第五章　相關議題探討與未來展望

Discussion & Future Work

在前一個章節我們有提到了實作時所遇到的問題，在這一個章節我們來做更進一步的討論，除此之外也為本研究提出未來展望。

第一節　相關議題探討

有關例外處理的部分，根據實驗的結果，伺服端無法直接擲出例外到客戶端進行處理，而客戶端所能捕捉到的例外，就只有 System::Net::WebException 以及 System::Web::Service::Protocols::SoapException 兩種，只要伺服端所回傳的 HRESULT 不是 S_OK，則客戶端就會捕捉到後者的例外；而像是服務逾時或是網路斷線等，客戶端就會捕捉到前者的例外，但是服務逾時的問題不只有網路可能會造成逾時，服務本身也有可能造成逾時，如系統過度忙碌或運算太繁雜，甚至服務設計不良等等，於是我們想到一個簡單的辦法來做服務逾時問題發生的原因判斷。

本研究建議伺服端在提供服務的同時也一併提供一個叫 echo 的服務，echo 服務內的動作很簡單，只要在一秒（或合理的時間）內回傳 S_OK 即可，不需有傳入傳出的參數也不需進行任何運算，目的就是確認網路的狀況是否暢通。有關 echo 的服務合約以及合約標題檔裡的內容如圖 12。

```
01  HRESULT echo() {
02      /*@CONTRACT
03      @RTT: 1000;
04      */
05      return S_OK;
06  }
//-----------------------------------------------------------------------------------------------
11  <operation name="echo">
12      <contracts>
13          <contract type="RTT">1000</contract>
14      </contracts>
15  </operation>
//-----------------------------------------------------------------------------------------------
21  void echo() {
22      WebService _ws;
23      try {
24          int _orig_timeout_ = _ws.Timeout;
25          _ws.Timeout = 1000;
26          _ws.echo();
27          _ws.Timeout = _orig_timeout_;
28      } catch ( System::Net::WebException^ ) {
29          throw gcnew ContractException ("網路可能發生問題導致服務逾時");
30      } catch ( System::Web::Services::Protocols::SoapException^ ) {
31          throw gcnew ContractException ("echo 服務發生錯誤");
32      }
33  }
```

圖 12：echo 服務的合約相關內容

圖 12 分了三個區塊，第一個區塊(01~06 行)是 echo 服務的函式及服務合約內容，只訂定了服務回應時間@RTT，服務內容也只是回傳 S_OK；而第二三個區塊(11~15 行及 21~33 行)是經過 Service Contract Parser 處理過之後產生 XML 格式的 echo 服務合約檔以及 echo 服務合約標頭檔。

當網路服務發生逾時的狀況時，會再執行 echo 服務的請求（如圖 9 的第 22 行），若 echo 服務正常，即表示網路品質沒有問題，而是網路服務發生了逾時的狀況；若 echo 服務也發生逾時甚至發生錯誤，則一般服務可能沒有問題，而是網路部分或是服務伺服器發生了問題。這樣的判斷雖然沒有非常的精確，卻也有效的將逾時的原因進行區分。

針對逾時部分的處理，本研究一開始的做法是想透過執行緒來計時。在進行服務請求前先建立一條執行緒，負責進行計時的動作，利用微軟.NET Framework 的非同步作業(Asynchronous Operation)方式來進行服務請求，藉由判斷非同步作業是否完成以及執行緒計時是否結束來決定服務是否逾時，在正常的狀況應當是非同步作業已完成但執行緒計時尚未完成，倘若執行緒計時的動作較非同步作業來得早結束，則表示網路服務已經發生逾時現象。但是隨著研究的發展，發現相同的功能可以利用 Timeout 的方式來簡單的達成，於是就決定捨棄已經撰寫完成的執行緒模組，全面更換成 Timeout 搭配 System::Net::WebException 的形式來達到偵測逾時的功能，我們將 Timeout 時間設定成服務合約內制定的服務回應時間@RTT，當服務發生逾時的現象，我們便會捕捉到 System::Net::WebException 的例外事件，之後再經由 echo 服務進而判斷服務逾時的原因是因為網路品質不佳亦或是服務本身的問題。

關於服務合約的部分，require（前置合約）與 ensure（後置合約）的利用在本研究中最大的好處就是在客戶端就可以直接進行服務合約先備條件的檢查，不

須透過網路傳送資料到伺服端之後才進行條件的檢查，如此一來不但可以有效降低網路及伺服器主機的負擔還可以減少因傳輸資料所造成的錯誤機率。雖然如此，但是如果網路品質的狀況真的不佳，還是有可能在傳輸的過程中資料發生錯誤，所以雖然已經在客戶端進行了先備條件的檢查，但仍無法取代伺服端在進行服務之前的檢查。但本研究對於服務合約先後備條件的制定方式並沒有設計得很周全，若未來能針對合約條件的制定提供更強大，更彈性的方法，那麼本研究將能更趨近完整。

第二節　未來展望

除了在服務合約的先後備條件的制定方式，在未來能提供更為彈性的設計之外，針對整合開發環境的支援度也期望能更完全。目前已經有整合開發環境支援直接匯入 WSDL 的檔案並主動轉換成網站服務的函式標頭檔，若 W3C 能接受服務合約並正式給予定義(WS-Contract)，進而納入 WSDL 正式規格，更進一步地各個整合開發環境也能加入支援，如此便能提供利用第三方網站服務做為元件進行整合服務軟體開發的程式設計人員更為便利的開發環境。