中文摘要

Web Service 是可程式化的實體，提供了特定的功能項目，並可供多個可能不同且使用常見網際網路標準(如 XML 及 HTTP) 的系統存取。Web Service 主要依賴 XML 和其他網際網路標準的廣泛接受度來解決先前無法解決的許多問題，進而建立支援應用程式互通性的基礎結構。

BPEL(Business Process Execution Language)是流程執行語言的標準。BPEL 扮演著服務之間合作的指揮者，描述了流程控制如分支、迴圈、平行處理、訊息處理及關連性、例外處理等。BPEL 是一個用 XML 來描述系統流程的方法，把不同的 web services 連結在一起而產生新的解決方案。這樣的組合方式與從前用程式把服務串在一起的方式相比較，顯的更有彈性且更容易管理。使用者可以透過不同的組合方式快速改變或產生新的解決方案。BPEL 對於程序實體容錯(process instance fault tolerance)仍有限且對於 Security 悉借助 WS-Security，然而 WS-Security 只針對 Web Service 而設計無法提供全方位的功能。

為了解決上述問題而提出一套 framework 達到程序實體容錯，程序身份確認性(process authentication)，程序資料隱密性(process confidentiality)，程序資料完整性(process message integrity)，程序內容不可否認性(process non-repudiation) 功能。

關鍵字：

BPEL、Web Service、程序實體安全、容錯、WS-Security

Abstract

Web Service is a programmable entity provided the specific function and will also be available use of different common Internet standards (such as XML and HTTP) to access the system. Web Service mainly relies on XML and other broad acceptance Internet standards to solve many problems that previously can not be solved, and then establish the infrastructure to support interoperability of applications.

BPEL(Business Process Execution Language) is a execution language standard and play a orchestrator between services, and describes the process control, such as branch, loop, parallel processing, information processing and related, the exception handling. BPEL is a business process approach using XML to describe, and generate new solutions by linking different web services together. This combination of these ways compare to the methods programming the services together, the former is significantly more flexible and easier to manage. Users can make use of different combinations to rapid change or create new solutions.

BPEL for the process instance fault tolerance is still limited and Security handling is base on WS-Security, however, WS-Security is only designed for the Web Service can not provide a full range of features. To conquer these problems and we propose a framework to achieve process instance fault tolerance, process authentication, process confidentiality, process message integrity, process non-repudiation function

Keyword: BPEL, Web Service, Process instance security, Fault tolerance, WS-Security.

附表目錄

附圖目錄

A Framework To Support Process Instance Security In SOA

1. Introduction

SOA 是一種架構模型，由網站服務技術等標準化元件組成，目的是為企業、學校或提供網路服務單位建構一個具彈性、可重複使用的整合性介面，促進內外部如內部應用程式、用戶、與部門(系所)等相關單位完美的溝通，盡速達到網路服務提升的目標。Information Technology (IT)產業的架構演進，由 1980 年代的主機(mainframe)架構，到 1990 年代的主從式(client server)架構[1]，到 1999 年時是 network centric 架構，而到 2004 年時已複雜到所謂的服務導向架構 (SOA, Service Oriented Architecture)[2]。

1.1 SOA

服務導向架構[2] 是一種新興的系統架構模型，其主要概念是針對系統需求組合而成的一組軟體元件。組合的元素通常包括：軟體元件、服務及流程三個部份，當企業面對外部要求時，流程定義外部要求的處理步驟；服務包括特定步驟的所有程式元件，而元件則負責執行工作的程式。服務導向架構[2] 已成為現今軟體發展的重要技術，透過 服務導向架構[2] 讓異質系統整合變得容易，程式

再用度也提高。不必自行開發或擁有所有程式元件，發展者可以視需要組合網路上最好的服務。不受限於特定廠商的產品功能或是平台，達到真正的開放性(Openness)[23]。服務導向架構[2]是一種架構風格(architectural style)，不是一種產品，也不是一組定型的解決方案；它是讓我們在分析設計系統時，能夠循序漸進遵循的精神與原則(principles)。如果我們能依據這些原則導入，不但阻礙較小，也比較容易設計出彈性而易重用(reusable)，更符合商業價值(business value)的系統。服務導向架構[2]把產品與應用程式造成的邊界完全抹去，讓我們用服務的觀點來看待整個企業裡資訊系統的組合。

軟體系統不管大小，兩個維度可以用來做初步判斷好壞的準則：凝聚力(cohesion) 和連結力 (coupling)[3]。凝聚力表示在一個邏輯單位(如 module 或 package)內，各組成單位(如 class) 彼此間的依靠程度 (dependency)；連結力表示在不同邏輯單位間組成單位的依靠程度。好的系統通常高凝聚力及低連結力，而最糟的系統低凝聚力與高連結力。我們做大系統的準則，就是想辦法把它切成一群子系統，使得子系統之間得連結力變得很小，但是在子系統內的凝聚力變大。如果每個子系統的凝聚力大，表示整個系統抽象切割合理；如果子系統間的連結力鬆(loose)，表示設計上封存(encapsulation)的工夫做得好。具體而言，我們在實踐服務導向架構[2]中，如果能利用以上兩種量度(metrics)來看看我們做得好不好，這樣會有個好的起點。首先要把相關子系統內的凝聚力增高，我們可以努力把子系統的邊界(boundaries)明顯切割好，也要努力讓子系統能夠

自主性(autonomous)，不會因為內部製作細節的改變，而影響了服務本身的正確性。這跟我們使用介面(interface)而不是類別 (class)來作為對外契約(contract)的原則很類似[24]。

1.2 Web Services

Web Service 在 World Wide Web Consortium(W3C)的定義為"A software system designed to support interoperable machine-to-machine interaction over a network"。Web Service 常常只是 Web application programming interfaces(API)，可以利用的網路，如網際網路，用以執行遠端系統主機所要求的服務，其他技術與 Web Service 幾乎相同的功能有 Object Management Group's (OMG)的 Common Object Request Broker Architecture (CORBA)、微軟的 Distributed Component Object Model (DCOM) 和昇陽的 Java/Remote Method Invocation (RMI)，W3C 的 Web service 的定義包含許多不同的系統，但在普遍是指客戶端和伺服器在使用 HTTP 協議上進行訊息通信，使用 XML(Extensible Markup Language)格式遵循 SOAP(Simple Object Access Protocol)[15]標準已普遍使用在一般傳統企業。

圖 1.1 在 SOA 架構中 Web Services 相關標準

Web Services 是一種軟體元件，它透過 Web 通訊協定及資料格式的開放式標準(例如 HTTP、XML、SOAP 及 WSDL 等)來為其他的應用程式提供服務，如圖 1.1 所示 HTTP(Hypertext Transfer Protocol)[19]是應用層次的通訊協定，SOAP[15]是為了與 Web Services 溝通所制定用來規範的信息交換結構，WSDL(Web Services Description Language)[14]描述了 Web service 的對外之介面，客戶端程序可以連接到 Web service 讀取其 WSDL，以確定可以使用 Web service 那些功能，WSDL 架構圖示意如圖 1.2。這裡面有兩個重點，一是它是一個提供服務的元件。二是它以 Web 的開放標準為基礎。根據以上的認識，我們可以看出 Web Services 的價值。作為提供服務的元件，它可用來建構分散式架構系統，實現分散式架構動態整合、平衡負擔、單元升級等優點。以 Web 的開放標準為基礎，在已經廣被使

用的 Web 網路架構上來運作，採用開放式標準讓 Web Services 具有良好互通性，在不同平台上用不同程式語言建置的系統也可以輕易整合，克服目前分散式系統各自使用不同機制造成整合困難的情形[25]。

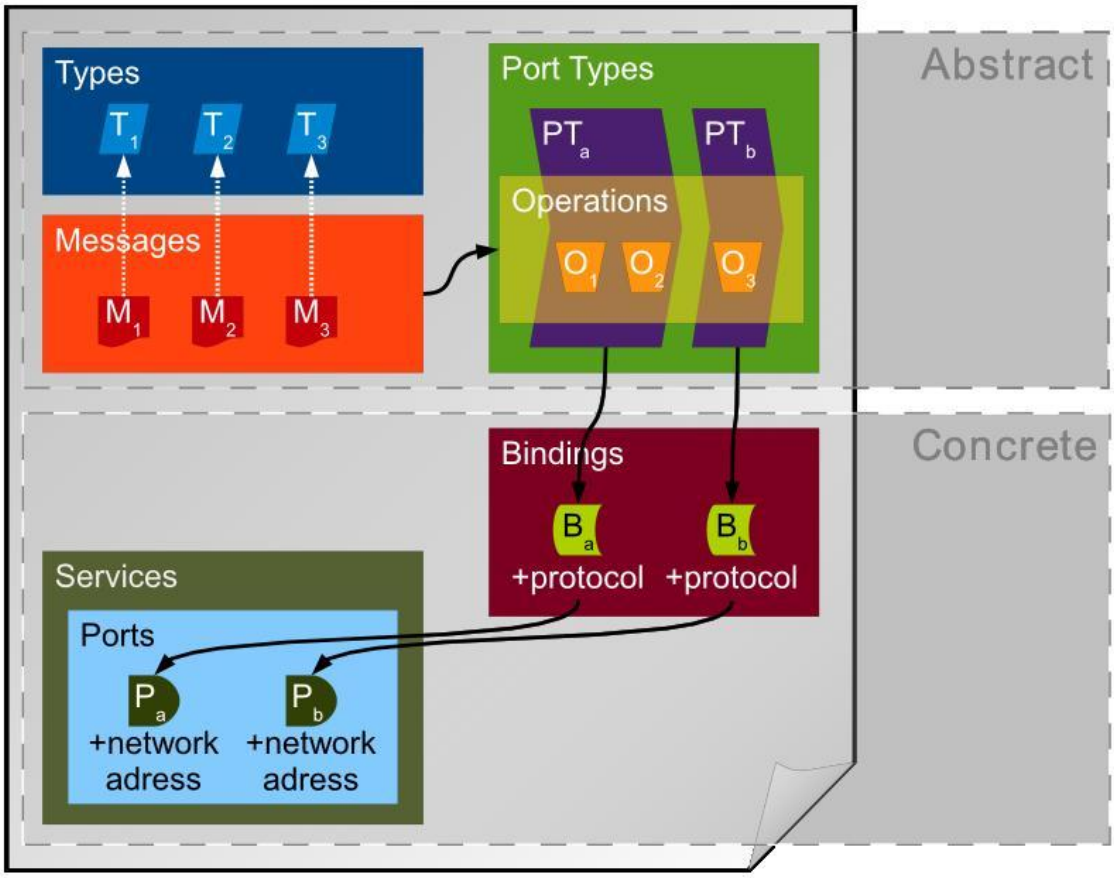


圖 1.2WSDL1.1 架構示意圖

1.3 SOA Web Services

服務導向架構[2]在 1990 年代提出之始，原意是在軟體設計的概念和整合性軟體的架構。 因此，諸如 CORBA[26]、RMI[27]、DCOM[28]或甚至使用 Socket 的技術，都可以建立具有服務導向架構[2]的系統，但是在強調鬆散耦合和技術中立的新定義下，服務導向架構[2]已逐漸定位為是建立於標準網路運算技術之上的架構與設計概念了，SOA Web Services 則是運用 Web Services 的技術來實踐 服

務導向架構[2]。Web Services 的技術成熟後不久很多人發現導入 Web Services 的技術於服務導向架構[2]是實踐跨平台分散式架構的很好選擇，可以享有 Web Services 分散式架構動態整合、平衡負擔、單元升級等優點、良好互通性，在不同平台上用不同程式語言建置的系統也可以輕易整合，克服目前分散式系統各自使用不同機制造成整合困難的情形，透過 服務導向架構[2] 讓異質系統整合變得容易，程式再用度也提高。不必自行開發或擁有所有程式元件，發展者可以視需要組合網路上最好的服務。不受限於特定廠商的產品功能或是平台，達到真正的開放性 (Openness)。

早期 Web services 標準實踐服務導向架構[2]如下，圖 1.3 為其架構示意圖：

- WSDL[14] 描述服務的公共介面。這是一個基於 XML 的關於如何與 Web 服務通訊和使用的服務描述；也就是描述與目錄中列出的 Web 服務進行交互時需要綁定的協議和信息格式
- SOAP [15]是一種標準化的通訊規範，主要用於 Web 服務（web service）中。SOAP 的出現是為了簡化網頁伺服器（Web Server）在從 XML 數據庫中提取資料時，無需花時間去格式化頁面，並能夠讓不同應用程式之間透過 HTTP 通訊協定，以 XML 格式互相交換彼此的資料，使其與程式語言、平台和硬體無關
- UDDI[5]是統一描述、發現和集成(Universal Description, Discovery,

and Integration）的縮寫。它是一個基於 XML 的跨平台的描述規範，可以使世界範圍內的企業在網際網路上發布自己所提供的服務

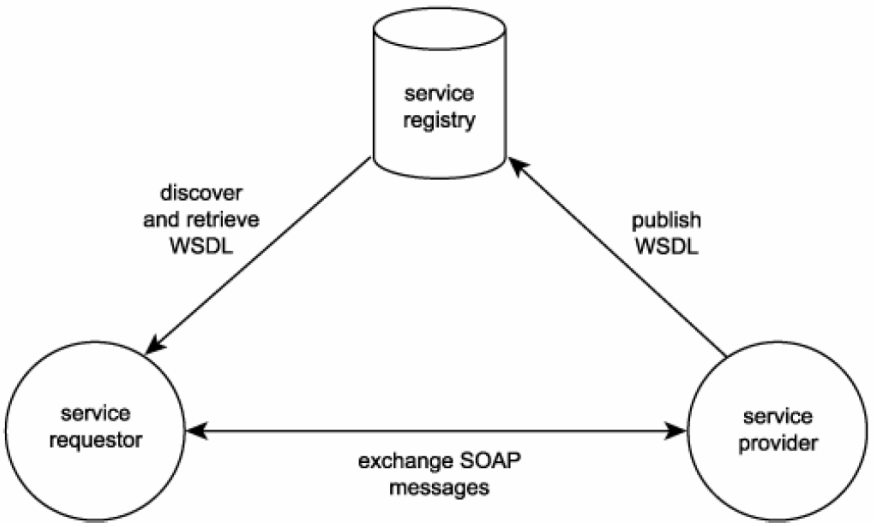


圖 1.3 架構示意圖

1.4 SOA Security

SOA Security 已經是不可忽視的一個重要議題，有鑒於服務導向架構[2]概念漸漸受到重視 SOA Security 的議題克服更是刻不容緩，Security 須要達到身份確認性 (Authentication)、資料完整性 (Integrity)、資料隱密性(Confidentiality)及內容不可否認性(Non-repudiation)而 SOA Security 大致可分類為兩種，一種是訊息傳遞時的安全 (Communication Security)也是大多數學者專家比較注意花費較多心思的部份，目前 Web Service 技術已日趨成熟，Web Service 相關延伸規格與開發套件也逐步發展成熟，其中有關 Web Service Security 標準已經相關成熟目前最新的標準為 SOAP Message Security 1.1 (WS-Security 2004)[18]，一般來說 Web Services 安全技術都以 XML 加密(XML

Encryption)、XML 簽名(XML Signature) 基礎，並使用 XML 檔案格式將訊息加密資料和嵌入數位簽章的 W3C 標準，因為在 SOA Web Services 是運用 Web Services 的技術，所以訊息傳遞時的安全也使用 SOAP Message Security 1.1 (WS-Security 2004)[18]。另一種是比較少去探討的程序實體的安全 (Process Instance Security)，程序實體(Process Instance)[4]是執行程序模型(Process Model)在運行時的實體，同一程序模型中不同的程序實體可能執行不同的任務集合而有不同的執行路徑，不同程序實體可以同時執行，程序實體的安全是指程序實體存在於系統時的實體之安全。

程序實體的安全包括執行時其執行序的安全和程序實體序列化到非揮發性儲存裝置的安全，並且都仰賴作業系統提供安全保護，目前並無標準規範，但是其重要性確是不容忽視。

1.5 Process Instance Security

程序實體的安全最須要注意的是程序實體序列化到非揮發性儲存裝置的安全，程序實體序列化到非揮發性儲存裝置的安全雖然也受檔案系統保護，但是相對來說其安全性其實相當薄弱，再說儲存到儲存裝置的程序實體會容易複製、移動、刪除的特性，故其安全性須要特別強化，所以這裡所說程序實體的安全主要指程序實體序列化到非揮發性儲存裝置的安全。程序實體的安全動機是程序實體必須確保存在於系統非揮發性儲存裝置時能符合身份確認性、資料完整性、資料

隱密性及內容不可否認性需求。目前這些需求必須訴諸作業系統或資料庫系統機制措施，因為技術的是不切實際的，不夠全面，或完全缺乏。常見的威脅模式為能夠取得資料的系統管理者或使用者無法完全信任，理由是:

1. 使用或傳遞資料時不注意或粗心
2. 就算系統管理者或使用者完全可信任但其使用的軟體可能有問題(例如:遭植入木馬程式)
3. 系統管理者或使用者可能真的有問題

須要確保程序實體中的資訊能被確實保護以符合身份確認性、資料完整性、資料隱密性及內容不可否認性需求，而保護機制不局限於特定平台環境及軟體。服務導向架構[2]程序實體除了上述四個需求須要考量外，還須要考量程序實體中不同狀態、不同對象、不同資料有不同身份確認、不同資料完整性、不同資料隱密性及不同內容不可否認性考量，這些考量往往造成服務導向架構[2]程序實體的安全實踐起來十分複雜而由於服務導向架構[2]針對程序實體的安全並無全面實際的標準，我們針對服務導向架構[2]程序實體的安全提出一套架構(Framework)使得服務導向架構[2]程序實體的安全有系統性的達到程序實體中不同狀態、不同對象、不同資料有不同身份確認、不同資料完整性、不同資料隱密性及不同內容不可否認性考量的功能。

此外此架構提供保護機制不局限於特定平台環境及軟體並且也提供錯誤回復(Fault Recovery) [13]機制甚至失敗轉移(Failover)的潛力。

程序實體的安全還有兩個非常特別重要的特性，那就是可以非常容易傳遞還有就是四個安全需求不會因為程序實體的傳遞而有安全上的瑕疵。例如一個跨國企業常常會利用全球每個國家不同成本及能力或者利用各地時差達到全天 24 小時工作或是風險分散等等考量的須要而須要將產品的生產分部在全球各地，比如說產品設計在台灣分公司，審核及預算在美國總公司，產品製造分別在中國大陸及越南，而產品設計、審核及預算、產品製造又有一些細項流程在當地執行甚至於還要跨國執行如此一個流程就須要全球往返，若是公司日益成長業務蒸蒸日上到網路頻寬不足或伺服器效能不敷使用，系統就會造成公司的負擔，但是若是採用程序實體的安全的機制就可以解決上述兩個問題，首先由於程序實體非常容易傳遞所以一些細項流程在當地執行的流程就可以在當地執行須要占用跨國網路頻寬，而當伺服器效能即將不敷使用時可以將程序實體傳遞到比較空間的伺服器也解決了伺服器效能不敷使用的困境，如此就可以隨公司成長而擴充伺服器使得效能也跟者成長。

由於採用公鑰基礎建設(PKI,public key infrastructure)[11]的基礎架構而程序實體的安全是文件為基礎，所以基於 XML 加密(XML Encryption)、XML 簽名(XML Signature)的使用不須要特定的工作環境，特殊的加解密及簽名驗證軟體，也不須要特定的資料庫系統，甚至不須要資料庫系統就可以達到身份確認性、資料完整性、資料隱密性及內容不可否認性需求，其實程序實體的主要優點就如上所說明的可以歸納如下:

1. 跨平台 (Cross Platform)

2. 容錯(Fault Tolerance)

3. 遷移(Migration)

4. 負載平衡(Load Balancing)

1.6 BPEL

服務導向架構[2]是一種新興的系統架構模型，其主要概念是針對企業需求組合而成的一組軟體元件。組合的元素通常包括：軟體元件、服務及流程三個部份，其中流程處理部份原本並沒有標準化，或是未被廣泛採用，但是 BPEL (Business Process Execution Language) [9] 標準制定打破了各家廠商各自制定自己流程語言的藩籬。2002 七月 IBM, Microsoft, BEA 三家軟體大廠制定 BPEL4WS 1.0，其中 BPEL4WS 1.0 採納 Microsoft 於 2000 十一月推出的 XLANG 及 IBM 於 2001 三月推出的 WSFL，在 2003 三月 BPEL4WS 1.0 提交 OASIS(Organization for the Advancement of Structured Information Standards)後 2003 五月 OASIS 推出 BPEL4WS 1.1 標準，再歷經四年的修訂強化後 2007 四月 OASIS 推出 WS-BPEL 2.0 標準。

BPEL 標準制定時要達到的目標如下:

1. 定義商業程序(business processes)與外部實體的互動，此互動利用 WSDL 1.1 定義的 Web service 達成，而此定義商業程序以使用 WSDL 1.1

定義的 Web service 方式展現。互動是仰賴 portType 定義以抽象概念(與此 Web service 執行環境與平台無關)表示，而不是以 port 定義以可執行概念(與此 Web service 執行環境與平台有關)表示。

2. 定義商業程序(business processes) 的語言使用符合 XML 的語言。不使用圖形符號描述程序（processes)，針對程序也不提供任何特別的設計方法論（design methodology)。

3. 定義一套 Web service，此套 Web service 具備集中式工作流程的概念（orchestration concepts）而此概念是以外部（抽象化（abstract））和內部（可執行（executable））觀點來描述。這樣的商業程序定義一個單一自動化實體的行為模式，此自動化實體基本上與其他類似的自動化實體互動。

4. 提供有等級制度(hierarchical)的和等同圖形符號描述(graph-like)控制制度，並使其盡可能完全完美混用。這應該減少程序建模破碎的空間(fragmentation of the process modeling space)。

5. 提供操作簡單的資料管理功能，此功能需要用來定義的程序資料和控制流。

6. 提供程序實體的識別機制，此機制允許的定義在應用資訊層面（application message level）之實體的識別子（instance identifiers)，此識別子應由夥伴（partners）界定且允許改變。

7. 提供隱含建立和終止程序實體為基本的生命週期機制。進一步生命週期的操作，如“暫停（suspend）”和“恢復（resume）”可能會增加在未來版本以加強生命週期管理。

8. 定義一個長時間運行的交易模式（long-running transaction model），此模式建立在已被證實為成熟技術如補償技術（compensation actions）及部分長時間運行的商業程序支持故障恢復的程範圍化技術（scoping）。

9. 使用 Web services 的模型來達到程序的分解（decomposition）和組裝（assembly）。

10. 盡可能以組合和模組化的方式（composable and modular manner）並且盡可能建立在 Web services 標準（核准和擬議（approved and proposed））上。

此架構使用 BPEL 來實踐服務導向架構[2]流程處理部份，BPEL 的架構示意圖如圖 1.4。

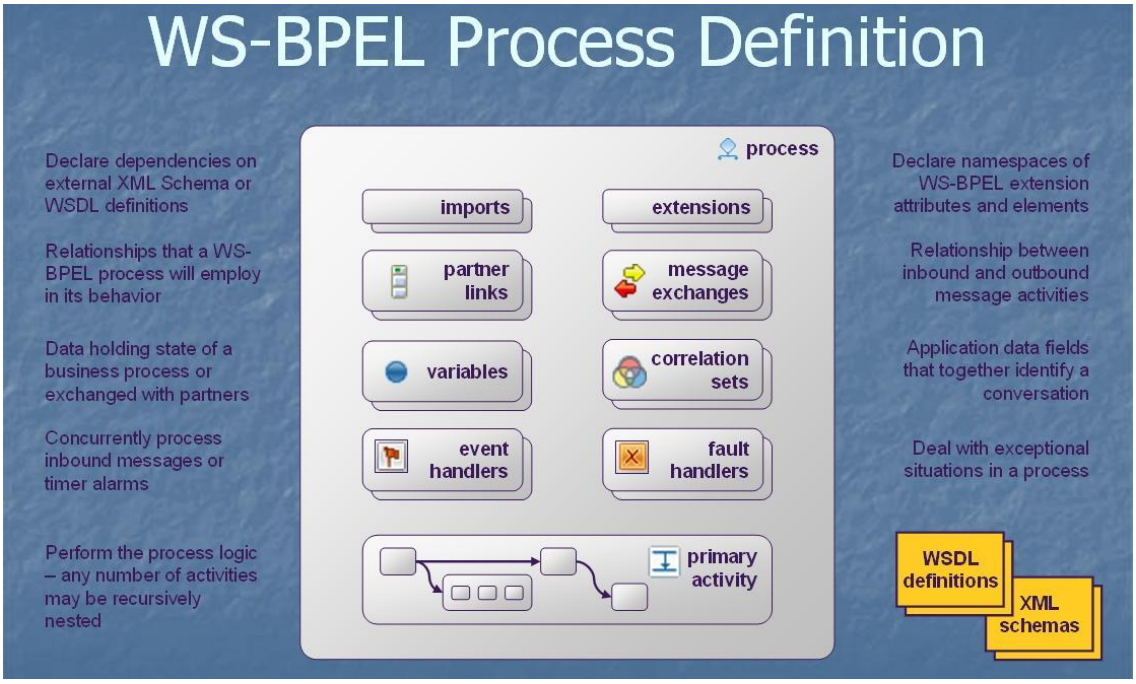


圖 1.4 架構示意圖

1.7 Dynamic behavior in workflow security issue

由圖 1.1 在服務導向架構[2]中 Web Services 相關標準中可以了解工作流程標準(例如 BPEL)在服務導向架構[2]中是一種屬於高階層標準架構在 WS-Security、WSDL、WS-Policy、SOAP、WS-Addressing 之上，所以較低階層標準對於屬於高階層標準來說是無法被察覺的，就好像(Open Systems Interconnection)OSI Reference Model[22]中的七層架構中網路層(Network layer)[22]是無法察覺資料聯結層(Data-link layer)[22]一樣，但是安全考量對工作流程而言確是常常需要知悉較低階層 WS-Security 執行時的狀況來判斷決定工作流程的執行路徑，目前工作流程標準缺乏兩種功能： Security 的狀況判斷及執行狀況判斷，這裡提出一個動機範例來說明 Security 的狀況判斷及執行狀況判斷。

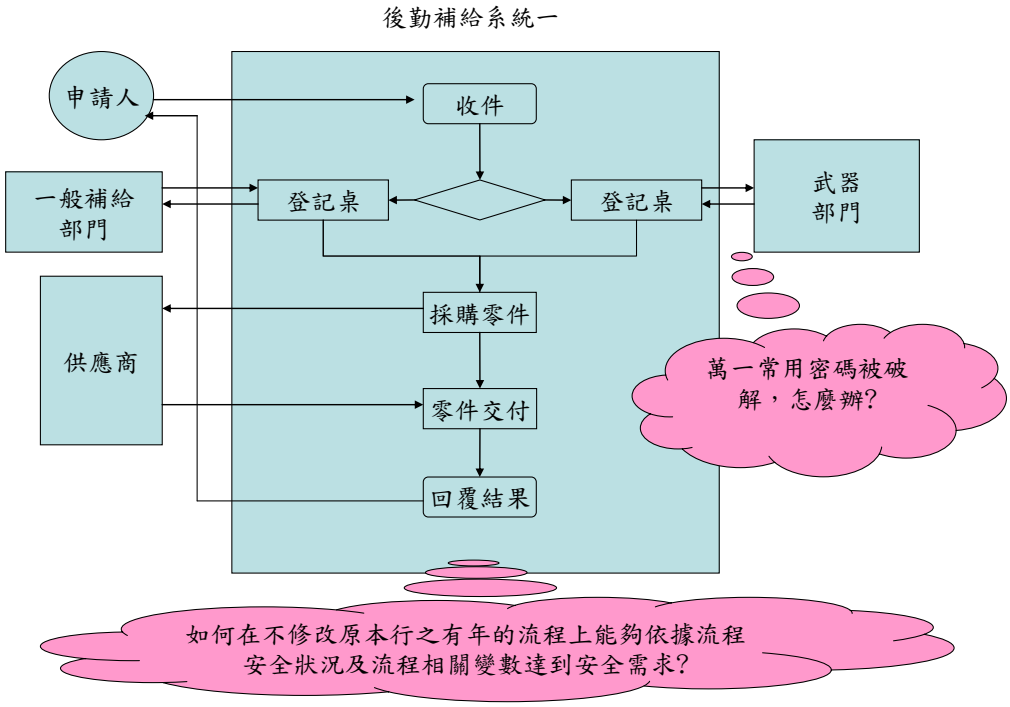


圖 1.5 後勤補給系統(一)

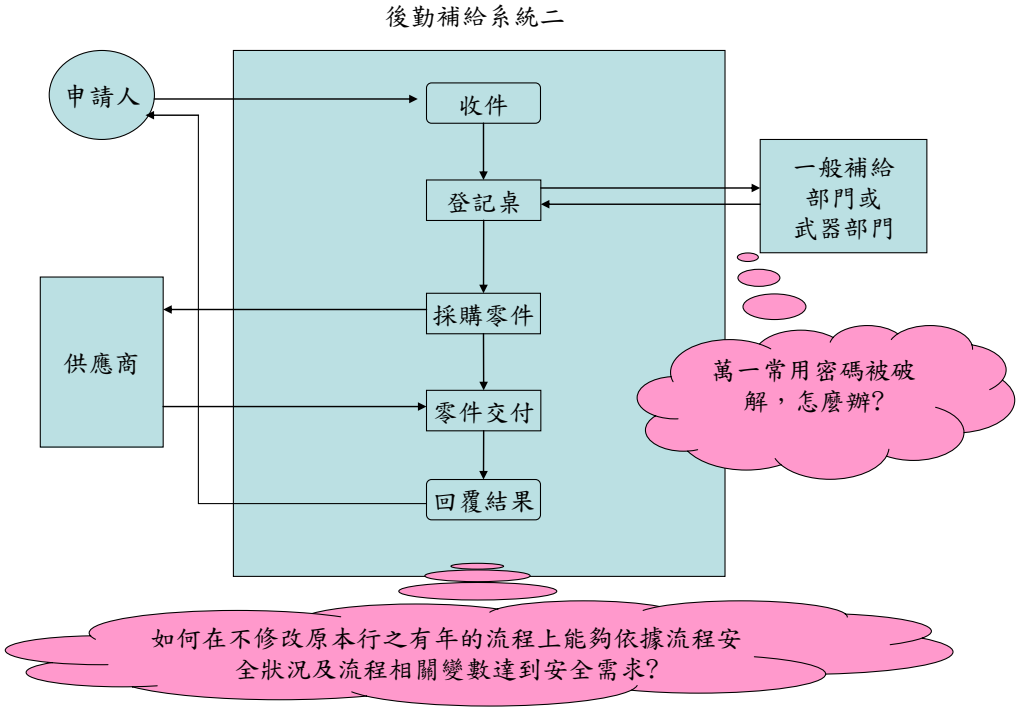


圖 1.6 後勤補給系統(二)

後勤補給系統為例如圖 1.5 後勤補給系統(一)申請人要申請補給用品，首先

送出申請單給系統後系統收件，再由收件單位分給登記桌簽收，登記桌簽收後給承辦單位處理，承辦單位處理後送交零件採購，零件採購後等待零件交付，零件交付後回覆結果，其中圖 1.6 後勤補給系統(二)和 1.5 後勤補給系統(一)區別為圖 1.6 範例須依據工作流程的變數區分一般補給部門或武器部門而圖 1.5 範例可依據流程區分一般補給部門或武器部門，如何在不修改原本行之有年的流程上能夠依據流程安全狀況及流程相關變數達到動態安全需求?

暴力法(brute force)解決方案，以圖 1.6 後勤補給系統(二)為例子針對”登記桌”步驟(Activity)若是不使用有系統的方法去解決透過代理人(Proxy)機制及序列化服務(Serialization Web services)功能的幫助，還是可以使用暴力法解決方案將”登記桌”步驟改寫如圖 1.7 暴力法解決方案，但是使用暴力法解決方案將改變原本工作流程，而且當政策改變(Policy)或單位異動時將再修改，試想經過一次又一次的修改結果將是資訊安全管理單位人員維護系統的一大夢魘，而且經過一次又一次的修改修改者職務又有異動的話，事後責任歸屬又是一個大問題，版本控制要如何確保呢?萬一改錯了又沒有儲存原本的版本又該如何呢? 暴力法解決方案雖然也可以解決問題但是也帶來很多執行及管理維護上很大的後遺症，前面所討論只是一個”登記桌”步驟，若是有數十個或上百個步驟怎麼辦呢。

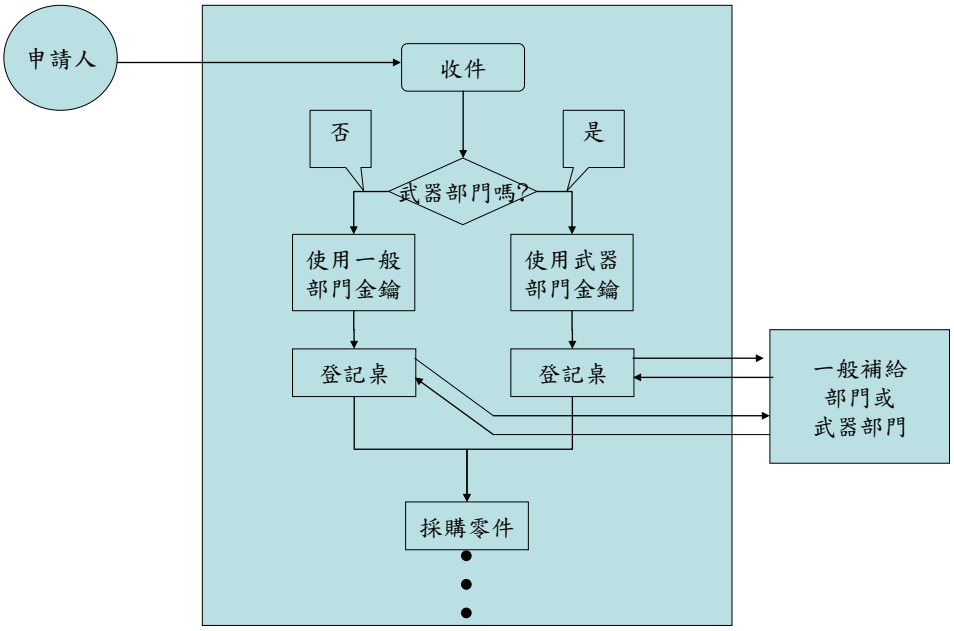


圖 1.7 暴力法解決方案

為了達到依據流程安全狀況及流程相關變數達到動態安全需求，提出一個動態行為處理邏輯(Dynamic behavior handler logic)方法處理流程安全狀況及流程相關變數達到動態安全需求，動態行為處理邏輯採用類似 BPEL If Conditional Behavior 語法如下：

```
<if>
      <condition>bool-expr</condition>
      ifactivity
      <elseif>*
            <condition>bool-expr</condition>
            ifactivity
      </elseif>
      <else>?
      ifactivity
      </else>
</if>
```

圖 1.8 動態行為處理邏輯 If 語法

針對圖 1.8 動態行為處理邏輯 If 語法分別就其中 bool-expr、ifactivity 分別加以描述。bool-expr 採用 BPEL 一樣布林表示法(Boolean expressions)[9]，因為 BPEL 採用數個標準其中包括 XPATH1.0 標準所以其布林表示法類似 XPATH 布林表示法，圖 1.10 後勤補給系統(二) 動態行為處理邏輯中的範例就使用了 XPATH1.0 中的 boolean contains(string,string) 布林函式。但是為了處理流程安全狀況及流程相關變數達到動態安全需求只用標準的布林表示法是無法實現的，必須使用四個擴充布林函數 ActivityIsExecuted (ActivityName)，SequenceIsExecuted (AN1,AN2,AN3,…) ， IsEncryptedValidated (ActivityName)，IsSignedValidated (ActivityName)，其中 ActivityIsExecuted 和 SequenceIsExecuted 在轉換成 BPEL’後會成為呼叫儲存程序實體所提供的 Web services 並根據所傳回的回覆得知所傳入為參數的步驟是否已執行過，而 IsEncryptedValidated 和 IsSignedValidated 在轉換成 BPEL’後會成為呼叫負責訊息傳遞時的安全的代理人所提供的 Web services 並根據所傳回的回覆得知所傳入為參數的步驟是否已通過加密驗證和簽章驗證。

ActivityIsExecuted(ActivityName)：步驟其識別 ID 為 ActivityName 在此程序實體是否已經執行過，已經執行過傳回真，否則傳回假。

SequenceIsExecuted(AN1,AN2,AN3,...)：步驟其識別 ID 依序為 AN1,AN2,AN3,…在此程序實體是否已經依序執行過，已經依序執行過傳回真，否則傳回假。

IsEncryptedValidated(ActivityName): 步驟其識別 ID 為 ActivityName 是否可解密，可解密則傳回真，否則傳回假。

IsSignedValidated (ActivityName): 步驟其識別 ID 為 ActivityName 是否簽章驗證有誤，簽章驗證有誤傳回假，否則傳回真。

ifactivity： 包括<if>，<sequence>，<throw>，兩個擴充函數 Encrypt (Key, S1, S2, …)，Sign (Key, S1, S2, …)。

Encrypt(Key,S1,S2,...)：使用 Key 加密 S1, S2, …所指示之資料。

Sign(Key,S1,S2,...)：使用 Key 簽章 S1, S2, …所指示之資料。

其中 Encrypt 及 Sign 中的 Key 由 3.1.2 章節所述 SDL 的 Key definition section 中定義的 X.509 certificates 格式的金鑰而 S1 及 S2 則是 BPEL 變數中要加密的元素(Element)。

以圖 1.5 後勤補給系統(一)及圖 1.6 後勤補給系統(二)為例當申請補給用品為武器類別則使用比較不易被破解金鑰加密的動態行為處理邏輯分別如圖 1.9 及 1.10

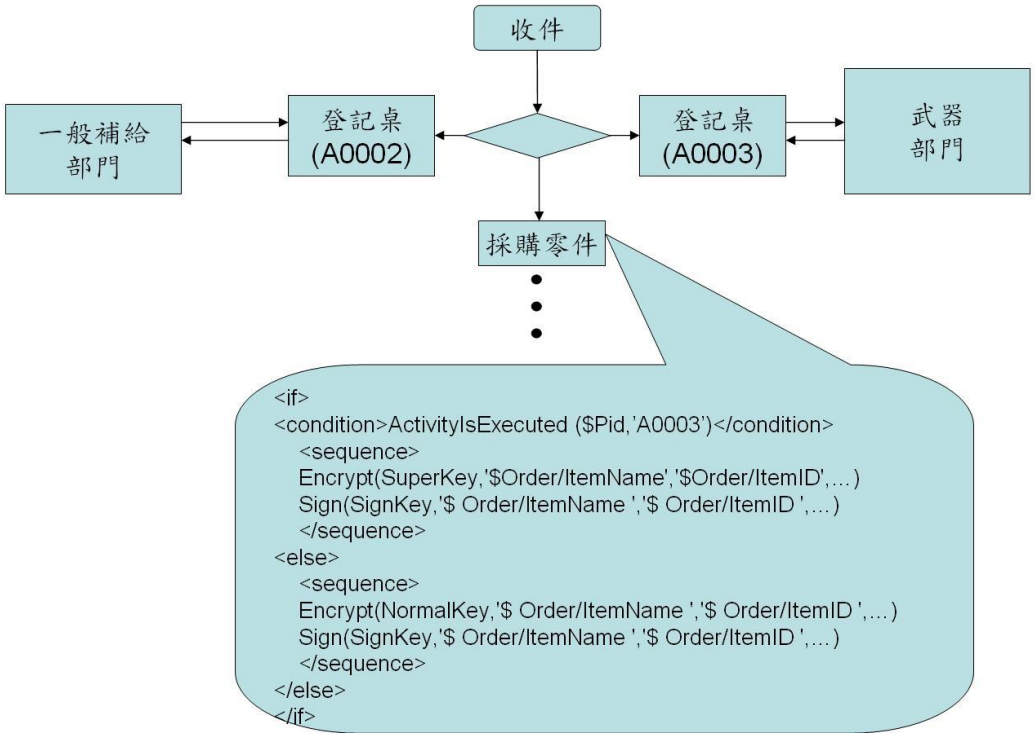


圖 1.9 後勤補給系統(一) 動態行為處理邏輯

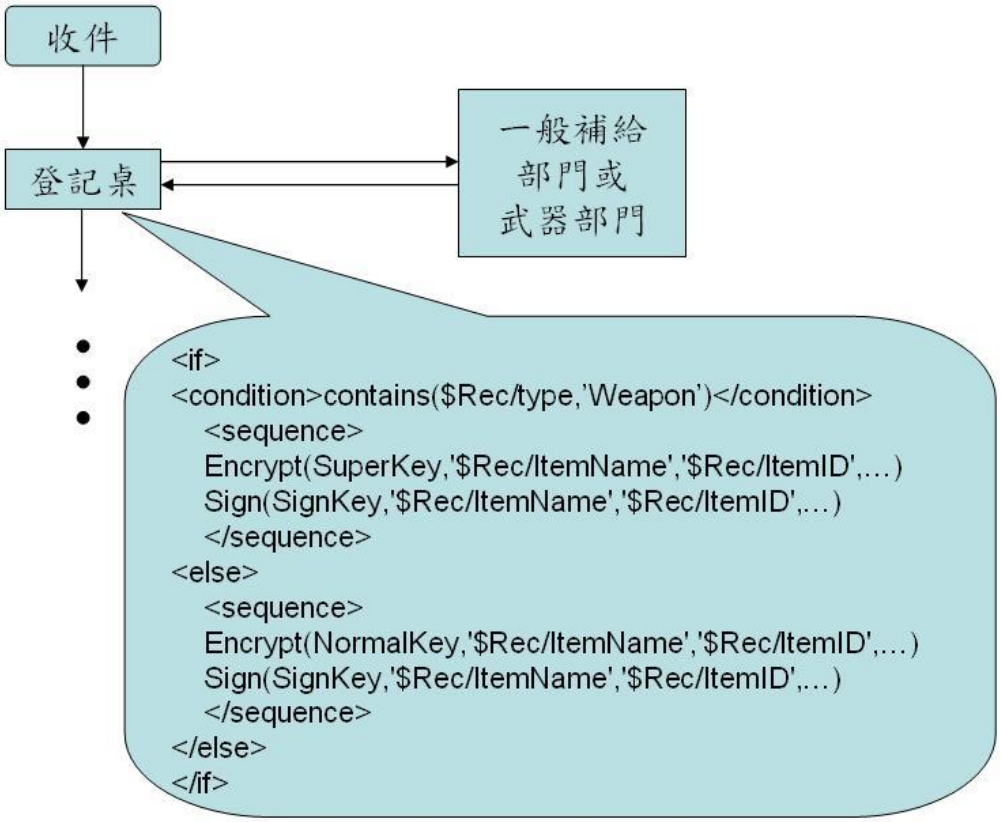


圖 1.10 後勤補給系統(二) 動態行為處理邏輯

圖 1.7 暴力法解決方案會有很多執行及管理維護上很大的後遺症而採用圖1.9及圖1.10動態行為處理邏輯方法處理流程安全狀況及流程相關變數達到動態安全需求則將原本工作流程與安全機制分離管理確又可以賦予原本工作流程動態行為安全機制，當政策改變時原本工作流程不須要改變只是將新的政策的每個動態行為處理邏輯寫下至 SDL(Serialization description language) 而因為SDL 也是 XML 格式的文件可以很容易儲存而不會有版本控制的問題，另外 SDL 有數位簽章可以知道該 SDL 由誰所撰寫不會有權責歸屬困難的問題。

1.8 Our framework

服務導向架構[2]其主要概念是針對企業需求組合而成的一組軟體元件。組合的元素通常包括：軟體元件、服務及流程三個部份，此架構制定出 SDL 將軟體元件、服務及流程三個部份整合並定義 Security 及 Serialization 功能及約束。

此架構軟體元件使用採用 ActiveBPEL(TM) engine4.0，Apache Axis2 1.3SOAP engine，Java SE Development Kit (JDK) is included in the Java EE 5 SDK，為實作軟體。

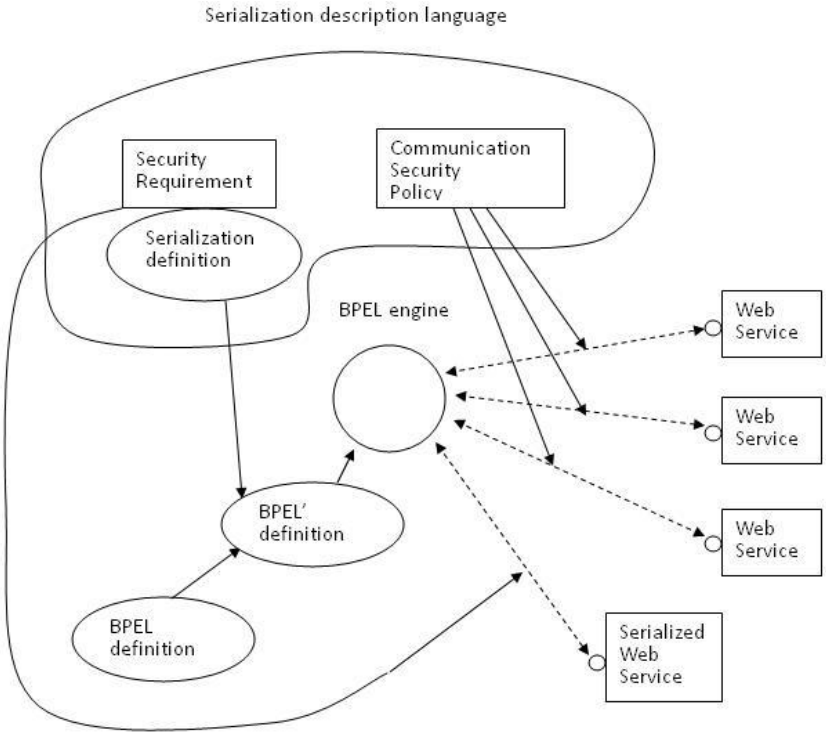


圖 1.11 Serialization description language 架構圖

服務部份包括三種：Apache Axis2 1.3SOAP engine 實作的 Web Service，Apache Axis2 1.3SOAP engine 實作的 Serialization Web Service 及 ActiveBPEL(TM) engine4.0 實作的 BPEL Web Service。

流程部份使用 BPEL 語言表達服務導向架構[2]流程控制及商業邏輯，透過 SDL Tool 所提供的演算機制產生具備程序實體序列化功能 BPEL' 語言進而實踐預期功能。SDL 功能方塊圖如圖 1.11。

就如同 1.4 SOA Security 節所說的 SOA Security 又可分類為兩種，一是訊息傳遞時的安全 (Communication Security)，二是程序實體的安全 (Process Instance Security)。提出的架構中有關訊息傳遞時的安全 (Communication Security)的達成是借由網路服務安全語言(WSSL,Web services security

language)[20]所提出的操作模型，SDL Workflow definition 定義出工作流程與網路服務安全語言對應關係，有關程序實體的安全的達成是借由 SDL 中的 Key definition，Serialization definition。

SDL 包括 Header definition，Key definition，Workflow definition，Serialization definition，Digital signature 等段落。以下分別概述其用途:

1. Header definition 定義 XML 格式定義，SDL 之根元素及 sdl 名稱空間(namespace)
2. Key definition 定義加密、解密及簽章所參考到的金鑰，在此架構主要以 X.509 certificates 為預設金鑰格式
3. Workflow definition 定義所針對 BPEL[9]中 partnerLink 中 myRole 和 partnerRole，由 myRole 的 partnerLinkType 進而對應到 web service 的 WSDL 中<portType>,<targetmyRole>中的<WSSL>對應針對此 WSDL 的網路服務安全語言文件進而取得確保訊息傳遞時的安全(Communication Security)相關安全機制,因為 partnerLink 是 BPEL[9]中呼叫其他 Web service 及提供 Web service 之 WSDL 定義點，透過網路服務安全語言操作模型除了滿足基本的安全需求，包括驗證，機密性，完整性及不可否認性外，它也提供了元素層次加密(element-wise encryption)及以時序為基礎的元素次層數位簽章(temporal-based element-wise digital signature)的安全機制

4. Serialization definition 定義以每一個 Basic Activities[9]為對象定義序列化內容及動態行為處理邏輯

5. Digital signature 定義簽章，以免 SDL 被串改，參考 XML-Signature Syntax and Processing W3C Recommendation 12 February 2002 標準

2.Related Works

近年來隨著異質系統整合及協同運算須要日漸殷切,原因是當企業組織成長到越來越大，跨國分支機構越來越多，組織內系統就越來越龐大且複雜，而且浪費及重複的系統也越來越明顯，經過前人前仆後繼的嘗試想要把所有系統統一最後往往未蒙其利，先受到因為要整合系統而使原本已經建立的系統修改而先蒙其害，加上電腦科技日行千里變化往往系統功能穩定後新的版本又出現或不同部門系統間異質系統也須要統合，經過無數次的嘗試不論是產業界或學術界都認為服務導向架構[2]是徹底解決異質系統整合的最可能的架構，加上軟體網際網路技術成熟且為異質系統最大的標準化問題也因為大家的努力制定了越來越多標準，讓異質系統整合及企業組織系統再造及流程精簡並且不會讓原本已經存在的系統重頭再改變原本已經成熟的舊有系統。

Workflow 概念是建立於建制一套系統若是從無到有往往是曠日廢時且歷時過久，有識之士透過分析發現任何一套系統其實就是 Workflow 加上其他特定功能單元的組合，而很多特定功能單元是原本已經存在的系統而 Workflow 則是可以獨立於系統之外進而標準化因為任何 Workflow 都不脫一般程式語言的流程控制範疇只須要制定一套跨平台、跨廠商的標準。

BPEL[9]標準制定已經解決 Workflow 的問題，WSDL、SOAP、UDDI 也解決功能單元的組合的問題，所以功能面的問題基本上已經相當程度解決了。

但是非功能面的問題解決了嗎?答案是目前正在努力以赴但是還須要大家的

努力，目前可見到的問題是安全相關議題及容錯相關議題。首先是有關安全相關議題，功能單元安全相關議也可以說就是 Web services 的安全相關議已經有相當成果，在 OASIS(Organization for the Advancement of Structured Information Standards)組織制定 SOAP Message Security 1.0 (WS-Security 2004)後，各界已經有了 Web services 的安全的倚靠，也就是說功能單元安全已經標準化了，但是 Workflow 並沒有被考量到其中，Anis Charfi[6]利用面向導向技術(AOP，Aspect-oriented programming)[12]將安全機制加以 Workflow 中，雖然可以達到安全的須求但是面向導向技術基本上是一種程式設計技術，修改 Workflow Engine 原始碼也會造成既有的標準被修改而無法達到原本跨平台的要求。

Workflow 中容錯相關議題也是須要更進一步探討的議題，針對 Workflow 中容錯方式我們可以分類成：靜態方式、變數動態、流程動態，針對 Workflow 中容錯等級我們可以分類成：單一 Activity 級，多 Activity 級，Process 級。

An Liu[7]中定義 Ignore、Skip、Retry、Alternate 幾種模式，再定義一些規則和條件及採取的模式之方式達到 Workflow 錯誤發生時採取所設定模式來處理錯誤，但是仍然有相當多的需求是無法用 Ignore、Skip、Retry、Alternate 模式解決的狀況，容錯方式為變數動態而容錯等級為多 Activity 級。

Glen Dobson[8]則是透過 BPEL[9]流程控制達到容錯提供兩種模式第一種稱為重試(Retry)利用 BPEL 定義的 PICK 步驟[9]特性當呼叫 Web service 時呼叫兩

個相同功能的 Web service 如此一來只要有其中一個成功就可以完成呼叫，另外一種重試(Retry)利用 BPEL[9]定義的範圍(Scope)步驟、 補償(compensate)步驟、補償處理者(Compensation Handler)、錯誤擷取(CatchAll)等特性呼叫 Web service 發生錯誤時，擷取錯誤再執行補償步驟而補償步驟會執行補償處理者，補償處理者會呼叫相同功能的 Web service 一樣達到單一 Web service 容錯功能，第二種稱為平行執行(Parallel Execution)利用 BPEL 流程(Flow)步驟，利用流程步驟平行執行的特性，呼叫三個相同功能的 Web service 再根據回覆得結果投票，選擇多數為結果一樣也可以達到單一 Web service 容錯功能但是針對 Workflow 本身確沒有容錯，容錯方式為靜態方式而容錯等級為單一步驟級。

Jim Lau[10]提出 FTWS-Orch model 利用 Engine Monitor、Primary Engine、Backup Engine 三個 BPEL Engine，Engine Monitor、Primary Engine、Backup Engine 利用 BPEL[9]流程控制達到容錯功能，基本上仍舊是透過 BPEL[9]流程控制達到單一 Web service 容錯功能，但是針對 Workflow 本身確沒有容錯，容錯方式為靜態方式而容錯等級為單一步驟級。

目前大多數學者專家以運用 BPEL[9]流程控制為主，但是針對整體程序而言沒有特別去探討然而我們所提出的架構透過程序實體序列化的功能容錯等級可以達到 Process 級，而透過動態行為處理邏輯方法容錯方式可以達到流程動態，使得程序容錯能力及安全機制透過動態行為條件判斷依據當下條件執行正確的安全措施並且將原本工作流程與所提供程序容錯能力及安全機制透過 SDL 結合，

以不須要更改原本工作流程精神將容錯能力及安全機制實現出來。

3. A framework to support process instance security in SOA

服務導向架構[2]顧名思義，以「服務」做為導向來出發，以設計並建構我們的系統構架。簡單來說，我們可以說：「服務導向架構」目的是要達到一個自動化的商業行為。服務導向架構[2]需提供一個「跨系統的資料交換與傳遞的規範與方法」，以便商業上的資訊交換。而在整個架構中，被實做出來以負責資訊的交換與傳遞的程式一個個的模組，我們可稱為服務元件。如果只是提供一個「跨系統的資料交換與傳遞的規範與方法」其實只要用 Web Service 就可以做到了。「服務元件」結合「流程」提供更多樣的自動化服務，所以自動化的商業行為並不只是資料交換那麼單純，例如，系統應該先進行身份確認（這裏是一個身份確認的服務元件）BPEL Process($BPEL_1$)，再來才進行詢報價程序（這裏應該又是另外一個服務元件）BPEL Process($BPEL_2$)，最後，當買方確認要進行下單的時候，才會進入採購程序（進入採購流程應該又是另外一組服務元件，甚至有可能對應到另外一個獨立的系統中）BPEL Process($BPEL_3$)。而組職、整合並決定所有的服務元件的使用順序地即是「流程」。因此商業自動化的目的下，如何進行跨程式語言、跨系統、跨組織、甚至跨企業的流程架構、規劃、設計與實作，也是服務導向架構[2]要思考與規範的重點項目。

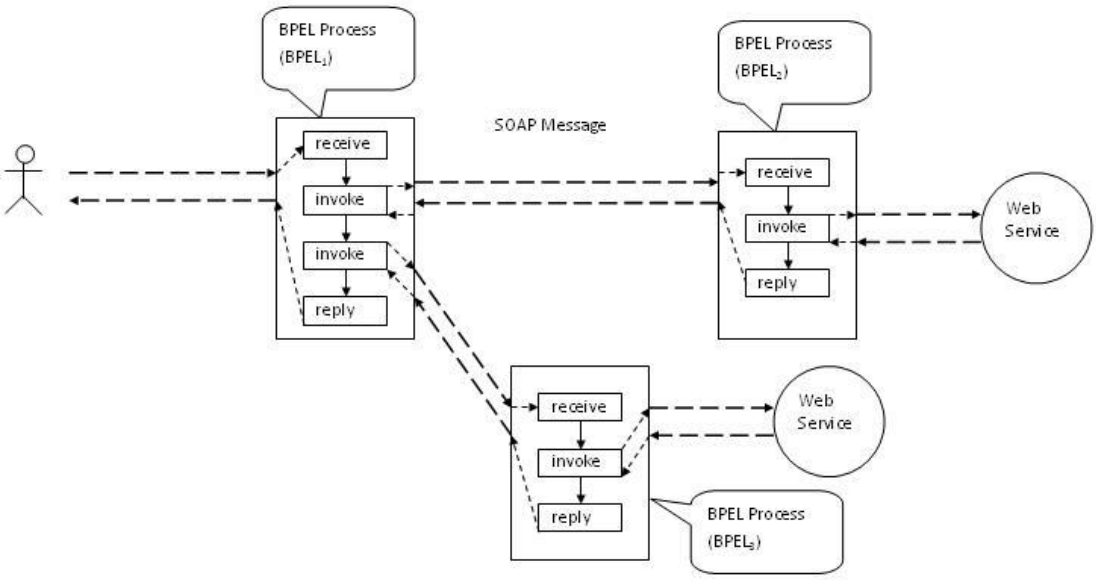


圖 3.1 簡單 SOA 操作模型

圖 3.1 是一個簡單服務導向架構[2]架構圖，每一個 BPEL Process 代表一個 Web Service 並且以 BPEL[9]來描述其流程，為有等級制度（hierarchical）的和等同圖形符號描述（graph-like）控制制度而一個 BPEL Process 可以再由任意個 BPEL Process 或其他軟體元件提供的 Web Service 加上 BPEL[9]流程語言描述其流程控制來達到程序的分解（decomposition）和組裝（assembly）。現象學提到「存有」與「關係」。「存有」在 OO（物件導向，Object-Oriented）中，可以視為是物件，而在服務導向架構[2]中，我們可以說它是服務元件。而關係，在 OO 中就是那個方法，而在服務導向架構[2]中，就是跟時間一起被定義在流程之中。服務導向架構[2]即是結合服務元件及流程的很嚴謹、很工整的架構，透過此簡單服務導向架構[2]架構圖可以將 OO 中強調之松耦合(Decoupling)充分發揮出來，松耦合的優點為在服務提供者和服務消費之間提供接口，這樣可以更改服務的具體實現而不影響服務消費。

圖 3.1 中簡單服務導向架構[2]架構圖可以發現就功能面而言已經達到服務導向架構[2]預期目標，但是缺乏兩樣重要需求:安全考量及容錯機制，而 SDL 將軟體元件、服務及流程三個部份整合並定義 Security 及 Serialization 功能及約束，使得圖 3.1 加入 Security 及 Serialization 功能及約束後而成圖 3.2，其中:

$BPEL_1$，$BPEL_2$，…，$BPEL_n$：BPEL Process 流程

SDL_1，SDL_2，…，SDL_n：SDL(Serialization description language)

$BPEL_i$=>SDL_i=>$BPEL_i$’ : BPEL’ Process 流程加入 Security 及 Serialization 功能及約束

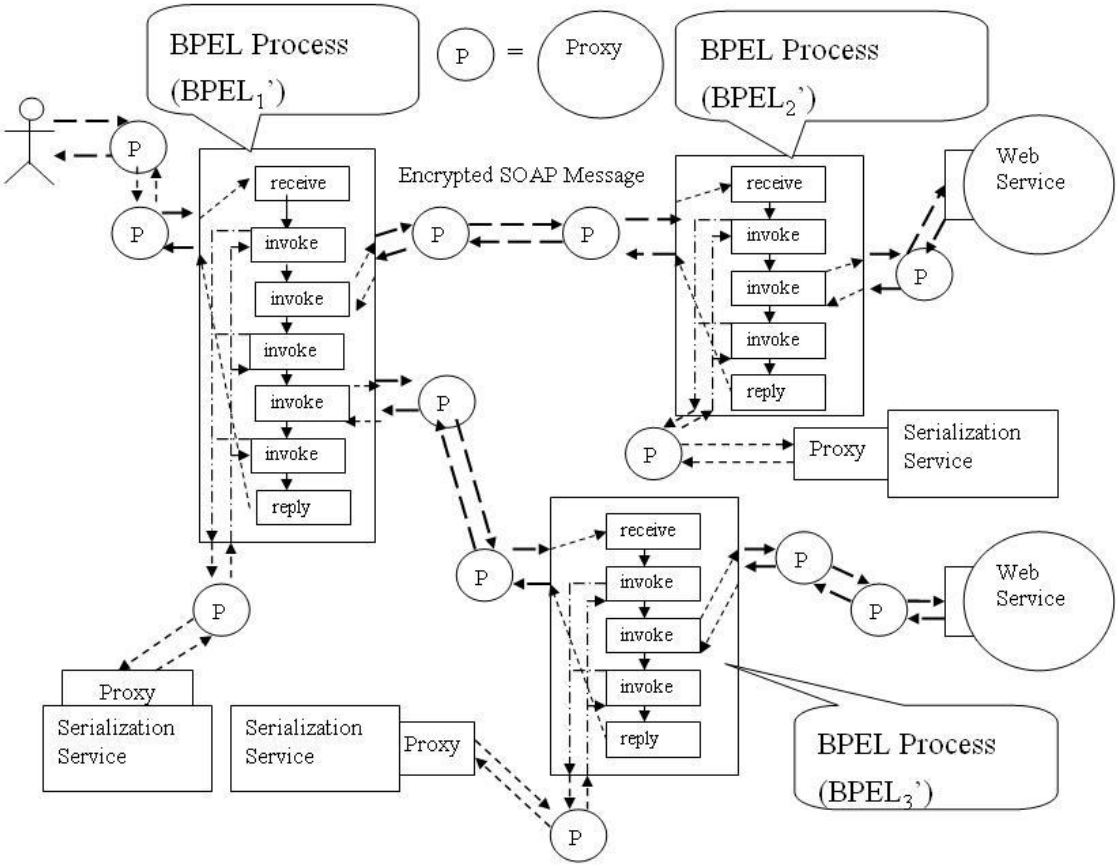


圖 3.2 是一個支援程序實體安全及訊息傳遞安全的 SOA 操作模型

所提出的架構中有關訊息傳遞時的安全的達成是借由網路服務安全語言[20]所提出的操作模型，該如圖 3.3

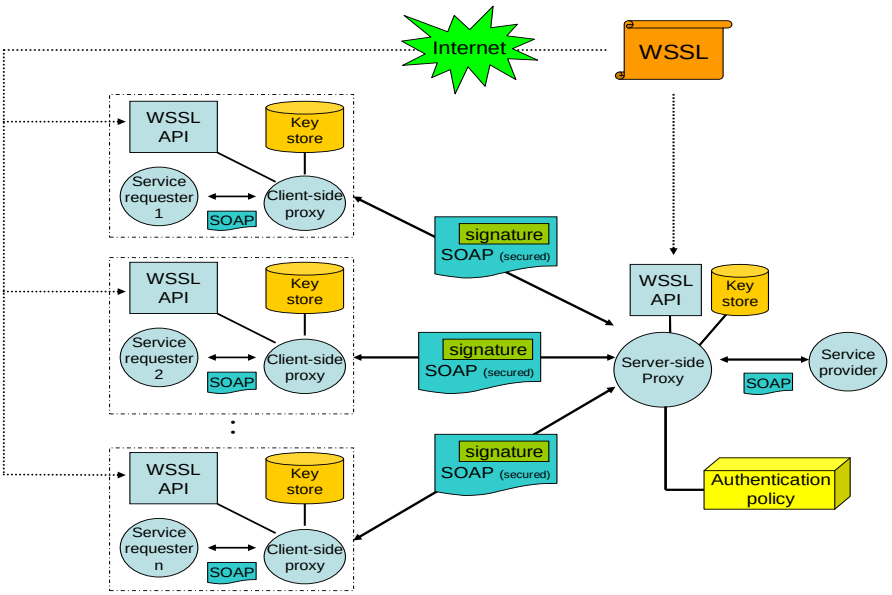


圖 3.3 WSSL 操作模型

這操作模型除了滿足基本的安全需求，包括驗證，機密性，完整性及不可否認性外，它也提供了元素層次加密(element-wise encryption)及以時序為基礎的元素次層數位簽章(temporal-based element-wise digital signature)的安全機制，而之所以能夠達到元素層次加密及以時序為基礎的元素次層數位簽章的安全機制則運用了 DSL(Document Security Language)[21]的技術。

網路服務安全語言操作模型支援一個具彈性的金鑰規格大綱，可以用來定義三種不同類型的金鑰，分別為靜態金鑰，動態選擇金鑰，以及採用數位簽章的金鑰。服務請求者可以決定使用金鑰的身份，而不需事先和服務提供者協商。在我們所提出來的操作模型中，設計出二種方法，可以用來減少系統開發與維護的成

本：(1)定義了一個網路服務安全語言，將網路服務中的服務實作與安全政策的規格分開。(2)藉由為網路服務安全語言設計的應用程式界面(API, Application Programming Interface)來支援所提供的操作模型。

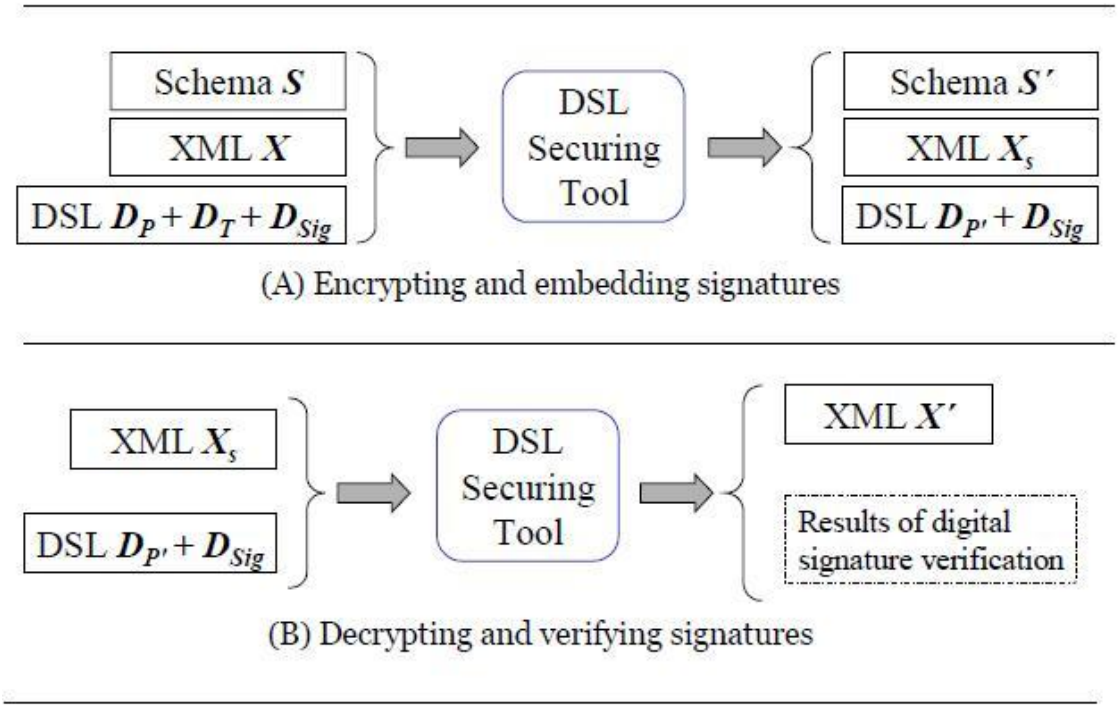


圖 3.4 DSL 針對 XML 文件安全機制之操作模型

所提出 SDL 架構有關訊息傳遞時的安全與網路服務安全語言 [20]所提出的操作模型如圖 3.5，主要重點是提供服務的代理人(Proxy)對應網路服務安全語言中伺服端代理人(Server-side proxy)而要求服務的代理人對應網路服務安全語言中客戶端代理人(Client-side proxy)則透過網路服務安全語言操作模型的元素層次加密及以時序為基礎的元素次層數位簽章的安全機制，可以確保有關訊息傳遞時的安全。

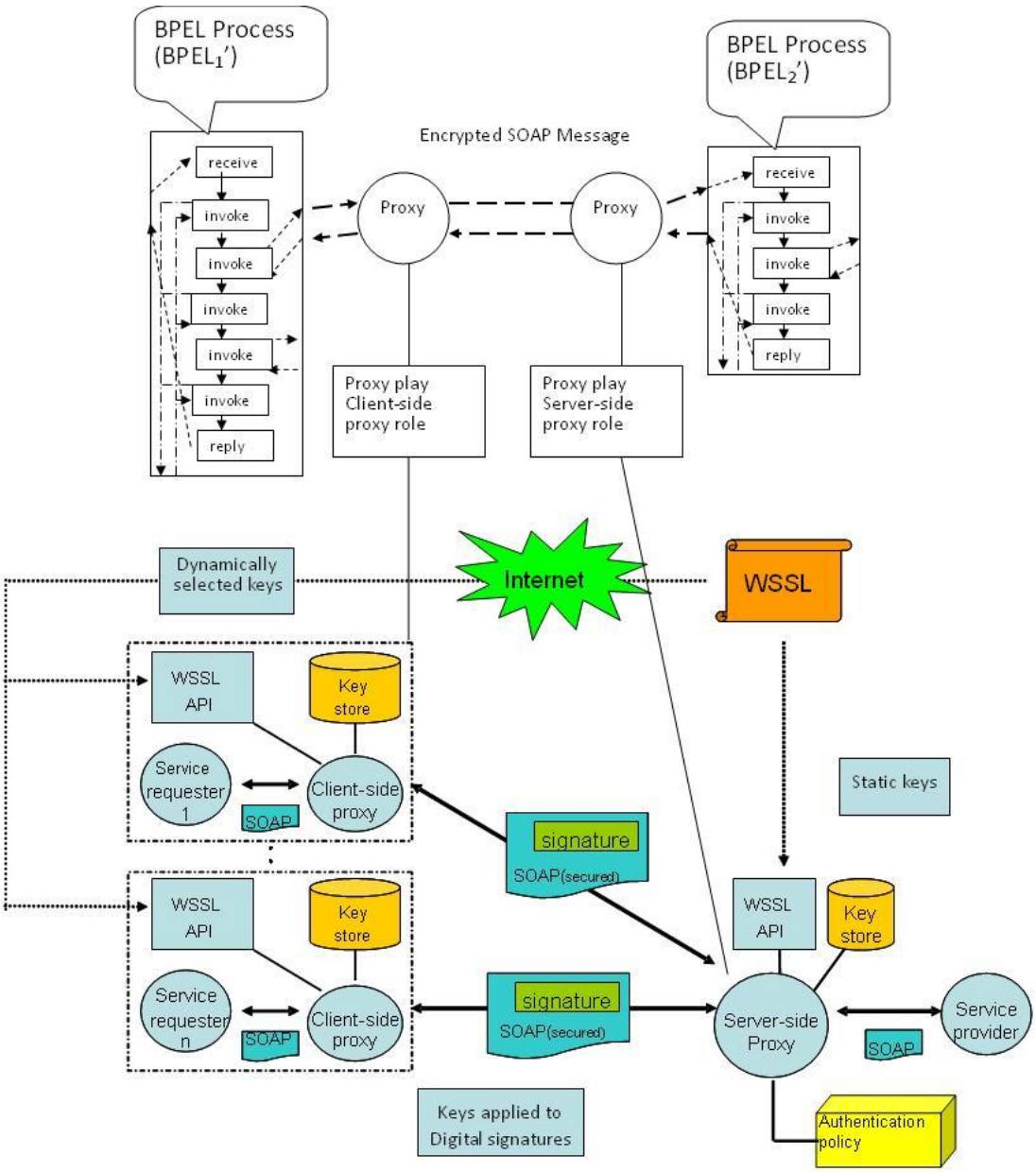


圖 3.5 BPEL' Proxy 與 WSSL Proxy 對應圖

3.1 SDL

在進一步探討 SDL 之前，首先讓我們針對 SDL 語法做一些說明。SDL 語法可分為五個段落，分別為 Header，Key definition，Workflow definition，Serialization definition，Digital signature 等段落，以下分別說明其內容。

Header section
Key definition section
Workflow definition section
Serialization definition section
Digital signature section

```
h1 <?xml version=1.0>
h2 <sdl:security_document xmlns:sdl="http://www.xml-dsl.com/2009/sdl/">
k1 <!--Key definition section>
k2 [K1]
k3 [K2]
.
k4 [Kx]
k5 <!--End of key definition section-->
w1 <!-- Workflow definition section-->
w2 [W1]
w3 [W2]
.
w4 [Wy]
w5 <!--End of workflow definition section-->
r1 <!--Serialization definition section-->
r2 [R1]
r3 [R2]
.
r4 [Rv]
r5 <!--End of serialization definition section -->
g1 <!--Digital signature section-->
g2 [G1]
g3 [G2]
.
g4 [Go]
g5 <!--End of digital signature section-->
</sdl:security_document >
```

圖 3.6 SDL 文件架構

3.1.1 Header section

因為 SDL 也是符合 XML 格式的文件，所以也必須符合 XML 格式定義故 Header 段第一行為<?xml version="1.0"?>而採用 1.0 版本所以 version 屬性值為 1.0，

而 SDL 之根元素為 <sdl:security_document xmlns:sdl="http://www.xml-sdl.com/2009/sdl/">，其中 xmlns:sdl 定義 sdl 名稱空間(namespace)，而 sdl 名稱空間的 URI 為 http://www.xml-sdl.com/2009/sdl/，名稱空間的使用是為了在網際網路的世界確保 XML 定義來源為何處。

3.1.2 Key definition section

近年來 Web Service 安全性標準制定已越來越趨成熟，為了能借鏡各方努力制定 Web Service 安全性標準，所以 SDL 也借用 Web Services Security:SOAP Message Security 1.1(WS-Security 2004)OASIS Standard Specification, 1 February 2006 有關 Binary Security Tokens 的定義。只是原本<wsse:BinarySecurityToken>定義須要在<wsse:Security> header 中，而在 SDL 中須要在<KeyDefinitionSection>中。

3.1.3 Workflow definition section

由於架構使用 BPEL[9]來實踐服務導向架構[2]流程處理部份，所以<Workflow DefinitionSection >中<targetBPEL>代表 SDL 所對應之 BPEL[9]的 URI，<targetBPEL>中<targetpartnerLink>代表 BPEL[9]所對應之 partnerLink，<targetpartnerLink> 中 <targetmyRole> 和 <targetpartnerRole> 代表 partnerLink 中 myRole 和 partnerRole，由 myRole 的 partnerLinkType 進而對應到 web service 的 WSDL 中<portType>，<targetmyRole>中的<WSSL>對應針對

此 WSDL 的網路服務安全語言文件進而取得確保訊息傳遞時的安全相關安全機制，<targetpartnerRole>中的<WSSL>對應針對此 WSDL 的網路服務安全語言文件進而取得確保訊息傳遞時的安全相關安全機制。範例如下：

```
< WorkflowDefinitionSection>
    <targetBPEL URI=" …" >
        <targetpartnerLink targetname=" …" >
            <targetmyRole>
                <WSSL>http://example.com/supply_system/registratio
            n_desk_1.wssl</WSSL>
            </targetmyRole>
            <targetpartnerRole>
                <WSSL>http://example.com/supply_system/registratio
            n_desk_2.wssl</WSSL>
            </targetpartnerRole>
        </targetpartnerLink>
        <targetpartnerLink>
        …
        </targetpartnerLink>
    </targetBPEL>
< /WorkflowDefinitionSection>
```

圖 3.7 Workflow definition section 範例

3.1.4 Serialization definition section

BPEL[9]使用步驟為單位達到流程處理的功能，步驟分為基本步驟和結構步驟，基本步驟為 Invoke、Receive、Reply、Assign、Throw、Wait、Empty、ExtensionActivity、Exit、Rethrow，結構步驟為 Sequence、If、While、RepeatUntil、Pick、Flow、ForEach。

Serialization 以步驟為單位將步驟使用的變數序列化，為了達到 Recovery

功能需要將步驟賦予一個可以唯一識別的代號，然而 BPEL[9]定義的步驟卻並無定義可以唯一識別的代號，為了達到這個功能特別借用 BPEL[9]中一個特別的步驟，它就是 Documentation Activity。Documentation Activity 就如其單字意義一樣只有註解功能，使人可以了解立即包含他的步驟的功用可以用 Documentation Activity 內之內容加以人們易讀的文字說明註解其功用。

SDL 特別利用 Documentation Activity 沒有流程處理的功能的特性，用以描述步驟之可以唯一識別的代號，此方法說明如下：

```
<Invoke …>
    <Documentation>
        [AId[A00001]]Initial Price Calculation
    </Documentation>
</Invoke>
```

圖 3.8 步驟唯一識別的代號

其中[AId[A00001]]中的 A00001 為 Invoke 的唯一識別的代號，而 Initial Price Calculation 說明註解，如此便克服了 BPEL[9]定義的步驟卻並無定義可以唯一識別的代號的問題。

Serialization definition section 中 <Serialization definition section>一個<Activities>。<Activities>中包含數個<Activity AId=”…” ActivityType = ”…”>，<Activity AId= ”…” ActivityType = ”…”>中包含一個 <DynamicBehaviorLogic>，一至數個 <SerializationVariable TargetVariable=”…” >。

<DynamicBehaviorLogic>中包含一個類似 BPEL[9] <if> 步驟[9]如圖 1.7 動態行為處理邏輯 If 語法，為了安全考量對工作流程而言確是常常需要知悉較

低階層 WS-Security 執行時的狀況來判斷決定工作流程的執行路徑，定義了四個擴充布林函數 ActivityIsExecuted (ActivityName)，SequenceIsExecuted (AN1,AN2,AN3,…)，IsEncryptedValidated(ActivityName)，IsSignedValidated (ActivityName)，兩個擴充函數 Encrypt(Key,S1,S2,…)，Sign(Key,S1,S2,…)

<DynamicBehaviorLogic>中的<condition>bool-expr</condition>如圖 1.8 動態行為處理邏輯 If 語法，採用 BPEL[9]一樣 WS-BPEL Boolean expressions 類似 XPATH Boolean expressions 加上四個擴充布林函數，在這裡特別再說明上四個擴充布林函數特性:

1. ActivityIsExecuted(ActivityName)：步驟其識別 ID 為 ActivityName 在此程序實體是否已經執行過，已經執行過傳回真，否則傳回假。
2. SequenceIsExecuted(AN1,AN2,AN3,...)：步驟其識別 ID 依序為 AN1,AN2,AN3,…在此程序實體是否已經依序執行過，已經依序執行過傳回真，否則傳回假。
3. IsEncryptedValidated(ActivityName): 步驟其識別 ID 為 ActivityName 是否可解密，可解密則傳回真，否則傳回假。
4. IsSignedValidated (ActivityName): 步驟其識別 ID 為 ActivityName 是否簽章驗證有誤，簽章驗證有誤傳回假，否則傳回真。

<DynamicBehaviorLogic>中的 if 步驟如圖 1.8 動態行為處理邏輯 If 語法定義了兩個擴充函數 Encrypt (Key,S1,S2,…)，Sign (Key,S1,S2,…)。

1. Encrypt(Key,S1,S2,…)：使用 Key 加密 S1, S2, …所指示之資料

2. Sign(Key,S1,S2,…)：使用 Key 簽章 S1, S2, …所指示之資料

而一至數個 <SerializationVariable TargetVariable=”…” >，<SerializationVariable TargetVariable=”…”>中包含數個<XPath>或不包含任何<XPath>，其中<XPath>內容表示該元素及該元素所包含之元素要序列化，比較特別的是當不包含任何<XPath>表示該變數全都要序列化，TargetVariable 指出要序列化的 BPEL[9] Variable，此外 <SerializationVariable TargetVariable=”…”>中也可以有一個<condition>bool-expr</condition>，Condition 採 BPEL[9] Boolean expression 類似 XPATH Boolean expressions 定義當條件為真(true)時序列化。Serialization definition section 範例如下:

```
<SerializationDefinitionSection>
    <Activities>
    <Activity AId=” A0001” ActivityType =” Receive” >
        <SerializationVariable TargetVariable=” input” />
            <Condition>…</Condition>
        </SerializationVariable>
        <DynamicBehaviorLogic>
            <sequence>
            Encrypt( ‘Keydef1’ ,’$Rec/ItemName’,’$Rec/ItemID’,…)
            Sign( ‘Keydef2’ ,’$Rec/ItemName’,’$Rec/ItemID’,…)
            </sequence>
        </DynamicBehaviorLogic>
    </Activity>
```

```
<Activity AId=" A0002"  ActivityType =" Assign" />
    <SerializationVariable TargetVariable=" invokeA" />
        <Condition>…</Condition>
    </SerializationVariable>
    <DynamicBehaviorLogic>
        <sequence>
        Encrypt( 'Keydef3' ,'$Rec/ItemName','$Rec/ItemID',…)
        Sign( 'Keydef2' ,'$Rec/ItemName','$Rec/ItemID',…)
        </sequence>
    </DynamicBehaviorLogic>
</Activity>
< Activity AId=" A0003"  ActivityType =" Invoke" >
    <SerializationVariable TargetVariable=" invokeB" />
        <Condition>…</Condition>
        <XPath>/Name</XPath>
        <XPath>/Phone</XPath>
    </SerializationVariable>
    <DynamicBehaviorLogic>
        <if>
        <condition>contains($Rec/type,' Weapon' )</condition>
            <sequence>
                Encrypt( 'KeydefS' ,'$Rec/ItemName',
                    '$Rec/ItemID',…)
                 Sign( 'Keydef2' ,'$Rec/ItemName',
                 '$Rec/ItemID' ,…)
            </sequence>
        <else>
            <sequence>
                 Encrypt( 'KeydefN' ,'$Rec/ItemName',
                 '$Rec/ItemID',…)
                 Sign( 'Keydef2' ,'$Rec/ItemName',
                 '$Rec/ItemID',…)
            </sequence>
        </else>
        </if>
    </DynamicBehaviorLogic>
```

```
    …
    </Activities>
</SerializationDefinitionSection>
```

圖 3.9 Serialization definition section 範例

SerializationVariable Condition 處理方式如圖 3.10 SerializationVariable Condition 處理流程，BPEL 轉變成 BPEL’後 Condition 邏輯會依據 SDL 的內容影響序列化的內容。

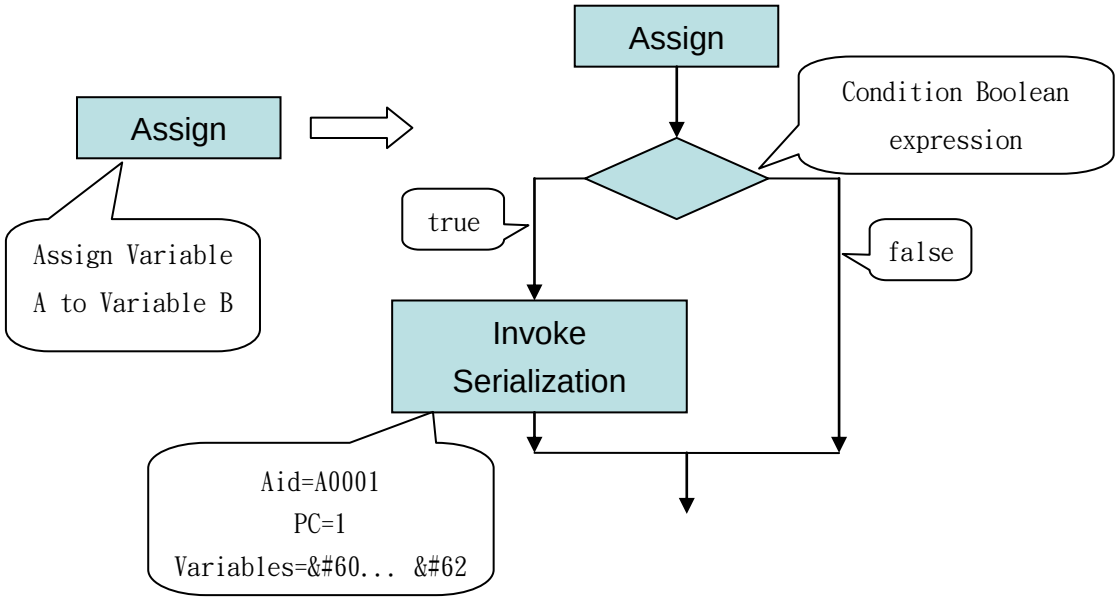


圖 3.10 SerializationVariable Condition 處理步驟

ActivityIsExecuted(ActivityName) 擴充布林函數處理方式如圖 3.11 轉換成 BPEL’後會呼叫序列化 Web service，因為序列化 Web service 透過程序實體才可能得到程序之前執行狀態，進而達到 1.7 節 Dynamic behavior in workflow security issue 中所提出的執行狀況判斷的功能，而 SequenceIsExecuted(AN1,AN2,AN3,…) 擴充布林函數處理方式也如圖 3.12 只是其傳入的參數多於一個，而傳入的參數的順序代表是否程序實體執行順序序列有

符合的，透過布林運算子的運用能夠達到平行運算模式條件判斷，例如 SequenceIsExecuted('A0001','A0002') and SequenceIsExecuted('A0003','A0004')可以表示 A0001、A0002 執行順序和 A0003、A0004 執行順序是平行運算。

IsEncryptedValidated(ActivityName) 擴充布林函數處理方式如圖 3.12 轉換成 BPEL' 後會呼叫 WSSL Proxy Web service，因為 WSSL Proxy Web service 透過 WSSL API 才可能得到 Encrypted 資料是否能夠解碼，進而達到 1.7 節 Dynamic behavior in workflow security issue 中所提出的 Security 的狀況判斷的功能，IsSignedValidated (ActivityName) 擴充布林函數處理方式轉換成 BPEL' 也後會呼叫 WSSL Proxy Web service，因為 WSSL Proxy Web service 透過 WSSL API 才可能得到簽章是否有效。

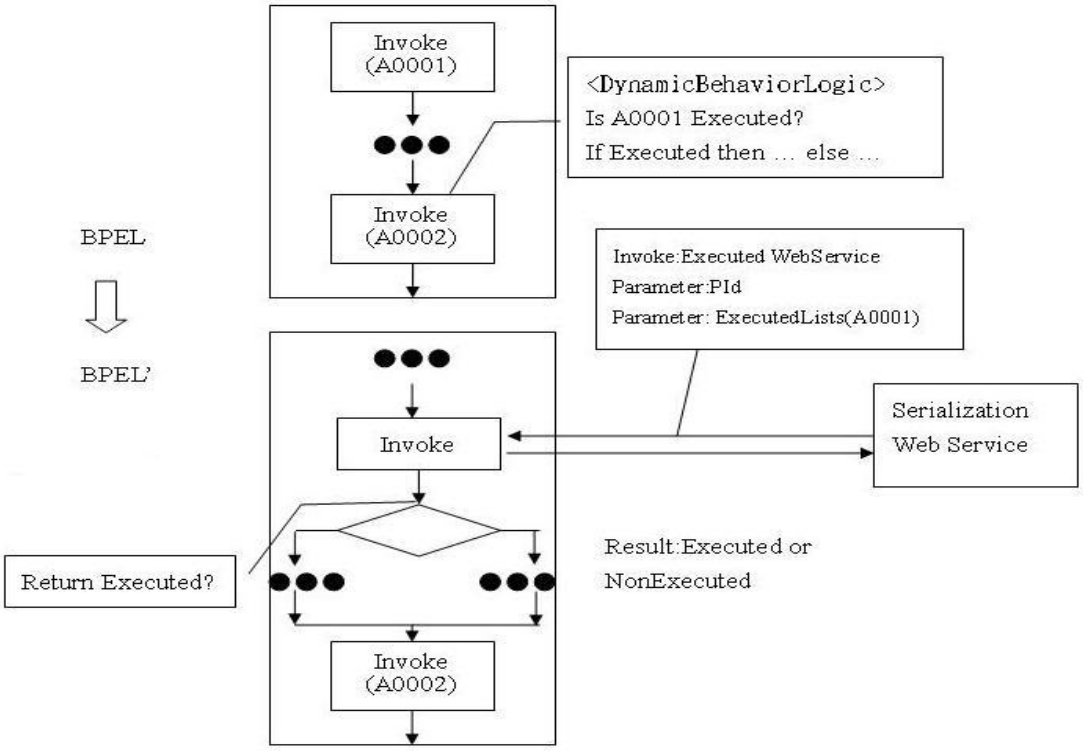


圖 3.11 ActivityIsExecuted(ActivityName)擴充布林函數步驟

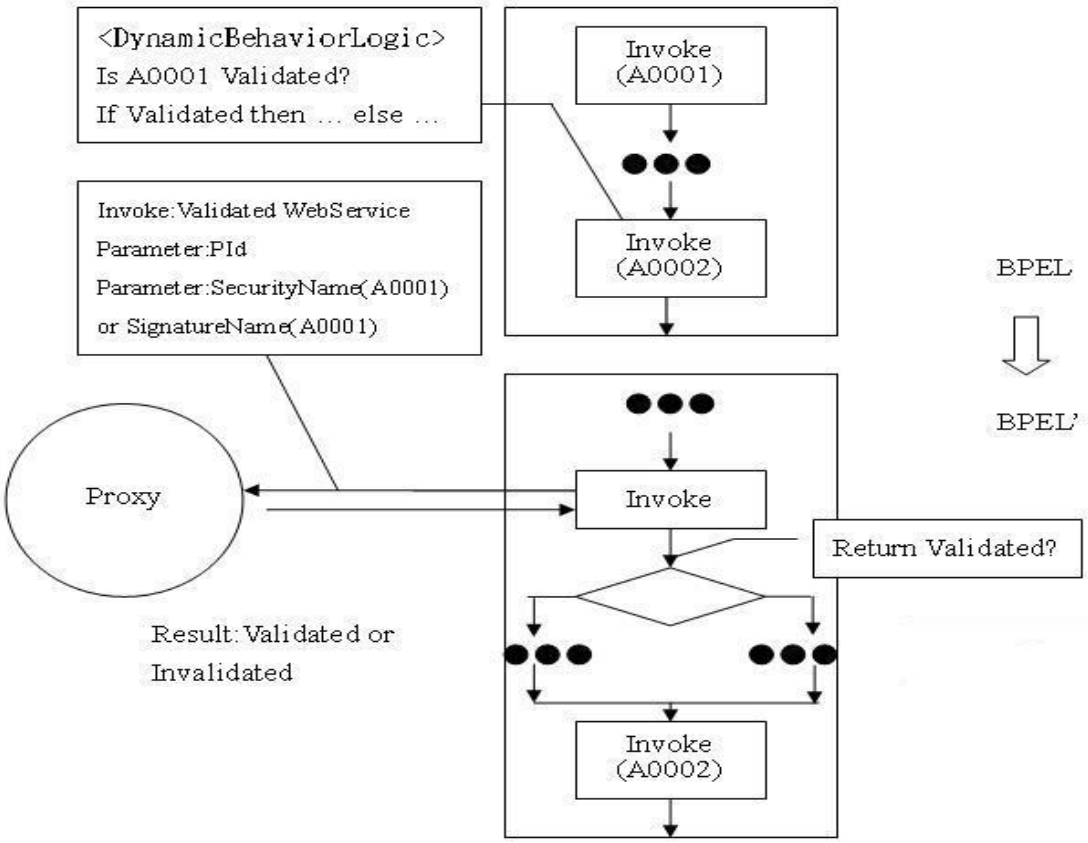


圖 3.12 IsEncryptedValidated(ActivityName)及 IsSignedValidated (ActivityName)擴充布林函數步驟

Serialization 以步驟為單位將步驟使用的變數序列化，透過 Serialization Web Service 將步驟執行前加上參數設定前置 Assign 步驟，將步驟執行後加上 Invoke 步驟 呼叫 Serialization Web Service，Serialization Web Service 包含:

1. Serialization Register Web Service：註冊 Serialization ID 作為 Serialization 識別用途，同時也可提供為儲存檔名，使用 UUID 為其格式可以有效確保，在網際網路上沒有重複一樣之 UUID，原因是 UUID 之產生以網路卡、時間、電腦環境等參數產生發生重複的機率很低很低。

2. Serialization Begin Web Service：註記序列化起始時間及第一個 Receive 或 Pick 步驟變數。

3. Serialization Activity Web Service：序列化步驟變數。

4. Serialization End Web Service：序列化 Reply 步驟變數，及註記序列化結束時間。

序列化文件為 XML 文件，其階層式表示法如下:

- ProcessInstant (Element)
- PId (Attribute)
- InitTime (Attribute)
- Mode (Attribute)
- Activity+ (Element)
  i. AId (Attribute)
  ii. PC (Attribute)
  iii. InvokeEngine (Attribute)
  iv. STime (Attribute)
  v. Mode (Attribute)
  vi. Variables (Text)
  vii. Tree Structure
- TimeStamp (Element)
  i. PId (Attribute)
  ii. FinishTime (Attribute)

圖 3.13 序列化文件為其階層式表示法

序列化文件為 XML 文件如圖 3.14

```
<?xml version="1.0" encoding="UTF-8" ?>
- <ProcessInstant PId="1c655bdf-f5ba-4002-8d66-9fbd91e804be" InitTime="''" Mode="Serialization">
  - <Activity AId="A0000" PC="0" InvokeEngine="" STime="2009-07-10T17:08:16.0828Z" Mode="REC">
    - <SupplyRequestOperation>
        <ItemName>F16 Engine</ItemName>
        <ItemID>F16_0001</ItemID>
        <Amount>2</Amount>
        <RequestPerson>Chao-Chen Chiang</RequestPerson>
        <RequestDate>2009-07-10T24:00</RequestDate>
        <Description>situation emergency</Description>
        <Type>weapon</Type>
      </SupplyRequestOperation>
    </Activity>
  - <Activity AId="'A0001'" PC="1" InvokeEngine="'140.122.184.92'" STime="2009-07-10T17:08:16.0921Z" Mode="'IN'">
    - <ReceiveOperation>
        <ItemName>F16 Engine</ItemName>
        <ItemID>F16_0001</ItemID>
        <Amount>2</Amount>
        <Date>2009-07-10T24:00</Date>
        <Description>Desktop Receive Data from Supply System [situation emergency]</Description>
        <Type>weapon</Type>
      </ReceiveOperation>
    </Activity>
  - <Activity AId="'A0002'" PC="2" InvokeEngine="'140.122.184.92'" STime="2009-07-10T17:08:17.0140Z" Mode="'EXT'">
    - <ReceiveOperationResponse>
        <ReceiveDate>2009-07-10T17:08:17.0062Z</ReceiveDate>
        <ReceivePerson>John</ReceivePerson>
        <Description>I got it</Description>
      </ReceiveOperationResponse>
    </Activity>
  - <Activity AId="'A0003'" PC="3" InvokeEngine="'140.122.184.92'" STime="2009-07-10T17:08:17.0234Z" Mode="'IN'">
    - <OrderOperation>
        <ItemName>F16 Engine</ItemName>
        <ItemID>F16_0001</ItemID>
        <ItemPrice>'100000'</ItemPrice>
        <Amount>2</Amount>
        <Description>'High Priority Please!'</Description>
      </OrderOperation>
    </Activity>
  - <Activity AId="'A0004'" PC="4" InvokeEngine="'140.122.184.92'" STime="2009-07-10T17:08:17.0437Z" Mode="'EXT'">
    - <OrderOperationResponse>
        <Result>OK</Result>
        <Description>Any thing is fine.</Description>
      </OrderOperationResponse>
    </Activity>

  - <Activity AId="'A0005'" PC="5" InvokeEngine="'140.122.184.92'" STime="2009-07-10T17:08:17.0515Z" Mode="'IN'">
    - <SupplyRequestOperationResponse>
        <Result>OK</Result>
        <Description>Any thing is fine.</Description>
        <ReplyDate>'2009-07-10T24:00'</ReplyDate>
      </SupplyRequestOperationResponse>
    </Activity>
    <TimeStamp PId="1c655bdf-f5ba-4002-8d66-9fbd91e804be" FinishTime="2009-07-10T17:08:17.0578Z" />
</ProcessInstant>
```

圖 3.14 序列化文件範例

XSLT 是 Extensible Stylesheet Language Transformations 縮寫，為 W3C 制定標準此處採用 XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999 版本，主要用途用來轉換 XML 文件成另一種格式的 XML 文件，此處為了達到 uniform representation 的目的將步驟變數序列化，使用如下列 XSLT：

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:output omit-xml-declaration="yes"/>
<xsl:template match="text()"/>
<xsl:template match="/" priority="1">
    <logcontent>
    <xsl:apply-templates/>
    </logcontent>
</xsl:template>
<xsl:template match="*" priority="2">
    &#60;<xsl:value-of select="name()"/>&#62;
    <xsl:apply-templates/>
    <xsl:if test="text()">
        <xsl:value-of select="text()"/>
    </xsl:if>
    &#60;/<xsl:value-of select="name()"/>&#62;
</xsl:template>
</xsl:stylesheet>
```

圖 3.15 為了步驟變數 uniform representation 之 XSLT

加上 BPEL[9]中特別定義將 XSLT 1.0 納入其 BPEL[9]標準中，並定義 bpel:doXslTransform 函數執行 XSLT 1.0，其函數語法如下：

object bpel:doXslTransform(string, node-set, (string, object)*)

透過 XSLT 及 bpel:doXslTransform 函數使得變數轉換為變數字串。

如圖 3.16 透過 Serialization Web Service 將 BPEL[9]的步驟後加上 Invoke Serialization Web Service 的步驟達到序列化的目的，為了能夠記錄程序的執行順序 BPEL’會加上兩個特別的變數 Pid 及 PC，其中 Pid 儲存 UUID 格式的字串用來識別程序實體而 PC 基本上隨著步驟執行而執行加一計算，用來針對平行運算執行時能確時記錄程序的執行順序其操作模型如下圖。

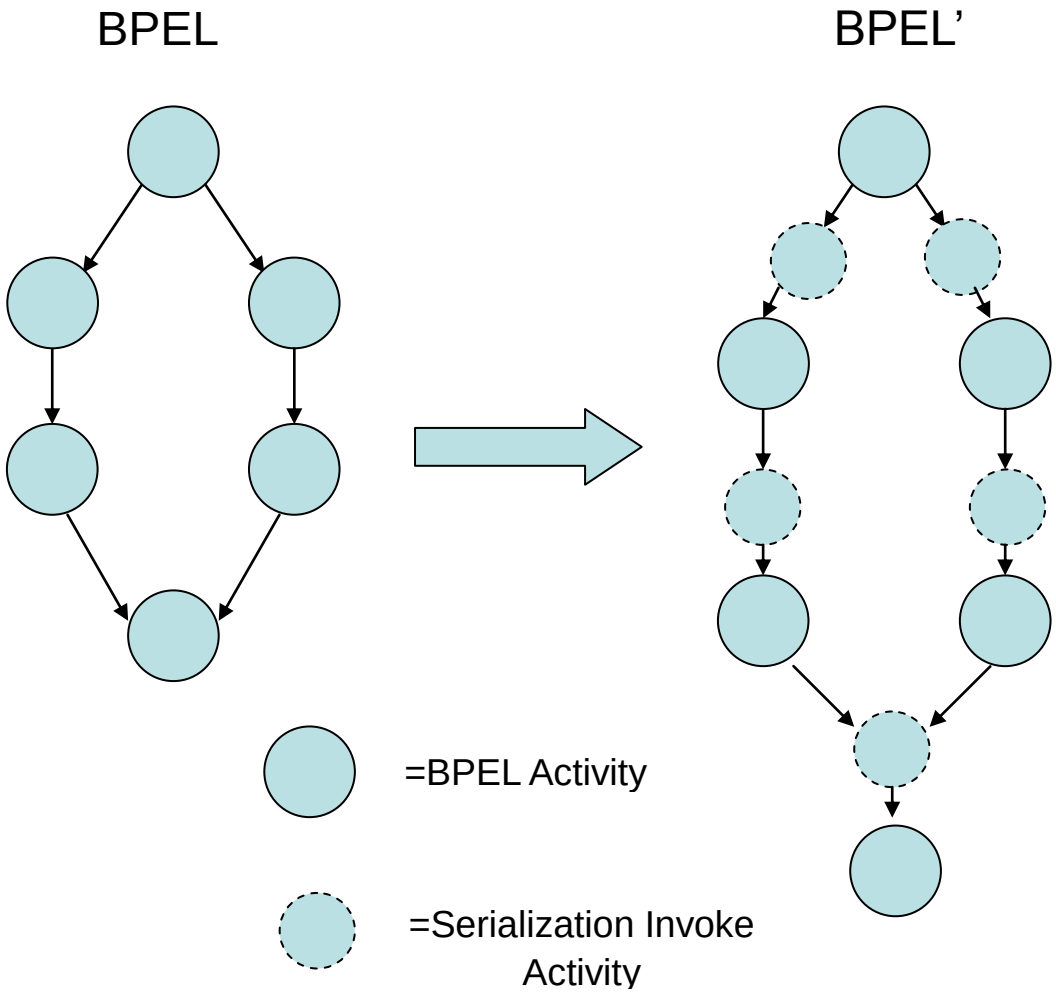


圖 3.16 Serialization operation model

3.1.5 Digital signature section

Digital signature section 主要是要定義簽章，以免 SDL 被串改，參考 XML-Signature Syntax and Processing W3C Recommendation 12 February 2002 標準，語法如下。

```
<DigitalSignatureSection>
  <Signature ID?>
    <SignedInfo>
      <CanonicalizationMethod/>
      <SignatureMethod/>
      (<Reference URI? >
        (<Transforms>)?
        <DigestMethod>
        <DigestValue>
      </Reference>)+
    </SignedInfo>
    <SignatureValue>
   (<KeyInfo>)?
   (<Object ID?>)*
  </Signature>
</DigitalSignatureSection>
```

圖 3.17 Digital signature section 語法

3.2 Fault Recovery Model

錯誤回復[13]一般來說是指當系統發生異常導致系統無法正常運作時所採取的一種處理使得系統狀況由錯誤狀態回復至正常狀況。錯誤回復處理方式一般又分為兩種：前向式錯誤回復[13]及後向式錯誤回復(Backward Recovery)[13]，回復技術主要目的是為了使系統狀況由錯誤狀態回復至正常狀況，而前向式錯誤回復採取的方式找到一個可以使系統正常運作狀態來達到錯誤回復功能，此種方式比後向式錯誤回復有較少重複資料(Overhead)，但是前向式錯誤回復雖然將系統狀況由錯誤狀態回復至正常狀況但是與真正的未發生錯誤前可能不是百分之百一樣。後向式錯誤回復則是採用將之前已儲存(一般來說是週期性的儲存)的系統狀況來達到由錯誤狀態回復至正常狀況，此種方式可以達到錯誤回復至真正的未發生錯誤前百分之百一樣，但是有較多重複資料。

在本架構下由於程序實體已序列化，所以採取後向式錯誤回復相當容易達成，主要要克服的問題是如何將程序實體由序列化轉換成可運作的程序實體，一般是要靠 BPEL Engine 來提供一些功能。在此提供 ActiveBPEL(TM) engine4.0 中相關的功能，由於 ActiveBPEL(TM) engine4.0 中提供一些管理功能 Web service 可能設定 BPEL Process 變數，其中 BpelEngineAdmin (wsdl) 中 getVariable Web service 可以用來取得 Process 的 Variable 變數內容而 setVariable Web service 可以用來設定 Process 的 Variable 變數內容 ，不過整體來說對錯誤回復由於須仰賴 BPEL Engine 來提供一些功能往往有很多的限制，提出一個 Failover Model 來達到失敗轉移，進而達到當錯誤發生時可以利用備份伺服器取代使得使用者根本不知道曾經有錯誤發生過。

3.3 Failover Model

失敗轉移是當故障發生時或異常終止以前的伺服器能夠自動切換到冗餘或備用伺服器，不須要任何人為干預，通常使用者沒有察覺伺服器已經自動切換到冗餘或備用伺服器。首先為了達到失敗轉移的功能除了原本主伺服器(Master BPEL Server)須要增加一個監督伺服器(Monitor Server)及一個備份伺服器(Backup BPEL Server)，監督伺服器功能為監督主伺服器是否正常運作並且作為提供服務窗口，當主伺服器發生故障或異常終止時監督伺服器會將備份伺服器啟動，利用之前序列化的程序實體回復至之前狀態再繼續運作，而使用者不會發覺主伺服器發生故障並且已切換至備份伺服器。

由於監督伺服器為監督主伺服器是否正常運作並且作為提供服務窗口，若是監督伺服器發生問題同樣會造成運作的問題，但是監督伺服器不負責太複雜流程發生錯誤機率相對來說較小，而且可以利用硬體技術可以克服此問題，比如說使用叢集伺服器(cluster server)來當監督伺服器。

若是單以功能面來說也可以使用 BPEL Engine 來達到監督伺服器的功能，監督伺服器每 5 分鐘會詢問主伺服器是否有問題，若是主伺服器沒有回復正常訊息則監督伺服器會負責啟動備份伺服器，備份伺服器會根據已序列化程序實體執行回復動作，當回復動作完成後再繼續執行原本流程。備份伺服器的流程為了達到根據已序列化程序實體執行回復動作的功能必須加上額外的步驟。

BPEL[9]為了提供程序執行時即時訊息處理能力特別定訂 Event Handlers 機制使得當程序執行時有針對此程序執事件發生時會有對應程序 Event Handlers 來處理此事件，監督伺服器利用主伺服器及備份伺服器 Event Handlers 來得知主伺服器及備份伺服器是否運作正常，判斷方式就是當主伺服器及備份伺服器正常運作時會回復 ok 訊息，當訊息不是 ok 或逾時還沒有 ok 訊息的話監督伺服器就斷定該伺服器已經異常終止。Failover Model 架構如圖 3.17

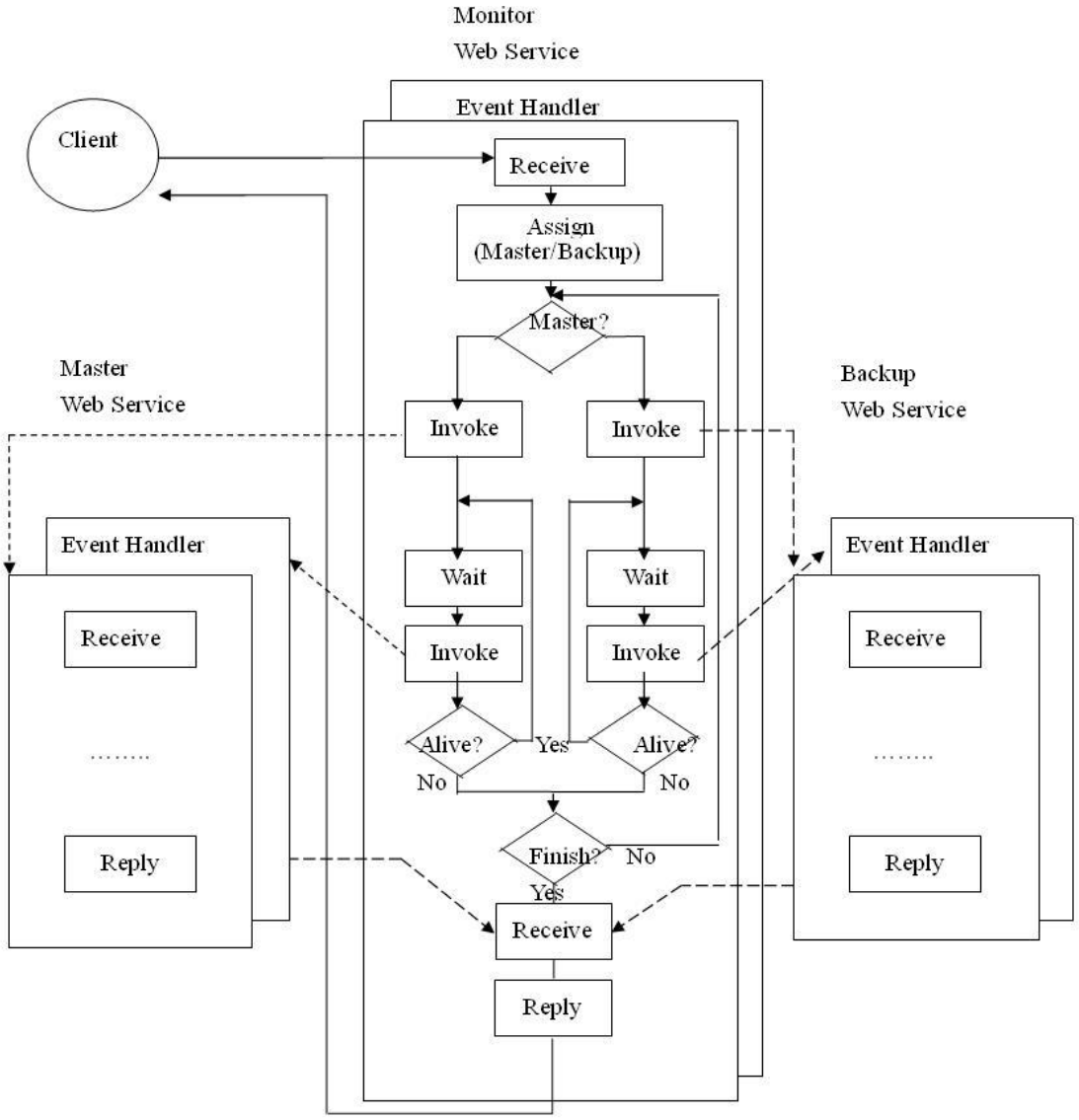


圖 3.18 Failover Model 架構

4. Implementation and experimental result

實作圖 1.6 後勤補給系統(二)工作流程 BPEL[9]使用序列化操作模型(Serialization Operation Model)為依據產生 BPEL’如 Appendix. A，而一般補給單位及武器部門及供應商使用 Axis2 提供 Code Generator 協助產生 Web Services 主要程式如 Appendix. B 所示，執行補給系統(二)請求及回覆 SOAP 如下:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sup="http://www.example.org/SupplySystem/">
   <soapenv:Header/>
   <soapenv:Body>
      <sup:SupplyRequestOperation>
         <ItemName>F16 Engine</ItemName>
         <ItemID>F16_0001</ItemID>
         <Amount>2</Amount>
         <RequestPerson>Chao-Chen Chiang</RequestPerson>
         <RequestDate>2009-07-10T24:00</RequestDate>
         <Description>situation emergency</Description>
         <Type>weapon</Type>
      </sup:SupplyRequestOperation>
   </soapenv:Body>
</soapenv:Envelope>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <aetgt:SupplyRequestOperationResponse
xmlns:aetgt="http://www.example.org/SupplySystem/">
         <Result
xmlns:ns1="http://www.example.org/ItemOrder/">OK</Result>
```

```
<Description xmlns:ns1="http://www.example.org/ItemOrder/">Any thing
is fine.</Description>
          <ReplyDate>'2009-07-10T24:00'</ReplyDate>
      </aetgt:SupplyRequestOperationResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

圖 4.1 後勤補給系統(二)申請人申請及回覆 SOAP 內容一

Pid	1c655bdf-f5ba-4002-8d66-9fbd91e804be		
步驟(Activity)	步驟代號(Aid)	序列化費時(毫秒)	程序實體大小(Byte)
Receive	'A0000'	15	536
Assign	'A0001'	16	911
Invoke	'A0002'	0	1215
Assign	'A0003'	0	1528
Invoke	'A0004'	15	1769
Assign	'A0005'	16	2067
Reply		16	2182

表 5.1 序列化時間及程序實體大小(一)

表 5.1 序列化時間及程序實體大小(一)中 Pid 為實體序列化之識別子，因為同一個 BPEL 工作流程會由成千上萬個程序實體，當還沒序列化時在作業系統可以識別不同程序實體，但是序列化後需要提供檔案系統一個識別子，Pid 就是為了這個目的而創造出來的而且透過 Serialization Register Web Service 產生同時 BPEL 為了協同運作 Web services 常常需要提供序列化 Web services 該程序實體 Pid，否則序列化 Web services 無法得知要求序列化程序之正確的程序實體，步驟(Activity)欄位表示原本 BPEL 之步驟而步驟代號(Aid) 欄位表示 BPEL' 所賦予之步驟代號，序列化費時(毫秒) 欄位則是序列化一個步驟花費多少毫秒，程序實體大小(Byte)欄位是序列化的程序實體大小，由表 5.1 可知因為序列

化是隨執行的步驟持續序列化所以程序實體大小是一直遞增直到工作流程結束。

請求 SOAP 改為如下：

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sup="http://www.example.org/SupplySystem/">
   <soapenv:Header/>
   <soapenv:Body>
      <sup:SupplyRequestOperation>
         <ItemName>鋼盔</ItemName>
         <ItemID>helmet</ItemID>
         <Amount>1000</Amount>
         <RequestPerson>Chao-Chen Chiang</RequestPerson>
         <RequestDate>2009-07-08T24:00</RequestDate>
         <Description>一般補給品,數量要正確</Description>
         <Type>normal</Type>
      </sup:SupplyRequestOperation>
```

圖 4.1 後勤補給系統(二)申請人申請及回覆 SOAP 內容二

執行序列化時間及程序實體大小如下：

Pid	95a2bc8d-ce6e-4b1c-86b6-894df4d2ea88		
步驟(Activity)	步驟代號(Aid)	序列化費時(毫秒)	程序實體大小(Byte)
Receive	'A0000'	16	562
Assign	'A0001'	16	963
Invoke	'A0002'	0	1267
Assign	'A0003'	0	1593
Invoke	'A0004'	0	1834
Assign	'A0005'	0	2132
Reply		0	2247

表 5.2 序列化時間及程序實體大小(二)

表 5.1 序列化時間及程序實體大小(二)則是和表 5.1 序列化時間及程序實體

大小(一)同一 BPEL 工作流程但是申請 SOAP 內容不同，可以發現 Pid 不一樣，基本上所有的程序實體的 Pid 都不應該重複，步驟(Activity)欄位和表 5.1 同而序列化費時(毫秒)則不同，檔案系統可能因為效能考量會有誤差不過一般來說每個步驟序列化平均約花十幾毫秒程序實體大小(Byte)欄位則跟序列化變數大小及所執行步驟多寡有關。

5.Conclusions & Future work

服務導向架構[2]的概念已經被提出相當一段時間，早期透過 CORBA[26]、RMI[27]、DCOM[28]或甚至使用 Socket 的技術都可以建立具有服務導向架構[2]的系統，但是都由於技術及系統平台沒有統一標準使得原意是在軟體設計的概念和整合性軟體的架構的理念大打折扣，Web Services 的技術成熟成為實踐服務導向架構[2]非常適合的基礎技術，但是針對服務導向架構[2]的安全機制卻沒有一個統一標準可以遵循。

本文針對服務導向架構[2]下探討程序實體的安全的重要性及其應用於動態行為的特性，還有它對於容錯能力及負載平衡能力的提升也有很好的潛力，提出一個架構透過所提出的 SDL 將軟體元件、服務及流程三個部份整合並定義 Security 及 Serialization 功能及約束，其中關於訊息傳遞時的安全(Communication Security) 借由網路服務安全語言[20]所提出的操作模型達成，並由 SDL 中 Workflow definition 定義該工作流程與對應網路服務安全語言關聯。

所提出的 SDL 主要將程序實體的安全定義於 Serialization definition 並透過 DynamicBehaviorLogic 定義步驟的動態行為處理邏輯及 SerializationVariable 定義序列化內容達到賦予動態行為邏輯處理安全機制又不會造成管理維護的夢魘並確保權責歸屬明確可信的優點。

關於未來工作主要有幾個方面，首先是關於訊息傳遞時的安全

(Communication Security)的結合的進一步研究，希望能夠達到代理人(Proxy)在訊息傳遞時能夠透過安全的機制與程序實體的安全更進一步整合，接著就是提供圖形化介面工具協助管理人員製作 SDL 文件而不須要了解 SDL 語言，希望製作 SDL 能夠透過拖拉的方式或圖形化操作介面減少錯誤政策制定的情況發展。

References

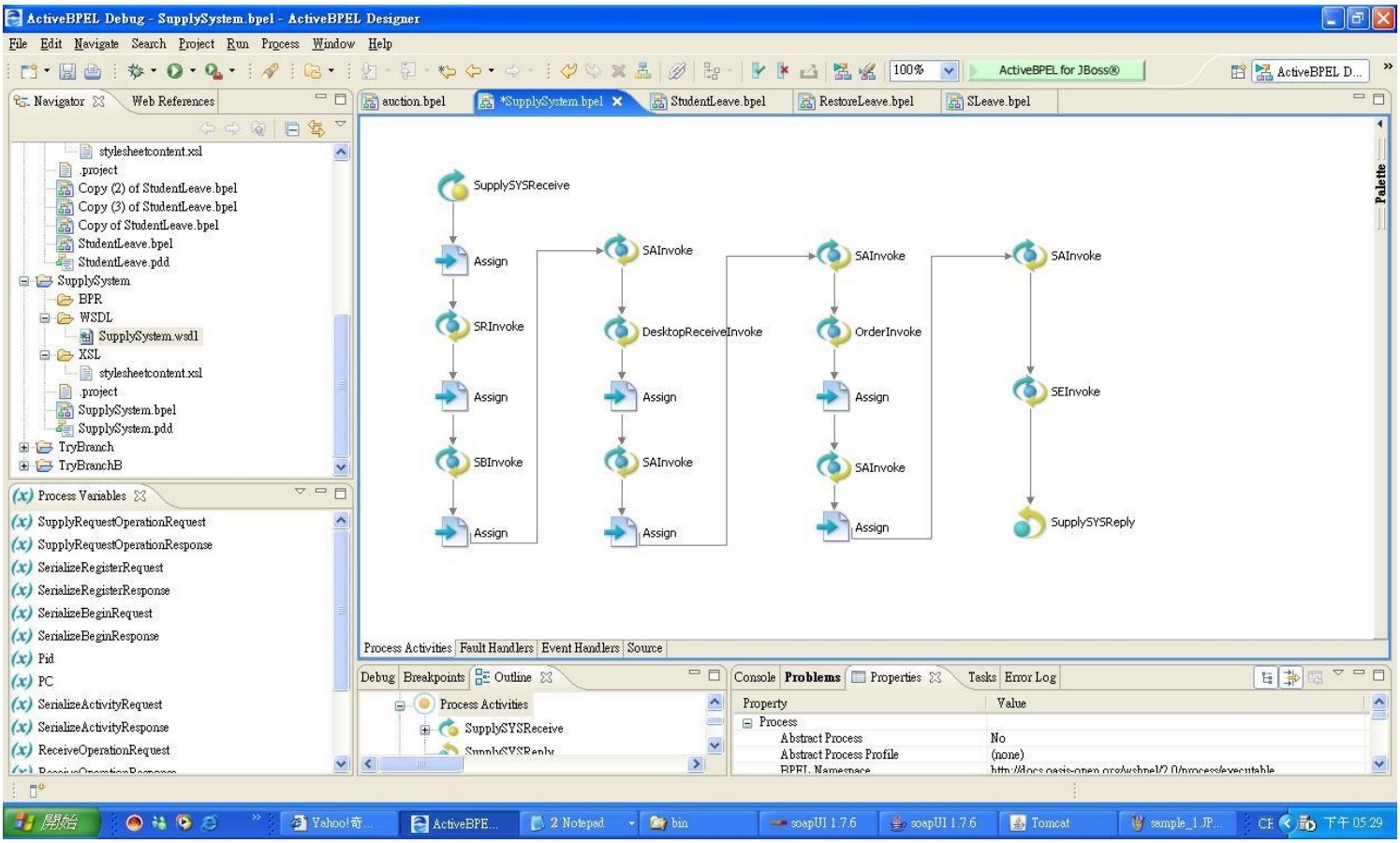
1. Berson, A, Client/Server Architecture, McGraw-Hill, New York, 1996.
2. T. Erl. Service-Oriented Architecture: Concept, Technology, and Design. Prentice Hall, 2005.
3. H.A. Reijers and I.T.P.Vanderfeesten. Cohesion and Coupling Metrics for Workflow Process Design. In J. Desel, B. Pernici and M. Weske, editors, Proceedings of the 2nd International Conference on Business Process Management (BPM 2004), Lecture Notes in Computer Science 3080, 290-305. Springer Verlag, Berlin, 2004.
4. Casati, F., Ilnicki, S., Jin, L.,Krishnamoorthy, V. and Shan, M. (2000):Adaptive and dynamic service composition in eflow.Technical Report, HPL-200039, Software Technology Laboratory, Palo Alto, USA.
5. UDDI Version2 Specifications - http://www.oasis-open.org/ committees/uddi-spec/doc/tcspecs.htm #uddiv2
6. Anis Charfi and Mira Mezini,Using Aspects for Security Engineering of Web Service Compositions，Proceedings of the IEEE International Conference on Web Services (ICWS' 05)
7. An Liu, Qing Li, Liusheng Huang, and Mingjun Xiao，A Declarative Approach to Enhancing the Reliability of BPEL Processes，2007 IEEE International Conference on Web Services (ICWS 2007)
8. Glen Dobson，Using WS-BPEL to Implement Software Fault Tolerance for Web Services，Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'06)
9. Web Services Business Process Execution Language Version 2.0,OASIS Standard,11 April 2007

10. Jim Lau, Lau Cheuk Lung, Joni da S. Fraga,Giuliana Santos Veronese，Designing Fault Tolerant Web Services Using BPEL，Seventh IEEE/ACIS International Conference on Computer and Information Science,2008
11. Radio Perlman， An Overview of PKI Trust Models，IEEE Network November/December 1999
12. Tzilla Elrad, Mehmet Aksit, Gregor Kiczales, Karl Lieberherr, Harold Ossher,Discussing aspects of AOP ,Communications of the ACM,Volume 44, Number 10 (2001), Pages 33-38
13. Pullum, Laura L.,Software Fault Tolerance Techniques and Implementation,ISBN：1580531377
14. Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001
15. Simple Object Access Protocol (SOAP) 1.1,W3C Note 08 May 2000
16. S.Bajaj, et.al.,Web Services Policy Framework (WS-Policy),September 2004
17. Web Services Security Policy Language (WS-SecurityPolicy) July 2005 Version 1.1
18. Web Services Security: SOAP Message Security 1.1(WS-Security 2004), OASIS Standard Specification, 1 February 2006
19. Hypertext Transfer Protocol -- HTTP/1.1,W3C June, 1999
20. G.H. Hwang, Y.H. Chang, T.K. Chang, An operational model and language support for securing web services, IEEE International Conference on Web Services, ICWS 2007, 9－13 July 2007,
21. Gwan-Hwan Hwang, Tao-Ku Chang, “Document Security Language (DSL) V2.0,” Technical report, National Taiwan Normal University, http://www.xml-dsl.com/DSL_Syntax_v2.pdf.
22. Herbert Zimmermann，OSI Reference Model-The ISO Model of Architecture

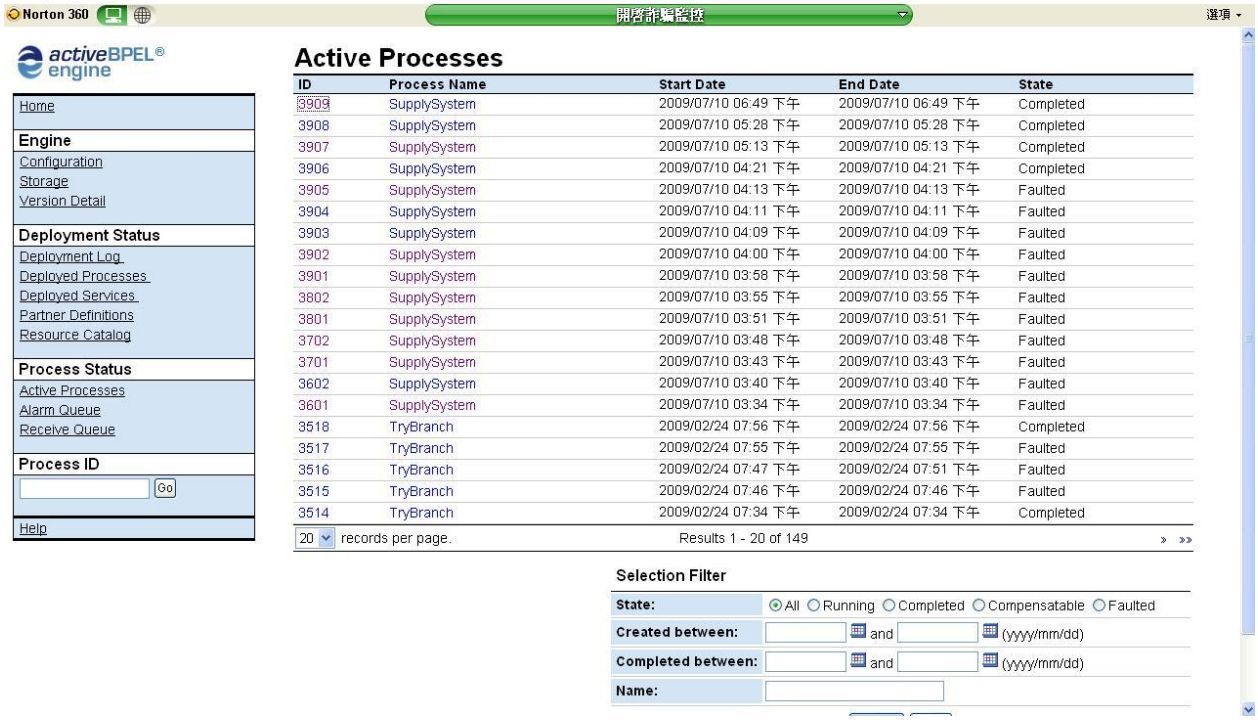
for Open Systems Interconnection，IEEE TRANSACTIONS .ON COMMUNICATIONS, VOL. COM-28, NO. 4, APRIL 1980

23. 服務導向架構 (Service Oriented Architecture) 應用專欄：服務導向架構 (Service Oriented Architecture) 應用,作者：簡西村(台灣微軟開發工具暨平台推廣處資訊平台策略顧問),2004 年 12 月,http://www.microsoft.com/taiwan/msdn/columns/soa/SOA_overview_2004112901.htm
24. 從原理面探討服務導向架構(SOA),倪文君 叡揚資訊 Architect,http://www.gss.com.tw/tw/eispage/vol52/eispage5202.htm
25. Web Services 介紹,資策會數位教育研究所講師 鄧文焯,http://www.iiiedu.org.tw/knowledge/knowledge20021231_1.htm
26. S. Vinoski, “CORBA: Integrating diverse applications within distributed heterogenous environments,” IEEE Commun. Mag., pp. 46–55, Feb.1997.
27. Troy Bryan Downing. Java RMI: Remote Method Invocation. Number 0764580434. IDG Books, 1998.
28. Chung, P., Huang, Y., Yajnik, S., Liang, D., Shih, J.,Wang, C.-Y., AND Wang, Y. 1998. DCOM and CORBA side by side, step by step, and layer by layer. C++ Rep. 10, 1 (Jan.), 18–29.

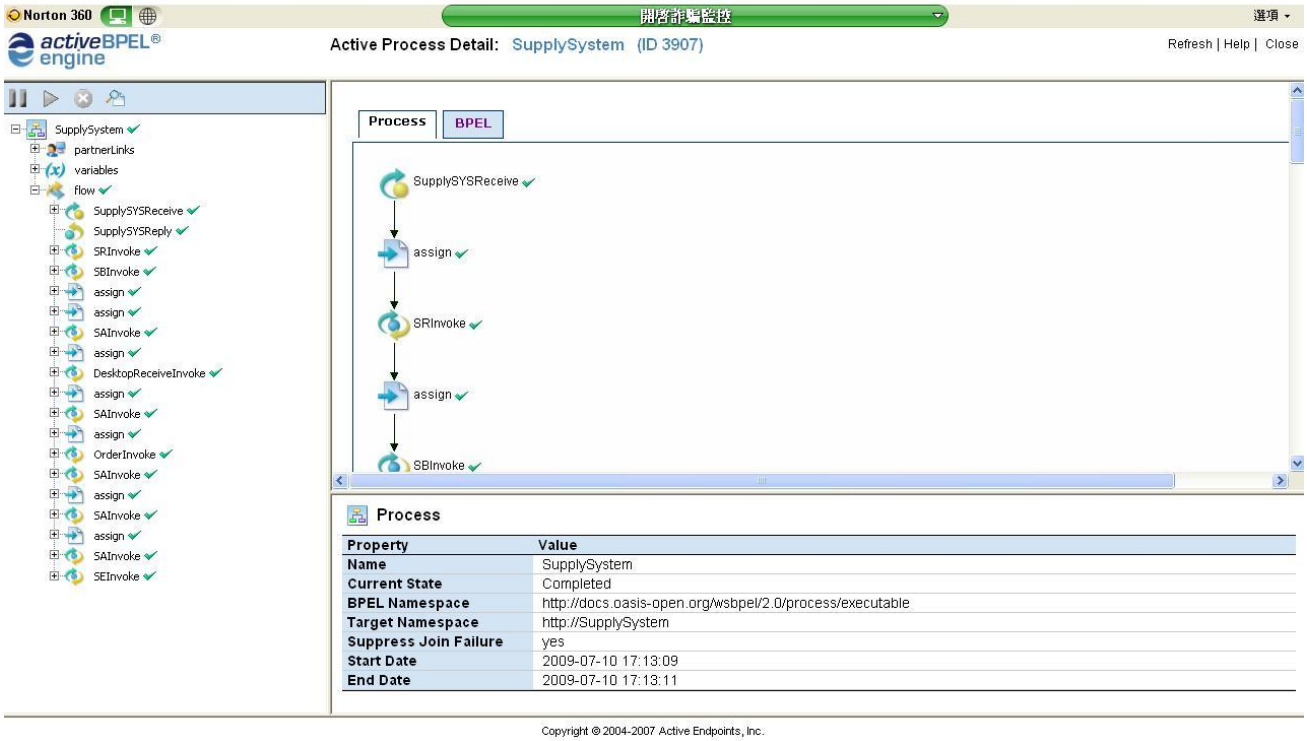
Appendix A.



BPEL 使用序列化操作模型為依據產生 BPEL’



後勤補給系統(二)工作流程 BPEL 之執行中程序(Active Process)



後勤補給系統(二)工作流程 BPEL 之程序執行過程

Appendix B.

一般補給單位及武器部門 Web Service 主要功能類別如下：

```
public class DesktopReceiveSkeleton implements
DesktopReceiveSkeletonInterface{
/**
* Auto generated method signature
* @param receiveOperation0
*/
public org.example.www.desktopreceive.ReceiveOperationResponse
ReceiveOperation(org.example.www.desktopreceive.ReceiveOperation
receiveOperation0)
{
     org.example.www.desktopreceive.ReceiveOperationResponse rtn=new
     org.example.www.desktopreceive.ReceiveOperationResponse();
     rtn.setReceivePerson("John");
     DateFormat dateformat;
     dateformat = new SimpleDateFormat("yyyy-MM-dd'*'HH:mm:ss.SSSS'/'");
     rtn.setReceiveDate(dateformat.format(new Date()).replace('*',
         'T').replace('/', 'Z'));
     rtn.setDescription("I got it");
     return rtn;
}
}
```

供應商 Web Service 主要功能類別如下：

```
public class ItemOrderSkeleton implements ItemOrderSkeletonInterface{
public org.example.www.itemorder.OrderOperationResponse OrderOperation
(org.example.www.itemorder.OrderOperation orderOperation0)
{
     org.example.www.itemorder.OrderOperationResponse rtn = new
     org.example.www.itemorder.OrderOperationResponse();
          rtn.setResult("OK");
          rtn.setDescription("Any thing is fine.");
          return rtn;
}
}
```