查詢多份 XML 文件的資料結構設計

張雅惠 吳俊頡 謝璨隆

國立台灣海洋大學

資訊科學系

摘要

近年來，在網際網路上從事商業行為，也就是所謂的電子商務，正蓬勃發展中，而如何處理企業與企業間大量資料的交換，則成為一個非常重要的課題，其中 XML (eXtensible Markup Language) 的提出受到很大的重視，我們預期未來將有很多企業將公司資料以 XML 格式存放。本篇論文討論管理眾多 XML 文件的方式，透過所設計的 B^+-Tree 與元素定位點兩種資料結構，可提高資料查詢的速度，我們並以測試資料提供實驗數據加以探討查詢處理的效率。

一、 簡介

為了提供企業間的資料交換，EDI (Electronic Data Interchange) 的應用已經是行之有年，所謂的 EDI，是指一台電腦的應用系統，運用協定的標準與資料格式，經電子化傳遞方式，將資料傳送到另一台電腦的應用系統，讓電腦能夠自動「瞭解」、「處理」和「回應」。然而傳統 EDI 的作法，如銀行間提供的跨行提款服務，是透過專有的加值網路 (VAN, value-added network)。此種作法所需要的設置時間和所耗費的經費都很可觀，並非所有中小企業皆能負擔。近年來，在網際網路 (Internet) 上從事商業行為，也就是所謂的電子商務，正蓬勃發展中，其所憑藉的網際網路與 Web 方面的技術，都比較容易取得。所以，當前電子商務一個很受重視的課題，便是處理企業與企業間 (B2B) 的資料交換，而 XML (eXtensible Markup Language) 的提出更加速了相關的發展。

XML 是由 W3C (World Wide Web Consortium) 所訂定的一個描述 Web 文件結構的標準規範，在 1996 年 11 月有了最先的雛形，並於 1998 年 2 月確立 XML1.0 規格書。XML 為 SGML (Standard Generalized Markup Language) 的子集，是一個結構性的、以文字格式儲存的標註語言 (markup language)。XML 具有可擴充性，各個團體可以依 XML 的規範訂定符合自己需求的資料結構，而該團體的使用者依其規則建立出來的資料便具有一致的表示方式，使資料的交換與流通更為方便。

目前有許多組織，包含銀行公會和高科技電子公會，正嘗試利用 XML 訂定符合自己需求的規則，作為資料交換的標準。而在電腦廠商中，康柏電腦主導的 [台威計劃]，設計 RosettaNet 提供全球供應鏈之協同資訊運作架構，即以 XML 來定義康柏與在台供應商之協同商務運作模式。供應鏈中的上下游廠商，只要透過雙方同意的 XML 標準經由網際網路來傳輸資料，就可以快速的建立交易關係或分享訊息。

因為預見未來大量的企業資料會以 XML 表示，各大資料庫廠商如微軟、IBM、甲骨文 (Oracle) 等，都在其既有的關聯式資料庫系統裡，擴充了處理 XML 的功能。但是，這些資料庫軟體本身相當昂貴，技術也很複雜，所以另一種作法是直接管理 XML 文件，如微軟提出 XML 的解析器(parser)，用來解析 XML 文件，以便於針對 XML 文件開發應用程式。所以，本篇論文提出的作法，是利用微軟處理 XML 文件之軟體開發套件，配合自行設計的特殊資料結構，包括元素定位點 (offset) 與 B^+-Tree，以加快查詢 XML 文件時的速度。

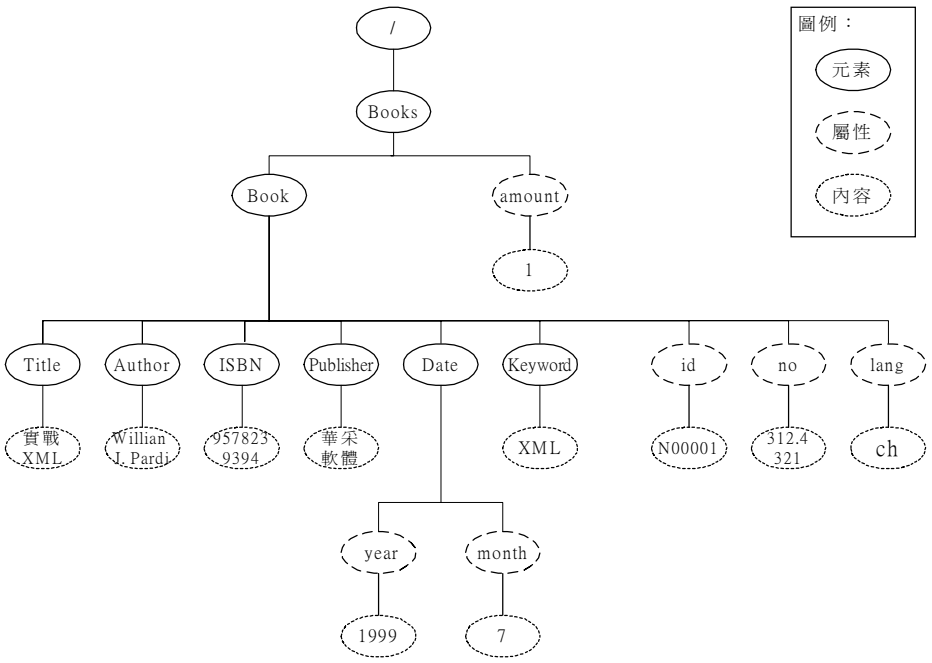
本篇論文的基本架構介紹如下：我們於第二節中，首先簡介 XML，說明其特殊的資料儲存格式及定義的方式；在第三節中，介紹我們為 XML 資料查詢處理所設計的資料結構，包括 B^+-Tree 索

```
L1  <?xml version="1.0" encoding="Big5" ?>
L2    <!DOCTYPE Books SYSTEM "Books.dtd">
L3     <Books amount="1">
L4       <Book id="N00001" no="312.4321" lang="ch">
L5           <Title>實戰 XML</Title>
L6           <Author>Willian J. Pardi</Author>
L7           <ISBN>9578239394</ISBN>
L8           <Publisher>華采軟體</Publisher>
L9           <Date year="1999" month="7" />
L10          <Keyword>XML</Keyword>
L11      </Book>
L12   </Books>
```

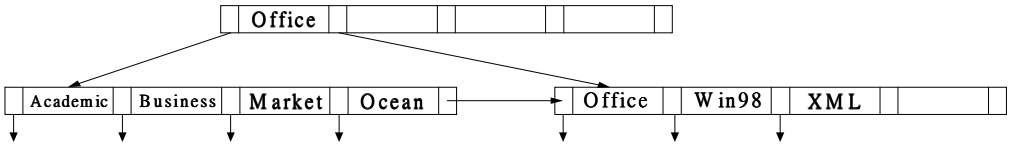
圖一：XML 範例

```
L1  <?xml version="1.0" encoding="Big5"?>
L2  <!ELEMENT Books (Book*)>
L3  <!ATTLIST Books amount NMTOKEN "1">
L4  <!ELEMENT Book (Title, Author+, ISBN, Publisher, Date, Keyword+,)>
L5  <!ATTLIST Book id ID #REQUIRED no NMTOKEN #REQUIRED lang (ch|en|jp|hz) "ch">
L6  <!ELEMENT Title (#PCDATA)>
L7  <!ELEMENT Author (#PCDATA)>
L8  <!ELEMENT ISBN (#PCDATA)>
L9  <!ELEMENT Publisher (#PCDATA)>
L10 <!ELEMENT Date EMPTY>
L11 <!ATTLIST Date year NMTOKEN #REQUIRED month NMTOKEN #REQUIRED>
L12 <!ELEMENT Keyword (#PCDATA)>
L13 <!ELEMENT UserID (#PCDATA)>
L14 <!ELEMENT LoanNo (#PCDATA)>
```

圖二：DTD 範例



圖三：DOM 結構範例



圖四：B^+-Tree 結構圖

引及元素定位點；我們對這兩個資料結構的效能所做的實驗，則在第四節中討論；最後，在第五節的結論裡，我們舉出相關的研究報告並指出未來的研究方向。

二、 XML 簡介

我們首先簡單說明 XML 文件的結構。一份 XML 文件主要是由宣告 (declaration，或稱 prolog)、處理指令 (processing instructions)、元素 (element) 以及註解 (comments) 所組成。圖一為一個 XML 文件的範例，圖中左欄的 Li 代表行數，其中 L1 行為 XML 宣告，說明此檔案採用的編碼為 Big5。L3 行的 Books 為本文件中的根元素，一個 XML 文件中只能有一個根元素。L4 行的 Book 是直屬於 Books 這個母元素下的子元素，它另外定義了三個屬性，屬性等號後面的值則為屬性值，譬如 id 屬性的值為 N00001，no 屬性的值為 312.4321，lang 屬性的值為 ch。而 L5 至 L10 定義的子元素如 Title 等，則是進一步提供 Book 元素的相關資料。

另外在圖一 L2 行裡使用了 Books.dtd，其檔案的作用是檢查此文件是否正確。DTD (document type definition) 是用來定義某種 XML 文件的格式，也就是將每一個元素包含哪些子元素或屬性、各元素出現的順序等，清楚地加以定義和規範。用 DTD 定義出來的一套元素組合，通常稱為「語彙」(vocabulary) 。圖二即為 Books.dtd 的內容。

在 DTD 文件中，ELEMENT 標籤之後放的是元素名，接著用小括號括起來的，是它的“內容模型”，也就是在對應的 XML 文件中可以出現的內容。圖二中的 L4 行說明，Book 這個元素包含了 Title、Author 等子元素，而在 L6 行中進一步註明 Title 元素存放的資料為 #PCDATA (parsable character data)，其為預先定義的標記，代表可解析的文字資料。ATTLIST 標籤則是宣告元素的屬性，包含了屬性名、屬性類別及預設行為的描述，若屬性不只一個時，可以用這三個部分為一個單位一直重複下去。譬如，由 L5 行開始，定義了 Book 這個元素具有 id、no、lang 這三個屬性，id 屬性的類別為 ID，表示該屬性值在同一個 XML 文件中不可重複，預設行為的描述為 #REQUIRED ，表示該屬性值必須存在。

為了分析 XML 文件，W3C 通過一種叫做 DOM (document object model) 的規格，提供了 XML 文件架構模組的文件物件模型。它指出一份文件均從一個根節點 (root node) 開始往下發展成一個樹狀架構。圖三的 DOM 範例對應到圖一的 XML 文件，高度為 2 的節點為 XML 文件之根元素 Books，根元素之下再建立其他節點，包括子元素 Book 及屬性 Amount，如此依序往下發展為一個樹狀結構。DOM 的內容包含了邏輯結構和實體結構，邏輯結構就如同圖中所示的每一個內部節點，它說明這份文件應該包含那一些元素及屬性，並規定了這一些元素及屬性的順序，而實體結構則為文件的真實資料也就是內容，如圖中所示的每一個外部節點。透過 DOM 模型，我們可以清楚地知道一個 XML 文件內部資料的相關性，以便於加以處理取出所需要的資料。

三、 資料結構

本節說明我們為查詢 XML 文件所設計的資料結構。在電子商務的環境中，企業將資料以 XML 檔案存放，而由於資料量極大，所以會表示成多個固定格式的 XML 檔案。不過，若要查詢某筆特定資料時，必須循序搜尋所有檔案，效率會非常不理想。因此，我們針對幾個比較重要的元素，建立查詢值 (search key) 與 XML 檔案對應的索引結構，如此便可直接開啟所需要的 XML 檔案，加快查詢的速度。

我們提出的索引結構，是依據目前傳統資料庫裡最常見的 B^+-Tree 樹狀結構所設計。一棵樹包含樹根 (root node) 、內部節點 (internal node) 及外部節點 (external node) ，每個節點皆存放數個查詢值與指標。如圖四所示，最上層的節點就是樹根 ，同時它也是此樹唯一的內部節點，會指向其他內部節點或外部節點。而最下層的兩個節點是外部節點

```
L1  <?xml version="1.0" encoding="Big5"?>
L2  <!ELEMENT Node (Pointer*,Next?)>
L3  <!ATTLIST Node type (ex|in) "ex">
L4  <!ELEMENT Pointer (#PCDATA)>
L5  <!ATTLIST Pointer key CDATA #REQUIRED>
L6  <!ELEMENT Next (#PCDATA)>
```

圖五：B+-Tree 節點檔案之 DTD

```
L1  <?xml version="1.0" encoding="Big5" ?>
L2  <!DOCTYPE BTree-Node SYSTEM "Btree.dtd">
L3  <Node type="ex">
L4     <Pointer key="Office">B0001.3,</Pointer>
L5     <Pointer key="Win98">B0002.2,</Pointer>
L6     <Pointer key="XML">B0001.1,</Pointer>
L7     <Next>B3.bt</Next>
L8  </Node>
```

圖六：B+-Tree 節點之範例檔案

00 01 00 61 00 50 00 44 00 66 00 7E 00 9E 00 BD 00 00 00 00

unit **1** unit **2** unit **3** unit **4** unit **5** unit **6** unit **7** unit **8** unit **9** unit**10**

圖七：元素定位檔範例

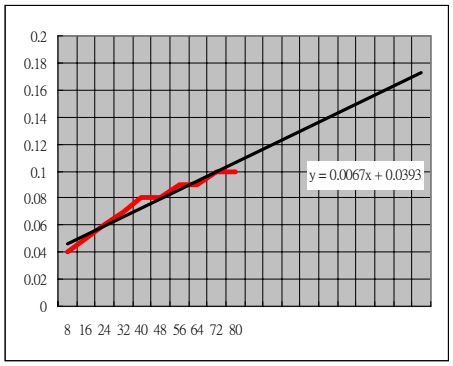
，指向真正資料存放的檔案位置。當我們要查詢某一個值時，必須從樹根開始，根據查詢值的大小往下走到最底層，然後循著指標將資料取回。

B+-Tree 的特色在於其樹狀結構是平衡的，也就是從樹根到外部節點的每條路徑都是相同的長度，所以搜尋每個值所需的時間大致相同。另外每個節點的大小有所限制。若 B+-Tree 的 order 為 n，則每個內部節點(除了根節點)最多有 n 個指標(或 n-1 個查詢值)，最少必須有$\lceil n/2 \rceil$個指標（或$\lceil (n-1)/2 \rceil$個查詢值），所以 B+-Tree 之高度不大於$\left\lfloor \log_{\lceil n/2 \rceil} K \right\rfloor$，這裡 K 為查詢值的個數。假設查詢值為 10,000 筆，當 order 為 20 時，則 B+-Tree 之高度只有 4，所以查詢的速度相當快。圖四的 B+-Tree 其 order 為 5。

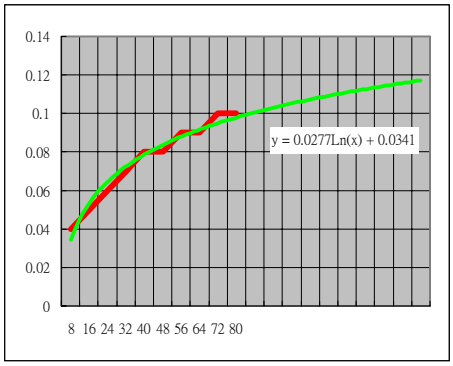
我們的設計是每一個 B+-Tree 的節點存成一個 XML 檔案，其 DTD 如圖五所示。在該圖中，L2 行表示 Node 元素為本文件之根元素，內含多個 B+-Tree 之指標，以 Pointer 元素表示之。L3 行定義了 Node 的一個屬性 type，用來代表對應節點之型態，若值為 ex 代表外部節點，若是值為 in 則代表內部節點。 L4 行說明 Pointer 元素的內容內含文字資料，若此節點為外部節點，則存放真實資料之檔名與位置，若為內部節點，則為下一層 B+-Tree 節點之檔名。此指標對應的查詢值，則在 L5 行定義為 Pointer 元素的屬性 key。L6 行表示的 Next 元素，是記錄本外部節點之下一個外部節點的檔名，以方便作範圍查詢時的處理。

圖六為 B+-Tree 節點的一個範例檔案，對應到圖四右下角節點，其格式遵循圖五之 DTD。L3 行表示此檔案對應到一個外部節點，L4 行至 L6 行表示了此節點內的三筆資料，第一筆資料其查詢值為 Office，B0001.3 則表示完整的真實資料為 B0001 這個檔案中的第三筆資料。值得注意的是，我們不只指出對應的檔案名，也一併記錄在該檔案裡的位置，如此可省卻比較檔案內容的步驟，更進一步加快查詢的速度。最後 L7 行表示此外部節點的下一個外部節點，是對應到 B3.bt 這個檔案。此外我們還必須記錄每一個 B+-Tree，其根節點的檔名，以作為查詢的起點，此類資料存放在設定檔內，該檔案也是 XML 格式。

所以，利用此 B+-Tree 索引結構，其查詢處理的步驟大致如下：



圖八 a：由 B^+-Tree 查詢法推算之直線圖



圖八 b：由 B^+-Tree 查詢法推算之對數曲線

資料筆數	查詢秒數	預測秒數
160	0.141	1.1113
320	0.26	2.1833
640	0.48	4.3273

圖八 **c**：B^+-Tree 之趨近直線秒數對照表

資料筆數	查詢秒數	預測秒數
160	0.141	0.174682
320	0.26	0.193882
640	0.48	0.213083

圖八 d：B^+-Tree 之趨近對數曲線秒數對照

1. 利用系統設定檔，找出所查詢元素，如 Title 或 Keyword 所對應的 B^+-Tree 根節點為何檔案，開啟該檔案。

2. 若開啟的檔案對應到 B^+-Tree 的內部節點，則比較查詢值的大小，繼續走到樹的下一層，也就是開啟另一個 B^+-Tree 節點檔案。

3. 若開啟的檔案對應到 B^+-Tree 的外部節點，且真實資料存放的位置假設為 Bxxxx.n，則開啟 Bxxxx 資料檔案，然後使用微軟提供的程式開發套件，利用 DOM 模式分析 Bxxxx 檔案內元素的相關位置，取出第 n 筆元素資料，處理後傳回結果。

不過我們進一步發現，XML 檔案內的元素具有固定的順序，所以可利用此順序性建立元素定位點，以記錄元素在檔案內的位置，如此則可省略利用 DOM 分析 XML 文件的步驟，快速搜尋到某元素的資料。在設計上，針對一個 XML 資料檔案存在一個對應的元素定位檔 (offset file)。如圖七所示，該檔案內每兩個位元組 (byte) 為一個單元 (unit)。第一個單元記錄此檔案之記錄 (record) 個數，每一筆記錄對應到 XML 資料檔裡每一個主要元素的所有資料，通常主要元素是根元素之下的第一個子元素，以圖一為例，根元素為 Books，而主要元素為 Book。每一筆記錄的第一個單元，記錄其對應的主要元素資料的位置，也就是在 XML 資料檔案中與前一個主要元素資料之距離。而其後之單元，則記錄各子元素與主要元素的距離。我們記錄元素間的相對位置，而非在檔案內的絕對位置，如此可使定位點資料不會因為 XML 檔案的資料修改而需頻頻修正。

圖七表示的元素定位點資料對應到圖一的 XML 範例檔案，其中 unit 1 表示檔案內有一個 record，這是因為圖一內的主要元素 Book 只出現一次。unit 2 則表示第一個主要元素 Book 元素為從檔頭開始第 97（61h）byte，在檔案內我們以 16 進位表示，unit3 表示第一個子元素 Title 的資料距 Book 元素 45 (2Dh) bytes，unit 4 表示第二個子元素 Author 的資料元素距 Book 元素 68 (44h) bytes，unit 5 到 unit 8 也是依相同的方式表示。最後兩個 unit，因為所對應的 UserID 子元素和 LoanNo 子元素沒有資料，所以以 0 (00h) bytes 表示。

利用元素定位檔，查詢處理的步驟如下：

1. 依序開啟每一個元素定位檔，找出欲查詢的元素所在的位置。

2. 如果該元素的定位點不是 00h，則表示有資料存在，開啟對應的 XML 檔案，根據元素位置直接取出資料，比較該資料與查詢值，若相同時則表示找到所要的資料，將結果傳回。

四、 實驗結果

我們依據圖二的 DTD 建立書籍資料 XML 檔案，以測試所設計的資料結構對查詢處理的影響。每八本書籍資料存成一個檔案，而實驗數據的收集是根據關鍵字的查詢，其所建立的 B^+-Tree order 設定為 8。實驗環境為 PentiumIII650 的個人電腦，使用 Win2000 Professional 的作業系統。

我們測量書籍數目為 8 的倍數時，查詢關鍵字所需要的時間秒數，前 10 組數據如圖八的折線所示，為了分析這些數據所代表的意義，我們分別找出一條最趨近於實驗結果的直線 ($y = 0.0067x + 0.0393$，如圖八 a 所示)，以及一條最趨近於實驗結果的對數曲線 ($y = 0.0277Ln(x) + 0.0341$，如圖八 b 所示)。根據這兩個函數，計算書籍數量在數百本時的查詢時間，結果觀察到，直線公式所預測的秒數，比實際查詢秒數多出數秒 (如圖八 c 所示) ，且隨著資料筆數的增加，誤差越來越大。另一方面，利用對數公式所預測的秒數，在 160 筆與 320 筆時，則只比實際查詢秒數少了百分之幾秒 (如圖八 d 所示)，如此可推論出我們設計的 B^+-Tree，的確如第三節所討論的，使查詢的速率，加快到對數時間，優於由循序查詢所需的線性時間。

不過我們也由圖八 d 觀察到，當書籍數量增加到 640 本時，對數公式提供的預測時間，與實際的查詢秒數，誤差有加大的趨勢，這是因為在建立書籍資料時，我們設計讓所有 XML 資料共有 10 個不同的關鍵字，所以 B^+-Tree 的高度在輸入一定數量的資料後就保持不變，但是 B^+-Tree 的外部節點中書籍檔案個數卻是成線性的成長，所以這種資料分佈的狀況，會讓查詢速率由起初的對數函數逐漸趨向線性函數。。

另外，我們也測量元素定位點所提供的查詢速率，圖九是相同書籍數量時，元素定位點與 B^+-Tree 分別提供的查詢秒數，結果發現元素定位點比 B^+-Tree 還要快上許多，儘管用 B^+-Tree 的方法所要開的檔案較少。我們進一步分析 B^+-Tree 的程式，發現絕大部份的時間是在利用 DOM 分析 XML 檔案（如圖十所示），而元素定位點查詢資料的方法並不需要此步驟。

資料筆數	Offset 查詢秒數	B^+-Tree 查詢秒數
160	0.04	0.15
320	0.05	0.28
640	0.07	0.50

圖九：元素定位點與 B^+-Tree 查詢秒數對照表

資料筆數	全部秒數	DOM 秒數
160	0.150	0.120
320	0.281	0.251
640	0.501	0.471

圖十：B^+-Tree 全部秒數與 DOM 使用秒數對照表

五、 結論

電子商務是未來的趨勢，而在網際網路上從事企業間的資料交換，是其中重要的課題，而 XML 的推出更加速了這方面的推展，我們相信未來有很多企業將會以 XML 表示其資料。本篇論文的目的，是討論加快眾多 XML 文件查詢速度的方式，我們設計了 B^+-Tree 及元素定位點的特殊資料索引結構，並實地建立測試資料測量效能。根據我們的實驗數據，B^+-Tree 的確提供了趨近於對數函數的效能，比單純的循序查詢所需要的線性時間有更佳的表現。

不過，我們也發現到，我們目前設計的 B^+-Tree 索引結構，雖然提供對應查詢值資料所在的檔案與元素序號，仍然需要透過微軟提供的 DOM 發展套件來分析該 XML 文件，耗時極大，相較之下，元素定位點的作法，記錄所有元素的位置，提供了較快的效率。所以，目前我們正針對如何加快 XML 的查詢速度，深入研究探討，未來可能改進的方向有下列幾點：

1. 探討結合 B^+-Tree 與元素定位點的可能性，也就是先利用 B^+-Tree 找到所在的檔案，然後再使用元素定位點取出特定的元素資料。

2. 討論 B^+-Tree 與書籍資料建置方式的關係，譬如目前是 8 本書籍放在一個檔案，則 B^+-Tree 的 order 應該是多少，才能達到最佳的效能。

最後列出目前國內外相關研究報告。XML 在

資料庫學門所帶來的影響以及可能的研究方向，在文獻中已可見廣泛的討論 [1]。而由史丹佛 (Stanford) 大學提出的 Lore 計劃，實做一套半結構化資料 (semistructured data) 的管理系統，並設計適合的查詢語言 Lorel 與討論索引結構 [2]，由於 XML 具有半結構化的特性，所以該系統有很大的參考性，而史丹佛大學也逐漸把該系統轉化為 XML 資料的管理系統。另外為了處理具有複雜特殊符號的文件 (如 Z Specifications)，並使其可在網際網路上快速傳遞與呈現，有研究者針對 XML 資料實作了 Java 翻譯引擎(rendering engine) [3]。XML-GL [4]，則針對 XML 文件定義圖形化的查詢語言，提供傳統關聯式資料庫的查詢功能。有的研究者則替 XML 查詢提供了一個代數的方法 [5]，來討論最佳的執行策略。這些結果都值得我們進一步參考與研究。

誌謝

此計畫由國科會贊助，編號為：NSC

89-2213-E-019-003

六、參考文獻

[1] J. Widom, Data Management for XML : Research Directions, *IEEE Data Engineering Bulletin*, volume 22, number 3, (1999).

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener, The Lorel Query Language for Semistructured Data, *International Journal on Digital Libraries*, volume 1, number 1, pages: 68 – 88, (1997).

[3] Ciancarini, P.; Vitali, F.; Mascolo, C. , Managing Complex Documents Over the WWW: A Case Study for XML, *IEEE Transactions on Knowledge and Data Engineering*, volume 11, issue 4, pages: 629 –638, (1999).

[4] S. Ceri, S. Comai, E. Damiani, P. Fraternali and L. Tanca, Complex Queries in XML-GL, *ACM Symposium on Applied Computing*, volume 2, pages: 888 – 893, (2000).

[5] Vassilis Christophides, Sophie Cluet and Jerome Simeon, On Wrapping Query Languages and Efficient XML Integration, *In Proceedings of ACM SIGMOD Conference on Management of Data*, pages: 141 – 152, (2000).