行政院國家科學委員會專題研究計畫　成果報告

Web 文件自動萃取系統之容錯研究

計畫類別：個別型計畫
計畫編號：NSC92-2213-E-009-116-
執行期間：92 年 08 月 01 日至 93 年 07 月 31 日
執行單位：國立交通大學資訊工程研究所

計畫主持人：吳毅成

計畫參與人員：蘇瑞元、姜智耀、翁仕全、王志祥

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 93 年 12 月 21 日

中文摘要

本計畫是第一個提出萃取容錯問題。主要工作項目包含如下部份：

1. 收集 DESDL 容錯案例。DESDL 是我們過去研究發展出的一套萃取容錯系統。
2. 定義萃取容錯問題。
3. 依據上述定義，分析具有高度容錯能力之 XPath 萃取表示模式。
4. 設計能產生高度容錯之 XPath 萃取表示的演算法。

我們相信此計畫，除了有相當的學術價值外，對於辦公室或企業組織或個人之自動資訊服務能有所貢獻，並且對台灣的電子商務網站發展有正面的助益。

Abstract：

As far as we know, this project is the first one that proposes the problem of fault tolerance of data extraction. This project mainly includes the following working items:

1. Collect the cases of data extraction on top of DESDL, a data extraction system done in one of our past NSC projects.
2. Define the problem of fault tolerance of data extraction.
3. According to the definition, analyze the model of XPath expressions with high fault tolerance.
4. Design an algorithm to produce XPath expressions with high fault tolerance.

Except for the academic contribution, our research is also important for the use of data extraction by office and enterprise. We believe that this would also have significant contribution for Taiwan industry.

Keyword: fault tolerance, data extraction, XPath, DESDL.

一、前言

全球資訊網(World Wide Web)可說是一個極大的資料庫，如何從中萃取出有用的訊息是非常重要的事情，例如：比價系統須萃取出其他電子商務網站中的有用訊息。

為此，我們在過去的國科會計畫中，設計了一個資料萃取語言叫做 DESDL (Data Extraction Service Description Language)及其系統，來解決網頁萃取的問題。在過去的實際經驗中，已能成功地協助使用者處理網站的資訊萃取。

然而，我們也觀察到許多網站，有時候會做一些網頁版面的調整，因而導致萃取錯誤。若能提升萃取的容錯能力，避免萃取錯誤，則網頁萃取的維護會更為容易。

二、研究目的

如前所述，網站會做一些網頁版面的調整。這時的問題是：若所要萃取的網頁，產生變化，則我們 DESDL 系統原先對此網頁所下的 XPath 表示式，可能就會萃取到錯誤的資料。如何能降低這個錯誤的機率，就是本計畫的主要研究課題。

三、文獻探討

本節我們將分兩個部份：XPATH 及 DESDL。

3.1. XPATH

XPath 是 W3C 制定的標準之一[15,16]，是一個 expression 語言，主要是用來指定或萃取 XML 文件中的部份資料。其資料模式考慮到 XML 的樹狀結構，並也包括整數、字串等資料的表示，甚至上述兩種所形成的資料列(sequence)。這語言的名稱，

主要是由於其表示法多以 Path 來指定資料位置。

XPath 語法用以下一些例子說明：

- child::para 選擇目前節點的 para 子節點。通常 child::可以省略不寫。
- child::text()選擇目前節點的所有文字子節點。
- child::node()選擇目前節點的所有子節點的文字部分。
- attribute::name 選擇目前節點的 name 屬性。通常可以用@name 來簡化這個寫法。
- self::para 如果目前節點是 para 元素則選擇自己，否則就不選擇任何節點。
- descendant::para 選擇目前節點的子孫中的 para 元素。若要多包含自己本身，則用 descendant-or-self。
- ancestor::para 選擇目前節點的祖先中的 para 元素。若要多包含自己本身，則用 ancestor-or-self。通常“/descendant-or-self::node()/”可以被縮寫為“//”
- / 選擇文件的根節點。
- para[@type="warning"]: 選擇 attribute type 之值為 warning 的所有節點。
- chapter[title="Introduction"]: 選擇有一 title 子節點含 Introduction 字串的值為 attribute type 之值為 warning 的所有節點。

此外 XPath 還包含條件敘述、比較、迴圈、Regular Expression 與 matching 等功能。目前則併入在 XQuery 中。

3.2. DESDL

過去有許多萃取相關系統，如[2,3,5,6,7,8,10,11,12]。我們在過去的國科會計畫中，設計了一個資料萃取語言叫做 DESDL (Data Extraction Service Description Language) 及其系統[18,19]，來解決網頁萃取的問題。

先來看圖一（如下）的一個 HTML 文件[13,14]，我們可以用圖二的 DESDL script 把 HTML 文件中的 Title 及 Author 資料，萃取出來，其中，變數萃取的表達格式是採用 W3C 的標準萃取格式 XPath。例如："//b[0]//text()"。

```
. . .
 <td><b><a href="http://...”>
       DESDL: A Data Extraction Service
       Description Language</a></b>
     <b>I-Chen Wu,  . . . </b>
     <i>Proceedings of . . .</i>
 </td>
 <td><b><a href="...”>
       WebOQL: Restructuring Documents,
       Databases, and the Web </a></b>
     <b> G. Arocena and ... </b>
     . . .
 </td>
. . .
```

圖一：一個 HTML 的例子

```
<DESDL>
  <INIT SERVICE="GetPaper"
       URL="..."/>
  <SERVICE NAME = "GetPaper"/>
    <VAR NAME="Title"
         PATH="//b[0]//text()"/>
    <VAR NAME="Author"
         PATH="//b[1]/text()"/>
  </SERVICE>
</DESDL>
```

圖二：萃取圖一的 DESDL Script

DESDL 支援 plug-in 的寫法，這樣可以讓程式設計者容易地加入自己所希望的功能。DESDL 的 plug-in 程式，我們稱之為 DESDLet，如圖三，在下一個萃取瀏覽動作前，要先執行 RemoveRep.dll 這個 DESDLet 來去除重複的網頁。

```
<DESDL>
  <INIT SERVICE="GetPaperAttr"
        URL="..."/>
  <SERVICE NAME="GetPaperAttr">
    <VAR NAME="SearchBase"
         PATH="//table/tr/td"/>
```

```
    <FOREACH FROM="$base">
      <VAR NAME="Title"
           PATH="b[0]//text()"/>
      <VAR NAME="Link"
           PATH="b[0]/a/@href"/>
      <INVOKE SERVICE="GetAbs"
           DESDLET="RemoveRep.dll"
           URL="$Link"/>
    </FOREACH>
  </SERVICE>
</DESDL>
```

圖三：DESDLet 的例子

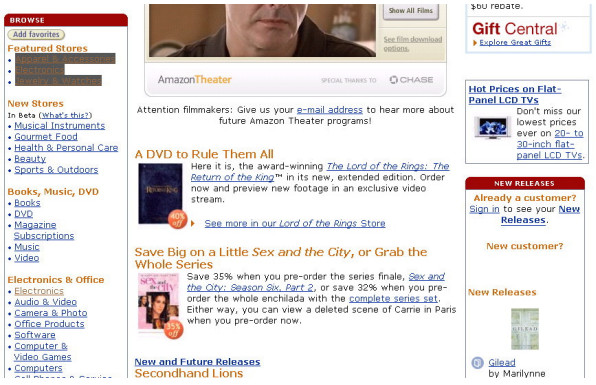
四、研究方法

本計畫的主要研究方法及項目如下[4,9,17]，並分述於各子節：

1. 收集 DESDL 容錯案例。
2. 定義萃取容錯問題。
3. 依據上述定義，分析 XPath 萃取表示法的容錯能力。
4. 設計具有高度容錯的演算法。

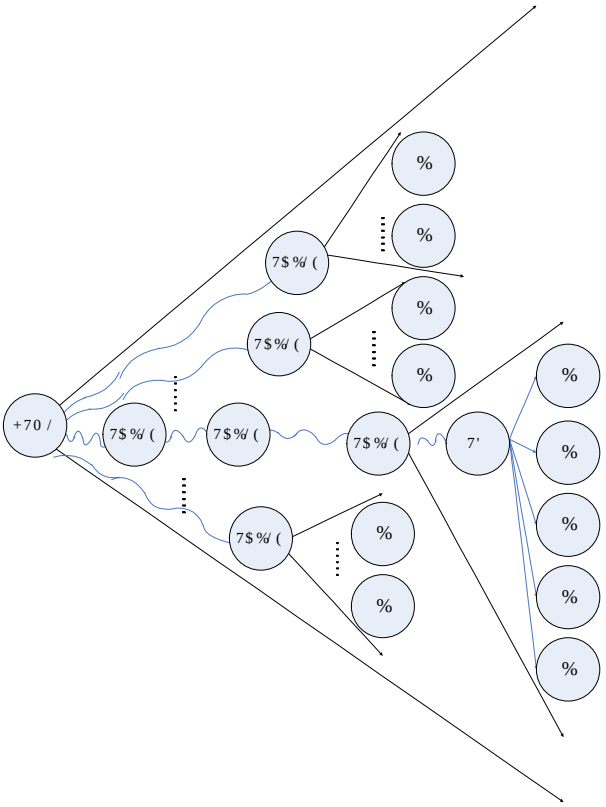
4.1. 收集 DESDL 萃取案例

為了研究萃取系統的容錯問題, 我們持續收集, 許多現有網站的網頁, 例如, Yahoo! 網站、Ebay 網站等。以觀察到的 Amazon 網頁為例[1]，如下圖（圖四)。



圖四：Amazon 網頁，含萃取部份

圖五：Amazon 網頁萃取部份放大



圖六：Amazon 網頁之 DOM Tree

這些項目(圖五)，可用以下 XPath 萃取出，`//HTML[0]/BODY[0]/TABLE[1]/TBODY[0]/TR[0]/TD[0]/P[1]/TABLE[0]/TBODY[0]/TR[1]/TD[0]/TABLE[0]/TBODY[0]/TR[0]/TD[0]/TABLE[0]/TBODY[0]/TR[0]/TD[0]/TABLE[0]/TBODY[0]/TR[0]/TD[0]//B`。此表示式看似合理，但若此網頁在該路徑上或 sibling 做任何簡單的修改，很有可能會造成萃取錯誤。若簡化為 `//TABLE[@width="170"]//B`, 則直覺上反而具有更高的容錯能力。本計畫所提出的分析方法，將會顯示，這樣的萃取有較高的容錯機率。

4.2. 萃取容錯問題之定義

為了簡化容錯的分析, 我們對萃取容錯問題做了以下的定義。

- 每天對任何一個網頁中做任何一項修改之機率均等且獨立，並設為 P_{item}。這修改包含節點修改（含新增、刪除、更改名稱）或 attribute 修改（含新增、

刪除、更改名稱、更改值）或內容修改（含新增文字、刪除文字、更改文字）。以我們觀察到的 Amazon 為例，此網頁有 1095 個節點，有 6717 個 attribute，有 5346 個文字。我們可簡單假設約有 10000 多個。為了更進一步簡化，我們將忽略節點新增及刪除，並假設網頁每 100 天修改一次，每次改 10 個地方。那麼我們可以用此來簡化估計 P_{item} 約為 10/(10000*100)=1/100000。

- 假設常用的節點標籤值有 M_{tag} 種，並假設每個節點的 Tag 被改變為另一個 Tag 的機率均等，設為 $P_{Mtag}=1/M_{tag}$。
  我們對某些網頁的觀察顯示，常用的節點標籤值大概有二十多個，因此 P_{Mtag} 可假想為約 1/20。
- 假設常用的 attribute 值有 M_{attr} 種，並假設每個 attribute 值被改變為另一個的機率均等，設為 $P_{Mattr}=1/M_{attr}$。
  我們對某些網頁的觀察顯示，常用的 attribute 值大概有十多個，因此 P_{Mattr} 可假想為約 1/10。
- 假設常用的文字有 M_{word} 種，並假設每個文字被改變為另一個文字的機率均等，設為 $P_{Mword}=1/M_{word}$。
  我們對某些網頁的觀察顯示，常用的文字大概有超過數千個，因此我們保守估算 P_{Mword} 為約 1/1000。

這樣的定義，原則上對一些具有 correlation 的修改比較不能彰顯。例如：對於節點中的文字改變，通常許多辭句都是一併修改的，或者要修改結構，就整個大塊來改。然而，為了簡化分析，我們仍視各個修改為獨立，忽略 correlation。

萃取表示式的容錯能力：給定一個網頁及一個 XPath(或類似 XPath)的萃取表示式，依照上述網頁修改之假設，此網頁不會發生錯誤的機率。

萃取容錯問題就是：在給定一個網頁及此網頁中所要萃取的資料後，如何選定最佳的萃取表示式，使得容錯能力最高。

4.3. 分析 XPath 萃取表示法之容錯能力

本節將以之前所提的 Amazon 的例子，來分析各種 XPath 表示法的容錯能力。

首先，萃取表示法會發生的錯誤，分為兩種：(1)原本應該萃取到的，網頁修改後萃取不到；(2)原本就不應該萃取到的，網頁修改後萃取到了。我們用 P_{miss} 來表示前者發生錯誤的機率，而用 P_{noise} 表示後者發生錯誤的機率。

我們先分析 Amazon 網頁的萃取表示式 `//HTML[0]/BODY[0]/TABLE[1]/TBODY[0]/TR[0]/TD[0]/P[1]/TABLE[0]/TBODY[0]/TR[1]/TD[0]/TABLE[0]/TBODY[0]/TR[0]/TD[0]/TABLE[0]/TBODY[0]/TR[0]/TD[0]/TABLE[0]/TBODY[0]/TR[0]/TD[0]//B`。對此表示式，我們分別估算 P_{miss} 及 P_{noise} 之機率如下：

- P_{miss}：此數約為
  $P_{item}*(1-P_{Mtag})*(N_{path(->B)})$ +
  $P_{item}*P_{Mtag}*N_{path(TD[0]—B)}$。

$N_{path(A<->B)}$ 代表從節點 A 到節點 B 的 Path 上面的節點數量，其中包括 A 和 B；若 A 不寫，表示從 root 開始。$N_{path(A->B)}$ 代表從節點 A 到節點 B 的 Path 上面的節點數量，其中只包括 B；$N_{path(A<-B)}$ 只包括 A；$N_{path(A-B)}$ 不包括 A 與 B。

此公式的第一項是指在 XPath 中 root 到節點 TD[0]之所有節點，及 B 的 Path 上所有節點，若改為其他任何一種標籤，則會找不到該節點。

此公式的第二項是指 TD[0]和 B 路徑中的標籤，本來不是 B 的標籤，若改為 B，則也會找不到該節點。

觀察之後，以下情形會讓 P_{miss} 降低：(a) $N_{path(->B)}$ 的數量愈少，或(b)TD[0]和 B 接近。

若用上節假設的值，則 P_{miss} 為

(1/100000)*(19/20)*33 +
(1/100000)*(1/20)*4 ~=
(1/100000)*33。

在這裡，第二項幾乎可以忽略掉。

- P_{noise}：此數約為
  $P_{item}*P_{Mtag}*(N_{des(TD[0])} - N_{M(B)})$

$N_{des(A)}$ 代表節點 A 的所有子孫節點數量；$N_{anc(A)}$ 代表節點 A 的所有祖先節點數量。$N_{M(B)}$ 代表

所有萃取到的 B 節點數量。

此公式是指所有 TD[0]的子孫節點的標籤，只要修改為 B，則會被萃取到。

觀察之後，以下情形會讓 P_{miss} 降低，以下情形會讓 P_{noise} 降低：TD[0]愈接近 B 或 leaves，也就是 $N_{des(TD[0])}$的數量愈少。

若用上節假設的值，則 P_{noise} 為 (1/100000)*(1/20)*237。如此，P_{miss} 與 P_{noise} 合約為(1/100000)*44。

現在，考慮另一個萃取表示式 //TABLE[@width="170"]//B，我們也分別估算如下：

- P_{miss}：此數約為

  $P_{item}*(1-P_{Mtag})*(N_{M(TABLE)}+ N_{M(B)}) +$
  $P_{item}*(1-P_{Mattr})*N_{TABLE[@width=170]} +$
  $P_{item}*P_{Mtag}*(N_{path(TABLE—B)})$

$N_{TABLE[@width=170]}$代表萃取到含有屬性值是 width=170 的 TABLE 節點數量。

此公式的第一項是指萃取到的 TABLE 節點或 B 節點的標籤，若改為其他任何一種，則會找不到原來的節點。

此公式的第二項是指含有屬性值是 width=170 的 TABLE 的屬性值如果改變，則會找不到原來的節點。

此公式的第三項是指 TABLE 和 B 路徑中的節點的標籤，若改為 B，則會找不到原來的節點。

觀察之後，以下情形會讓 P_{miss} 降低，以下情形會讓 P_{miss} 降低：(a)$N_{M(TABLE)}$的數量愈少，或(b)TABLE 和 B 愈接近的話，$N_{path(TABLE—B)}$的數量愈少，或(c)$N_{TABLE[@width=170]}$的數量愈少。

若用上述假設的值，則 P_{miss} 為 (1/100000)*(19/20)*(1+10) + (1/100000)*(9/10)*1 + (1/100000)*(1/20)*4 ~= (1/100000)*12。

- P_{noise}：此數約為

  $P_{item}*P_{Mtag}*(N_{des(M(TABLE))} - N_{M(B)}) +$
  $P_{item}*P_{Mattr}*N_{des(M(TABLE[@width=170]))\cap M(B)}$

此公式的第一項是指萃取到含有屬性值是 width=170 的節點，如果標籤改成 TABLE，則會被萃取到。

此公式的第二項是指含有屬性名稱是 width、屬性值是 170、子孫節點標籤為 B 的這些節點，如果屬性值改成符合 width=170，則會被萃取到。

觀察之後，以下情形會讓 P_{miss} 降低，以下情形會讓 P_{noise} 降低：(a)TABLE 愈接近 leaves，也就是 $N_{des(TABLE)@width=170])}$的數量減少，或(b)B 的祖先中，具有屬性值@width=170 的節點數量愈少。

若用上述假設的值，則 P_{noise} 為 (1/100000)*(1/20)*231 + (1/100000)*(1/10)*50 ~= (1/100000)*16。如此，P_{miss} 與 P_{noise} 合約為 (1/100000)*28。

以上分析顯示後者的萃取公式可獲得較佳的容錯能力。這確實反應我們在前節的直覺。

本計畫更進一步地，提出四類容錯能力較好且較為簡單的萃取表示式。其容錯能力之分析，在此報告省略。

- //T//U
- //T[.//V//W]//U
- //T[@A=V]//U
- //T[Match(text(),"pattern")]//U

T 可以有幾個節點的 path，例如：table/tr；同樣地，U 也是。在我們正在撰寫的論文中，我們有分析出這幾類的萃取有較高的容錯能力，此報告略過。

4.4. 萃取容錯演算法

在本計畫中，我們發展了一個演算法來找出具有較高容錯能力的萃取表示法。

1. 找出所要萃取資料的共同祖先 CM (common root)。以圖六為例，TD[0]即為共同祖先。
2. 從這個 CM 開始，選用最少的標籤，做為 XPath 表示式，來萃取所要的萃取項目。這個 XPath 表示式即是/…/CM//U。
3. 簡化 CM 之前的 XPath 為//T[..]。

本計畫，我們也提出了一套資料結構表示法，使得演算法，有以下的效率：若 XPath，可簡化為//T[]//U 模式，且 T 與 U

只有一個標籤的話，執行時間可以在 O(N) 時間完成。N 是此網頁的總標籤數。

五、成果與討論

本計劃對萃取容錯能力做了研究與分析。主要成果如下：

1. 完成 DESDL 容錯案例之收集與分析。
2. 完成萃取容錯問題之定義。這提供分析者一套萃取容錯之分析模式。
3. 依據上述定義，完成 XPath 萃取表示法的容錯能力分析。
4. 從前述分析，提出一些具有高度容錯的 XPath 萃取表示模式。
   我們未來將會持續研究分析案例，更進一步地分析其他高度容錯的表示模式。
5. 完成具有高度容錯演算法之設計。
   此演算法，若可用 //A[]//B 表示出 XPath 萃取表示法，時間複雜度為 O(N)。我們未來將有更多時間複雜度的研究分析。

六、計畫成果自評

由於本計畫是第一個提出萃取容錯問題，加上有許多新模式的發展及新的演算法設計。我們預計將有會有一至三篇論文，並將投稿至有權威的期刊。

另外，DESDL 已經經由國科會技術移轉至業界。由於本計畫所要解的問題，也是由業界合作中觀察到的，因此我們相信本計畫的成果，對業界也將十分重要。

因此，自評此計畫對學術研究及台灣相關業界有相當的貢獻。

References:

[1] Amazon.com, “Amazon Online Store”, 2002, http://www.amazon.com.

[2] Gustavo Arocena and Alberto Mendelzon. “WebOQL: Restructuring Documents, Databases, and the Web”, In *Proceedings of ICDE*, 1998, Orlando, Florida.

[3] G. Arocena. “WebOQL: Exploiting Document Structure in Web Queries”, Master's Thesis, University of Toronto, 1997.

[4] Richard E. Bablow, Jerry B. Fussell, and Nozer D. Singpurwalla, “Reliability and Fault Tree Analysis : Theoretical and Applied Aspects of System Reliability and Safety Assessment”, Society for Industrial and Applied Mathematics, 1975

[5] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. “XML-QL: A Query Language for XML”, In *Proceedings 8th International World Wide Web Conference (WWW8)*, 1999. Computer Networks 31(1116) : 1155-1169.

[6] D. Konopnicki, O. Shmueli. “W3QL: A Query System for the World Wide Web”, in *Proceedings of the 21th International Conference on Very Large Databases*, Zurich, 1995.

[7] David Konopnicki , Oded Shmueli. “Information gathering in the World-Wide Web: the W3QL query language and the W3QS system”, *ACM Transactions on Database Systems (TODS)*, Volume 23 Issue 4, Dec. 1998.

[8] Phillip Merrick, Charles Allen. “Web Interface Definition Language”, W3C NOTE, Sep. 1997. http://www.w3.org/TR/NOTE-widl.

[9] Martin L. Shooman, “Reliability of Computer Systems and Networkds : Fault Tolerance, Analysis, and Design”, John Wiley & Sons, Inc. 2002.

[10] W3C Consortium, “XML Query”, Apr. 2000. http://www.w3c.org/XML/Query.

[11] W3C Consortium. “XQuery 1.0: An XML Query Language”, W3C Working Draft, 16 Aug. 2002. http://www.w3.org/TR/xquery/.

[12] W3C Consortium. “Extensible Markup Language (XML) 1.0 (Second Edition)”, W3C Recommendation, Oct. 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[13] W3C Consortium. “Hyper Text Markup Language”, Jan. 1998. http://www.w3c.org/Markup/.

[14] W3C Consortium, “HTML 4.01 Specification” W3C Recommendation, Dec. 1999. http://www.w3.org/TR/html4/.

[15] W3C Consortium. “XQuery 1.0 and XPath 2.0 Data Model”, W3C Working Draft, Aug. 2002. http://www.w3.org/TR/query-datamodel/.

[16] W3C Consortium. “XML Path Language (XPath) 2.0”, W3C Working Draft, Aug. 2002. http://www.w3.org/TR/xpath20/.

[17] Yau, Hiu Wah, “The Analysis and Optimization of Fault Tolerance in multiprocessor systems : A Graph Theoretic approach ”, UMI, 1989.

[18] I-Chen Wu, Jui-Yuan Su, Loon-Been Chen, W. C. Chien, and C. T. Lee, "DESDL: A Data Extraction Service Description Language", in 2002 *International Computer Symposium* (*ICS2002*), Hualien, Dec. 2002.

[19] I-Chen Wu, Jui-Yuan Su, and Loon-Been Chen, "Browser-Oriented Data Extraction", in 2004 *International Computer Symposium* (*ICS2004*), Taipei, Dec. 2004.