一個支援網際服務監督及測試的工具
A Monitoring and Testing Tool for Compositions of Web Services

劉建宏
國立台北科技大學資訊工程系
cliu@ntut.edu.tw

林修儀
國立台北科技大學資訊工程系
s3598011@ntut.edu.tw

摘要

近幾年以服務導向架構(Service Oriented Architecture, SOA)為基礎的網際服務(web service)逐漸受到重視,被認為是解決異質系統整合問題的有效方案。如同其他的軟體一樣,網際服務亦必需被嚴謹地測試,以確保其品質與可靠性。本論文提出一個支援測試及監督網際服務互動行為的工具,採用行動代理人的架構來攔截與模擬不同服務間的 SOAP 訊息,並實際透過設計和執行測試案例,以驗證網際服務間互動流程的正確性。

關鍵詞:服務導向架構,網際服務,網際服務測試,行動代理人

Abstract

In recent years, web services have draw many attentions along with the widespread acceptance of the Service-Oriented Architecture (SOA). They have been considered a promising solution and standard for integrating heterogeneous and distributed systems. Like other traditional software, web services also require to be tested thoroughly in order to ensure their quality and reliability. In this paper, we propose a tool to support monitoring and testing compositions of web services. The tool is developed based on a mobile agent-based framework for intercepting and simulating SOAP messages between different web services. Test cases are developed and exercised to show the effectiveness of the tool for verifying the correctness of interactions among web services.

Keyword : Service-Oriented Architecture, web services, web service testing, mobile agent

一、前言

在日常生活中,我們經常使用許多商業服務,例如:金融服務和物流服務等。這些商業服務大多都需要用到資訊系統來實現。但各家企業所使用的系統不同,系統之間的溝通協定也不一樣,使得系統間的整合一向被視為是件困難的工作。近年來網際網路技術的蓬勃發展,使得以服務導向架構(Service Oriented Architecture, SOA)為基礎的網際服務(web service)逐漸受到重視,由於網際服務的技術是架構在 web 的平台上,加上既有資訊系統的功能很容易轉換成網際服務[7],因此網際服務被視為是解決異質系統整合問題的有效方案。

跟傳統的軟體一樣,網際服務亦必需被嚴謹和徹底地測試,以確保其品質與可靠性。然而測試網際服務被認為與測試傳統軟體有所不同。主要是因為網際服務具有多種程式語言和異質平台的特性,因而增加了測試的複雜度。且測試人員僅擁有描述網際服務介面與訊息格式的 WSDL 資訊,無法獲得網際服務的原始碼(source code)來進行分析和瞭解網際服務的結構與行為。

目前市面上網際服務的測試工具大多僅能提供 SOAP 輸入輸出的介面,並由測試人員藉分析 WSDL 所訂定的服務介面和參數格式來產生 SOAP 的訊息,用以測試網際服務的功能,或者是驗證 SOAP 訊息格式是否正確。然而這樣只能對單一網際服務進行測試,對於由多個網際服務所組成的網際服務應用程式而言,網際服務之間必需有適當的協調(orchestration)和互動,並依照所訂定的流程(workflow)提供正確的服務。但簡單的 SOAP 訊息測試並不能滿足對驗證網際服務協調互動的需求,例如一個旅遊業的網際服務可能包含飛機訂票服務、租車預定服務、和旅館訂房服務。這些服務的協調互動有一定的流程規範,例如租車的取車時間必需在飛機抵達時間之後;若取消飛機班次,則預定的租車和旅館皆一併取消。若要測試這個旅遊業的網際服務是否正確,則飛機訂票、租車預定、和旅館訂房服務間的互動必需驗證以確定其流程規範是否能夠被滿足。

有鑑於此,本篇論文提出一個網際服務測試和監督的工具,協助開發人員驗證網際服務組合互動的行為。此工具是以一個行動代理人的架構為基礎,可以攔截與模擬不同服務間的 SOAP 訊息,使得在不干擾遠端服務的情況下,可以測試和監督本地端與遠端服務間的互動是否正確。此外,我們亦描述如何以 XML 來訂定測試案例,以便測試案例能被傳送至不同的網際服務端進行模擬和測試網

際服務間的協調與互動。

本論文組織如下：第二部份回顧目前網際服務測試的相關研究，第三部份針對本論文所使用的相關技術做一簡單介紹，第四部份描述欲解決的問題及解決方法，第五部份對本文所提出的測試工具架構加以介紹，第六部份以一個範例說明如何設計測試案例與如何執行，第七部份則為本篇論文結論及未來研究方向。

二、相關研究

Tsai[1]等人提出 WSTF (Web Services Testing Framework)的測試框架。以 agent 技術為基礎，可運用於 SOA 架構系統。由於網際服務描述規格(WSDL)僅描述了呼叫界面的規格，對於測試來說，不管是測試案例的推導或呼叫流程的檢驗，WSDL 所提供的資訊都有所不足，Tsai 提出包括 I/O Dependency 與 Invocation Sequences 等四項 WSDL 的延伸規格，並且使用了 SMtSS 方法模型化多個 service 的同時性行為 (concurrent behavior)。透過這些延伸功能的協助，在 SOA 系統的 black-box testing 與 regression testing 方面能夠更容易的進行。在測試案例的推導上，是使用 Scenario Specification[7]的方法進行推導。

Looker[3]等人提出以錯誤注射(Fault-Injection)為基礎的測試方法。由於網際服務用以交換資料的 SOAP 封包格式是基於 XML 而來，Looker 透過修改網際服務的容器(container)的方式，得以在 service provider 與 client 中間加入一個 Injector Server，用以監控 service provider 與 service requester 兩端所有的訊息交換，並且可根據測試案例的設定，將可能引發錯誤的資料插入在正常的訊息中，觀察受測的網際服務是否能夠正確的對應異常的訊息，如：含有錯誤內容、內容缺漏等等。

Haslinger 和 Dustdar [2]提出一個 SITT (Service Integration Test Tool)的工具來支援測試服務導向架構(SOA)的應用程式。這個工具的架構主要包含有 Test Agent、Master Agent、Test Daemon、及 Test Database，其中 Test Agent 被部署到各個服務伺服器上收集伺服器所記錄的 Log，包含 Message ID、ServiceName、Timestamp、MessagePriority 等資訊。然後將所收集到的資料送往 Master Agent 做彙整，最終會集中到一個 Test Database 上，並由 Test Daemon 進行分析驗證測試結果。SITT 的架構可適用於 Intranet 或 Internet 的服務，同時該工具並提供兩種模式：Online 或 Batch。其中 Online 模式可進行即時線上的測試，而 Batch 模式則是將收集到的 Log 在某個時段送回給 Master Agent 後再進行測試。然而 Haslinger 和 Dustdar 所提出的工具著重在資料的收集與監督，並未詳細說明如何測試網際服務的協調與互動。本論文所提出的架構中，Test Agent 的責任並不止收集伺服器上的 Log，同時能根據我們所設定的資料，模擬網際服務行為。而使用者亦可透過界面將新的測試資料傳送給 Test Agent，並能清楚的瞭解測試案例是否通過。

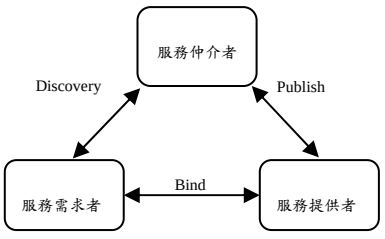
Offutt 和 Xu [5]提出利用資料擾亂(Data Perturbation)的方式產生不同的 SOAP 參數資料，並分析其回應的訊息以驗證單一(peer-to-peer)網際服務的正確性。Offutt 和 Xu 共提出三種資料擾亂的方法：Data Value Perturbation (DVP)、RPC Communication Perturbation (RCP)、和 Data Communication Perturbation (DCP)。其中 DVP 主要是根據 WSDL 所定義的參數資料格式，如字串或數值，透過 Boundary Value Analysis 的方法來產生不同的參數資料。RCP 則是透過某些 mutation operators 對參數進行運算以產生不同的參數資料。而 DCP 則是針對使用 XML Messages 傳送資料的網際服務進行測試。Offutt 和 Xu 透過一些規則修改 SOAP 訊息中 XML Messages 的內容，並藉由這些 XML Messages 來測試網際服務其資料庫的存取是否正確。

Yu[8]等人提出一個能夠彈性定義流程的開發與測試環境，使得組合網際服務能夠以十分方便的方式進行。並且提出 Virtual Partner 與 Inspector 兩個機制來協助開發人員檢驗流程的正確與否。Yu 指出，由於 BPEL 執行環境並沒有定義標準的界面供外部取得相關的資訊，如網際服務用以交換資料的變數內容等，因此在一個流程的執行過程當中必須要透過一個稱為 Inspector 的網際服務來探測執行過程中發生的變化，藉由將 Inspector 插入到流程的各個部份，以取得流程運作中的動態資訊，使得測試工作得以進行。

三、背景介紹

3.1 服務導向架構系統

隨著網際服務漸受重視，以網際服務之間的互動構成的服務導向架構亦趨重要。圖一說明服務導向系統中的參與角色。與傳統的主從式(client/server)架構不同的是除了服務的提供者與發出請求的服務需求者之外，增加了一個居中的服務仲介者(service broker)。



圖一 網際服務的架構

服務提供者能將網際服務透過服務仲介者發佈出去，而服務需求者則向服務仲介者詢問是否有合乎需要的服務可用，因此能夠動態的組合出一個系統。而做為 SOA 系統的礎石，網際服務奠基於許多標準之上，包括以 XML 為基礎的訊息交換機制(Simple Object Access Protocol, SOAP)、超文件傳輸協定(HTTP)及網際服務描述語言(WSDL)[11]等等，由於 SOAP 是一個以文字為基礎的(text-based)訊息交換協定，因此比傳統的 CORBA[10]或 DCE RPC[11]等分散式環境所使用的二進位訊息更容易瞭解與整合。

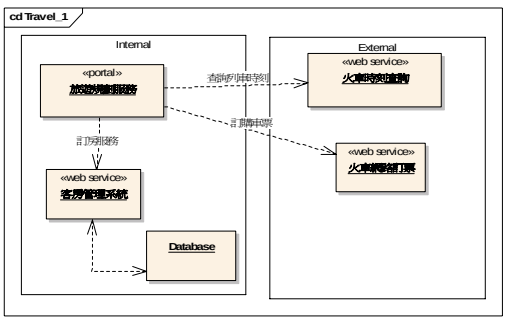
3.2 行動代理人

行動代理人[6]是一個具有能夠在異質網路中動態的進行移動的程式。由於動態移動的特性，所以能夠根據我們賦予的任務，移動到適當的地方執行。並且代理人程式具有自主執行的特性，同時也能夠與其他的代理人程式互相溝通，並具有高度的擴充性，因此除了功能性測試之外，尚可取得如回應時間、平均傳輸速度等數據，可協助維護人員瞭解系統的運轉情況。

四、問題描述與測試方法

在一個網際服務應用系統的開發過程中，除了需要對企業內部自行開發的網際服務進行許多測試，以便驗證功能是否正確之外，如果使用到了其他廠商所提供的網際服務，也必須要檢驗是否已經正確的到整合到系統中。我們以一個提供旅遊套餐服務的旅遊規劃網站為範例，說明一個網際服務應用系統在企業內部與外部的網際服務整合上會產生的問題。

圖二為一個旅遊規劃服務網站，其中旅遊規劃服務與客房管理系統屬於企業內部，但是若要結合大眾運輸系統提供給客戶更完整的服務，就必須要整合相關的服務，如：火車時刻查詢、訂票服務等等，而這些服務不可能完全由企業內部自行開發，因此必須要由其他的廠商提供。

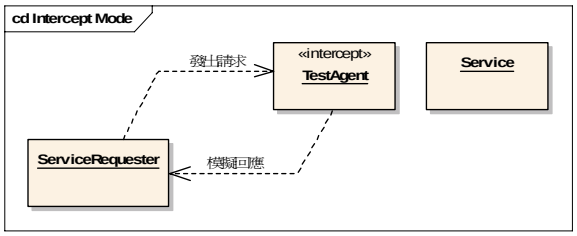


圖二 一個旅遊規劃服務網站架構

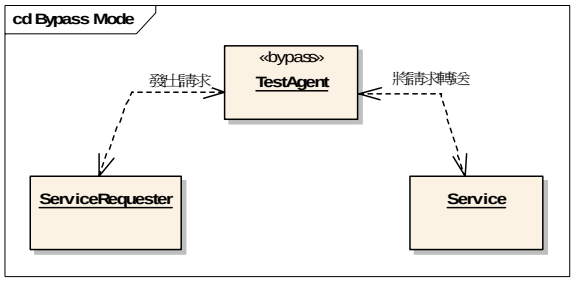
為了驗證企業外部的網際服務是否已正確的整合進來，必須針對各方面進行測試，如資料格式是否正確、呼叫順序是否正確等等。但這些由其他廠商所提供的網際服務可能已經正式對外發佈，而且已經有許多的客戶正在使用，這些服務無法知道我們所發出的訊息是一個測試用的訊息，而會將其視為真正的客戶所發出的請求，如此將造成許多不良的結果，如：訂購了許多的車票導致其他的客戶訂不到票等等。為了避免干擾到已上線的網際服務，必須要求外部廠商提供專屬的測試用的特殊版本服務以供系統整合測試階段叫用。此舉可能導致廠商需要另外進行開發，使得系統整體開發時程將往後遞延，並且增加許多額外的費用及時間，使得開發成本上升。

而當系統開發完畢，在上線測試的階段有許多調校工作待進行，因此必須持續的監督系統行為，並且收集許多相關數據如：平均回應時間、單位時間內的訊息流量等，以做為調校的依據，更重要的是瞭解系統在實際運轉的時候是否穩定。常見的解決方法是利用探針(probe)，即擷取資料的程式片段，將探針安插到系統各部份取得數據，並將其數據傳輸到其他地方，但由於網際服務應用系統是一個分散式結構，各元件彼此只以訊息互相溝通，通常無法取得原始程式，因此也就無法將擷取資料用的探針插入其中。

為此我們提出一個以行動代理人技術為基礎的網際服務測試架構來驗證網際服務之間協調互動的正確性。在這個架構中，我們利用 socket 技術在客戶端與服務提供者之間建立一個轉接層，並將之實作在行動代理人平台上。透過轉接技術，代理人程式能夠如同網際服務一樣的回應訊息給客戶端，使我們能夠進行整合測試而不會干擾到真正提供服務的系統。我們為代理人程式的轉接功能設定了兩種運作模式，分別解決不同的問題。首先，代理人程式能夠藉由訊息攔截的方式，攔住原本要送往網際服務的訊息，而當收到訊息時並不將這個訊息傳送給原本應處理的網際服務，而是以定義好的回應訊息回傳給服務請求者，如圖三，此時我們稱這個代理人程式處於攔截模式(intercept mode)。代理人程式的另一種運作模式稱為轉送模式(bypass mode)，如圖四。亦可將收到 SOAP 訊息時會，此種。當代理人程式處於轉送模式時，一旦收到客戶端的請求訊息時將原封不動的轉送給對應的網際服務；同時將一些數據如：訊息抵達時間、訊息大小等，收集起來後回傳給另外一個代理人程式。



圖三 攔截模式



圖四 轉送模式

進行測試時，我們將測試資料傳輸給代理人程式並設定其運作在攔截模式，以確認網際服務之間的互動是否正常，例如前述旅遊規劃服務系統中，一個交易必須要同時訂到客房與車票才能算是成功，因此我們可根據訂到車票與否這個條件，訂出兩個測試案例。為了能夠完整的執行這兩個測試案例，我們需要佈署一個代理人程式並設定其攔截火車網路訂票服務，當我們令代理人程式回傳訂不到車票的回應訊息給旅遊規劃服務系統時，則可預期旅遊規劃服務系統將因無法訂到車票而告知使用者交易失敗，但若此時旅遊規劃服務系統仍然告知使用者交易成功，則表示有錯誤存在。緊接著我們令代理人程式回傳訂到車票的回應訊息，可執行另一個測試案例。另外當系統進入維護階段，可將代理人程式設定在轉送模式，藉以取得系統運作的相關數據。

4.1 測試案例結構

一般而言，欲測試網際服務之間的互動關係無法只單純的呼叫待測的函數並觀察其結果就可完成。但當整合牽涉到非自行開發的網際服務時更有許多的困難，所以需要使用行動代理人技術，來協助我們攔截或監督指定的網際服務，並根據測試資料產生模擬訊息。因此一個測試案例的內容除了包括待測的網際服務函數、輸入參數及預期結果為何之外，還需要包含代理人程式的模擬回應，而這些資訊必須要以結構化的方式有效的加以管理，為此，我們提出一個適用於測試網際服務互動的測試案例結構，其中包含數個部份，詳述如下：

一、關連宣告(partner)。由於網際服務之間的互動需要使用到許多的網際服務，但每一個網際服務的服務位址皆為一個 URL，由於是一個長度相當長且複雜的字串，非常容易造成輸入上的錯誤，為了減低發生錯誤的機會，因此將使用到的網際服務位址宣告一個較簡短而有意義的名稱，使得引用上較為方便。

二、前置條件(precondition)。指明有那些代理人程式需要產生模擬回應，並且包含產生模擬回應所需要的測試資料。相關的代理人程式開始執行測試案例前就會收到此處所紀錄的資料，當測試案例開始執行，被設定為攔截模式的代理人程式將根據前置條件中的資料製造出模擬訊息回應給請求者。

三、待測函數(target)。指明該測試案例所要進行測試的是哪一個函數。

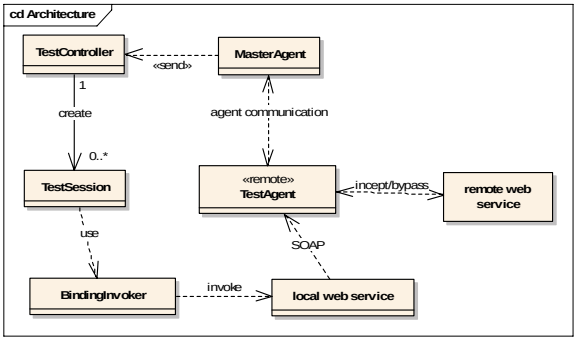
四、輸入參數(input)。指明呼叫待測函數時傳入的參數值。

五、預期結果(assert)。根據呼叫待測函數的回傳結果，與預期獲得的結果相比對，做為判斷是否成功的依據。

為配合測試網際服務測試協調互動所訂定的測試案例範例，詳細內容於第六節中說明，完整的測試案例如附錄一所示。

五、系統架構簡介

為了測試網際服務組合互動的正確性，我們採用一個以行動代理人為基礎的測試架構，如圖五所示。



圖五 系統架構

透過這個架構，可以執行測試案例以監督和測試網際服務的行為。為簡化說明，此處只列出五個核心類別，功能詳述如下：

MasterAgent

MasterAgent 為實作行動代理人技術的類別，佈署於測試主機上。MasterAgent 負責居中掌控所有在遠端的 TestAgent，其任務有三：

一、建立並派遣 TestAgent 到指定的位置

二、接收來自 TestAgent 的訊息

三、與 TestController 互動，將訊息呈現到使用者界面

TestAgent

為實作行動代理人技術的類別，佈署於提供服務的遠端主機上，負責監控所有流往網際服務的 SOAP 訊息封包。對每一個流往網際服務的訊息，TestAgent 執行以下的任務：

一、將封包攔截下來，並根據測試案例產生一個模擬的回應訊息送回給服務請求者，此處我們假設測試案例中已包含了網際服務所產生的回應資料

二、將封包的相關資料紀錄下來送回給 MasterAgent

TestController

負責與 MasterAgent 溝通，控制測試案例的執行。當使用者開始執行一個測試案例時，系統將建立一個 TestController，負責將前置條件傳遞給 MasterAgent 並監督接下來的執行動作。在測試過程進行當中可能因為各種原因如伺服器當機或網路斷線等而導致無法收到回應，TestController 將根據逾時(timeout)機制或使用者取消測試而中斷測試案例的執行。

TestSession

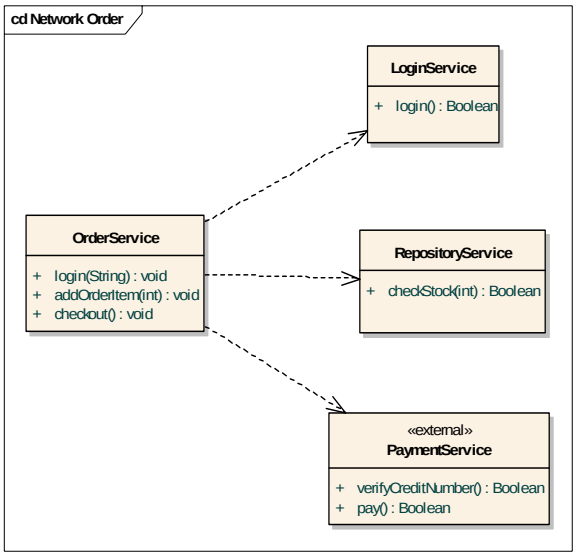
對每一個函數而言，可能會在許多不同的測試案例都會被呼叫到，但是對 TestAgent 來說無法分辨目前的呼叫動作是屬於哪一個測試案例，並且為了兼顧效率因素，TestAgent 是以非同步方式將訊息回傳給 MasterAgent，實際上每個訊息被送出的時機無法事先決定，因此造成抵達 MasterAgent 的先後順序上發生混亂。因此針對每一個呼叫動作，TestController 建立一個 TestSession 進行區隔，其中包含了一個獨一無二的識別碼，MasterAgent 可依據該識別碼將訊息正確的予以區分。

Binding Invoker

根據測試案例所指定的測試目標，以動態方式呼叫待測函數。網際服務所使用的 SOAP 訊息格式是以 XML 為基礎，因此若要直接產生一個 SOAP 訊息則需要建立許多 XML 標籤(tag)，使用上殊為不便，因此有許多函式庫均提出了便捷的解決方案；以 Java 語言為例[13]，常見的作法是使用一個代理類別(proxy class)負責將呼叫網際服務的動作包裝成為 SOAP 訊息封包。但是由於網際服務是一跨語言的標準，而我們無法預先得知待測的網際服務是以何種語言實作的，若待測的網際服務並非以 Java 實作時，就無法取得其代理類別。有鑑於此，我們使用 Java 語言提出的動態呼叫界面(Dyncmic Invocation Interface, DII)來解決此一問題。透過動態呼叫界面，可在執行期解析欲呼叫的網際服務對應的 WSDL，動態的解析待測函數的參數型態，並進行呼叫，而不需要透過代理類別。

六、系統使用範例

茲以網路訂貨及付款服務為一個範例，展示我們如何為待測的網際服務設計其測試案例，及如何運用本工具來執行這些測試案例。圖六描述一個整合了線上付款機制的網路訂貨服務。圖中的每一個物件均是一個網際服務，每個網際服務各自包含了數個函數。

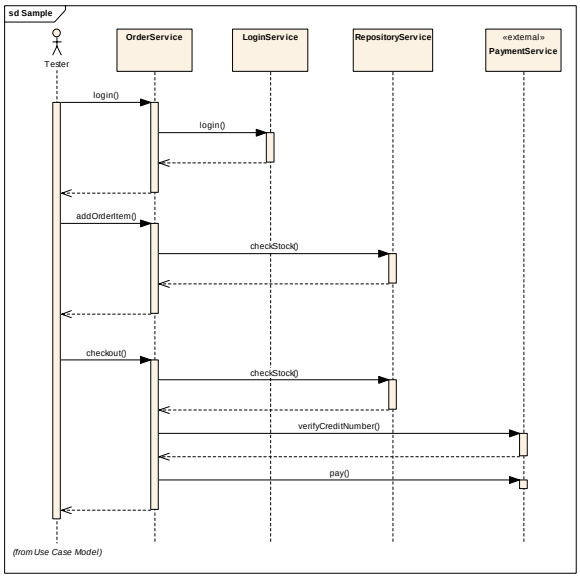


圖六 網路訂貨及付款服務

在這個範例系統中，OrderService 是為了能夠提供使用者線上訂購而新開發的網際服務，為了能夠使付款更方便，因此使用了其他廠商所提供的付款機制服務(以 PaymentService 表示)。除此之外，客戶管理與倉儲管理則是由舊有系統所延伸出來的網際服務，分別以 LoginService 與 RepositoryService 表示之。以下使用外部服務一詞來描述像 PaymentService 這種不是自行開發而是由別的服務提供者供應的網際服務。

由於這個訂貨系統只開放給已註冊的客戶訂購貨品，因此在開始訂購任何物品前，客戶必須先行登入，訂貨系統會利用 LoginService 中的 login 這個函數來處理登入動作。登入完畢後，客戶就能夠開始訂購物品，訂貨系統將使用者欲訂購的品名與數量傳送給 RepositoryService 的 checkStock 函數，以便確認庫存量足以應付此筆交易；一旦確定數量充足，訂貨系統將利用 OrderService 進行訂貨並加入到購物籃中。當客戶挑選完所要訂購的物品之後，訂貨系統將再次呼叫 OrderService，利用 checkout 函數處理所有的計價及結帳工作，我們設定客戶必須使用線上付款，因此系統將要求客戶提供信用卡號，當訂貨系統收到客戶輸入的信用卡號後，將呼叫 PaymentService 的 verifyCreditNumber 函數以確定使用者提供了正確的信用卡，通過驗證之後訂貨系統才會正式承認此筆交易並收取貨款。圖七中描述了這個定貨系統中每個服務之間的互動關係。

在此例中，我們希望能夠設計一些測試案例，對 OrderService 這個網際服務的正確性加以驗證，以便瞭解這些相關的網際服務是否已正確的整合到系統中。



圖七 網路訂貨系統的循序圖

根據規格，我們可以知道如果客戶訂購的貨物存量不足或是客戶的信用卡無法通過驗證則交易將失敗，因此 checkout 這個函數的回傳結果受兩個因素所影響：

1、 呼叫 RepositoryService 確認存貨是否充足。

2、 若使用者以信用卡線上付款，則需要呼叫 PaymentService 檢查卡號是否正確。

我們根據 Decision Table 的方法來推導出測試案例，如表一所示。

表一 checkout 函數的 Test Case

Condition	TC1	TC2	TC3	TC4
checkStock 的回傳值	true	true	false	false
verifyCreditNumber 的回傳值	true	false	true	false
checkout 函數成功	V			
checkout 函數失敗		V	V	V

在表一中，我們將 verifyCreditNumber 及 checkStock 兩個函數的回傳值視為 condition，以此推得的四個測試案例分別以 TC1 到 TC4 表示。因篇幅所限，我們以表一中 TC2 為例詳述如何執行這個測試案例。

除了對上述的功能性需求，我們希望能夠驗證其正確性之外，我們也希望瞭解訂貨系統的一些非功能性需求，如：每次 login 處理客戶登入的需求需要耗時多久。若登入處理的時間過長，容易造成客戶不耐等待，則需要加以改善。由於完整的測試案例內容較長，以下分段加以說明，完整的內容詳列於附錄。

由表一中 TC2 的定義，可知這個測試案例目的在測試當 checkStock 回傳 true 而 verifyCreditNumber 回傳 false 時，checkout 能否正確處理。因此我們將 OrderService 中的 checkout 設定為待測函數，並且將預期的回傳結果部份設定為 false，表示若 checkout 回傳 false 則視為執行成功，否則視為失敗。

```
<target>
  <name>checkout</name>
</target>

<assert>
  <value type="Boolean">false</value>
</assert>
```

圖八 待測函數與預期結果

由於 checkout 這個函數受 checkStock 與 verifyCreditNumber 兩個函數回傳值所影響，若其中一個檢查失敗，表示無法完成結帳。因此我們將 checkStock 與 verifyCreditNumber 這兩個函數視為 checkout 的前置條件，我們將前置條件以 <precondition>標籤分別表示之，並在<response>註明回應訊息型態為 Boolean，如圖九所示。

```
<precondition partnerLink="RepositoryService"
        name="checkStock">
  <response type="Boolean">true</response>
</ precondition >

< precondition partnerLink="PaymentService"
        name="verifyCreditNumber">
  <response type="Boolean">false</response>
</ precondition >
```

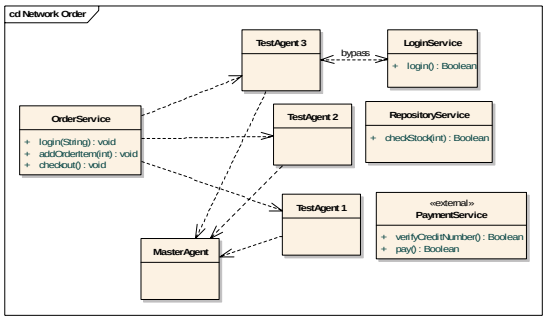
圖九 前置條件

由於 checkStock 與 verifyCreditNumber 分屬兩個網際服務，因此我們設定 verifyCreditNumber 這個函數的 partnerLink 設定為 PaymentService，並將回傳結果將被設為 false，表示不論客戶端傳送任何信用卡號碼，都將收到一個 false 代表驗證失敗；其次設定 RepositoryService 的 checkStock 函數回傳值設為 true，代表貨物庫存量總是充足。藉由前置條件的設計，就能夠清楚的得知待測函數的回應是否如預期。

整個測試案例定義完成之後，我們需要開始執行測試，由於 TC2 中包含了兩個網際服務的函數，因此需要佈署兩個 TestAgent，並且分別令其對應到 PaymentService 及 RepositoryService 的攔截動作，如圖十所示。

我們以 TestAgent 1、TestAgent 2 標示被佈署到 PaymentService 及 RepositoryService 的 TestAgent。另外我們需要量測 LoginService 中的 login 執行一次所耗用的時間，因此我們將監控用的 TestAgent 3 則佈署到系統中，並將 TestAgent 3 設為轉送模式。

此處需要特別說明的是，TestAgent 3 被設為轉送模式，會將資料轉送給 LoginService，因此兩者之間有所聯繫，在圖十中以虛線表示，但 TestAgent 1 與 TestAgent 2 為攔截模式，故並不與任何的網際服務有所聯繫，因此在圖十中 TestAgent 1 與 PaymentService 之間沒有任何線段相連；TestAgent 2 與 RepositoryService 之間亦是如此。



圖十 加入代理人程式後的佈署圖

所有需要的 TestAgent 都佈署完畢之後，我們就能夠開始執行測試案例，執行過程敘述如下：

一、系統將使用者指定的測試案例中所定義的 Precondition Task 的設定個別取出，並將其值傳輸給對應的 TestAgent。此例中我們希望對 PaymentService 及 RepositoryService 採取攔截動作，故將 TestAgent 2 設定攔截 checkStock 並回傳 true，而 TestAgent 1 設定攔截 verifyCreditNumber 並回傳 false。而 TestAgent 3 由於只需要將回應時間送回給 MasterAgent，故設定為轉送模式。

二、TestController 建立一個 TestSession，並針對每一個函數的呼叫動作，賦予 TestSession 一個獨一無二的識別碼以便 MasterAgent 能夠識別 TestAgent 所回傳的訊息是屬於哪一個測試案例。

三、TestSession 呼叫 BindingInvoker，使用動態呼叫界面解析測試案例中待測函數 WSDL，產生並傳遞訊息封包到網際服務。根據測試案例的定義，BindingInvoker 將呼叫 checkout 這個函數。

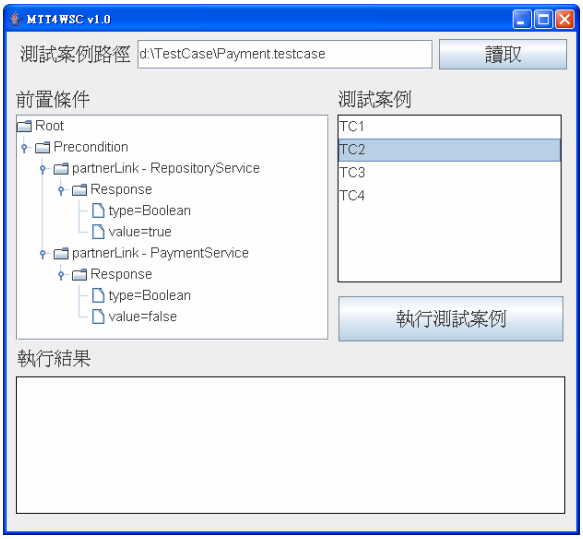
四、當 BindingInvoker 將結果回傳時，TestController 將該回傳值與預期結果進行比對，若相符則視為成功，否則視為失敗。此例中的 TestAgent 1 已被設定攔截呼叫 verifyCreditNumber 的動作並且回傳 false，故 checkout 將因信用卡號碼驗證失敗而告知使用者無法完成結帳。

五、當 TestAgent 所監控的網際服務被呼叫時，TestAgent 會將訊息傳送給 MasterAgent：

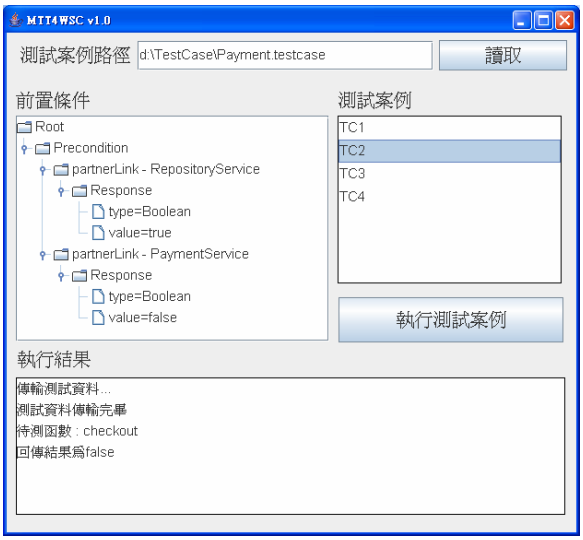
(一). 設定為攔截模式的 TestAgent 如 TestAgent 1 與 TestAgent 2 在進行攔截動作時將告知 MasterAgent 已發生攔截動作。

(二). 設定為轉送模式的 TestAgent 如 TestAgent 3 在將訊息轉送給網際服務時將收集相關數據傳回給 MasterAgent。因此當 LoginService 的 login 函數被呼叫的時候，TestAgent 3 會將每次網際服務的執行 login 所耗用的時間回傳給 MasterAgent，根據這些數據我們可 LoginService 的回應時間是否過長而需要加以改進。

圖十一為本系統執行畫面，畫面右方將顯示目前已讀入的測試案例，畫面左方則顯示目前已佈署的 TestAgent。圖十二為測試案例執行畫面，在呼叫待測函數之前，所有定義在測試案例中的前置條件將透過網路被送達 TestAgent，隨後才呼叫待測函數，並比對其回傳值是否與預期結果相符。



圖十一 測試案例讀取完成



圖十二 測試案例執行完畢

七、結語與未來研究方向

在這篇論文中我們提出一個以行動代理人技術為基礎的網際服務應用系統測試架構，利用訊息攔截技術截取流往網際服務的訊息，並根據測試資料產生回應，同時我們提出一個以 XML 為基礎的測試案例結構，讓使用者能夠很容易的指定前置條件、待測函數及預期結果。透過行動代理人的兩種轉送模式，能夠發揮探針的功用，偵測系統運作中的各項數據，使本系統不但能夠在開發階段使用，在正式運轉時仍然能夠肩負起監督的工作。

目前有許多用以描述網際服務之間互動關係的規範，如 BPEL。未來將擴充本系統，使其能夠根據流程描述語言來推導測試案例，並分析可能的執行路徑，以更有效的方式測試 SOA 架構系統。此外由於行動代理人能夠自主執行，我們亦將延伸代理人的功能，使其支援更多的測試方法，並可於系統離峰時間進行，如：錯誤注射測試[3]和資料擾亂測試[5]等，俾使系統運作能夠更加穩固。

八、參考文獻

[1] X. Bai and W. T. Tsai et al, "Scenario-Based Modeling And Its Applications," Proceedings of the Seventh International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002), May 2002, pp.253-260.

[2] S. Haslinger and S. Dustdar, "Testing of Service Oriented Architectures - A Practical approach," Net.ObjectDays, Lecture Notes in Computer Science, 3263, Springer, 2004, pp. 97-109.

[3] N. Looker and J. Xu, "Assessing the Dependability of SOAP RPC-Based Web Services by Fault Injection," Presentation at Leeds University, 31th July 2003, February 2004, pp.163-170.

[4] A. Mani and A. Nagarajan, "Understanding quality of services for Web services," IBM developerWorks, January 2002.

[5] J. Offutt and W. Xu, "Generating Test Cases for Web Services Using Data Perturbation," ACM SIGSOFT Software Engineering Notes, Vol. 29, No. 5, 2004, pp. 1-10.

[6] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview," IEEE Communications Mgazine, July 1998, pp. 26-37.

[7] I. Singh et al, Designing Web Services with the J2EE 1.4 Platform：JAX-RPC, SOAP, and XML Technology, Addison Wesley, June 2004.

[8] W. T. Tsai, Ray Paul, Lian Yu, Akihiro Saimi, and Zhibin Cao, "Scenario-Based Web Service Testing with Distributed Agents," IEICE Transaction on Information and System, 2003, Vol. E86-D, No. 10, January 2002, pp. 2130-2144.

[9] X. Yu et al, "WSCE: A Flexible Web Service Composition Enviorment," Proceedings of the IEEE International Conference on Web Services (ICWS'04) , July 2004, pp. 428-435.

[10] Business Process Execution Language for Web Services Specification, version 1.1, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[11] CORBA Component Model, version 3.0: Object Management Group, 2002, http://www.omg.org/technology/documents/formal/components.htm.

[12] DCE 1.1: Remote Procedure Call, Issue 1.1; The Open Group, 1997, http://www.opengroup.org/onlinepubs/9692999399/toc.htm.

[13] Web Service Description Language, version 1.1, http://www.w3.org/TR/wsdl.

[14] Apache Extensible Interaction System - Axis, version 1.2.1, http://ws.apache.org/axis/.

[15] Simple Object Access Protocol, version 1.2, http://www.w3.org/TR/soap.

附錄一 範例中所使用的測試案例

```
<?xml version="1.0" ?>
<root>
 <partners>
  <partner>
   <name>LoginService</name>
   <url>http://localhost:8080/Services/LoginService</url>
  </partner>

  <partner>
   <name>RepostoryService</name>
   <url>http://localhost:8080/Services/RepostoryService</url>
  </partner>

  <partner>
   <name>PaymentService</name>
   <url>http://PayMoney.com/services/PaymentService</url>
  </partner>
 </partners>

 <testcase id="1.1">
  <precondition partnerLink="RepostoryService" name="checkStock">
   <response type="Boolean">true</response>
  </ precondition >
  < precondition partnerLink="PaymentService" name="verifyCreditNumber">
   <response type="Boolean">false</response>
  </ precondition >

  <target>
   <name>checkout</name>
   <input>100</input>
  </target>

  <assert>
   <value type="Boolean">true</value>
  </assert>
 </testcase>
</root>
```

宣告部份

前置條件

待測函數與輸入參數

預期結果