第四章　系統設計與實作

System Design & Implementation

本研究是偏理論型研究，重點在提出服務合約的構想，但是要怎麼樣說明服務合約這個構想是有效且可行的，才能為本研究提出更有力的證據，於是我們著手進行網站服務合約系統的建置。網站服務有著伺服端及客戶端，於是本研究也分成兩個部分進行。服務合約全系統流程圖如圖 4。

第一節　系統簡介

網站服務的伺服端，即服務提供者，提供了網站服務及服務合約。將服務合約以註解的形式寫在服務的程式碼裡，再透過本研究的伺服端工具－「Service Contract Parser」，剖析(parse) 網站服務程式碼內的服務合約區塊，並且將其轉成 XML 格式的服務合約檔，提供給服務請求者透過本研究的客戶端工具使用。

而網站服務的客戶端，即服務請求者，對服務提供者提出服務請求。本研究的客戶端工具－「Service Contract Injector」，可以將 Service Contract Parser 所產生的服務合約檔和使用網站服務的程式碼一併處理，將服務合約的內容整合至客戶端的程式碼中，再經過編譯後即完成使用包含服務合約的網站服務的軟體了。並且在整合的過程中也順利將合約例外的處理機制放入客戶端的程式裡，程式設計人員只需要一套合約例外處理機制就能處理在使用網站服務中，不管是預期的或是非預期的錯誤了。

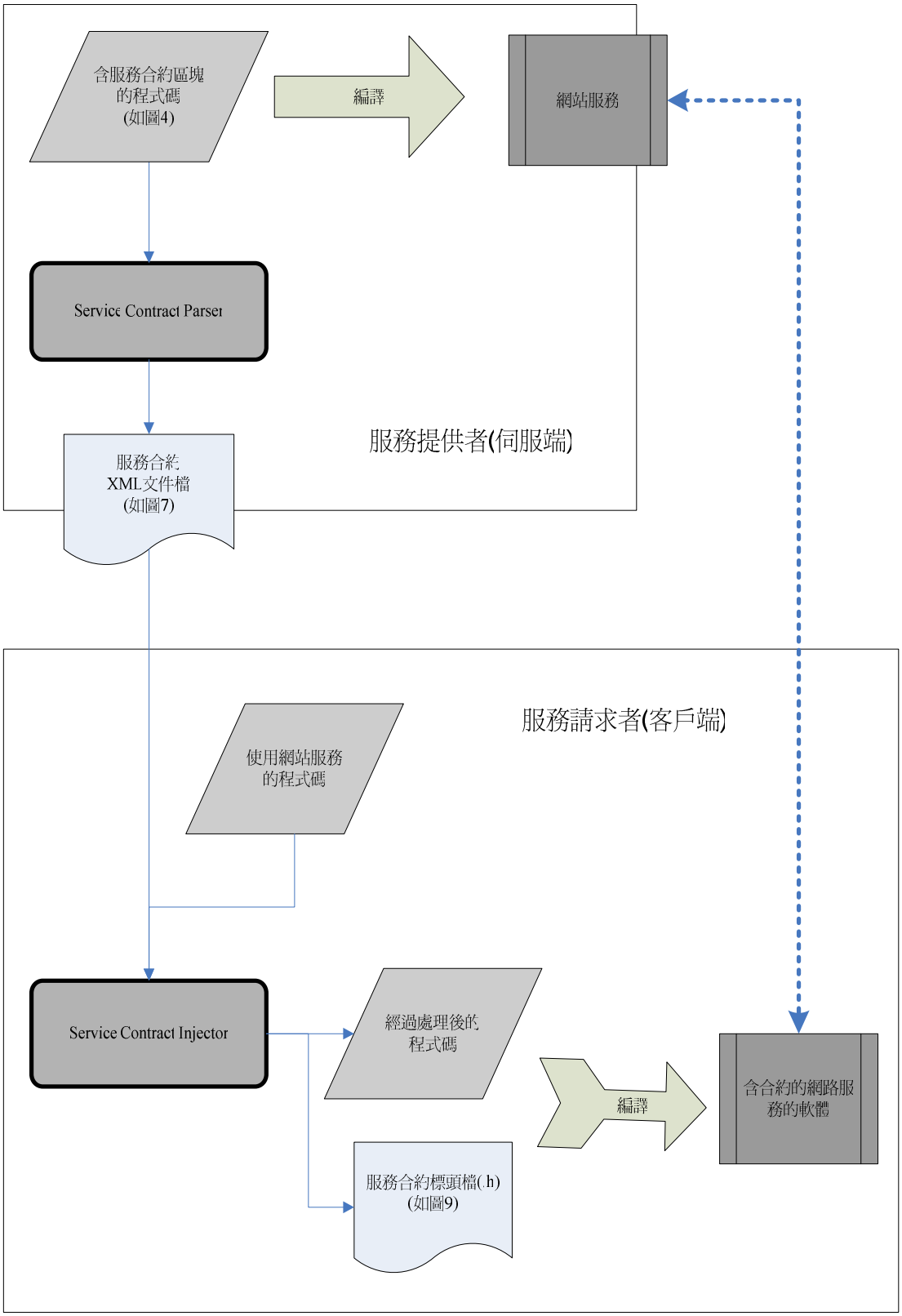

圖 4：服務合約全系統流程圖

第二節　系統架構

本研究的系統主要分成兩大部分，服務伺服端及服務客戶端。原則上是分開開發的，但是有共用兩個類別「ProgrammarParser」及「XMLTree」，全系統架構圖如圖 5 所示。

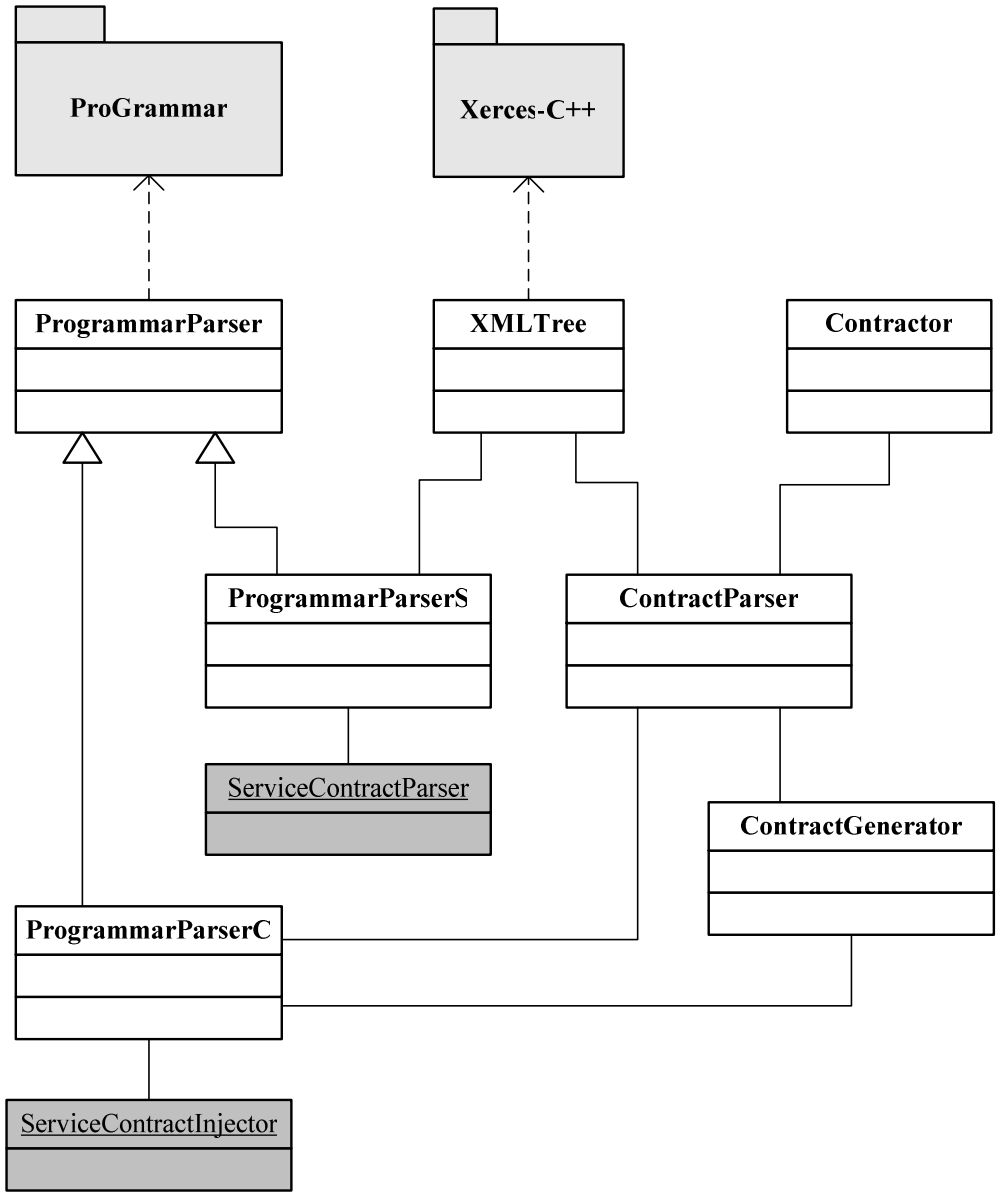

圖 5：服務合約全系統架構圖

ProGrammar™是一套 Parser Generator，只要給它合法的 Language Grammar，透過 ProGrammar™幫我們編譯成 Parse Engine，再將要處理的程式碼交給 Parse Engine 處理，就可以產生 Parse Tree，接著利用 ProGrammar™所提供的 API，就可以直接對 Parse Tree 進行資料檢索。

我們利用 ProGrammar™所提供的 C++ Grammar 加以修改，將原來的 /**/ 註解區塊與 /*@CONTRACT … */ 合約區塊分開，再加上部分微軟在 .NET Framework CLR(Common Language Runtime)才出現的參考類別(ref class)符號，如^、%等，以及我們所制定的服務合約的語法與文法，成為符合我們所需要的文法檔。

Xerces-C++則是 C++版本的 XML Parser，是 Apache XML Project 的一支，提供 C++版本的 XML API 供大家使用。我們利用這一套工具在伺服端產生以及在客戶端解析這一個 XML 格式的合約檔。

順帶一提，ProGrammar™及 Xerces-C++都需要在 Microsoft Visual C++ 6.0 的環境下進行開發。

第三節　Service Contract Parser

Service Contract Parser 是本研究實作的服務伺服端工具，協助服務提供者將服務合約（如圖 3）轉成 XML 格式的合約檔，如圖 6 所示。

合約檔的 XML 文件是樹狀架構，以<Contractor>為起始標籤(root tag)，接下來第二階層的<class>指的是服務函式的類別，標籤內有兩個屬性 name 及 namespace，記錄該服務的名稱及命名空間。第三階層分別是數個<operation>，屬性 name 記錄著服務函式的名稱，而第四階層開始屬於服務合約的部分，分別是<param>、<retval>及<contracts>，前兩者含有屬性 type 記錄著傳入參數與傳回值的資料型態，而後者則是合約的主要內容，包含了 require、ensure 及 RTT，記錄在第五階層<contract>的屬性 type 中。

```
<Contractor>
    <class name="…" namespace="…">
        <operation …>…</operation>
        <operation name="divide">
            <param type="int" />
            <param type="int" />
            <retval type="int" />
            <contracts>
                <contract type="require">_para1 != 0</contract>
                <contract type="ensure">_para0 == _return_ * _para1</contract>
                <contract type="RTT">3000</contract>
            </contracts>
        </operation>
        <operation …>…</operation>
    </class>
</Contractor>
```

圖 6：合約檔的 XML 格式

圖 7 為 Service Contract Parser 的使用方法，當使用者執行 Service Contract Parser 時，若未給定參數或參數數量錯誤，則會顯示出 Usage。若未指定輸出檔名，則預設輸出檔名為輸入檔名後加上「.contract.xml」。若輸出檔名和輸入檔名相同，則會在輸入檔名後加上「.bak」做好備份。此外如果要處理的檔案內容有誤，造成 parse 錯誤的話，也會顯示出錯誤訊息。

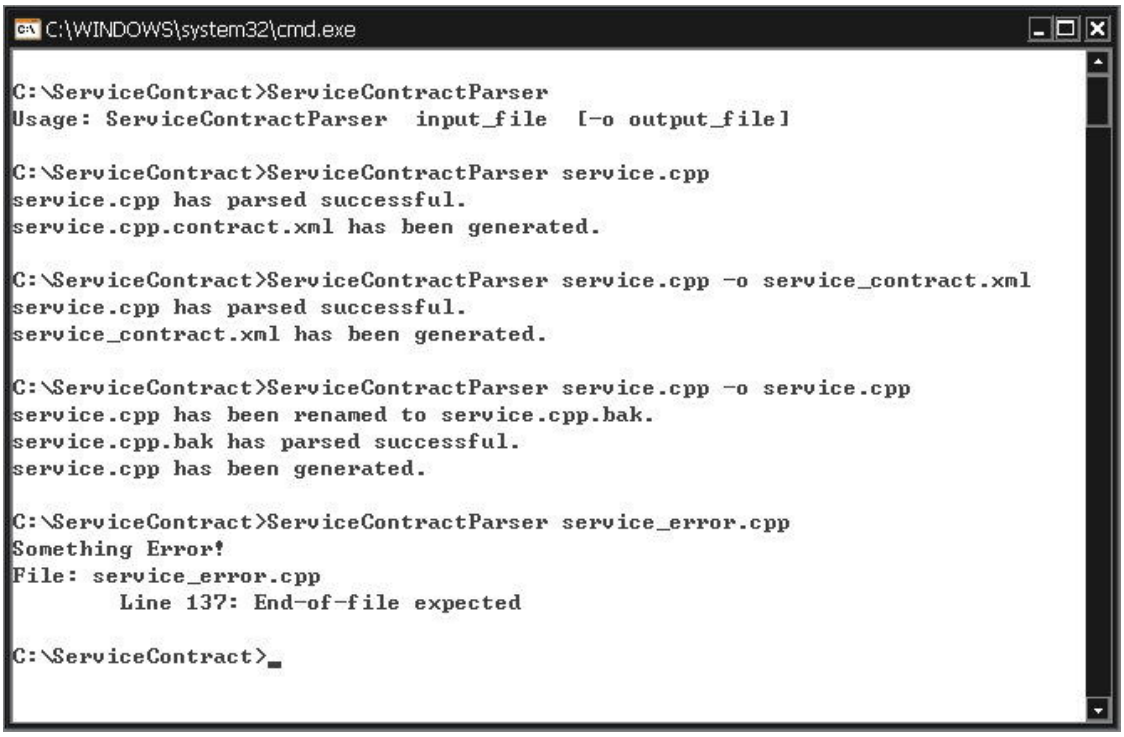

圖 7：Service Contract Parser 的使用方法

第四節　Service Contract Injector

Service Contract Injector 是本研究實作的服務客戶端工具，協助服務請求者將服務合約匯入程式中，將原來有使用網站服務但不含服務合約的程式（如圖 8）處理成包含服務合約內容與合約例外處理的程式檔（如圖 10）及另外產生的一個合約標頭檔（如圖 9）。

```
…
01    int main() {
02          WebReference::WSService ws;
03          …
04          try {
05                ws.divide(3, 0);
06          } catch ( System::Net::WebException^ ) {
07                // do_something...
08          } catch ( System::Web::Services::Protocols::SoapException^ ) {
09                // do_something...
10          }
11          …
12    }
```

圖 8：不含服務合約的原始程式碼

經過實驗結果，正常的網站服務只會捕捉到兩種 System::Exception，分別是 System::Net::WebException 以及 System::Web::Service::Protocols::SoapException。後者處理的是網路服務相關的問題，只要伺服端所回傳的 HRESULT 不為 S_OK，則客戶端就會捕捉到後者的例外，而前者主要處理的是網路相關的問題，如逾時

或斷線，但是逾時的問題不只是網路可能造成的問題，服務本身也有可能造成逾時，而本研究針對這個部分有提出判斷的方法，待下一章節詳述。

```
01  ref class ContractException : public System::Exception {…};
02  ref class EnsureException : public ContractException {…};
03  ref class RequireException : public ContractException {…};
04  ref class RTTException : public ContractException {…};
…
11  int divide(int _para0, int _para1) {
12          WebService _ws;
13          int _return_;
14          try {
15                  if (!( _para1 != 0 )) throw gcnew RequireException (…);
16                  int _orig_timeout_ = _ws.Timeout;
17                  _ws.Timeout = 3000;
18                  _return_ = _ws.divide ( _para0,   _para1);
19                  _ws.Timeout = _orig_timeout_;
20                  if (!( _para0 == _return_ * _para1 )) throw gcnew EnsureException (…);
21          } catch ( System::Net::WebException^ ) {
22                  echo(); throw gcnew RTTException (…);
23          } catch ( System::Web::Services::Protocols::SoapException^ ) {
24                  throw gcnew ContractException (…);
25          }
26          return   _return_;
27  }
```

圖 9：合約標頭檔的內容

合約標頭檔的內容一開始是宣告合約例外的參考類別(ref class)，當然要繼承自 System::Exception，再來是根據服務合約產生包含服務合約例外處理的新服務函式。以圖 9 為例，divide 服務有兩個的傳入參數，以及傳回值都是 int。接著是

檢查合約先備條件、設定服務回應時間，最後檢查合約後備條件。Service Contract Injector 會修改使用者原來的程式碼，將呼叫原來的服務函式改變成呼叫經過包裝的函式，這些函式經過包裝後已經含有合約例外的內容，當服務不符合合約的內容，便會丟出合約例外。而逾時的時間設定，我們在研究初期是利用執行緒(thread)進行計時，隨著研究不停的進展，我們找到了更為簡易的設定方法，設定網站服務的 Timeout 屬性，則若在時間內未收到網站服務的回應，便會捕捉到前者 System::Net::WebException 的例外，判斷成網路服務逾時，我們再將其轉成服務合約逾時例外(RTTException)拋出，如此更能明確細分各種例外，並加以進行適當的例外處理。

以圖 8 來說，在沒有使用服務合約的情況下，如果服務伺服端沒有處理除數為零的情況下進行除以零的動作會造成伺服器主機的當機，使得網路服務因收不到回應而造成逾時，此時會收到 System::Net::WebException 的例外，但事實上並不是因為網路的問題而造成逾時。若使用了服務合約的情況下，此時便會判斷成先備條件不符而以 RequireException 進行合約例外處理，如此一來也比較貼近事實。

我們會處理服務使用者原來的程式，將原本直接的服務呼叫改成呼叫我們包裝過服務合約的函式，再經由我們的函式呼叫原來的服務函式，這些動作都可以透過我們的客戶端工具直接完成。

```
…
01   int main() {
02         MyWebService _my_ws_;
03         …
04         try {
05               _my_ws_.divide(3, 0);
06         } catch ( System::Net::WebException^ ) {
07               // do_something...
08         } catch ( System::Web::Services::Protocols::SoapException^ ) {
09               // do_something...
10         } catch ( ContractException ) {
11               // do_ContractException_handling...
12         } catch (RequireException) {
13               // do_RequireException_handling...
14         } catch (EnsureException) {
15               // do_EnsureException_handling...
16         } catch (RTTException) {
17               // do_RTTException_handling...
18         }
19         …
20   }
```

圖 10：使用服務合約後的程式碼

我們認為原先的網路服務所提供的兩種例外並不足夠，舉例來說如服務發生逾時，便會收到 System::Net::WebException，而逾時卻不只是因為網路的問題造成，如服務設計不當造成伺服器當機而無法回應也會造成逾時的現象；又如果在服務提供端因為檢查發現除數為零而不執行，以非 S_OK 的 HRESULT 值回傳，則服務使用端會收到 System::Web::Services::Protocols::SoapException 的例外，無法得知發生例外的原因，這都會對服務使用端的程式設計人員帶來莫大的困擾。

所以我們希望利用服務合約的加入來協助我們更能清楚且明確的區分各種例外發生的原因，以便採取合理且最為適當的例外處理。

圖 11 為 Service Contract Injector 的使用方法，當使用者執行 Service Contract Injector 時，若未給定參數或參數數量錯誤，則會顯示出 Usage。若未指定輸出檔名，則預設輸出檔名為輸入檔名在副檔名前加上「.new」。若輸出檔名和輸入檔名相同，則會在輸入檔名後加上「.bak」做好備份。此外如果要處理的檔案內容有誤，造成 parse 錯誤的話，也會顯示出錯誤訊息。

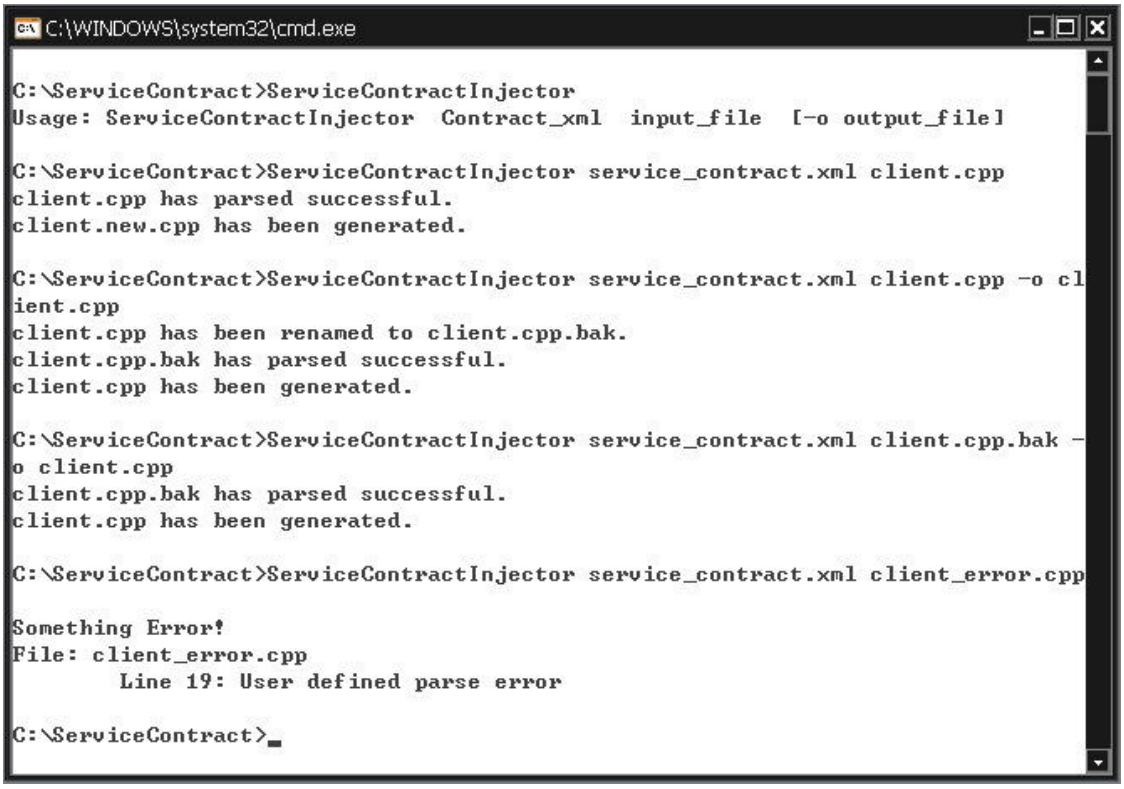

圖 11：Service Contract Injector 的使用方法