃採用VB6.0及ASP.NET程式語言為主要設計的工具。

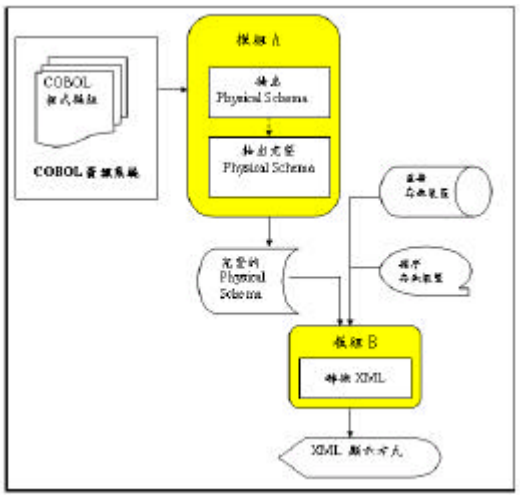


圖 3.1 系統架構圖

模組A主要功能乃針對傳統COBOL系統中實體檔案及邏輯檔案等關連性作分析，彙整後並抽出有效的資訊。因此乃著重在COBOL的原始程式碼將一些繁雜冗長的程式碼剔除，找出可轉換成XML 格式所需的有用資訊。透過模組A，逐步將COBOL系統中全部將使用到的所有實體檔案，都能透過順利找出相對應的完整Physical Schema。

模組B主要功能乃針對當順利找出完整Physical Schema，將所得之完整Physical Schema轉換成可交換格式的XML。使用者即可將任一傳統COBOL資訊系統中的原始程式，即可順利轉成跨程式語言的XML 的格式。

（一）萃取出COBOL資料結構之完整Schema

本模組利用COBOL程式結構固定之特質及結合程式分割的技術進行程式模組的撰寫。詳細地說，運作原理如下：

(1) COBOL程式結構固定之特質：

1、 邏輯資料結構的宣告皆包含在DATA DIVISION 及 WORKING-STORAGE 這兩個主要程式碼區塊中。

2、 敘述句開始位址起始於第八欄，結束位址終止於第七十二欄。

3、 且欄位宣告的結構主要有四大部份：階層、欄位名稱、欄位屬性及欄位長度。

4、 階層宣告必以為數字為首(如 01、05 等)。

5、 並且以上四大部份中間必以空白字元作為區隔。

6、 欄位屬性之宣告定字為 PIC，屬性特性是包含在 PIC 及欄位長度中間，例如 PIC X(40) 表示宣告該欄位屬性為 X。

7、 欄位長度之宣告必尾隨在 PIC 之後，且以()括住。

8、 敘述結束必以「．」做為結束。

(2) 程式分割的技術：

利用以上之特質，執行以下程式分割技巧。

1、 分析該區塊的程式碼，自第八欄至第七十二欄之間的字元為有效字元。

2、 在有效字元範圍中，判斷當分析該字元時為非空白字元時，將字元放入 STACK 中，當分析該字元為空白或”(“ 時，POP出 STACK 中的所有字元，並串成一連串的字元組(如

圖 3.3 右半部)。(空白表示 STACK 中所存放的是一串字元，”(“表示 STACK 中所宣告的是欄位屬性)。

3、 將每一 POP 出來的字元組以”，”分隔。

4、 判斷 POP 出來字元組為”01”表示之後 POP 出來之字元組為記錄名稱。

5、 當敘述中沒有”PIC”字元存在時，表示本身為一個GroupName，在下一個敘述句中可能有作欄位的細分。

由原始程式碼將原始程式的邏輯資料結構萃取的流程如圖3.2：

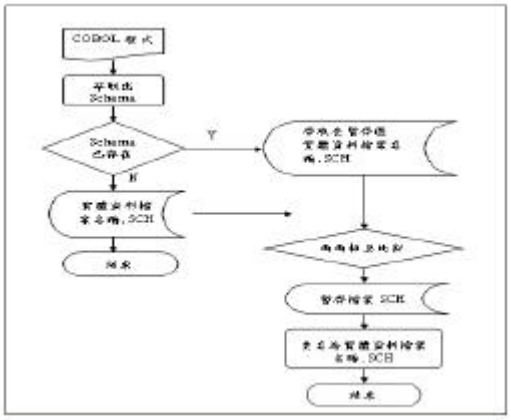


圖 3.2 萃取出完整 Schema 流程圖

由原始程式碼將原始程式的邏輯資料結構萃取出欄位名稱、欄位屬性、欄位長度、自身節點特性及父節點等有效的資訊作為中間產物 Schema 形式之檔案 (如圖3.3)。

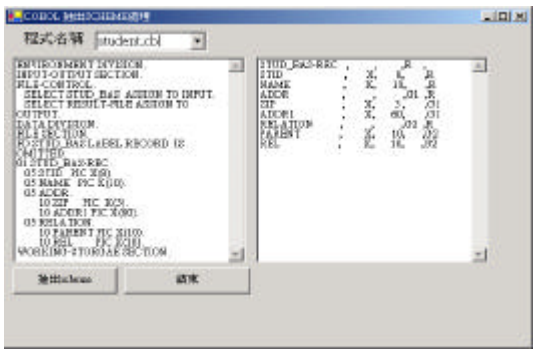


圖 3.3 萃取 COBOL Schema 模組所產生之中間產物 Physical Schema形式檔案(圖 3.3 右半部份)之欄位定義如表 3.1。

表 3.1 Schema 欄位定義一覽表

	說明
欄位名稱	對應於COBOL程式碼中的邏輯資料結構欄位名稱
屬　性	對應於COBOL程式碼中的邏輯資料結構欄位屬性
長　度	對應於COBOL程式碼中的邏輯資料結構欄位長度
自身節點特性	(1)最終節點(以空白表示) (2)群組節點(以G表示) (3)根節點(以R表示)
父 節 點	上一層節點

同時在程式分割處理過程中，將COBOL原始程式中實體資料檔案名稱找出並設定成為該 Schema 檔名，當經過「萃取 COBOL Schema 模組」處理後，產生中間產物的 Schema，此時該Schema 已經以實體資料檔案名稱存檔。

但實際作業的應用系統中可能會存在多個不同的COBOL 程式共同存取同一個實體資料檔案，雖共同存取同一實體資料檔案，卻極有可能在 File Section 中的每一個COBOL 程式所宣告的邏輯資料結構欄位方式不盡相同(如圖 3.4 COBOL模組下程式一及程式二)。因此透過「抽出的Physical Schema 處理模組」，判斷當該Physical Schema已經存在時，表示該Physical Schema 已經曾被抽出處理過，因此則將仍對目前程式碼進行「抽出的 Physical Schema 模組」處理，並暫存在暫存區中, 再將暫存區的Schema檔案與既存之檔案兩兩相互比較，並進行調整。因此COBOL系統中每一個COBOL程式皆能透過這樣的方式，逐步萃取的Physical Schema中間產物，並經「抽出完整的Physical Schema」 處理，調整成為完整的Physical Schema 產物。

（二）結合完整的Physical schema與實體資料並產生 XML 格式

本研究主要採用ASP.NET中所提供的特定物件GETXMLSCHEMA()及GETXML()等

技巧，將上一章節所彙整得到的資訊轉換成XML的格式。

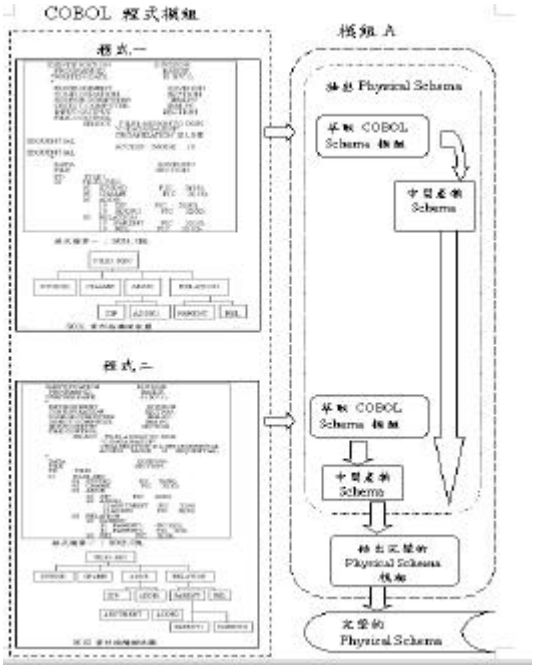


圖 3.4 完整的 Schema 處理圖

在前一節中，我們已經從原始COBOL程式中的DATA DIVISION及ENVIRONMENT DIVISION中萃取出完整 Physical Schema格式的 Schema 檔案。也由 ENVIRONMENT DIVISION中取得之資料檔案名稱(亦即真正的儲放資料的檔案)。藉由 Schema 檔案名稱及資料檔案名稱這兩個重要資訊，作為呼叫轉換XML模組的參數，當被呼叫之轉換XML模組開始執行時，其基本原理如下(如圖 3.5)：

(1) 先建立一暫存的datatable。

(2) 讀取 Schema 檔案，並將每一筆記錄中的欄位名稱，逐一建立 table column。

(3) 等讀取 Schema 檔完畢且table column之欄位名稱皆建立完成後，再讀取資料檔案，逐一將資料檔案中每一筆資料建立成一筆table row。

(4) 當資料讀取完畢後，將datatable加入dataset 後，並利用ASP.NET 所提供的dataset的特性，經程式撰寫並呼叫物件GETXMLSCHEMA()動態產生 XML的格式(如圖3.6)。

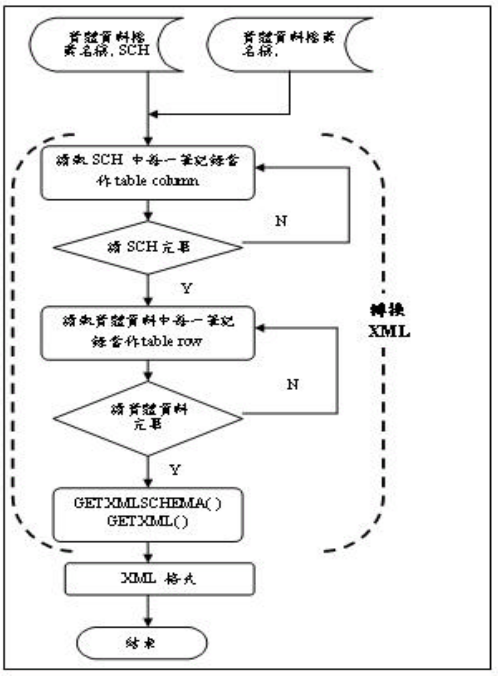


圖 3.5 轉換為 XML 格式流程圖

圖 3.6 結合完整 Physical Schema 及 Physical Data 產生 XML

（三） 實際案例

本研究所採用的案例是以某大學實際上線應用系統之程式來進行實作，該上線之應用系統之程式皆在DOS 6.0環境下執行，且所有程式皆以ANSI 85/ RMCOBOL 軟體所撰寫，以文字模式方式表現系統功能(如圖3.7)。

系統實作過程中面臨的困難在於實體資料檔案中索引檔的存取。傳統COBOL的檔案組織分為索引檔及循序檔，實作部份原針對索引檔及循序檔兩種主要的檔案組織進行探討；但因索引檔實體資料的組織方式涉及不同的COBOL軟體產品，其內部實體資料的存取控制碼不同。故讀取索引檔時，無法依據其特

定存取控制碼來取其每一筆的實體資料。

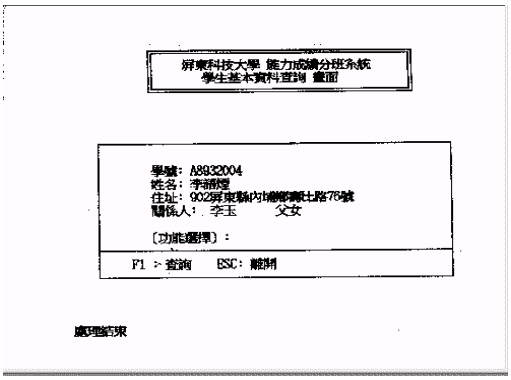


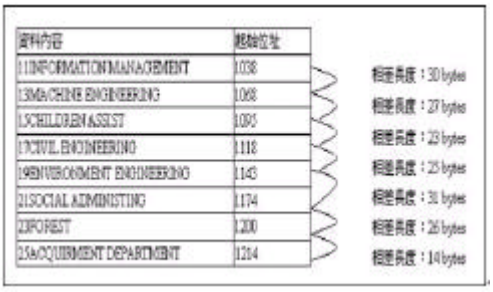
圖 3.7 文字模式的 COBOL 應用系統作業畫面

即使在原始程式碼中所宣告的資料結構長度固定(如表 3.2 資料的記錄長度為 42 bytes) 但實際讀取出來的每一筆記錄長度(包括中、英文字元)卻是不固定(如表3.3) 。故由實體資料內容 dump 出來，一來無法得知產品實體資料的特定存取控制碼，以便於正確取得每一筆資料長度；二來無法分析每一筆不固定長度資料的規則性，故經多方面嘗試仍無法針對索引檔做實例探討。因此本案例以循序檔之檔案組織檔案存取來進行實作。

表 3.2 原始程式檔案組織及資料結構宣告

```
FILE-CONTROL.
           SELECT  DEPT-FILE
                   ASSIGN TO RANDOM
"C:\ENGLISH\DATA\DEPT11"
                   ORGANIZATION  IS  INDEXED
                   ACCESS MODE   IS  DYNAMIC
                   RECORD KEY    IS  DEPT-KEY
                   FILE   STATUS IS  STATUS-DEPT.
      *
       DATA                        DIVISION.
       FILE                        SECTION.
      ***
       * 系列檔
       * RECORD KEY  :DEPT-KEY    (系列碼)
       * OTHER       :DEPT-NAME   (系列名稱)
       *
       FD     DEPT-FILE
              LABEL    RECORDS    ARE    STANDARD.
       01     DEPT-REC.
              05  DEPT-KEY       PIC    X(02).
              05  DEPT-NAME      PIC    X(40).
```

表 3.3 讀取 COBOL 實體資料記錄起始位址比較一覽表



使用我們的程式工具模組，將原來在 DOS環境下文字模式的顯示畫面轉成網際網路的Web 介面(如圖 3.8) ，進入系統後，點選左半部份之學生資本資料查詢，所得XML顯示結果會展示在圖右半部份(如圖3.9)。

圖 3.8 系統實作整合畫面

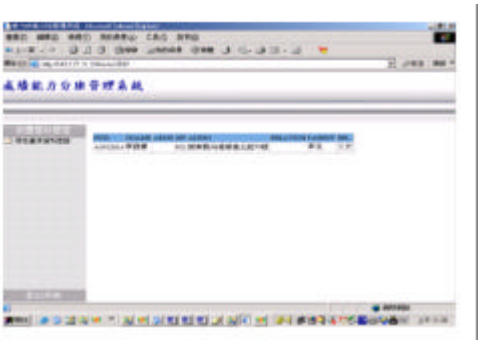


圖 3.9 系統實作整合畫面(續)

四、其他整合 COBOL 系統與 WWW 的方案之比較

使用第三代程式語言(COBOL) 來撰寫傳統資訊系統或維護舊有的 COBOL系統的技術人員，隨著時代變遷而越來越少。對於一個龐大複雜且正在運作的資訊系統而言,完全改寫整個資訊系統對資訊部門是一項重大負擔，而且誰也無法預期新舊系統移轉時資料遺失的風險。

實際上,在業界線上執行的 COBOL 程式種類不一(如 RMCOBOL 、FIJUSTI COBOL 等) 其檔案實體存放方式及其所在平台(Platform)可能會依其供應廠商而有所不同。目前網路上已有部份應用軟體如 XML4COBOL 、 Redvers COBOL XML Interface 亦提供如何整合 COBOL 軟體及 XML 的技術，但經研究調查，多是在既存舊

有的 COBOL 程式碼加入呼叫 XML 轉換函數模組[8]。XML4COBOL 軟體主要是在原始 COBOL 程式碼 DATA DIVISION 加入呼叫 XML Data Area，而在 PROCEDURE DIVISION 中呼叫相關轉換模組；Redvers COBOL XML Interface 亦呼叫軟體既有的函數模組：RCCOBXML、RCXMLCOB 。

以上兩種產品的作法，技術人員仍需要針對既有的 COBOL系統下的原始程式碼做修正並加入新的程式碼 (如 DATA DIVISION 之 XML data area 及 PROCEDURE DIVISION 中的 Call XML4cobol 兩個區塊），因此對於企業資訊部門的技術人員而言，仍需要對既有的 COBOL 系統下的原始程式進行程式修改，方能透過網際網路存取既有的應用系統資訊。

五、結 論

本研究運用資料反向工程的觀念與技術將 COBOL 的程式片斷進行分割，透過程式碼模組之分析處理，抽出有用 Physical Schema，整合傳統資訊系統的資料，顯示在 WWW 的平台上。本文中主要的貢獻是希望資訊人員能透過本文所提之模組，便能輕易將封閉式的傳統資訊系統中的資料轉換成開放性的 WWW 介面，資訊部門無須花很多人力及成本再重新開發新的系統或使用界面，進而延伸傳統應用系統原始程式之生命週期，減輕資深程式開發人員面臨學習新技術所產生之無力感與所耗費時間；並且減少系統重新開發及資料移轉時所產生不可預期之風險。

另一方面，本研究中所提工具主要是結合 COBOL 通用語法及程式分割原理，故模組 A 在任一平台(PC/Mainframe)皆可正常運作。但模組 B 所使用之軟體是 ASP.NET，針對 ASP.NET 伺服器平台的系統需求而言，必須在 Windows 2000 或是已安裝 Service Pack 6a 的 Windows NT4.0。因此在其它 IBM、NEC 或 HP 等 Mainframe 是否可進行，目前仍有待評估。

感謝國科會計畫 Ncs 90-2213-E-020-001 經費補助。

參 考 文 獻

[1] 陳長念,陳勤意，"活用 XML "，知城數位科技股份有限公司。

[2] 梁中平,徐子淵,謝鎮澤,"XML 與電子商務標準",財團法人資訊工業策進會 編印。

[3] R.S. Arnold, "A Roadmap Guide to Software Reengineering Technology in Software Reengineering", IEEE Computer Society Press , Los Alamitos , 1993.

[4] O. Lassila, , "Web Metadata： A Matter of Semantics", IEEE Internet Computing, 2(4), p30-37,July-August 1998.

[5] Shari, Lawrence, and Pleeger, "Software Engineering, Theory and Practice",Prentice Hall,1998.

[6] Ry.M.Sneed, "Wrapping Legacy COBOL Programs behind an XML-Interface", WCRE'01, October 02 - 05, 2001 Suttgart, Germany.

[7] Y.Zou, Kostas, and A.Kontogiannis, "Web-based Legacy System Migration and Integration" ，Dept. of Electrical & Computer Engineering University of Waterloo.

[8] Http://xml4cobol.com