國立政治大學資訊管理學系

博士學位論文

指導教授：陳春龍 博士、吳忠敏 博士

派翠網路運用於建構雲端服務之研究

Using Petri Net for Cloud Computing

研究生：陳俊廷

中華民國一百年七月

摘　要

雲端運算是近來全世界最熱門的資訊議題，任何設備只要能夠連上網路，就可以享有各種不同的網路服務，這些服務可能建置在不同的機器設備上，甚至可能在相隔遙遠的雲端裡，因此也稱之為雲端服務，在操作過程中，使用者不需要擔心服務安裝在哪裡，亦不用擔心服務如何達成，可想而知網路服務的背後，隱藏著很複雜的技術與架構。

網路服務的開發方式相當地簡單且快速，為系統帶來更大的便利與彈性，不過在管理的部份，卻變得越來越複雜，例如不易瞭解網路的結構與狀態、潛在的無窮迴圈及冗餘的流程問題，甚至是在資源共享的情況下，因等待或其他因素而導致死結的情況產生。這些問題將使得網路服務組合在執行時期，可能發生無法預期的錯誤。情況嚴重時，系統可能會完全鎖死或停止，對公司將造成重大的財務及商譽上的損失。這些流程的問題，需要在網路服務組合執行前先進行驗證，本研究透過流程轉換成派翠網路的分析，以確保流程設計的正確性與穩定性。

因此，本論文運用網路服務標準，將雲端服務轉換成派翠網路模型，再以派翠網路中的針織法為分析基礎，檢驗模型的狀態，避免死結發生，並提供網路特徵報告書，以降低管理的複雜度，進而提升服務的穩定性。

關鍵字：網路服務、雲端服務、派翠網路、針織法、死結。

Abstract

Cloud computing is regarded as the most popular recent ICT topic. Any equipment which can be connected to Internet can provide its user have the access to the various services, which may be built on different machines, or even may exist in the distant 'clouds' far away. However, in the operating process, the user has no need to worry where the service locates, needless to say how the service operates, which means that a considerably amount of data and techniques hide behind the Web service.

Web service is an artifical piece of art, the user and the manager can concatenate several Web services online into a bigger one according to the conditions they need, which is a considerably simple and fast developing method bringing more convenience and flexibility for the system. Nonetheless, the managerial part also becomes more complex in problems like potential infinite loops and abundant procedures, for instance. What is worse is under the circumstance of resource sharing, the deadlocks happen while pending or other factors occur. These issues will lead to severe errors while the Web service compound operates. If the system completely locks and stops, serious financial damage and loss of commercial reputation will be caused to the company. Hence the design of procedures needs to be validated and transformed as Petri Net analysis before the Web service compound operates in order to assure the accuracy and stability of the procedure design.

Therefore, this paper uses Web service standards, turning Web services into Petri Net models with knitting technique as analytical base, to validate the accuracy and stability of the model to avoid deadlock to happen in order to enhance the reliability of the service.

Keywords: Web service, Cloud services, Petri Net, knitting technique, deadlock.

誌　謝

終於完成論文了，首先感謝我的兩位指導老師，吳忠敏老師和陳春龍老師，在我的學習過程中，從碩士班到博士班的辛苦指導，一直不斷地幫助我、鼓勵我，讓我有足夠的動力堅持到最後，也很謝謝趙玉老師在修課與投稿部份，給予我這麼多的幫助與指導，另外要特別謝謝口試委員林我聰老師、季延平老師、耿慶瑞老師的寶貴建議，和老師們的諄諄教誨，論文才能順利完成。另外也要感謝系上曾經指導與照顧我的余千智老師、楊建民老師、湯宗益老師、周宣光老師、管郁君老師、苑守慈老師、張維中老師、曾淑峰老師、謝明華老師和經濟系的陳樹衡老師，還有系辦的詩晴助教、雨儒助教。

要感謝的人很多，要特別感謝從大學時代就一直指導我的世新大學資管系主任陳育亮老師、創新與產學中心主任廖鴻圖老師，還有世新的長官與同事們，計算機中心主任葉炳倉老師，巧菱、阿富、秀芬老師、心潔、惠雯、美娜、威伯，教務處陳麗卿組長、英語系李玉華老師、台大經濟系的吳聰敏老師。政治大學博士班的同班同學、同門的展勝學長、俊龍學長、孝慈，此外還有婷妤學姊、寶弟、豬毛、映曄、麗旭、于真，遠在美國的小 p 和雅雅、阿匡學弟、廣寧、元佳、宇軒、雁予、昀瑾，感謝你們在我最需要幫忙的時候，給我最大的支持和鼓勵，真的謝謝你們。

要感謝的人真的很多很多，最後要特別感謝我的父母親與我的太太淑美、佑佑和怡君，還有在天國的弟弟俊成，謝謝你們在我最辛苦的時刻，體諒我、幫助我，讓我能專心於課業，感謝你們付出的一切，也希望這輩子能回報你們給我的恩情，感恩。

陳俊廷 謹誌

于國立政治學資訊管理學系博士班

中華民國一百年七月

目　錄

圖目錄

表目錄

第一章　緒　論

網路服務(Web Services)近年來發展迅速，其運用的分散式應用程式架構與服務導向架構(Service Oriented Architecture; SOA)，讓網路服務能在標準架構規範下快速整合。結合目前雲端環境，使資訊系統的服務能快速的建置，提供多樣化的服務。也因為服務需求快速成長，服務數量急速增加，造成服務與系統的管理上變得非常複雜，因此，本論文提出一個轉換與分析的架構，透過系統的實作，解決服務與服務組合後所產生的問題。

本章針對論文的動機與目的進行介紹，其中共分為四節，第一節針對研究背景與動機進行討論，說明網路服務的發展與其衍生的問題，第二節則對本研究之目的進行探討，並擬定出研究方向與架構，第三節介紹本研究之範圍與步驟，最後，第四節介紹研究架構。

1.1 研究動機

近年來，由於全球化和服務導向系統的發展趨勢，企業面臨內部及外部龐大的服務需求，必須尋求一種更好的解決方案，因此，正確快速與彈性便成為企業系統建置時追求的目標，此外，服務導向系統架構與網路服務等理論與技術，也在近期蓬勃發展，例如 Microsoft、IBM、BEA 等軟體大廠，都積極推動此技術架構的應用。

網路服務的應用，可加速系統的開發，並增加系統彈性，使得企業可以快速地適應新環境(Cherbakov et al., 2005)。以開發資訊系統的角度來看，網路服務的架構，是一種可以建置在任何作業系統上，且可透過網路連結的一種應用程式，簡而言之，網路服務的概念是將需求或功能，包裝成單一個服務，再透過統一的

介面進行操作，其好處是能夠讓系統方便快速地抽換元件，或是依照環境需求加以重組，讓企業能夠快速因應外部環境的變化，以組合的方式建置系統(Chiu and Yang, 2001)。如此不但可縮短系統開發時間，也可以減少重覆開發資源浪費的問題(Wang and Stroulia, 2003)。

但在網路服務發展的過程中，也面臨許多問題需要被克服，這些相關問題可歸納為安全性問題、整合性問題以及語意傳達性問題(Wang et al., 2004)，其中整合性問題會衍生出網路服務的管理問題，如先前所言，網路服務是將許多不同的功能，重組包裝成一個新的服務。因此，如何有效地整合與管理網路服務，便成了網路服務應用時重要的議題。

1.2 研究目的

網路服務的概念與架構是很簡單的，但實際上，在服務組合完成後，進入執行階段時，問題就會變得越來越複雜。在目前的網路服務架構中，有三種不同的角色，其中包含(1)服務提供者，(2)服務的需求者，以及(3)服務註冊管理者，它們之間的關係，並不一定屬於同一個組織。簡而言之，服務提供者就是提供其所能執行的服務；而服務註冊管理者，是管理服務提供者所註冊的各種服務；服務需求者可以透過搜尋的機制，尋找其所需要的服務再進行連結；因此，搜尋的結果，如果可以找到很多堪用或是相似服務，是個很好的結果，但要嚴謹地判斷是否為適當的服務，甚至是否為好的服務，可否順利的執行，及連結後會不會產生冗餘的流程，而造成執行效率不彰。除此之外，公開的服務，本身就可以視為一個可分享的資源，因此有很多的資源，很可能會處於一個或多個執行中的狀態，如此可能造成處理程序因等待資源而鎖死，而變成難以預期的結果。

綜合以上所言，在眾多的服務的情況下，服務的組合與使用，在企業內部應如何管理，是相當重要的問題，因此，本論文將解決當前的幾個議題：

(1) 服務合成後自動轉換成派翠網路模型，以利後續分析與管理。

(2) 找出服務組合中冗餘的流程，提升效率。

(3) 取代人工分析網路，系統提供分析後資訊，提升管理效能。

(4) 以有效率的演算法，找出服務組合中隱藏的死結。

因此，本研究希望提出網路服務自動轉換機制，轉換後，提供圖形化的網路服務架構圖，以方便管理者檢視，並且自動檢驗網路結構中，是否有特殊結構或隱藏死結的存在，以提升系統檢驗效率。過程中需先將雲端中的網路服務，轉換成派翠網路，再針對派翠網路進行分析，最後針對結果提供分析建議書。

1.3 研究範圍與步驟

本論文之研究範圍，包括企業內部與外部的服務流程，在處理過程中，先針對流程中的服務進行拆解與分析，流程部份是使用標準的目錄伺服器上所提供之服務，針對服務流程的組合，進行分析與轉換，其中分析與轉換的基礎，都使用網際網路上的標準規範，利用網路服務商業流程執行語言(Web Services Business Process Execution Language; WSBPEL)進行分析轉換，再針對轉換後之網路結構進行進一步的分析。

因此，本論文之研究步驟，從蒐集文獻開始，再針對雲端運算、網路服務，網路轉換、化簡驗證與死結判斷等模組進行整合，建置一個實作系統，最後再針對案例進行模擬，研究流程如圖 1-1 所示。

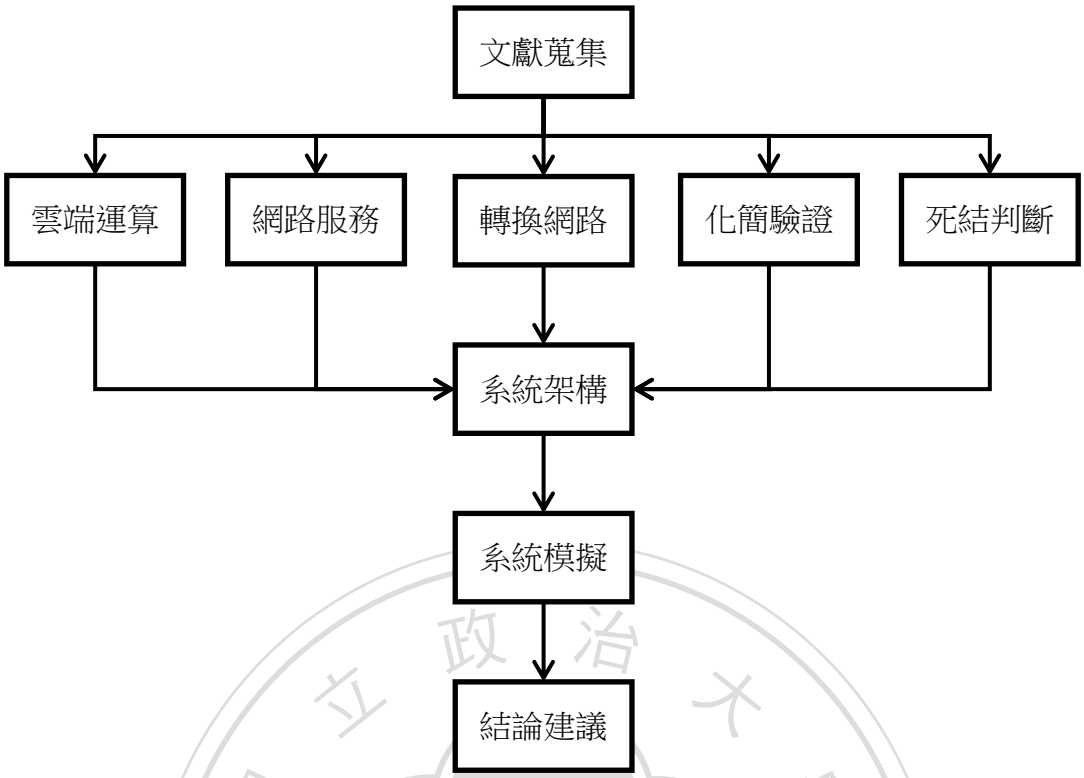

圖 1-1 研究步驟架構圖

1.4 研究架構

本研究架構總共分為五個章節來進行討論，以下將依序進行介紹：

第一章 緒論

本章針對本研究之動機與目的進行介紹，並將本研究之主要問題進行釐清，以利閱讀後續章節時，可更清楚瞭解相關重點。

第二章 文獻探討

針對本研究所探討的主題，進行文獻的蒐集與討論，其中包含網路服務、本體論、語意網路、網路服務組合、雲端運算、派翠網路與死結處理。藉此介紹內容，瞭解技術與問題之相關性。

第三章 研究方法

本章依據第二章文獻探討延續，進行研究方法的分析與規劃，其中包括網路服務轉換成派翠網路的規則、結構化活動的處理、網路轉換演算法、派翠網路中針織法與編織的規則，並分別實作成數個網路服務，以方便後續移植至其它系統使用。

第四章 系統實作與結果

本章以實際之網路服務程序，使用網路服務商業流程執行語言輸入系統，進行實證測試，以驗證本研究之可行性。

第五章 結論與建議

總結前述各章對本研究之討論，並提供研究結果、研究貢獻與未來延伸研究之建議。

第二章　文獻探討

本章依據研究的動機與目的，提出幾個重要部份，進行相關文獻的討論，第一節是介紹網路服務，第二節介紹服務組合流程中，重要的關鍵技術，包括語意網與本體論，第三節介紹網路服務組合的定義、採用網路服務組合之目的，以及網路服務組合的效益為何，第四節介紹雲端運算之環境與問題，第五節介紹派翠網路，最後一節介紹派翠網路中死結的判斷。

2.1 網路服務

網路服務最初的發展目的，主要是用來幫助電腦與電腦之間資料的傳遞與展現，不論是文字、圖形或多媒體，這些功能使人們得以快速地閱讀新聞、查詢天氣、分享資訊或是訂購商品等等。但是這種以文字或是圖片為傳遞基礎的網路，本質上是無法讓電腦軟體或系統間，自動地傳遞訊息，因此，一種透過網際網路，讓分散在各地的應用程式或系統，能夠自動彼此互相溝通服務的想法就應運而生。根據全球資訊網協會(World Wide Web Consortium; W3C)對網路服務的定義：「一個應用程式可經由延伸標記語言(eXtensible Markup Language; XML)來描述或查詢，透過標準的資源辨識規則，此應用程式的介面與連結方式，可以被完整的定義，且支援其他應用程式與系統，藉由延伸標記語言形式的訊息，再透過網際網路的協定來直接驅動。」簡單來說，就是很多功能性元件，透過特定的溝通管道相互配合，在需要時互相溝通，以提供適當的服務。

使用網路服務模式，將可以讓應用程式之間的溝通更為容易，並且透過標準規範的方式，可以讓分散在不同地點的網路服務，透過網際網路整合在一起，形成一個大型的資訊系統。與以往侷限於單一主機上的資訊系統比較起來，這樣的

分散式服務的方式，可以使得主機的運作負荷減輕，並使服務形式更加多元化。以軟體開法的角度來看，每一個單一的網路服務，均可視為一個軟體元件，在需要服務的時候，不需要重複開發既有的元件，因此可以快速地連結並提供服務，還可降低軟體開發成本。

接下來，首先要先定義網路服務的概念與架構，網路服務是由服務的提供者、服務的需求者與服務的管理者，三個不同的角色所組成的(Curbera et al., 2002)，網路服務架構圖如圖 2-1 所示：

(1) 服務需求者：需要某項系統服務的服務要求者，其會透過網際網路連線到網路服務管理者，找尋適合的網路服務。

(2) 服務提供者：服務提供者會將所提供的網路服務，包括相關的詳細資訊、使用的位置、方法等資訊，註冊在網路服務管理者系統當中，並且提供服務及服務本身的執行環境規格，給服務要求者。

(3) 服務註冊管理者：提供一個公開且標準的機制，讓所有提供服務的網路服務提供者，可以登錄它們所提供的網路服務相關資訊，並且讓需要服務的網路服務需求者，可以透過此機制找尋適合的網路服務。

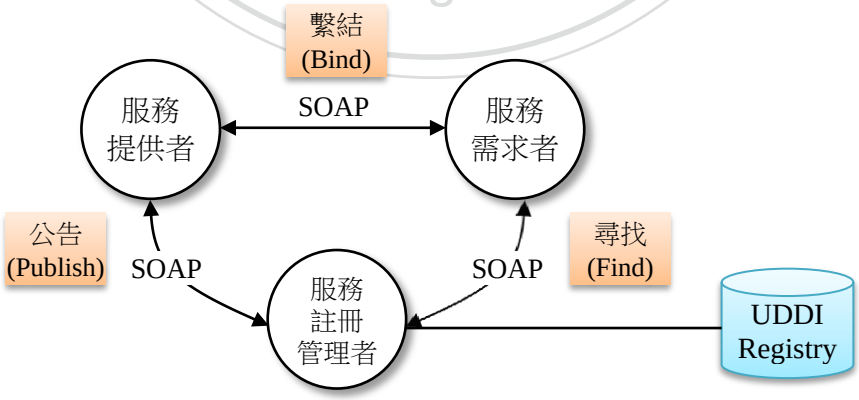

圖 2-1 網路服務架構圖

網路服務結合了跨平台執行的特性，和網際網路分散架構的概念特性，使得在網路上資料的傳遞、展現與應用，變得非常容易，並且在技術性與相容性上也有相當程度地改善。透過網路服務的方式，使得不同服務的系統平台、硬體或是軟體能夠彼此溝通，服務之間也能自動地運作。

2.1.1 網路服務的特性

網頁的技術標準，是網際網路發展過程中，影響相當深遠的一種技術，其規範出一種在網際網路上資源相互連結與共享的機制，讓網路上的溝通與連結更為方便，除了改變生活與工作的模式外，也帶動整個網路上的商務發展。雖然此技術，可以透過網頁瀏覽器來操作，達到標準且一致的瀏覽介面，但是這僅僅只是將資料呈現在網頁上，並不能讓不同的系統間，進行資料的存取與交換，因此系統與系統之間的「資料存取介面」問題還是存在。

網路服務是一種網路上可提供服務的資源，處理的過程中，會透過標準資料存取介面的方法，讓所有在網路上的程式，能透過標準的協定，夠彼此相互溝通。若將網路服務導入企業中，更可以降低系統整合與軟體開發的成本，並且能夠有效地促進商業運作流程的效率，而且其採行目前廣泛使用的標準資料傳輸介面，比起以往的商業流程整合，或是電子資料交換(Electronic data interchange; EDI)機制來說，在系統整合與傳輸模式上，更是簡便有效率。

網路服務是使用標準的網際網路技術，以超文件傳輸協定(HyperLink Text Makeup Language; HTTP)為傳遞資料的通訊協定，並以延伸標記語言為資料傳輸的標準格式，採用網路服務軟體元件，來整合商業流程的運作，過程中，服務需求者並不需要對這些技術底層，是如何去傳輸、呼叫與運作，只需要瞭解如何透過標準介面，呼叫網路服務，取得所需要的服務即可。

網路服務還可用來解決分散式或異質系統的整合問題，將傳統的應用程式介面方式，提升到與底層技術無關的層次，並且使用延伸標記語言來定義標準的資

料傳遞規格，以及配合網路服務描述語言(Web Services Description Language; WSDL)，來描述網路服務運作時，相關的輸出與輸入資料和服務呼叫的方法。

簡易物件存取協定(Simple Object Access Protocol; SOAP)是配合網路服務的運作架構，所制訂出來的標準通訊協定，能夠讓程式與程式之間，透過標準的簡易物件存取協定方式，存取它遠端的相關服務。除此之外，由於簡易物件存取協定，採行延伸標記語言的標準語法，因此，除了應用簡單之外，也可以增進資料通訊的傳遞效能，使得系統開發人員在建置系統的過程中，可輕易地整合不同的平台，來實作多種不同的系統功能。

除了上述幾項特性之外，網路服務對於整個軟體開發與整合上，所帶來的效益也是令人矚目的，網路服務可以讓不同種類的系統間，以有效率的方式來進行整合，更能提升資訊系統對於企業內部及外部的協調處理能力。

2.1.2 網路服務技術

網路服務技術是定位在「機器與機器之間溝通的橋樑」。從技術的觀點來看，網路服務就是一種以網路為基礎，將物件封裝與元件模組化技術，運用延伸標記語言等相關的技術標準，它能夠輕易地在網站與網站之間進行溝通，其中包括服務自我規格描述(description)、服務的發佈(publish)、服務的搜尋(search)以及服務的取用(retrieval)等自動化功能(葉俊仁, 2002)。這樣運作架構的好處，在於可以大幅降低企業或組織內、外部系統間服務整合的困難度，而且更可以為企業與企業間系統的溝通，提供一個具有高度彈性的模式。

網路服務架構最大的優點，是它突破了以往系統整合與開發時，應用系統與程式元件間緊密耦合的運作架構模式，而是採用耦合鬆散的方式，來建構網路環境的資訊系統。以耦合鬆散的方式來建置網路服務的環境，其最大的好處是讓應用程式的系統整合工作，不再限定於特定的作業平台、特定的資料結構，與特定的合作伙伴，而是更具彈性與空間。此外網路服務對於解決網際網路上異質作業

環境的系統整合，也很有成效，透過網路服務可以讓應用程式透過網路或任何的平台與裝置，使用多元化的服務，或與其它的系統協同運作。

維持網路服務的運作，需要配合幾項以延伸標記語言為基礎的相關技術，這些技術利用標準化方式進行溝通，因此也增加其通透性。以下針對這幾個相關的技術，做進一步的說明。

(1) 延伸標記語言：是整個網路服務建構的基礎，由全球資訊網協會，所制訂公佈的一項標準規格，它的作用在於制訂出一套標準的資料傳輸，與資料定義的規格，透過這樣的標準規格，網路上的應用程式與程式之間，才能夠依此標準傳輸與取用資料。

(2) 網路服務描述語言：網路服務描述語言，主要是描述網路服務的細節，是使用延伸標記語言格式為基礎之技術，讓網路服務應用程式，能以一種標準方法來描述自己，擁有哪些功能與使用方法，並且定義網路服務的介面、資料與訊息的型態，以便讓互動更容易進行。例如藉由網路服務，把遠端執行的程式、函示等，當成本機端的資源來執行，關鍵就在於網路服務描述語言，網路服務才能夠啟動。

(3) 簡易物件存取協定：簡易物件存取協定，是一種提供給網路服務的標準通訊協定，這仍然是以延伸標記語言為基礎，所發展出來的技術。簡易物件存取協定的目的，就是讓程式與程式之間，能自動地相互溝通，但兩者之間，不需要知道彼此的作業平台是那一種，或是兩邊各自如何實作等細節，只需要透過網路服務描述語言，瞭解互動模式如何操作即可。例如：電子郵件是藉由簡單郵件傳輸協定(Simple Mail Transfer Protocol; SMTP)的標準傳送資料，在一封電子郵件中，除了文字外，也定義了簡單郵件傳輸的協定內容，如果欲將此郵件傳送出去，則郵件的封裝格式，必須是符合簡單郵件傳輸協定看得懂的格式，才能夠傳送成功。

(4) 通用描述、探索與整合(Universal Description, Discovery and Integration; UDDI)：通用描述、探索與整合，指的是一種提供網路服務註冊與搜尋的機制，其架構也是以延伸標記語言為基礎，其主要的目的為提供企業或組織，註冊通用描述、探索與整合，告知其他人提供者有提供網路服務，並尋找其所需要的服務。因此通用描述、探索與整合的功能也類似目前常見的電話簿，或是稱為黃頁的功能，目的就是要提供註冊與查詢服務使用者，可利用的網路服務有哪些，以及服務的位置在哪裡。

2.1.3 網路服務管理議題

學者 Younas 等人認為，當單一網路服務無法滿足使用者需求時，就需將多組服務加以整合(Younas et al., 2005)，學者 Limthanmaphon 和 Zhang 則認為動態整合服務，較能滿足使用者需求(Limthanmaphon and Zhang, 2003)，除此之外，學者 Berardi 指出使用者需求的服務，有時必須把服務加以整合後，才能得到完整的功能(Berardi, 2003)，然而整合本身卻具有許多需要被考慮的議題，且需要逐一地克服，學者 Medjahed 等人就強烈建議，如果語意沒有被詳細表達，服務整合就會有困難(Medjahed et al., 2003)。然而語意的傳達中，依現在環境來看本身就有困難存在，但仍可透過其他方式提供相似的服務。

學者 Rao 和 Su 曾經針對整合網路服務的方法進行整理，文中提到整合網路服務是複雜度相當高的工作，由於網路服務的整合，是建構在需求者計畫進行的流程，於是將複雜的原因歸納為三點(Rao and Su, 2004)：

(1) 可使用的網路服務，在近年來數量有明顯的成長，因此透過通用描述、探索與整合搜尋時，所找到的服務數量也相當可觀。

(2) 網路服務可以隨時被創造與上傳，負責管理的通用描述、探索與整合，也必須能即時的運作，並同步更新相關的資訊。

(3) 網路服務可以由不同的單位或組織提供，而這些單位或組織之間，並沒

有特別的邏輯概念，來創建這些服務，亦沒有規範統一的語言或方法來解釋服務的涵義，因此可能有同樣的服務，但有不一樣的解釋，或是不一樣的服務，但有相同的解釋。

面對這些問題時，若以自動或半自動的方式，將服務進行整合，是相當重要的工作。現行主要的方法是利用工作流程進行整合，或是以人工智慧的方式進行規劃(Rao and Su, 2004)。以工作流程為整合方法來說，適用於已經有明確定義流程的模型，但自動化系統的功能，需要自動地找到服務，來滿足各式各樣的需求。人工智慧方法，較適合用於沒有先行定義程序模型的情況，但亦需要使用者提供一些限制條件或是使用的偏好。人工智慧的方法，泛指一些分類方法，而這些方法的主要目的，在於先針對服務自己內部的一些條件，進行上下文脈(Context)的規劃。同樣的針對上下文脈，學者 Maamar 利用上下文脈的概念，將本體論應用於網路服務，並指出上下文脈的方法，對服務表現與使用者需求滿足的重要(Maamar, 2005)。

在服務多元化的情況下，許多研究也針對情境感知(Context Awareness)與語意網等議題進行探討，試圖從中尋求網路服務管理問題解套的方式，資訊過量的問題其實也與 Tim Berners-Lee 所提出了語意網(Semantic Web)的概念相符(Berners-Lee, 1998)，由於服務不斷地被創造與提供，最終的目的，就是為了提供使用者更好的服務品質，因此問題關鍵，在於系統如何提供好的、正確的服務。好的服務必需依情境調整，提供適當的服務，這就成了重要的關鍵，傳統的系統開發流程，是從使用者需求分析做起，再建立整體系統之規劃；但如今網路服務的開發環境，服務提供者將特定功能之元件，包裝成網路服務後，透過連結就可以在雲端使用，但要如何連結使用，目前並無一定的規範，僅依據使用者提出的需求整合而成，但如何滿足使用者需求就無法得知。若要知道外界情境，就必須從前後關係進行分析，語意網是一個可行的方式。

2.2 語意網路服務

2.2.1 語意網的介紹

語意網的概念，是將網路上有意義的內容結構化，藉由共享的、通用的知識本體建置，使得網路上的資源及服務，更容易取得和分享(Hendler, 2001)。在此環境下，提供了智慧的方式存取異質性(heterogeneous)與分散性的(distributed)資訊，使得代理人系統有能力，處理使用者之間的需求，以及處理可利用的資訊資源(Fensel, 2000)。目的是希望在機器可以理解的情況下，各種自動化服務可以幫需求者達成目標。

全球資訊網協會所制定的語意網的階層架構，此架構亦說明了網路技術與標準，在語意網各階層中執行的原則，語意網的階層架構如圖 2-2 所示。各階層的說明如下(Guarino, 1998; Hendler, 2001; Horrocks and Patel-Schneider, 2003)。

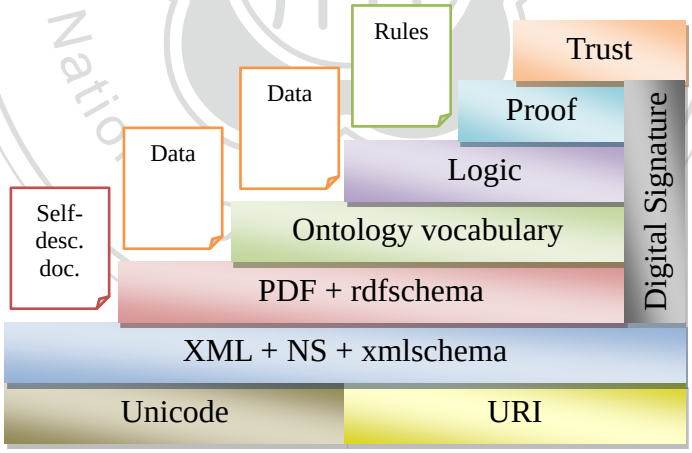

圖 2-2 語意網階層架構圖

資料來源：Berners-Lee, 1998

(1) 萬國碼(Unicode)與通用資源標識碼(Uniform Resource Identifier; URI)階層，是確保在語意網環境處理文字時，可以使用國際通用的字集(international characters sets)，即可支援全世界主要語文的文件，進行交

換、處理或是顯示，並透過通用資源標識碼，提供識別物件，或是標示出資源在語意網中的位置。

(2) 延伸標記語言層、命名空間(namespace)和延伸標記語言綱要(XML Schema)的規定，是以機器能理解的延伸標記語言編碼為基礎，以確保各知識本體的整合性。

(3) 資源描述框架(Resource Description Framework; RDF)與資源描述框架綱要階層，是用來定義網路上資訊的架構，使用資源描述框架所提供的表達技術，可以使電腦能夠迅速處理詞彙和概念的意義，以描述資源或是資源與資源之間的鏈結型態；除此之外，使用通用資源標識碼來標示實體、概念、特性和關係。

(4) 本體論是一種以資源描述框架之類型的語言寫成的文件，文件中，概念之間的關係和推理邏輯規則，都已經清楚地被定義完成。透過網頁指向的本體連結，電腦將能理解網頁上語意資料的含意。

(5) 邏輯(Logic)、證明(Proof)和信任(Trust)是目前許多研究正在進行的階層。邏輯階層所賦予撰寫規則的能力，證明階層主要是用來執行這些規則，而信任階層則是需要一個授信的機制，來幫助整個過程的評估，是否要信任證明層的結果。

(6) 數位簽章(Digital Signature)階層，是作為偵測各種的文件，用以確保其安全性的階層，是未來能否實現可信任網路的關鍵技術之一。

其中本體論是建立整個邏輯架構與關係規則的部份，而這部份在(Bada et al., 2004; Li et al., 2004; Arpinar et al., 2005; Grau et al., 2006)等學者的研究中有說明，用於建構相關的領域概念。

2.2.2 本體論定義

本體論一詞，最初使用在哲學領域中，表示「存在的且有意識的實體或是主體」之意，後來延伸應用在人工智慧領域方面，表示用來描述與說明特定領域下的概念與知識，藉以分享與表達該知識領域中，存在的事件與彼此間的關係。

學者 Gruber 指出，本體論是一個對特定詞彙，具有邏輯性的解釋，亦即世界中特定部份概念化的表稱(Gruber, 2000)。當使用本體論來描述特定領域的知識時，可以把本體視為是概念(Concept)、物件(Object)或是類別(Class)、屬性(Attribute)、特性(Property)或是角色(Role)、實例(Instance)與關係(Relation)這些元素的組合，以下分別說明這些元素：

(1) 概念：概念就是以多個底層物件，所組成的描述範圍，也就是由多個字彙(Vocabulary)所組成的集合，這個集合可以作為一個概念性的描述，描述出主體的基本範圍，透過這個字彙的集合，能讓系統可以自動地瞭解到定義概念所代表的意思。

(2) 屬性:屬性可以當作是該物件的一個描述,描述該物件的特性或是特徵，當使用本體論表達某個特定知識領域時，概念就是其中的子集合，這些集合可以將它看成物件，在物件與物件之

(3) 間會有各種關係存在，而且每一個物件本身，也會有各種屬性存在。實際上，物件擁有屬性所建構出整個本體論的資料架構，在應用上將提供更多元且有用的訊息，不但可得知概念與其它概念之間的關係，從單一的概念也可得知概念本身的特性或重要性,另外,若是要將一個本體論，作充分有效的利用，屬性對於訊息的多樣化，是最有幫助的項目。

(4) 關係:若只使用物件與屬性來描述特定知識領域內,其概念與結構時候，在知識領域的表達上來說，還不足以提供物件之間相關的細節訊息，所以當建構出整個本體論的架構後，除了清楚地描述出物件與物件屬性之

外，還可以為這些物件定義其彼此間所有的關係。由於目前自動建構本體論的技術尚未成熟，所以大都是透過領域專家，以人工方式提供一個有系統的知識領域架構，這個架構可以用來描述整個領域中，抽象的結構與關係，提供相關應用系統的使用。

(5) 實例：實例可以更清楚地表達上層的概念，因此實例與上層概念，通常會存在著某種特殊關係，並繼承某些上層概念的屬性，除此之外，實例也可以擁有自己更詳細的屬性，以表示實例本身與其它實例之間的差異。此時，也會產生一個問題，就是在建構本體論的過程中，對於一個項目，如何定義它是概念或是實例。根據 Maedche 的建議：「實例是用來更清楚地表達概念」，所以通常在本體論架構中，用最底層的部份來定義實例(Maedche, 2003)。

圖 2-3 說明了一個簡單的本體架構，其中最大概念的階層是「東西」，接下來分成「動物」和「植物」，再依此兩類進行細分，動物或植物可以再依特徵細分，可以細分到最後的實體，甚至還可以再繼續細分。

而這些概念與概念之間，分支後又有分支，彼此存在著特殊的關係，圖 2-4 則為學者 Sensoy 和 Yolum 針對服務所建立的基本本體架構，透過此本體說明可以觀察到，需求者在使用服務時，可能會面對的各種不同的情境(Sensoy and Yolum, 2006)。由上述可得知，利用本體論可以簡單的針對這些知識領域分類，定義與描述該特殊領域的專業術語、項目與彼此之間的關係，也能瞭解特定領域裡，所擁有的元件、關係、屬性。

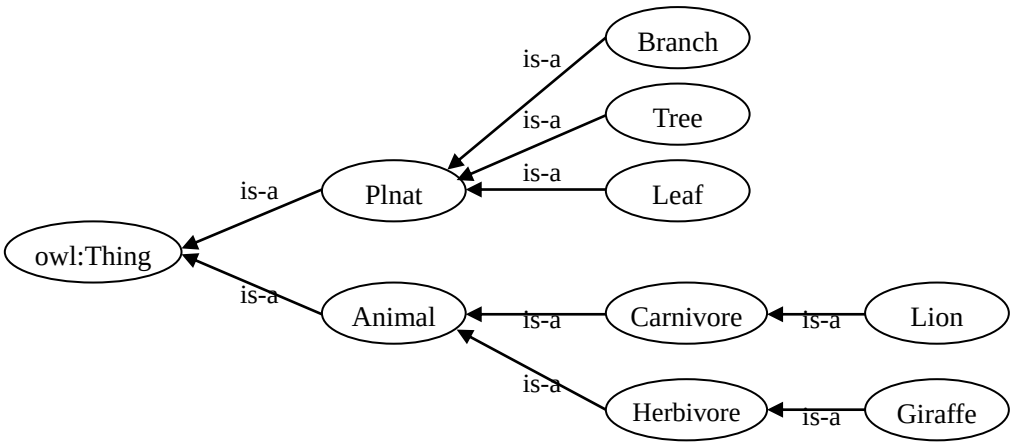

圖 2-3 本體架構圖

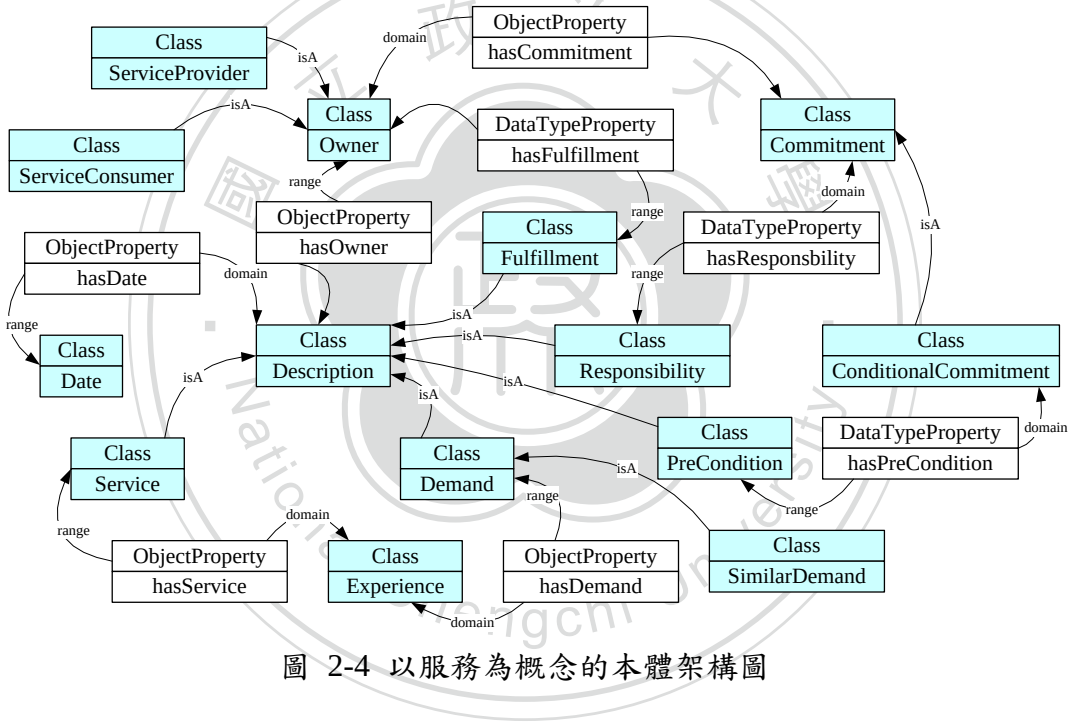

圖 2-4 以服務為概念的本體架構圖

將語意網的概念，結合到網路服務系統中，是可行的構想，特別是網務服務組合自動化，將會有很好的效果，但目前語意網的建置與使用，在網路服務相關領域的應用尚未普及，因此，本研究繼續針對網路服務其他的方式再進行探討，其中，實作方面較成熟的技術為網路服務組合。

2.3 網路服務組合

2.3.1 網路服務組合定義與目的

網路服務組合是取得現有的服務，並且結合這些服務，形成新的服務能力(Marton et al., 1999)，也就是將現有的網路服務加以組合，可以解決更複雜問題的網路服務，這也讓網路服務可以應用在企業流程。學者 Zeng 指出，以流程為基礎的網路服務組合，是一種新興的服務方式，可運用在組織內，或跨組織的自動化企業流程方法(Zeng, 2003)。以流程為基礎的網路服務組合，被認為是有效地整合分散式、異質平台、獨立的應用程式的一種方式，構成網路服務組合的特性，區分為三種類型(Zeng et al., 2004)，其中包含基礎網路服務、組合網路服務網路服務社群，說明如下：

(1) 基礎網路服務：單獨的一個網路服務，不需要依賴其它的網路服務。

(2) 組合網路服務：聚集數個其它的網路服務組合而成。

(3) 網路服務社群：一群具有相同功能的網路服務集合，但是可能具有不同的屬性，這些屬性是和功能是不相關的，例如：有同樣的功能，不同的提供者、不同的服務品質等等。

利用網路服務組合管理複雜的交易流程，是企業對企業(Business to Business; B2B)協同運作的方式(Benatallah et al., 2003a)，最好的情況下，能根據環境條件的變動，系統具備自我調整的能力，並且盡量減少人為的介入(Casati et al., 2000)。網路服務組合之目的，是讓企業與企業之間的流程，進行無縫隙的整合，這和交易的生命週期，具有相同的型態，並且能讓其它的網路服務使用；網路服務組合的另一項目的，就是要讓具有企業流程功能的網路服務，能夠藉由發佈的程序，讓更多的應用程式連結使用(Snell, 2001)。

當個別的服務，受限於它們所提供的功能時，就必須以網路流程的方式，重

新組合現存的網路服務，以建立新的服務功能(Chandrasekaran et al., 2003)。網路服務組合是一個包含許多組成的結構。它聚集了多個基礎網路服務或組合的網路服務，並且根據流程模型，進行彼此互動(Zeng et al., 2004)。

學者 Lehmann 提出，網路服務組合的目的，有以下四點(Lehmann, 2003)：

(1) 藉由提供加值性的整合服務，來達成與其它企業的結盟，而加值性的整合服務，是重新結合現有的網路服務而獲得的。

(2) 現有的網路服務，功能的延伸及再利用。

(3) 網路服務的數量增加。

(4) 支援 e 化服務組合的規劃、定義和實作。

網路服務組合對於系統開發來說，是很重要的基礎。傳統的系統開發是從無到有，必須依據系統開發流程，從需求訪談、系統分析與設計開始，一個元件一個功能辛苦地開發，而現今的方式是藉由現有網路服務的搜尋，配合特殊需求功能的服務開發，共同組合成一個新的系統。這樣的開發方式不僅可以減少系統開發的時間，並且可以增加網路服務的再利用性。

網路服務的組合，當需求者和單一的網路服務互動時，很難或甚至不可能達到服務功能的價值(VanderMeer, 2003)。如果系統可以在單一的入口，呈現相關的服務給使用者，並且依據使用者的需求，產生和執行組合計劃的能力，這樣的系統所提供的價值，明顯地超過個別網路服務所提供的價值。

因此，網路服務組合的必要性，在於它不僅可以將企業的流程，依據新的需求自動化產生，並且能快速地提供服務。

2.3.2 網路服務組合方法

目前已經有許多研究針對網路服務組合的相關做法，進行分類與討論(李智

偉，2004)，包括：網路服務組合的階段、組合的需求規格、組合方式和執行方式等，提出相關定義，網路服務組合相關方法比較，如表 2-1 所示：

表 2-1 網路服務組合相關方法比較表

網路組合議題	方法	文獻作者
網路服務組合階段	(1) 規劃階段 (2) 定義階段 (3) 實作階段	(Yang and Papazoglou, 2002)
網路服務組合需求規格	(1) 程序導向規格 (2) 介面導向規格	(Matskin and Rao, 2002)
網路服務組合方式	(1) 探索式 (2) 半固定式 (3) 固定式	(Yang and Papazoglou, 2002)
	(1) 靜態	(Benatallah et al., 2003b)
	(2) 動態	(Chandrasekaran et al., 2003)
網路服務組合執行方式	(1) 中央集權模式 (2) 點對點模式	(Benatallah et al., 2003b)

以下將針對網路服務組合的階段、組合的需求與規格、服務組合方式和執行的方式，進行重點摘要說明。

2.3.2.1 網路服務組合階段

學者 Yang 和 Papazoglou 針對網路服務組合，將整個網路服務的組合過程，分為規劃、定義與實作三個階段(Yang and Papazoglou, 2002)：

(1) 規劃階段：尋找所有可能的候選服務，並且確認它們的組合能力和符合性。此階段，會產生替代的組合計畫，並且提給應用程式的開發者。

(2) 定義階段：產生實體的網路服務組合流程。此階段的產出是網路服務組合的規格。

(3) 實作階段：根據定義段產的網路服務組合規格，實作出網路服務組合的繫結。

2.3.2.2 網路服務組合需求規格

學者 Matskin 和 Rao 將網路服務組合的需求規格，區分為以下兩種(Matskin and Rao, 2002)，包含了程序導向規格和介面導向規格，說明如下：

(1) 程序導向規格(Process Oriented Specification; POS)：包含許多抽象化的網路服務元件，結合企業應用邏輯，以工作流程的概念，描述這些網路服務之間的協同關係；達成方式是以企業邏輯模型為基礎，配合工作流程的規劃，來實作網路服務的協同組合。

(2) 介面導向規格(Interface Oriented Specification; IOS)：組合服務需求者，針對所需求的內容，提供輸入和輸出的相關資訊即可，但需求者不需要瞭解複雜的細節，例如在組合服務中，需要包含那些網路服務元件，要以何種的結構組合這些服務。整個處理的方式，是透過程式自動化組合的方法，以達成網路服務的協同組合。

2.3.2.3 網路服務組合方式

學者 Yang 和 Papazoglou 則將網路服務組合的方式，區分為三種組合(Yang and Papazoglou, 2002)，其中包括探索式、半固定式與固定式組合，說明如下：

(1) 探索式(explorative)組合：網路服務的組合是依據使用者的需求，在執行時期產生組合流程。使用者必須先描述所需要的服務，然後，服務中介者會比較需求服務的特性，和已註冊服務中，可能符合的服務規格進行配對選擇，最後產生所有可能的網路服務組合計畫。這些產生的網路服務組合計畫，可以被排序或是由使用者再根據某些條件選擇，例如：可用程度、成本、效能等條件進行挑選。這種網路服務組合方式，是在執行時期被決定，並且需要動態的組合與協同服務。

(2) 半固定式(semi-fixed)組合：此種網路服務組合方式，有部份的服務連結

是在執行時期才決定。當此種網路服務組合被呼叫時，實際的網路服務組合計畫才會被產生，系統也會根據組合中所指定的服務項目，和所有可能可以使用的服務，兩者進行比較而產生結果。

(3) 固定式(fixed)組合：固定式的網路服務組合，事先就決定好要參與組合的服務有哪些。組合的結構與服務元件都是靜態連結。傳達給此網路服務組合型態的需求，再分送至各個參與組合的網路服務執行。

其中探索式組合，相似於動態組合方式，固定式組合，相當於靜態組合方式，與 Benatallah 和 Chandrasekaran 所提的觀點((Benatallah, 2003b; Chandrasekaran, 2003)相似，區分為靜態與動態的方式，主要取決於網路服務組合流程設計的時間點，靜態流程為事前即設定完成，其特性為流程固定不變，也因此缺乏彈性；而動態流程則是在執行時期才決定組合的順序，其特性為富有彈性且具延展性，不過因為即時決定，因此難度也相對提高。兩種組合型態，都包含了網路服務的搜尋。

比較不一樣的是半固定式組合，是一種介於探索式組合和固定式組合之間的新組合方式(Yang and Papazoglou, 2002)。此種方式的網路服務組合，有部份的服務是在執行時期，才決定連結哪個的服務，其餘則是事先就決定的連結方式。

2.3.2.4 網路服務組合執行方法

學者 Benatallah 指出，網路服務組合的兩種主要執行方法，包括中央集權(Central Authority)模式和點對點(Peer-to-Peer; P2P)模式(Benatallah, 2003b)。

(1) 中央集權模式：中央集權模式的網路服務組合，是由獨立的服務提供者管理。該提供者必須具有一個網路服務組合排程器。排程器也必須根據網路服務組合的控制流程，初始化整個網路服務組合，並依序分派與執行。為了達到這個目的，排程器要根據控制流程中規則的結果，呼叫每個網路服務的元件。此外，排程器必須能夠接收，並處理來自需求者的

要求。排程器也必須負責控制和處理資料的往來。

(2) 點對點模式：將工作平均分給每個管理服務元件的提供者。這樣的網路服務組合執行模式，不只需要中央排程器的協助，還必須依靠網路服務組合中的協調器，經由協調器的輔助，才能正常運作。多個協調器工作時，以點對點的方式互相溝通，確保每一個網路服務組合的執行，都遵照控制流程及資料流程的規格設計。

2.3.3 網路服務組合平台

本節針對目前網路服務組合的相關研究，進行歸納整理。其中包括 Self-Serv、FUSION 和 SCET 等三個網路服務組合平台，本文將平台整理分析，並比較個別的優與缺點，各網路服務組合平台的比較，詳細如表 2-2 所示：

表 2-2 網路服務組合平台比較表

組合平台名稱 / 比較項目	**FUSION**	**Self-Serv**	**SCET**
需求規格類型	介面導向規格	程序導向規格	程序導向規格
網路服務組合語言	WSESL	無	WSFL
執行方式	中央集權	點對點	中央集權
執行後分析	執行結果正確性分析	效能分析	效能分析執行模擬分析
特性	點對點為基礎的協同組合模型 有服務容器的概念	服務入口的為概念以流程需求為主的平台	效能分析以及模擬分析 有回覆機制
提出者	VanderMeer et al., 2003	Benatallah et al., 2003a	Chandrasekaran, 2003

以下將分別介紹 FUSION、Self-Serv 和 SCET 等三個網路服務組合平台的特性。

(1) FUSION：由 VanderMeer 等學者所提出，以服務入口的概念，同時也藉由 FUSION 系統的實作，提供共同的基礎架構。服務入口必須具備以下

的幾項功能：(a)蒐集使用者的目標資訊，(b)將使用者的目標，對應到實際可執行的流程，產生滿足需求者條件，且提供邏輯性示意圖，(c)轉換需求者的抽象目標，變成可執行的流程，且是最佳化的執行方案，(d)將執行方案對應成網路服務，並且管理方案的執行，(e)接收執行的結果，並且檢驗需求者的條件是否被滿足,(f)若執行結果不符合需求者的要求，系統將會啟動適當的復原程序，回復到執行前的狀態，(g)將處理結果傳遞給需求者(VanderMeer et al., 2003)。FUSION 的系統架構圖，如圖 2-5 所示。

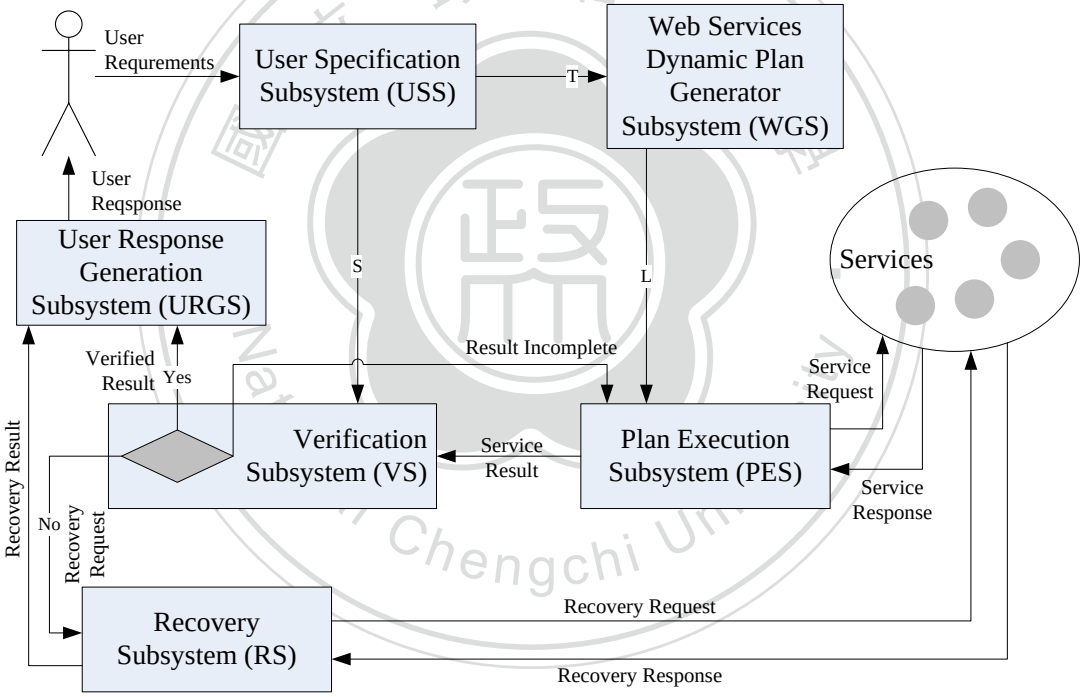

圖 2-5 FUSION 系統架構圖

資料來源：VanderMeer et al., 2003

在 FUSION 系統架構中，包含了六個子系統，摘要如下：

(a) 使用者需求規格系統(User Specification Subsystem; USS)：這是一個圖形化的介面，提供給需求者描述他的相關需求，並且轉換需求者提供的條件，形成結構化的架構，以便讓網路服務動態方案產生子

系統來處理。

(b) 網路服務動態方案產生子系統(Web Services Dynamic Plan Generator Subsystem; WGS)：經由網路服務動態方案產生子系統的處理，可以產生正確並且是最佳化的執行方案。此執行方案是以網路服務執行規格語言(Web Services Execution Specification Language; WSESL)來表達。處理的結果會輸入到方案執行系統。

(c) 方案執行系統(Plan Execution Subsystem; PES)：將方案對應到實際可執行的程式碼，然後真正執行方案中的每個服務，最後將執行結果傳遞給驗證系統。

(d) 驗證系統(Verification Subsystem; VS)：驗證系統會驗證服務的執行結果，是否符合需求者所指定的各項條件。

(e) 復原系統(Recovery Subsystem; RS)：若驗證結果不符合預期條件，則會將結果傳遞到復原系統，在復原系統收到之後，會啟動一連串的復原程序。

(f) 使用者回應產生系統(User Response Generation Subsystem; URGS)：負責蒐集驗證系統或復原系統的輸出，並且以適當的方式將執行結果呈現給需求者。

從以上的說明可以發現，希望提供一個完整的網路服務組合平台，包括服務的取得、服務的組合、組合後服務的執行和結果的呈現等。

服務入口的概念，是以需求者的角度，考量網路服務的組合。由於需求者的流程經常是多變的，所要求的限制條件也不會相同。服務入口所提出的方式，可讓使用者指定最低要求的機制，使得整個網路服務的組合更有彈性。

(2) Self-Serv：Self-Serv 專案，是由 Benatallah 等學者提出。Self-Serv 實作

出以點對點為基礎的網路服務組合模型(Benatallah et al., 2003a)。Self-Serv 是一個網路服務動態組合的中介架構，主要包含兩個概念：合成服務與服務容器，摘要如下：

(a) 合成服務：在 Self-Serv 中，合成服務又分為兩種：基礎網路服務與組合網路服務。基礎網路服務是指一個服務，不需要明顯的依賴其它網路服務。組合網路服務則是聚集數個其它的服務組合而成。在服務組合的流程邏輯描述方面，採狀態圖的方式表示。

(b) 服務容器：由於服務提供者對於移除服務、修改服務、變更服務位置的情形非常頻繁，因此，網路服務組合常常在高度變動的環境中運作。服務容器的概念，就是要幫助這種高度變動的服務進行組合。服務容器是指提供相同功能、可互相替代的服務聚合。服務容器有精確、查詢與註冊三種模式。精確模式是指在容器建立時，就能精確地指定容器中，網路服務的成員為何。查詢模式是指容器中的網路服務成員，必須透過服務中介者，例如必須透過通用描述、探索與整合的查詢來處理。在註冊模式中網路服務必須透過註冊程式，才能成為容器中的成員。

Self-Serv 採用點對點協同方式，來執行網路服務組合，其中包含以下兩項基本元素：狀態協調者與路由表。

(a) 狀態協調者：由 Self-Serv 中的狀態協調者，依據狀態圖的流程描述產生，並且由每個服務的提供者自行管理。狀態協調者必須接收來自其它狀態協調者的通知。並且從這些通知決定目前的流程狀態為何。當所有的狀態條件符合時，狀態協調者會呼叫相關服務。當服務執行完成時，會通知狀態協調者進入下一個狀態。

(b) 路由表：協調者必須依賴兩個路由表，以獲取所有的路由訊息。一

個路由表記錄在進入前的狀態，必須符合先決的條件。另一個路由表記錄著當狀態改變或離開時，必須通知那一個協調者。

Self-Serv 提出了實作必須具備的配套機制。並且對動態網路服務組合的實踐方式，提出實際可行的方法。

(3) SCET：服務組合和執行工具(Service Composition and Execution Tool; SCET)，主要是改善效能的問題，建構的一個完整的網路服務組合平台(Chandrasekaran et al., 2003)。

使用者可以透過 SCET 的圖形化設計功能，處裡靜態的組合網路服務，然後以網路服務流程語言(Web Services Flow Language; WSFL)規範的形式儲存。網路服務流程語言，支援動態的網路服務挑選，因此可以達到動態網路服務組合的功能。在執行流程方面，SCET 會根據網路服務流程語言的描述，自動產生相對應的 Perl 執行程式碼，來完成服務的組合。在效能分析方面，SCET 採用 Java 模擬動態環境(Java-Based Simulation and Animation Environment; JSIM)的模擬方式，來分析網路服務組合的效能。

本研究將建置實作轉換與分析系統，因此將參考並整合前述系統特性，使用網際網路標準建構。以下將介紹服務流程部份所使用的標準執行語言。

2.3.4 服務流程執行語言

企業需要因應外界變動，需要有更彈性的企業流程，隨著網路技術演進與服務導向概念的成熟，可透過網路服務組合語言(Web Service Composition Language)來達成。網路服務組合語言以流程導向方式，組合出新的網路服務，其與網路服務描述語言緊密結合，且概念是直接建構於網路服務描述語言之上(van der Aalst, 2003)。

目前網路服務語言主要有網路服務商業流程執行語言、商業流程執行語言

(Business Process Modeling Language; BPEL)(W3C, 2010)、延伸標記流程描述語言(XML Processing Description Language; XPDL)(W3C, 2010)、網路服務編排介面(Web Service Choreography Interface; WSCI)(W3C, 2010)、延伸標記流程敘述語言(XML LANGuage; XLAN)(W3C, 2010)等，其中又以網路服務商業流程執行語言(Web Services Business Process Execution Language; WSBPEL)為市場主流，廣泛地應用在目前系統上。

商業流程執行語言，是一種以延伸標記語言，描述企業內部流程的語言，於2002 年 8 月由微軟公司、IBM 公司和 BEA 等三家廠商合作開發，原名為商業流程執行語言使用於網路服務(Business Process Execution Language for Web Services; BPEL4WS)，在 2002 年出現第一版、於 2003 年更新為 1.1 版，直至今日的 WSBPEL 2.0 版(Andrews, 2003)。其定位為整合網路服務方面的標準，這些標準也相繼在其商業流程軟體中支援。該語言用於商業流程描述，強調規範結構化與標準化，其中包含多種網路服務的整合，並能將系統內部和業務夥伴之間的資訊交換標準化，使原本建立在不同系統上的商業流程，也能像網路服務可以跨平台互通有無。

網路服務商業流程執行語言文件，主要分為兩個部份：一個是描述構成流程的定義，例如，某項服務處理的訊息(Message)格式定義；另一個則是描述一個完整流程，如何串起多個不同的服務，以及處理路徑的判斷規則。在描述流程如何串起服務的部份，網路服務商業流程執行語言，使用延伸標記語言標籤，定義商業流程中的網路服務，以及商業流程活動(Activity)，活動的類型的描述包含資料操作、關聯、錯誤處理、補償及結構化活動；除此，網路服務商業流程執行語言，可以支援長時間的交易活動，並確保其狀態可以被保存，在實際商業活動進行時，若遭遇到重大事故，例如停電，可以確保狀態可以獲得保存，而不會因為突發狀況而流失，因此可增加交易的安全性。在網路服務商業流程執行語言 1.X 版中，常用的標籤如下(Fransisco et al., 2002)：

<partners>：包含一連串網路服務組合執行時，所要呼叫的網路服務。

<variables>：在網路服務組合流程中，所需要用到的變數。

<sequence>:以串列方式執行所包含的內容。

<flow>：以平行方式執行所包含的內容。

<while>：以迴圈方式重覆執行所包含的內容。

<switch>：以程式語言常用的 case 方式，在所包含的內容中選擇一種執行。

<pick>：配合外在的事件執行活動。

<invoke>：呼叫執行一個特定的網路服務。

<receive>：接收網路服務執行後所傳回的結果訊息，或呼叫該網路服務組。

<reply>：傳送啟動網路服務執行所需要的參數訊息，或回覆網路服務組合。

<assign>：指派給變數一個特定的值。

<empty>：不執行任何行為之活動。

<throw>：遇錯誤時負責接收錯誤，並進行進階錯誤處理。

<wait>：指令流程暫停一段時間。

<terminate>：立即結束流程。

以上是網路服務商業流程執行語言，於版本 1.0 與 1.1 時之常用標籤，在 2005 年商業流程執行語言，更新為網路服務商業流程執行語言 2.0 後，提供了更多定義活動的標籤，並停用了 1.0 與 1.1 版之<switch>與<terminate>兩個標籤 ，其餘新增標籤如下(Assaf et al., 2005)。

<validate>：驗證變數的結果。

<exit>：立即離開處理程序，替代了前版的<terminate>。

<rethrow>：可將接收到的錯誤，送向更高層的錯誤處理器。

<if>：提供依條件分支的活動，擴充了前版<switch>之功能。

<forEach>：以迴圈方式處理每一個子項目。

<repeatUntil>：自動重複執行某個活動，直到一定的條件成立，或是時間條件成立才停止。

2.3.5 網路服務組合樣式

為了能夠將各種複雜的網路服務組合，和流程設計都加以塑模，因此必須探討網路服務組合流程中，所有可能出現的組合方式。Aalst et al. (2003)將工作流程的樣板，分為六大類共二十種，研究發現，目前常用的網路服務組合語言WSBPEL、XLANG、WSFL、XPDL 等還無法完全的實作二十種工作流程樣板。

除此之外，還有其他的網路服務組合語言，例如工作流程延伸標記語言(Workflow XML; WFXML)、網路服務編排介面(Web service choreography interface; WSCI)、網路服務代理標記語言(DARPA agent markup language for services; DAML-S)等也有相同問題，然而，目前網路服務組合的執行，需要依賴各種網路服務組合語言，因此，本研究在對網路服務組合流程，進行塑模與轉換時，將會考量到各項細節與限制。

2.4 雲端服務

雲端是過去電腦網路剛剛起步時，描述系統的架構圖中，系統連線到遠方機器或設備時，會經過很多的網路線路與設備，中間經過的複雜路徑，經常會使用雲朵的圖形來表示，之後用來描述網路連線至其他網路環境時，就稱為連線到雲

端，結合網路服務與運算的環境時，就使用端運算這個名詞。

雲端運算是個涵義廣大的概念，而非定義，主要目標是希望運算或儲存，能像生活中使用水電方式一樣，容易取得也容易使用，只需要透過網路連線，可選用免費或依使用狀態付費的服務。雲端中包含龐大的運算力與多樣化的網路服務，因此，企業不需要建置自己的資訊中心，即可以在雲端中使用不同的服務，這種新的服務模式，廣泛地應用在目前的系統中。

雲端運算的架構，大致上可分成三個階層：應用程式、服務平台與基礎設施，三種型態服務的關係，如圖 2-6 所示。

依雲端技術部署的所在地，與不同的雲端服務使用者，區分為公有雲、私有雲與混合雲(IBM, 2010)，例如 Amazon.com 與 Google 等網路服務供應商，所部署的雲端服務稱公有雲，企業或組織內部網路為私有雲，其中，公有雲的服務又可區分為軟體即服務(Software as a Service; SaaS)、平台即服務(Platform as a Service; PaaS)與基礎即服務(Infrastructure as a Service; IaaS)等三類(IBM, 2010)。

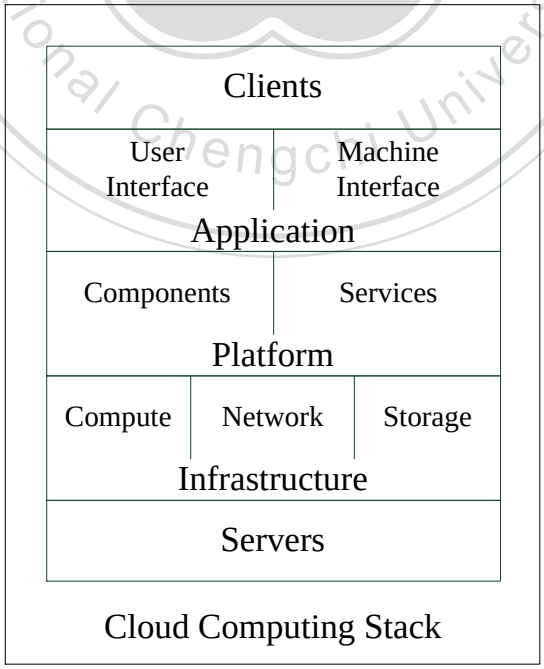

圖 2-6 雲端服務三種服務型態圖

(1) 軟體即服務：將軟體視為是一種服務的概念，軟體服務供應商，以租賃的概念提供客戶服務，而非購買。該服務的軟體安裝於遠端的網路伺服器，並確保在網路環境下可執行其功能，因為其具有高度的靈活度，與強大的擴展性，亦具有較低維護成本的的特性。

目前網路上有很多的軟體即服務的提供廠商，其中最著名的代表就是 Saleforce.com 和 Google Apps。此外，還有國內的趨勢科技雲端防毒、YouTube、Facebook、Twitter 等網際網路應用程式，皆屬於不同類型的軟體即服務。微軟公司在商用的 Office 軟體上，另外提供相對應的線上 Office 軟體服務，這也算是軟體即服務的一種方式。在儲存方面，目前 Amazon 所提供的自助式內容分派服務 CloudFront，及支援檔案共享與資料同步化服務的微軟 Live Mesh，都是採用分散式雲端儲存技術的軟體即服務應用。

軟體即服務是有效的方式，比直接購買與安裝應用程式的成本低，通常軟體的使用是須要付費，目前以月租或是年租方式計價。軟體即服務的用戶，在軟體的管理上，可以不必擔心日後所有的安裝與升級問題。

(2) 平台即服務：因軟體即服務的需求快速擴展，連帶地使得平台即服務也有大量需求。平台即服務是指提供了一個資訊開發與使用的平台。系統人員撰寫程式碼於平台即服務的平台上，系統平台提供上傳的介面，或應用程式介面(Application Programming Interface; API)服務，這些服務都是透過網際網路進行的。SalesForce.com 也是平台即服務的例子。平台即服務的發展提供了服務、測試、部署和維護網路服務的主機，在整合的開發環境上。它還提供了系統基本的創建與網路服務程式。因此，平台即服務的架構提供了一種更快、更有效的模型，並提供系統的應用、開發和系統交付的平台。平台即服務供應商也提供系統升級服務、更新

和其他日常系統維護。管理者或是需求者只需支付使用部份的費用。使用過程中，用戶只需要瞭解有什麼服務，服務要怎麼使用，而不必擔心系統內部的複雜性。

平台即服務形式目前有幾種類型，例如：社交應用平台、服務執行與運算平台、網絡應用平台和商務應用平台等等。例如，Facebook 是一個社交類型的應用平台，其中第三方組織可以撰寫新的應用服務，提供更多服務給使用者。又例如提供客戶關係管理服務的機構，提供的是商務應用平台，以支援客戶提出的相關問題與服務。在服務執行與運算平台上，開發人員可以在亞馬遜公司提供的平台上，上傳和執行他們的服務程式。此外 Google 也提供應用程式介面服務，來建構網際網路應用系統。

(3) 基礎設施即服務：基礎設施即服務是指硬體部份的基礎設備，可部份或完全委外處理的基處服務。提供基礎設施服務的廠商例如 Google、Amazon，中華電信等等，除提供傳統的主機託管服務外，主機租用服務的部份，是透過虛擬主機的架構處理，能提供動態的運算資源服務。企業或使用者可以依據需求，購買基礎設施的服務量，概念上只針對購買的硬體資源或運算能力付費，這些資源可能屬於短期性或暫時性服務，並且可依需求變化，進行動態調整。基礎設施即服務的商業模式屬於“使用多少付多少”的模式，確保用戶僅需支付他們使用的資源服務。若企業自行建構機房或基礎設施，其建構成本與維護成本可能會非常高，而基礎設施即服務的供應商，是透過虛擬化提供服務，使承載的硬體效益更高、更具彈性。因此，能夠有效地降低成本，並獲得有效率的資訊技術資源，這就是基礎設施即服務最大的優勢。尤其對於企業來說，可以有效地降低持續性的資本支出，並提供穩定的系統服務；而透過這樣的服務模式，中小企業能擁有簡單便利，屬於企業等級的硬體和數據中心。

雲端技術的發展，離不開傳統的技術與環境，部份議題是傳統平台中，就存在的問題，但在雲端環境中，因為被賦予新的內涵，也因此產生了一些新的議題，這些新舊議題整合後包括：快速佈署系統、資源彈性化、資源穩定度問題、巨量資料的處理、訊息的傳遞與交換、巨量的分散式處理、安全問題、可用性與可擴充性問題、如何自動化與標準化等等。上述議題中，資源彈性化之後服務與資源穩定度就會受到影響，執行過程中，就有因為資源共享問題，造成等待或死結問題，此外信息的交換與自動化、標準化，與本研究有緊密的相關性，因此本研究使用網路服務標準協定，加入網路活性判斷，以確保系統的效率與穩定度。

2.5 派翠網路介紹

派翠網路(Petri Net; PN)是 1962 年，由西德數學家 Carl A. Petri 博士提出，適合於描述非同步的、並行的電腦系統模型。派翠網路有嚴格的數學表述方式，也有直觀易懂的圖形表達方式，是同時擁有圖形化與數學化的正規模擬工具，跟其它圖形建模工具不同的地方，在於它可以表現平行與分散式系統的動態行為，除此之外，也提供一套數學理論，支援系統描述方法和系統行為分析技術。為紀念 Carl 博士所做的貢獻，這種網路分析理論定名為派翠網路。

派翠網路應用在各種不同的領域，已經有相當豐富的成果，像是在知識庫系統、專家系統、學習評估、彈性製造系統(Flexible Manufacture System; FMS)、電腦整合製造(Computer Integrated Manufacturing; CIM)、資料庫設計、資料傳輸及電子電路設計等等，也廣泛地使用於系統的設計、建構與分析之中。由於派翠網路能表達並行的事件，被認為是自動化理論的一種。研究領域趨向認為，派翠網路是所有流程定義語言之母。

派翠網路有其邏輯上的定義(Murata, 1989)，派翠網路由五個部份組成 PN = (P, T, F, W, M_0)，這些組成元素分別為狀態(Place)、標記(Token)、轉移(Transition)

和有方向性的有向弧(Arc)，主要圖形是由有向弧連接狀態與轉移兩種節點，

$P = \{p_1, p_2, \ldots, p_m\}$，$m \geqq 1$，$p$ 為一組有限的狀態集合；

$T = \{t_1, t_2, \ldots, t_n\}$，$n \geqq 1$，$t$ 為一組有限的轉移集合；

$F \in (P \times T) \cup (T \times P)$，是一組有向弧集合的流程關係；

$W: F \rightarrow \{1, 2, 3, \ldots\}$，是一個權重函數；

$M_0: P \rightarrow \{0, 1, 2, 3, \ldots\}$，是初始派翠網路呈現的狀態；

其中 $P \cap T = 0$，$P \cup T \neq 0$。

其意義解釋如表 2-3 所示。若以圖形呈現來看，類似流程圖(Flowchart)，因為它具有平行(Parallel)與同步(Concurrent)的模擬能力，故至今已被廣泛用在不同領域作為系統之模組化建構、分析與模擬。

表 2-3 派翠網路組成單元整理表

派翠網路之單元	意義解釋
狀態	資源狀態、條件、緩衝或儲存空間
轉移	作業、過程、活動及事件
標記	物料、資源、資訊
有向弧	物料、資源、資訊、控制之方向

資料來源：Murata, 1989

派翠網路主要精神是將模型化的目標系統，解析為有限的狀態與動作，藉以表現出系統的行為模式。以狀態和所包含的標記(一到多個)表示系統中的一個特定狀態，而派翠網路中所有狀態，稱為標記狀態(Marking)。

轉移用來表示系統中，會發生的幾個狀態轉移事件；而有向弧為連接狀態及轉移的有向線段，用來表示兩者之間的關係。當發生某一個事件時，會使得派翠網路由目前的狀態轉移至另一個狀態。而利用系統中狀態的變化，便可觀察出系

統本身的運作情況。

以下用一個簡化的例子，來說明派翠網路的執行動作。圖 2-7 為一個派翠網路的模型，圖中的每一個轉移，能擁有一個到多個輸入或輸出狀態，例如 p_1 與 t_1 之間存在一個有向弧連接兩者，且方向是由 p_1 連向 t_1，所以 p_1 稱為是 t_1 的輸入狀態，同理 p_2 也是 t_1 的一個輸入狀態。相反的，t_1 與 p_k 存在一個方向，也就是指向 p_k 的有向弧，所以 p_k 就是 t_1 的輸出狀態。在此可將輸入狀態視為是一個轉移的啟動條件，而輸出狀態則是轉移被啟動後所引發的結果。當一個轉移所有的輸入狀態，都擁有標記時，也就表示此轉移的啟動條件已被滿足，所以會引發特定的事件觸發，將轉移輸入狀態的標記轉移到輸出狀態去；如圖 2-7 的派翠網路基礎模型圖內，p_1 與 p_2 皆有標記的存在，會導致 t_1 被觸發，使得輸入狀態的標記轉移至輸出狀態 p_k 上，而 p_1 和 p_2 在轉移後其本身所擁有的標記也會跟著消失。

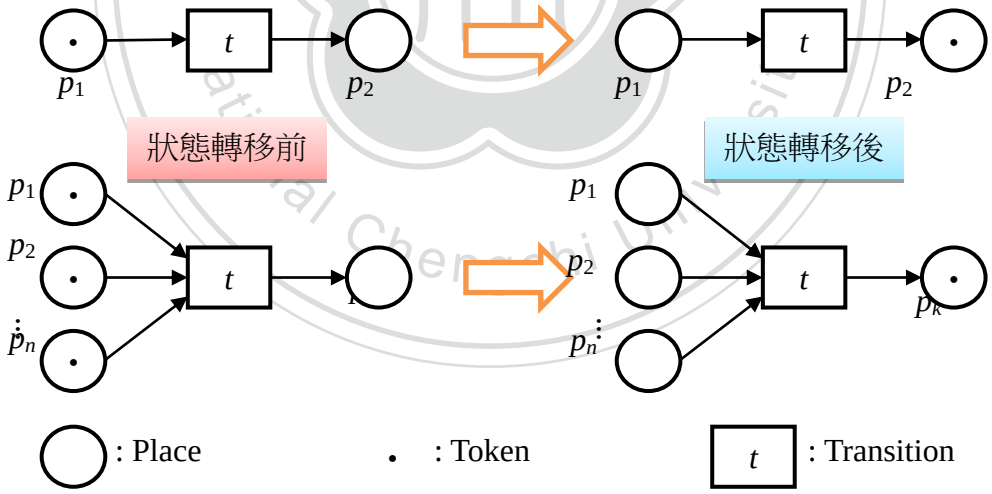

圖 2-7 派翠網路基礎模型圖

派翠網路具有 4 種特性：(1)可達性(reachability)，(2)有限性(boundedness)，(3)安全性(Safeness)，(4)活性(Liveness)，其特性整理如表 2-4 所示。

(1) 可達性(reachability)：在一個派翠網路中，存在著一個觸發的順序，假設

起始狀態是 M_0，可轉換至 M_N 的狀態，這樣稱為標記狀態 M_0 到達之狀態，由 M_0 到 M_N 到達的狀態之集合，以 $R(M_0)$來表示。

(2) 有限性(boundedness)：在一個派翠網路中，假設起始狀態是 M_0，經過一連串的觸發後，狀態中的標記數小於有限數 k，且 k 大於零稱為 k-bounded。

(3) 安全性(Safeness)：假設一個派翠網路為 1-bounded，則稱此派翠網路符合安全性。

(4) 活性(Liveness) ：在一個派翠網路中，假設，若每個屬於 $R(M_0)$的 M，及每一個轉移 *t*，存在有一觸發序列，其中包含 *t*，則稱此派翠網路符合活性，這也意味著系統中每一個活動，都可以順利被執行，不會有死結情形發生。

表 2-4 派翠網路特性整理表

派翠網路之特性	派翠網路原本意義	網路服務中的含意
可達性	標記狀態可由初始標記狀態，經轉移觸發序列而到達	流程中每一條件都有可能為真
有限性	無緩衝區不足的情況	無緩衝區不足的情況
活性	避免死結的發生，確保模式中之事件、作業、過程或活動都可以進行	流程中每一個服務都可以被執行

派翠網路本身即具備模擬與分析之特性，根據學者 Zhou 等人之研究，使用派翠網路的優點如下(Zhou et al., 1996)：

(1) 可用圖形化方式，建立系統模型。

(2) 可由派翠網路的模型，直接進行模擬分析。

(3) 可檢查系統之超載與死結情形，適合用於排程分析。

(4) 不限定於特定的系統，具有獨立性。

(5) 可藉由派翠圖形產生控制碼。

(6) 可以即時監控狀態。

本研究主要觀察網路服務的狀態，包括組合流程的正確性，執行流程與路徑的穩定性，因此，需在組合完成後判斷其活性，此外也希望瞭解系統流程架構，方便系統建置與管理，依據學者 Zhou 等人之研究(Zhou et al., 1996)，派翠網路是一個很適合的工具，因此，本研究將雲端中的網路服務，轉換成派翠網路，利用派翠網路來呈現系統結構與分析結果。

2.6 派翠網路死結的判斷

死結的判斷規則有四個必要條件，若用網路服務觀點重新描述作業系統中，死結的發生情形，則修改後如下：

(1) 互斥(Mutual Exclusion)：資源具有獨佔性，須等待此資源被釋放後，才能再度競爭此資源。

(2) 不可換出(Non-Preemption)：若資源正在使用中，則不可以強取此資源，而必須等待資源使用後被正常釋放，才可繼續使用。

(3) 持有與等待(Hold and Wait)：某個資源被一個處理單元擁有，並且正在使用，因為它具有獨佔性，其他處理單元必須等待此資源被釋放後，才能再競爭此資源。

(4) 循環等待(Circular Wait)：一組處理元 p_0， p_1， …， p_n，其中 p_0 正在等 p_1 所擁有的資源，而 p_1 正在等 p_2 所擁有的資源，…，p_n 正在等 p_0 所擁有的資源，如此稱為循環等待。

作業系統中的系統資源，是指程序處理時所獲得的中央處理器時間，而網路服務流程中的資源，是指所獲得資源的控制權，流程中的活動需要得到控制權才能夠執行，就好像在環形(token ring)網路環境中的工作站，必須得到標記後，才能執行工作。網路服務流程控制權是不能共享的，所以互斥的條件必須成立。網路服務沒有優先權概念，且不可中途換出控制權，故不可換出條件也必須成立。

因此，當流程中的循環等待，或是相互持有條件皆成立的時候，流程就會有死結產生。

雖然派翠網路應用，因其具有獨立性與方便性，使其被廣泛的使用在各個領域，但仍有三個缺點，第一、派翠網路沒有階層的概念，因此無法建構子模式，無法成為統一化的溝通介面，因此派翠網路常需針對不同的系統，進行不同模式的修改。第二、無法呈現資料的概念，派翠網路中只有標記資料，定義方向與資料型態的概念。第三、因所有資料都必須符合轉移和狀態的表示方法，使得派翠網路架構龐大而顯得複雜難懂。

有鑒於上述三點缺點，本研究在研究商業流程 WSBPEL 2.0 轉換派翠網路時，會針對派翠網路的龐大複雜難懂，以及複雜結構化的問題進行改進，使 WSBPEL 2.0 轉換為派翠網路後之模型，依然具有可讀性及保持資料的敘述性，以方便管理者判讀。這部份將於第三章與第四章進行說明。

第三章　研究方法

本研究進行的流程主要包含兩大部份，第一部份是將網路服務流程，透過 WSBPEL 轉化為派翠網路；第二部份是針對派翠網路進行分析。本章的分節主要就是處理這兩部份，第一部份再拆解成 3.1 和 3.2 兩節，3.1 節先討論 WSBPEL 轉派翠網路機制，3.2 節再針對 WSBPEL 2.0 轉派翠網路的流程及演算法，3.3 節是描述針織法如何分析網路。

3.1 WSBPEL 轉派翠網路機制

WSBPEL 為一描述商業流程之程式語言，主要是讓商業處理的程式、流程和服務間能夠藉由 BPEL 程式語言來進行溝通。基本上 BPEL 可以藉由不同的視覺化工具來產生，如 Oracle JDeveloper、IBM Websphere、Eclipse，皆可描述出可執行且具有抽象化之商業流程。但如前述，BPEL 為一程式語言而非模型化描述語言，如同一般程式語言，會在執行期間發生錯誤，例如本研究之死結問題 (dead lock)，此類問題，目前需要以人工進行統整，並手動修復問題。如何讓管理者減少手動處理 BPEL 問題，或是處理時能夠精簡、方便且易於瞭解狀態，進而找到問題點的所在，並有依據地歸納統整問題，便是 WSBPEL 轉為派翠網路之重點及精神所在。

學者高慶霖(2004)，將 WSBPEL 劃分為基本元素、結構化元素、並行處理元素三大部份。在 WSBPEL 中基本元素部份包括 receive、reply、invoke、throw、empty、wait、assign 等項目；結構化元素則包含 sequence、pick、while；並行處理元素部份包含 flow。

文獻中，學者高慶霖(2004)之研究，將 BPEL 轉換成派翠網路之主因，主要

希望能偵測死結的發生，故將基本元素全部轉換為派翠網路中之轉移，並於轉移之名稱中加入行號與 WSBPEL 活動名稱。

但是在 BPEL 轉換成派翠網路的過程與最後結果，仍有許多尚未解決的問題存在，例如(1)因 WSBPEL 版本已更新，舊有版本中的部份指令與敘述，目前已不再使用，因此需要修正，(2)新版 WSBPEL 增加了許多新指令與活動，在本研究中會一併納入處理，(3)轉換為派翠網路後，有些網路結構龐大複雜，且會衍生出失去結構化等問題，故轉換後之格式及邏輯，需再加以重新定義處理，使其保留原始網路意義，(4)轉換後可讀性與關連性變弱，一旦發生問題，問題的點的位置難以掌握，因此，轉換過程中須針對此問題須強化，使管理者能依網路架構找到對應程式的問題點。以上四點為本研究針對 WSBPEL 轉換至派翠網路部份，最重要的貢獻。

3.1.1 WSBPEL 2.0 轉換為派翠網路之規則

由於 WSBPEL 轉換成派翠網路，是採取一對一轉換方式，因此，過程中主要以 WSBPEL 2.0 官方規格書上之分類為基礎(Assaf et al., 2005)，並增加關於 WSBPEL 2.0 在商業流程中，處理順序活動之類別。分別是基礎活動(Basic Activities)、結構化活動(Structured activities)及流程活動(Process activities)。針對所有的活動，將會在轉換過程中，加入該活動在文件中的活動範圍，使其成為派翠網路中轉移或狀態識別的部份。

3.1.1.1 WSBPEL 2.0 基礎活動轉換至派翠網路規則

在 WSBPEL 2.0 的規範中，包含 receive、reply、invoke、assign、throw、exit、wait、empty、rethrow 及 validate，這些指令為基礎活動(Basic activities)。基礎活動中又分為基本處理活動、指派活動、驗證活動、控制活動、錯誤處理活動，以下將分開討論。

基本處理活動包含 receive、reply、invoke 等三項活動。雖然商業流程定義複雜，但最基本的服務與流程之間，常用的溝通方式，最主要是靠這三種活動方式，所以死結比較容易發生在這些地方。

在商業流程處理的活動中，有循序處理(Sequence)、並行處理(Concurrent)以及非同步處理三種，其中又以非同步處理最為常見(Foster et al., 2003; Fu, 2005)。非同步處理時，商業流程(Business process)與服務(Service)、或服務與流程、或流程和流程之間會進行溝通，其大部份情形，是以圖 3-1 商業流程與服務溝通圖的情況存在。

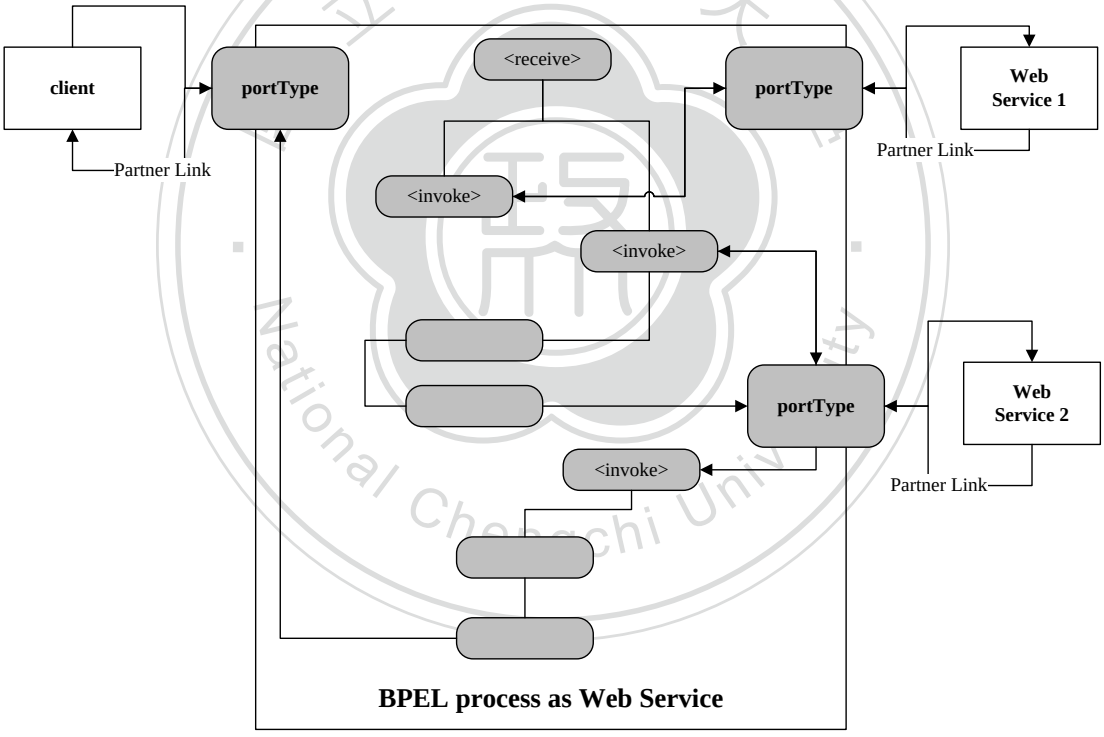

圖 3-1 商業流程與服務溝通圖

如圖 3-1 所示，在商業流程與客戶端，或是服務之間的溝通，都是依靠 Partner Link 以及 Port Type。基本處理的活動轉換時，格式將包含這三種類型活動的共同屬性(attributes)，也就是 partnerLink、portType 及服務中被呼叫到的操作 operation。

(1) receive 轉換

receive 表示在商業流程中，或是某特定的服務內，負責等待，直到接收到由客戶端或是其他網路服務傳來訊息。依照 WSBPEL 2.0 規格書中之規範，<receive>的語法規則定義與範例如表 3-1 所示：

表 3-1 receive 語法規則與範例表

語法規則	範例
<receive partnerLink="NCName" portType="QName"? operation="NCName" variable="BPELVariableName"? createInstance="yes|no"? messageExchange="NCName"? standard-attributes> standard-elements <correlations>? <correlation set="NCName" initiate="yes|join|no"? />+ </correlations> <fromParts>? <fromPart part="NCName" toVariable="BPELVariableName" />+ </fromParts> </receive>	<receive partnerLink="client" portType="com:InsuranceSelectionPT" operation="SelectInsurance" variable="InsuranceRequest" createInstance="yes" > </receive>

上述右方範例表示，目前所在之服務或商業流程正在等待接收 client 經由 com:InsuranceSelectionPT 中的 SelectInsurance 處理方式傳回至 InsuranceRequest 的資訊。

依照前述的轉換規則，receive 必須加上基本處理活動類型中，共有之屬性，

加入方式為@[partnerLink] : [portType] (operation)；並且須加上所有 WSBPEL 2.0 活動所共同加入之文件範圍，且此活動將會以派翠網路中的轉移形態來表示。依照上述範例轉換後，派翠網路圖形如圖 3-2 所示。

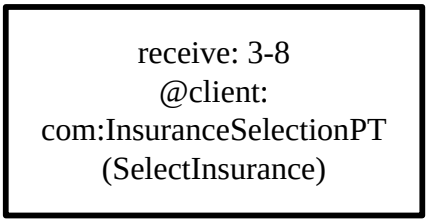

圖 3-2 receive 派翠網路圖

(2) reply 轉換

reply 可針對特定服務中，處理的過程進行回覆，常與 receive 相互搭配，使商業流程與服務或服務與服務間進行溝通。其語法規則與範例如表 3-2 所示。

表 3-2 reply 語法規則與範例表

語法規則	範例
<reply partnerLink="NCName" portType="QName"? operation="NCName" variable="BPELVariableName"? faultName="QName"? messageExchange="NCName"? standard-attributes> standard-elements <correlations>? <correlation set="NCName" initiate="yes|join|no"? />+ </correlations> <toParts>?	<reply partnerLink="client" portType="com:InsuranceSelectionPT" operation="SelectInsurance" variable="InsuranceSelectionResponse" > </reply>

<toPart part="NCName" fromVariable="BPELVariableName" />+ </toParts> </reply>	

依上述右方範例所示，其所在之商業流程或服務，回覆其他服務或客戶端 client 時，回覆的方式則是透過 com:InsuranceSelectionPT 的 portType 呼叫 SelectInsurance 的 operation。根據轉換規則，reply 轉換為派翠網路中的轉移，並包含 reply 所涵蓋之範圍，及基本處理活動之共有屬性，轉換結果如圖 3-3 所示。

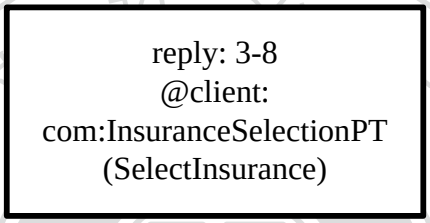

圖 3-3 reply 派翠網路圖

(3) invoke 轉換

invoke 在 WSBPEL 2.0 功能中，是呼叫其他的網路服務，如同其他的基本處理活動，是透過 partnerLink 指定服務，再藉由 portType 呼叫特定 operation。語法規則與範例如表 3-3 所示。

表 3-3 invoke 語法規則與範例表

語法規則	範例
<invoke partnerLink="NCName" portType="QName"? operation="NCName" inputVariable="BPELVariableName"? outputVariable="BPELVariableName"? standard-attributes>	<invoke partnerLink="insuranceA" portType="ins:ComputeInsurancePremiumPT" operation="ComputeInsurancePremium" inputVariable="InsuranceRequest" outputVariable="InsuranceAResponse" > </invoke>

standard-elements <correlations>? <correlation set="NCName" initiate="yes|join|no"? pattern="request|response|request-response"? />+ </correlations> <catch faultName="QName"? faultVariable="BPELVariableName"? faultMessageType="QName"? faultElement="QName"?>* activity </catch> <catchAll>? activity </catchAll> <compensationHandler>? activity </compensationHandler> <toParts>? <toPart part="NCName" fromVariable="BPELVariableName" />+ </toParts> <fromParts>? <fromPart part="NCName" toVariable="BPELVariableName" />+ </fromParts> </invoke>	

上方之範例，則是目前所在的商業流程或服務呼叫 insuranceA 服務，並藉由 portType ins:ComputeInsurancePremiumPT 呼叫 operation ComputeInsurancePremium，並將呼叫時的參數存於 inputVariable InsuranceRequest

中，以進行傳遞，而 insuranceA 回覆時，則將結果儲存於 outputVariable InsuranceAResponse 中。依照基本處理活動的處理方法，找出本類型活動共同之屬性，並附帶所有活動共同之範圍資訊，轉換成為派翠網路中之轉移；轉換結果如圖 3-4 所示。

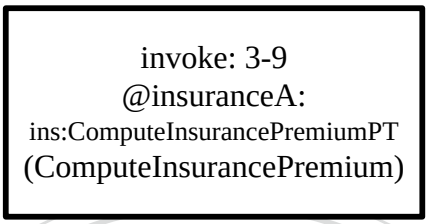

圖 3-4 invoke 派翠網路圖

3.1.1.2 指派活動轉換

指派活動包含 WSBPEL 2.0 中的 assign 活動，不同於基本處理活動，指派活動是針對服務中，或服務與服務間之資料進行複製，以下將針對 assign 轉換進行說明。

assign 活動提供了變數的變數值、與其他服務之間進行參照，進行複製的功能。在 assign 中使用指令 copy，在指令 copy 中又包含了複製的來源 from 以及複製的目標 to。其語法規則及範例可見表 3-4 所示。

表 3-4 assign 語法規則與範例表

語法規則	範例
<assign validate="yes|no"? standard-attributes> standard-elements (<copy keepSrcElementName="yes|no"?	<assign> <copy> <from variable="InsuranceAResponse" /> <to variable="InsuranceSelectionResponse" />

ignoreMissingFromData="yes|no"?> from-spec to-spec </copy> | <extensionAssignOperation> assign-element-of-other-namespace </extensionAssignOperation>)+ </assign>	</copy> </assign>

上述範例是將變數 InsuranceAResponse 複製到 InsuranceSelectionResponse。assign 的轉換規則，首先須包含 assign 活動的範圍，且因 assign 屬 WSBPEL 2.0 中的基礎活動，所以轉換為派翠網路中的轉移，並且搭配帶有資訊的 copy 指令，表明複製的內容由來源(from)至目的(to)，中間以「→」相連，顯示出資料的流向，轉換後可在轉移中清楚表示 assign 活動之目的。轉換結果如圖 3-5 所示。

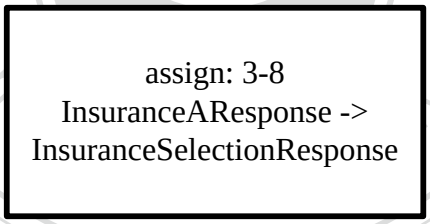

圖 3-5 assign 派翠網路圖

3.1.1.3 驗證活動轉換

驗證服務包含 WSBPEL 2.0 中的 validate，主要驗證變數數值是否合理，以確保商業流程能夠正確執行，以下將針對 validate 轉換進行說明。

validate 可在變數經過處理後，特別是在進行過上一小節所述的指派活動後，確認數值是否正確。在 validate 中，藉由屬性 variables 指定需驗證之變數，其中

可定義一到多個變數，變數與變數之間，以空白字元分開。validate 的語法規則及範例如表 3-5 所示。

表 3-5 validate 語法規則與範例表

語法規則	範例
<validate variables="BPELVariableNames" standard-attributes> standard-elements </validate>	<validate variables="InsuredPersonRequest InsuranceRequest PartialInsuranceDescription " />

依照上述，validate 活動驗證了三個變數 InsuredPersonRequest、InsuranceRequest 以及 PartialInsuranceDescription。因為 validate 屬於 WSBPEL 2.0 中的活動，故包含其在文件中之活動範圍；並且取出屬性 variables 之值，轉換為派翠網路的轉移。轉換結果如圖 3-6 所示。

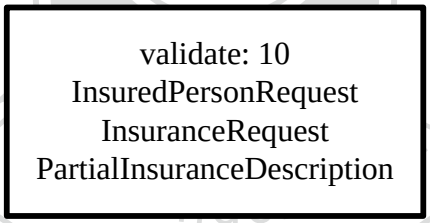

圖 3-6 validate 派翠網路圖

3.1.1.4 控制活動

控制活動包含 WSBPEL 2.0 中的 empty、wait 及 exit，主要負責立即結束流程、等待處理以及特定情況下，需要使用沒有功能的活動。

(1) empty 轉換

在 WSBPEL 2.0 中，有些情況必須得定義活動，但卻不需要做任何處理，這

便是 empty 的功能，例如在後續會說明的條件式活動，有些狀況發生是不需要做任何處理，但依照 WSBPEL 2.0 標準，必須定義該狀況的活動，這也就是 empty 的功用。其語法規則與範例如表 3-6 所示。

表 3-6 empty 語法規則與範例表

語法規則	範例
<empty standard-attributes> standard-elements </empty>	<empty />

empty 沒有特殊的轉換規則，因為該活動的目的在名稱上已表示，故依最基本的轉換原則，加入該活動在文件中所含的活動範圍。轉換結果如圖 3-7 所示。

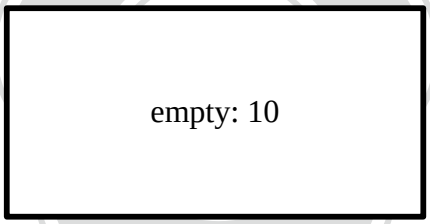

圖 3-7 empty 派翠網路圖

(2) wait 轉換

在商業流程中，可以由 wait 活動定義固定時間的暫停，商業流程便會暫停，直到定義的時間點，或定義的相對等待時間到達，才會進行下一步驟。wait 中有兩個指令，until 與 for，分別指定絕對時間及相對等待時間。其語法規則與範例如表 3-7 所示。

表 3-7 wait 語法規則與範例表

語法規則	範例
<wait standard-attributes> standard-elements (<for expressionLanguage="anyURI"?>duration-expr</for> | <until expressionLanguage="anyURI"?>deadline-expr</until>) </wait>	<wait> <until>'2011-06-18T21:00:00+01:00'</until> </wait>

依照上述範例，有 wait 活動等待直到 2011 年 6 月 18 日的晚間九點。在轉換至派翠網路時，wait 屬於 WSBPEL 2.0 的基本活動，故轉換為轉移型態，且包含其指令 until 或 for 之內容，以及 wait 活動在文件中之範圍，轉換後結果如圖 3-8 所示。

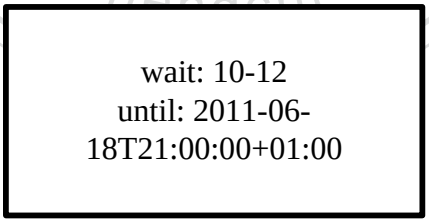

圖 3-8 wait 派翠網路圖

(3) exit 轉換

exit 可以在流程中的某些特定情況下，立即中止商業流程。其語法規則與範例如表 3-8 所示。

表 3-8 exit 語法規則與範例表

語法規則	範例
<exit standard-attributes> standard-elements </exit>	<exit />

exit 活動屬於 WSBPEL 2.0 基本活動，故轉換為派翠網路時，屬於轉移的型態。因 exit 活動的目的單純，故只需符合一般轉換原則，加入其活動在文件中所含蓋的範圍。轉換後的結果如圖 3-9 所示。

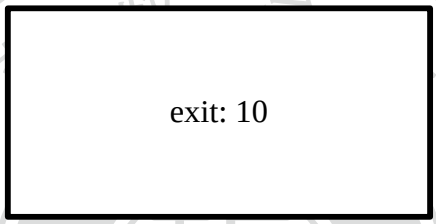

圖 3-9 exit 派翠網路圖

3.1.1.5 錯誤處理活動

錯誤處理活動，包含 WSBPEL 2.0 之 throw 以及 rethrow 命令，其內容如下所述：

(1) throw 轉換

throw 活動可以提供商業流程中，針對特定情形發出特定的錯誤訊息，接下來，再進行其他活動來處理。其語法規則與範例如表 3-9 所示。

表 3-9 throw 語法規則與範例表

語法規則	範例
<throw faultName="QName"	<throw faultName="WrongEmployeeName" />

faultVariable="BPELVariableName"? standard-attributes> standard-elements </throw>	

如上述範例，則是針對某特定情形，發出一個 throw 活動，其名稱定義在 faultName 中。在轉換為派翠網路時，將其轉換為轉移形態，並且包含該活動在文件中的範圍，以及屬性 faultName。轉換結果如圖 3-10 所示。

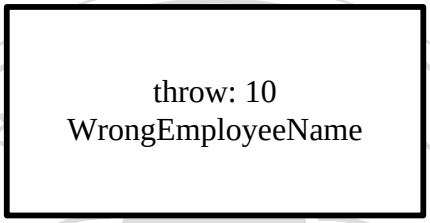

圖 3-10 throw 派翠網路圖

(2) rethrow 轉換

當錯誤藉由 throw 活動送出後，處理的商業程序可以藉由 rethrow 將錯誤送往更高階的錯誤處理器進行處理。其語法規則和範例如表 3-10 所示。

表 3-10 rethrow 語法規則與範例表

語法規則	**範例**
<rethrow standard-attributes> standard-elements </rethrow>	<rethrow />

rethrow 活動沒有特別的屬性需要設定，因為該活動是搭配接收到的錯誤，再往上層送出。轉換為派翠網路時，將其轉為轉移狀態，並且加入該活動在文件中之範圍。轉換結果如圖 3-11 所示。

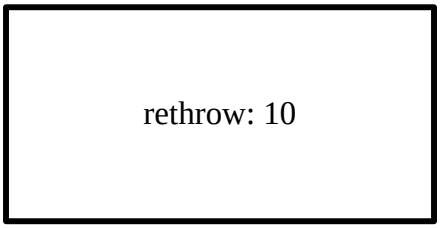

圖 3-11 rethrow 派翠網路圖

3.1.2 WSBPEL 2.0 結構化活動轉至派翠網路規則

WSBPEL 2.0 中結構化活動(Structured activities)，包含了 if、while、repeatUntil、pick、forEach、sequence 及 flow，但 sequence 與 flow 的性質與其他結構化活動不同，故於 3.1.3 再予以說明。所有結構化活動被轉換為派翠網路時，也必須包含該活動之範圍；結構化活動又分成條件式活動、循環式活動、選擇式活動及多重分支式活動四大類。

3.1.2.1 條件式活動

條件式(Conditional)活動，包含 WSBPEL 2.0 中之 if 活動，主要藉由動態執行的情況為條件，再決定商業流程之處理方式，以下將針對 if 轉換進行說明。

if 活動提供了可條件判斷的商業流程，可依照不同的情況建立流程的分支，有如一般程式語言，if 可搭配多個 else if 來進行分支定義。其語法規則和範例如表 3-11 所示。

表 3-11 if 語法規則與範例表

語法規則	範例
<if standard-attributes> standard-elements <condition expressionLanguage="anyURI"?>bool-expr</condition>	<if> <condition> $InsuranceRequest.insuredPersonData/ins:Age > 50 </condition>

activity <elseif>* <condition expressionLanguage="anyURI"?>bool-expr</condition> activity </elseif> <else>? activity </else> </if>	perform activities for age 51 and over <elseif> <condition> $InsuranceRequest.insuredPersonData/ins:Age > 25 </condition> perform activities for age 26-50 </elseif> <else> perform activities for age 25 and under </else> </if>

如上述範例，條件皆定義在 if 活動中的 condition，而 else if 也是如此。if 活動轉換為派翠網路時，if 活動本身以狀態表示，後續的 condition 則以轉移做表示，condition 的轉移須表明 condition 條件；依不同條件進入不同分支，分支活動以狀態與條件相連，分支處理結束，再以狀態搭配 if 結尾及其結尾的範圍，即完成轉換。轉換結果如圖 3-12 所示。

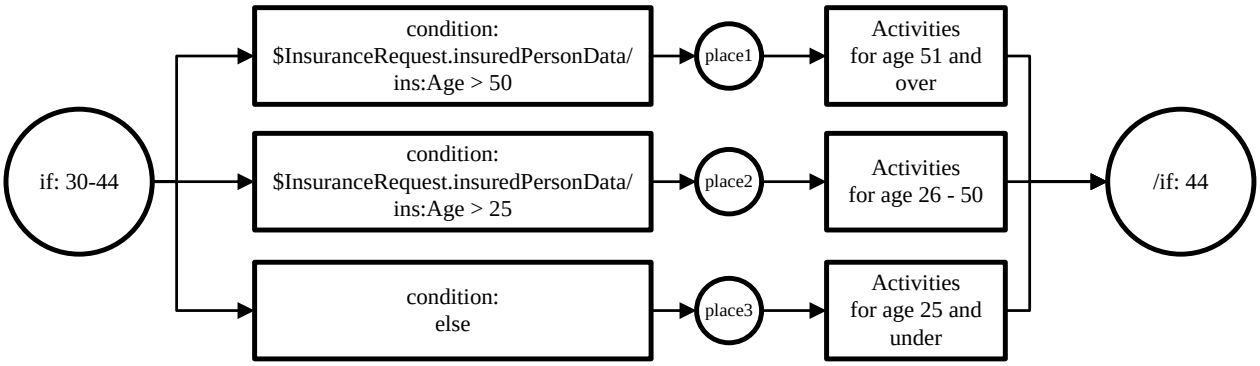

圖 3-12 if 派翠網路圖

3.1.2.2 循環式活動

循環式(Loops)活動包含了 WSBPEL 2.0 中的 while、repeatUntil 等活動。共通特性也是以 condition 來決定該活動是否重複進行。

(1) while 轉換

while 活動依照 condition 中的布林條件，決定其內部的處理活動，是否持續進行，其語法規則與範例如表 3-12 所示。

表 3-12 while 語法規則與範例表

語法規則	範例
<while standard-attributes> standard-elements <condition expressionLanguage="anyURI"?>bool-expr</condition> activity </while>	<while> <condition> $Counter < $NoOfPassengers </condition> perform activities </while>

上述範例，while 根據其下 condition 之定義條件，條件結果若是 TRUE 則進行後續活動，若結果為 FALSE 則結束 while 活動。轉換為派翠網路時 while 以狀態表示，且包含文件中之範圍；condition 則以轉移表示，搭配其 condition 條件；若條件結果正確之活動，以狀態與 condition 相接；不合之條件則以 while 結尾與其相接，while 結尾須包含其結尾在文件中之範圍。轉換結果如圖 3-13 所示。

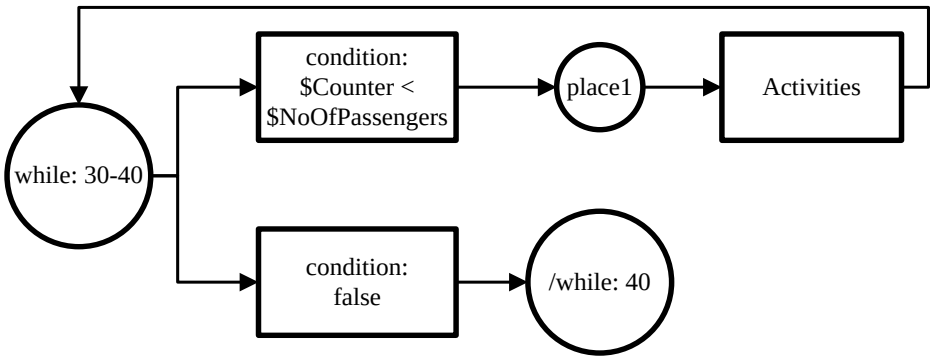

圖 3-13 while 派翠網路圖

(2) repeatUntil 轉換

repeatUntil 活動與 while 活動類似，以 condition 判斷之布林結果，決定活動是否持續進行，不同處在於 repeatUntil 是先執行活動，再判斷 condition 決定是否持續執行。其語法規則與範例如表 3-13 所示。

表 3-13 repeatUntil 語法規則與範例表

語法規則	範例
<repeatUntil standard-attributes> standard-elements activity <condition expressionLanguage="anyURI"?>bool-expr</condition> </repeatUntil>	<repeatUntil> perform activities <condition> $Counter <= $NoOfPassengers </condition> </repeatUntil>

依照上述範例，則是先進行活動，再判斷 condition 中之條件結果如何，再決定是否重複執行活動。轉換為派翠網路時，以狀態表示 repeatUntil 並包含其範圍，爾後進行的活動則視活動內容進行轉換，再以狀態接 condition，condition 以轉移搭配條件內容表示，最後結尾則以狀態表示 repeaetUntil，並搭配其在文件之範圍表示。轉換結果如圖 3-14 所示。

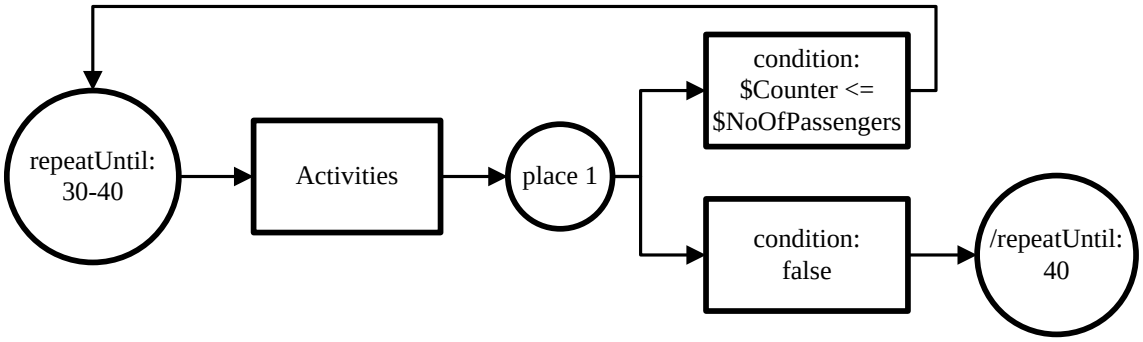

圖 3-14 repeatUntil 派翠網路圖

3.1.2.3 選擇式活動

選擇式(Selective)活動，類似於條件式活動，依照不同情況為商業流程建立不同的處理分支。在選擇式活動中包含了 WSBPEL 2.0 的 pick 活動，以下將針對 pick 轉換進行說明。

pick 活動是等待發生的事件，依結果進行不同的分支，包含了以 onMessage 傳入的事件或以 onAlarm 決定的時間發生點。語法規則與範例如表 3-14 所示。

表 3-14 pick 語法規則與範例表

語法規則	範例
<pick createInstance="yes|no"? standard-attributes> standard-elements <onMessage partnerLink="NCName" portType="QName"? operation="NCName" variable="BPELVariableName"? messageExchange="NCName"?>+ <correlations>?	<pick> <onMessage partnerLink="AmericanAirlines" portType="aln:FlightCallbackPT" operation="FlightTicketCallback"

```
                <correlation set="NCName"
initiate="yes|join|no"? />+
                </correlations>
                <fromParts>?
                <fromPart part="NCName"
toVariable="BPELVariableName" />+
                </fromParts>
                activity
        </onMessage>
        <onAlarm>*
                (
                <for
expressionLanguage="anyURI"?>duration-ex
pr</for>
                |
                <until
expressionLanguage="anyURI"?>deadline-ex
pr</until>
                )
                activity
        </onAlarm>
</pick>
```

```
variable="FlightResponseAA">
                <empty/>
                <!-- Continue with the rest
of the process -->
        </onMessage>
        <onMessage

partnerLink="AmericanAirlines"

portType="aln:FlightCallbackPT"

operation="FlightNotAvaliable"
                variable="FlightFaultAA">
                        <throw
faultName="trv:FlightNotAvaliable"
faultVariable="FlightFaultAA"/>
        </onMessage>
        <onAlarm>
                <for>'PT30M'</for>
                <throw
faultName="trv:CallbackTimeout" />
        </onAlarm>
</pick>
```

如上述範例，pick 活動總共定義了三個分支，兩個是由服務 AmericanAirlines 藉由 portType aln:FlightCallbackPT 執行的 operation FlightTicketCallback 及 FlightNotAvaliable 所決定，另一個為時間到了 30 分鐘後，會進行觸發。在 pick 中的兩個指令，onMessage 其屬性與基礎活動中的基本處理活動類似，藉由 partnerLink、portType 及 operation 所定，而 onAlarm 則與基礎活動中的 wait 類似，藉由指令 for 及 until 定義。

轉換為派翠網路時，pick 以狀態表示在文件中之範圍；而不同的分支觸發點皆以轉移狀態表示，如同 WSBPEL 2.0 基礎活動轉換為派翠網路後的結果相似，故 onMessage 轉換後須包含@partnerLink:portType (operation)，而 onAlarm 則須包含 for 或是 until 等指令內容；每個分支後續處理的活動，則以狀態與觸發點的轉移相連，最後再以狀態表示 pick 活動的結束。轉換結果如圖 3-15 所示。

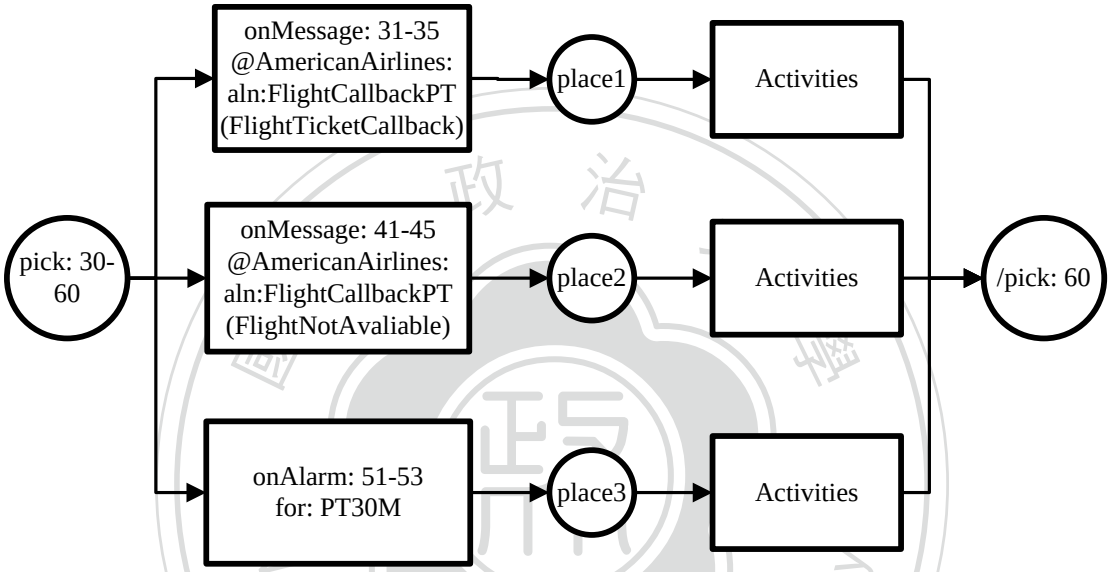

圖 3-15 pick 派翠網路圖

3.1.2.4 多重分支式活動

多重分支式(Multiple branches)活動，包含了 WSBPEL 2.0 中的 forEach 活動，類似於循環式活動，但執行條件的成立決定於要進行活動的次數，以下將針對 forEach 轉換進行說明。

forEach 的活動類型，非常類似於 Java 語言中的 for-each 迴圈。forEach 的執行決定在其下定義的 startCounterValue 及 finalCounterValue。其語法規則與範例如表 3-15 所示。

表 3-15 forEach 語法規則與範例表

語法規則	範例
<forEach counterName="BPELVariableName" parallel="yes|no" standard-attributes> standard-elements <startCounterValue expressionLanguage="anyURI"?> unsigned-integer-expression </startCounterValue> <finalCounterValue expressionLanguage="anyURI"?> unsigned-integer-expression </finalCounterValue> <completionCondition>? <branches expressionLanguage="anyURI"? successfulBranchesOnly="yes|no"?>? unsigned-integer-expression </branches> </completionCondition> <scope ...>...</scope> </forEach>	<forEach counterName="Counter" parallel="no"> <startCounterValue> number(1) </startCounterValue> <finalCounterValue> $NoOfPassengers </finalCounterValue> perform activities </forEach>

在上述範例中，定義的 startCounterValue 值還沒等於 finalCounterValue 值時，活動將會持續的執行，轉換為派翠網路時，forEach 以狀態的方式表示，並標示其在文件中之範圍；startCounterValue 及 finalCounterValue 則為 condition，以轉移表示並加上 startCounterValue 及 finalCounterValue 的條件；forEach 結束以狀態加上文件中之位置。轉換結果如圖 3-16 所示。

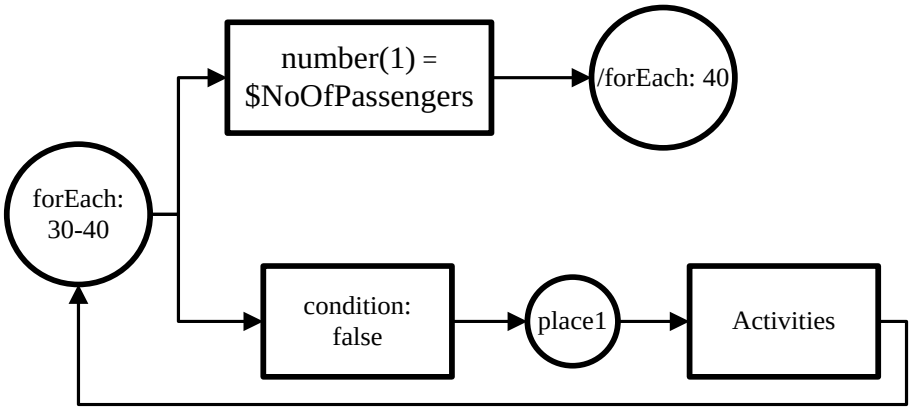

圖 3-16 forEach 派翠網路圖

3.1.3 WSBPEL 2.0 流程活動轉換至派翠網路規則

流程活動(Process activities)在 WSBPEL 2.0 中，屬於結構化活動，但其目的與其他結構化活動不同，故特別獨立出來討論。在一般商業流程的活動處理上，可分為循序處理或是並行處理，也就是 WSBPEL 2.0 的 sequence 及 flow，故流程活動包含了 WSBPEL 2.0 中的 sequence 及 flow，此外也針對 flow link 轉換進行說明。

(1) sequence 轉換

sequence 活動屬於循序活動類型，其定義在 sequence 活動之下的各種 WSBPEL 2.0 活動流程中，皆需依照定義的順序依序處理。其語法規則與範例如表 3-16 所示。

表 3-16 sequence 語法規則與範例表

語法規則	範例
<sequence standard-attributes> standard-elements activity+	<sequence> perform activity 1 perform activity 2

</sequence>	… perform activity n </sequence>

依據上述範例，定義在 sequence 中的所有活動，會依照次序處理，也就是會先處理活動 1、活動 2 直到定義的活動 n，全部處理完畢後 sequence 活動才會結束。轉換為派翠網路時，sequence 則以狀態表示，也需包含其 sequence 整塊活動在文件中之範圍；定義在 sequence 下的每個活動，都已轉移狀態表示，活動與活動之間以狀態相連結；sequence 結束時以狀態表示，並包含結束時之位置。轉換結果如圖 3-17 所示。

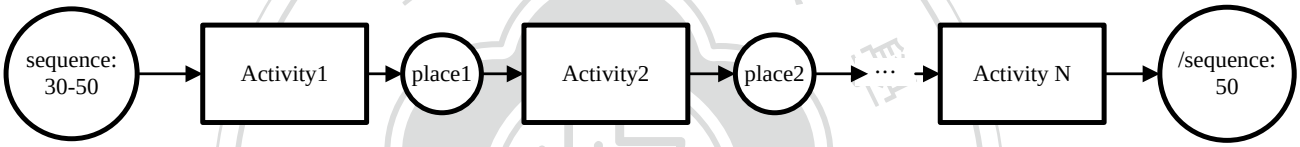

圖 3-17 sequence 派翠網路圖

(2) flow 轉換

flow 活動為並行(Concurrent)處理，定義在 flow 下的活動，會在 flow 活動開始執行時開始運作。語法規則與範例如表 3-17 所示。

表 3-17 flow 語法規則與範例表

語法規則	範例
<flow standard-attributes> standard-elements <links>? <link name="NCName" />+ </links> activity+ </flow>	<flow> perform activity A perform activity B </flow>

如上述範例，flow 活動會在開始時，儘快地讓之後定義的活動 A 與活動 B 開始進行。轉換為派翠網路時，flow 以轉移表示，並包含 flow 活動在文件中的所在位置；定義在 flow 下的活動則以轉移表示，活動與活動之間是並排的，而非循序式的，活動與 flow 之間以狀態做連結；flow 的結尾也以轉移做結束。其轉換結果如圖 3-18 所示。

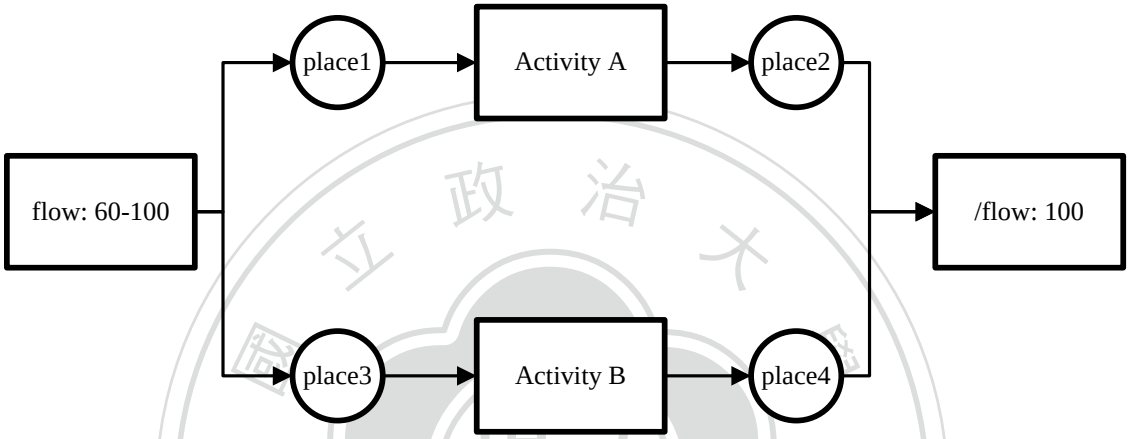

圖 3-18 flow 派翠網路圖

(3) flow link 轉換

所有在 flow 中的 WSBPEL 2.0 活動，皆有 link 的特性(Matjaz et al. ,2010)，link 需在 WSBPEL 2.0 商業流程中定義，而之後定義的商業流程活動才可以使用，link 提供了 flow 處理過程中，並行處理活動同步化的能力。其語法規則與範例如表 3-18 所示。

表 3-18 flow link 語法規則與範例表

定義語法規則	活動使用語法規則
<links> <link name="NCName"/> </links>	<source linkName="NCName"/> <target linkName="NCName" />
範例	

```
<flow>
    <links>
        <link name="A to B"/>
    </links>
    <Activity A>
        <source linkName="A to B"/>
    </Activity A>
    <Activity B>
        <target linkName="A to B"/>
    </Activity B>
</flow>
```

依照上述範例，在活動 A 中，設定了 link 的來源，活動 B 中，設定了 link 的目的。轉換為派翠網路時，link 以狀態表示，且包含定義的 linkName，由 source 的活動，連結至 link 的狀態，再由 link 狀態，連結至 target 活動，其餘的轉換就如同一般 flow 的轉換。其轉換結果如圖 3-19 所示。

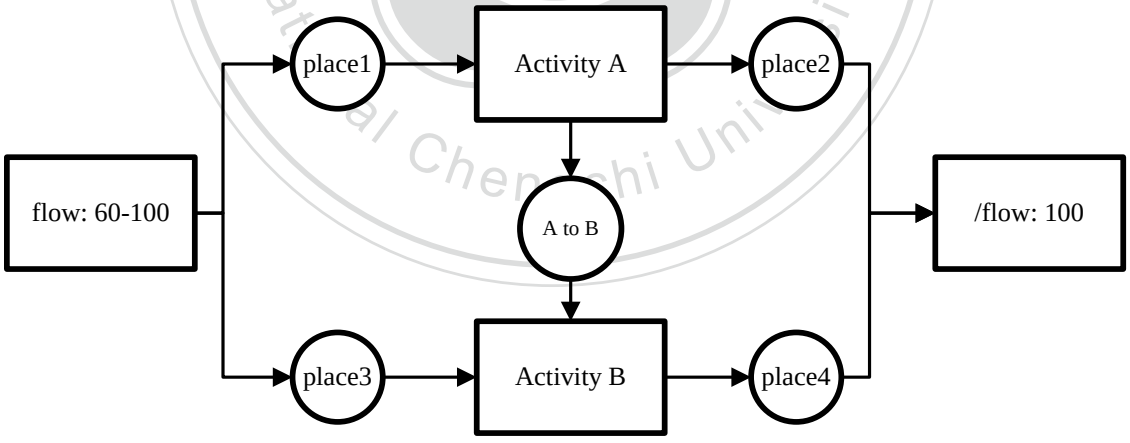

圖 3-19 flow link 派翠網路圖

3.2 WSBPEL 2.0 轉派翠網路流程及演算法

在 WSBPEL 轉換成派翠網路的流程中，使用本研究自行開發之 BPEL-PN 引擎(BPEL-PN Engine)，本引擎以文件物件模型(Document Object Model; DOM)為基礎。以下將分為兩小節介紹 BPEL-PN 引擎架構及引擎轉換過程，並且針對轉換演算法部份進行說明。

3.2.1 BPEL-PN 引擎

本研究採用以文件物件模型為基礎的 BPEL-PN 引擎，進行 WSBPEL 2.0 轉換成派翠網路，轉換完成後再進行派翠網路死結偵測。圖 3-20 是轉換引擎之系統架構圖。

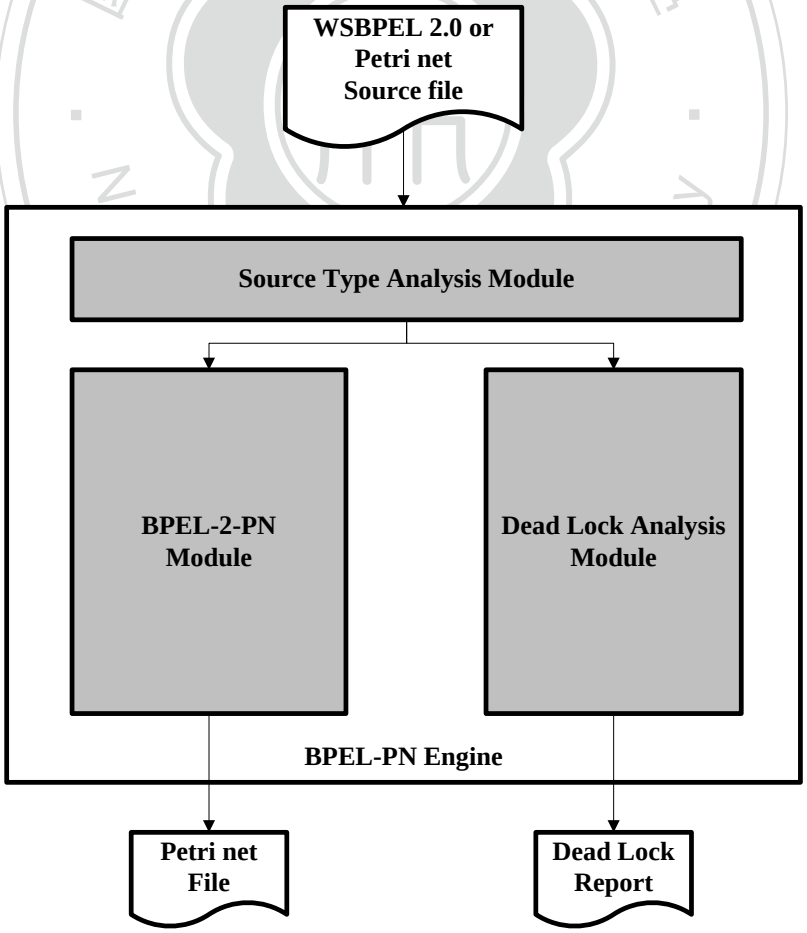

圖 3-20 轉換引擎之系統架構圖

BPEL-PN 引擎主要分成兩大部份，第一部份是針對 WSBPEL 2.0 轉換為派翠網路的部份，此模組稱為「BPEL-2-PN Module」；第二部份是死結偵測的部份，該模組稱為「Dead Lock Analysis Module」。當檔案輸入引擎時，會藉由 Source Type Analysis Module 自動偵測檔案類型，並且將檔案送入正確的模組進行轉換為派翠網路或是網路死結偵測；送入 BPEL-2-PN 模組中之 WSBPEL 檔案，將會轉換後輸出派翠網路，之後再送入另一模組，之後會針對該派翠網路，產生死結偵測報表，死結偵測的部份，將於下一節進行說明。首先先針對 BPEL-2-PN 模組進行說明。

此模組共分為三個處理器，分別是 Normalized Processor、Format Processor 以及 WSBPEL to Petri net Processor。Normalized Processor 的目的為檢查、產生具可讀性的文件，供後續處理使用，並精簡 WSBPEL 2.0 的文件，以增加後續處理效能；Format Processor 負責將 WSBPEL 2.0 中的活動範圍記錄下來，以及將各類活動進行分類；WSBPEL to Petri net Processor 則負責處理 WSBPEL 2.0 轉換到派翠網路的過程。除此之外，為增加系統擴充性，還搭配兩個管理器，分別是 Parser Registration Manager，與 WSBPEL to Petri net Processor 搭配的 Converter Manager。Parser Registration Manager 會針對 WSBPEL 的文件，將延伸標記語言文件轉換為相對應的 WSBPEL 2.0 各類活動；Converter Manager 則是將每個 WSBPEL 2.0 活動的細節，轉為派翠網路的每個節點。圖 3-21 BPEL-PN 是引擎架構圖。

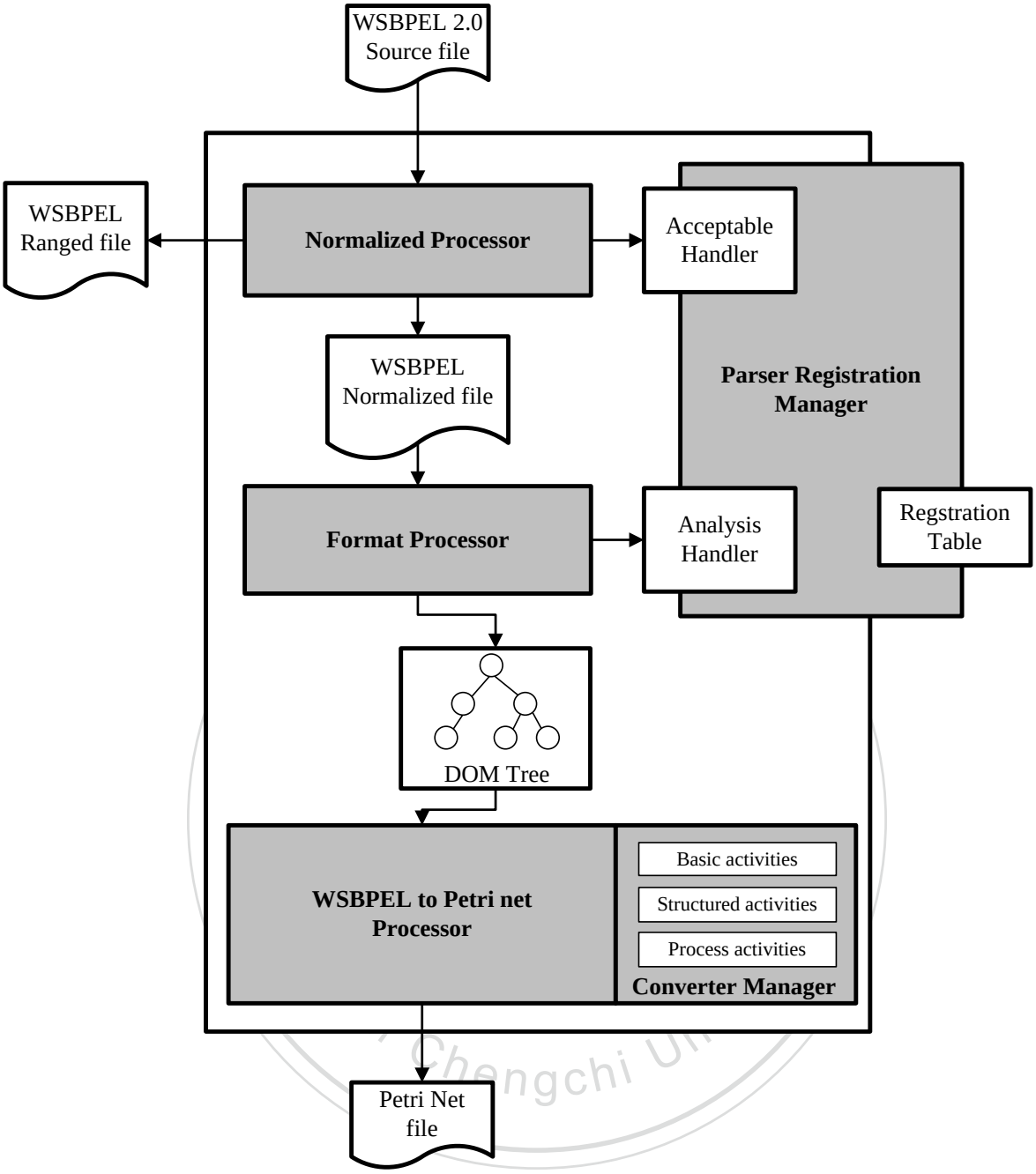

圖 3-21 BPEL-PN 引擎架構圖

首先，輸入 BPEL-2-PN Module 的 WSBPEL 2.0 文件，會先由 Normalized Processor 進行正規化處理，將文件中不正確的換行、冗餘的文字與之後轉換時無關的內容先進行濾除，之後會產生第一份文件，存於執行引擎的主機上，稱為 WSPBPEL Normalized file，此檔案的內容經過了精簡程序，可增加後續轉換的效能，並減少轉換過程中出現錯誤的機會。

決定文件何處精簡，是由 Parser Registration Manager 判斷而定，本節剛開始有提到，BPEL-PN 引擎是以文件物件模型為基礎來處理的，而 Parser Registration Manager 中的 Acceptable Handler，將會針對 Normalized Processor 在處理的每一個文件物件模型節點，進行是否處理的判斷，Acceptable Handler 將節點送入註冊的剖析器中進行檢驗，而剖析器內部有哪些是可接受的指令，可藉由 Registration Table 來進行註冊，有註冊過的才會處理，因此，企業內部若有特殊定義命令需要處理，系統會變得比較有彈性。

除此之外，依照上一節的內容可知道，WSBPEL 2.0 轉換為派翠網路後，每一個轉移和狀態，都會包含最基本 WSBPEL 2.0 活動的範圍；故產生第二個目的檔案，稱為 WSBPEL Ranged file，提供使用者後續若有需要藉由 WSBPEL 2.0 轉換為派翠網路的文件，進行問題查詢時，可以藉由派翠網路中的轉移和狀態呈現的資訊，反向查閱本檔案以方便找到問題的範圍。

接下來 WSPBPEL Normalized file 檔案，會在引擎內部直接送入 Format Processor，在 Format Processor 中會取得每一個 WSBPEL 2.0 中活動的範圍，並記錄於活動的根節點中；此外，依上一節的說明，WSBPEL 2.0 的活動分成基礎活動、結構化活動以及流程活動。所有的 WSBPEL 2.0 活動在經過 Format Processor 時，將會送入 Parser Registration Manager 中進行分類。

經過 Format Processor 後，會產生儲存於記憶體中的文件物件模型二元樹，此物件將送入 WSBPEL to Petri net Processor 中，進行 WSBPEL 2.0 轉換為派翠網路的處理。在轉換過程中，WSBPEL to Petri net Processor 會走訪文件物件模型節點中的每個節點，並送入 Converter Manager 進行判斷，依照不同的活動類型，再送入不同的活動轉換器中進行轉換，轉換細節將於下一部份來說明演算法；經過 WSBPEL to Petri net Processor 後，便會產生轉換成功的派翠網路文件。

3.2.2 WSBPEL 2.0 轉派翠網路演算法

以下 WSBPEL 2.0 轉派翠網路，將以虛擬碼的方式說明演算法，其內容屬於 BPEL-PN 引擎的 WSBPEL to Petri net Processor 部份，說明如下。

首先為主流程的部份,首先為最後要轉出的派翠網路建立其根節點 PNML，並且建立狀態表示 process 的開始。接下來將經過 BPEL-PN Engine Format Processor 處理過的根節點，取出子節點的列表，送入 Parse()中進行處理；當整個處理完畢後，針對 Link 建立連結；最後針對 process 的結尾建立狀態，再將 PNML 輸出成為檔案。主流程演算法範例如表 3-19 所示：

表 3-19 主流程演算法範例表

Function	**Main flow**
Description	Input: WSBPEL, root of DOM tree Variable: PNML, root of Petri net Ps, place of Petri net to represent the start of process Pe, place of Petri net to represent the end of process

Algorithm	Main()
	{
	Create PNML as petri net root;
	Create place, Ps, as the start of process;
	PNML.add(Ps);
	Parse(WSBPEL.getChildList(), PNML); // Traversal tree
	Link();
	Create place, Pe, as the end of process;
	Nn = PNML.getLastNode();
	if (Nn == “transition”)
	{

	Create transition Tn;
	PNML.add(Tn);
	Create arc link from Nn to Tn;
	Create arc link from Tn to Pe;
	}
	else
	{
	Create arc link from Nn to Pe;
	}
	PNML.add(Pe);
	Output(PNML);
	}

Parse 功能針對傳入的子節點列表，進行循環處理，針對每一個節點在轉換前，先由 BPEL-PN 引擎判別節點的類別，自 ConverterManager 中，再藉由節點類別取得轉換器之形式，並呼叫 convert 進行轉換。如表 3-20 所示，為遞迴剖析演算法範例表。

表 3-20 遞迴剖析演算法範例表

Function	**Traversal Parse**
Description	Input: NL0, Node List PNML, root of Petri net Variable: Bt, WSBPEL Type

Algorithm	Parse(NL, PNML)
	{
	if (validate(NL))
	{
	For each node in NL

	{
	Bt = node.getWSBPELType();
	converterManager(Bt).convert(node); // Main convert flow in here
	}
	}
	}

呼叫 ConvertManager 中的 convert 後，會依照本研究前述的規則，將 WSBPEL 2.0 依活動的分類進行剖析，首先是基礎活動，在基礎活動中，針對傳入的節點建立轉移以表示該活動，建立目前 PNML 中，最新的節點與轉移間的連線。除此之外，flow 中的 Link 將會出現在基礎活動中，所以針對基礎活動的每個子節點進行查驗，找出 Link 的 Source 與 Target 儲存於 LinkMap 中，待後續處理。如表 3-21 所示，為基礎活動轉換演算法範例表。

表 3-21 基礎活動轉換演算法範例表

Function	**Basic activities of WSBPEL convert**
Description	Input: N, current Node PNML, root of Petri net Variable: LinkMap, the map holds source and target of link Nn, the last child of PNML

Algorithm	basicActivitiesConvert(N, PNML)
	{
	Nn = PNML.getLastChild();
	Create transition T as N in Petri net;
	Create an arc to link from Nn to T;
	if (Nn == "transition")
	{

```
            Create place Pn;
            Create arc link from Nn to Pn;
            Create arc link from Pn to N;
            PNML.add(Pn);
        }
        else
        {
            Create arc link from Nn to N;
        }
        PNML.add(T);
        Get children nodes list NL of N;
        For each node of NL
        {
            if (node.getNodeName() == "sources")
            {
                For each source of sources
                {
                    Get attribute linkName value of source, Ls;
                    LinkMap.put(Ls, T);
                }
            }
            else if (node.getName() == "targets")
            {
                For each target of targets;
                Get attribute linkName value of target, Lt;
                LinkMap.put(Lt, T);
            }
        }
    }
```

處理結構化活動時，這裡以 pick 為例子，其他類似或相同的活動，皆以同樣模式進行處理。首先先建立兩個狀態，以表示結構化活動的開始與結束，並且建立 PNML 最後一個節點與開頭狀態的連結進行連接。pick 中每一個分支都建構於 pick 其下子節點定義的 onMessage 與 onAlarm 中，故需走訪 pick 的子節點，取得 onMessage 或 onAlarm 的集合。針對兩者建立轉移，並且建立 pick 的狀態與此轉移連結，至於 onMessage 或 onAlarm 觸發後要處理的活動，則將 pick 子節點的子節點列表，再傳入 Parse()中進行遞迴處理。最後將遞迴處理後的子節點與 pick 結尾的狀態相連結。流程如表 3-22 所示。

表 3-22 結構化活動轉換演算法範例表

Function	**Structured activities convert**
Description	Input: N, current Node PNML, root of Petri net Variable: LinkMap, the map holds source and target of link Nn, Nnn, the last child of PNML Nn1, the first child of PNML

Algorithm	structuredActivitiesConvert(N, PNML)
	{
	Nn = PNML.getLastChild();
	if (N.getNodeName() == "pick")
	{
	Create place Ps as start of pick
	PNML.add(Ps);
	Create place Pe as the end of pick
	Create arc link from Nn to Ps;
	NL = N.getChildNodes();

```
For each node of NL
{
    if (node.getNodeName() == "onMessage" ||
        node.getNodeName() == "onAlarm")
    {
        Create transition Tn as onMessage;
        PNML.add(Tn);
        Create arc link from Ps to Tn;
        Parse(node);
        Nn1 = PNML.getFirstChild();
        Create arc link from Tn to Nn1
        Nnn = PNML.getLastChild();
        Create arc link from Nnn to Pe
    }
}
PNML.add(Pe);
}
}
```

在流程化活動部份，若是 sequence，則先建立兩個狀態，分別表示開始與結束，並且建立開始狀態與 PNML 中最後節點的連結；再針對 sequence 底下的子節點進行遞迴 Parse()處理，最後再將 PNML 處理完遞迴後的最後節點，與 sequence 結尾的狀態相連即完成。

若是處理 flow 的部份，首先先建立 flow 開始和結束的轉移，並連結開始的轉移與 PNML 目前最後的節點。接下來每個 flow 中的子節點處理，其中的 link 定義且於 LinkMap 中，並進行註冊；除此之外，再用遞迴尋訪的方式 Parse()每一個節點，最後將 PNML 中最後的節點，與 flow 結尾的轉移連結。過程如表 3-23 所示。

表 3-23 流程活動轉換演算法範例表

Function	Process activities convert
Description	Input: N, current Node PNML, root of Petri net Variable: LinkMap, the map holds source and target of link Nn, Nnn, the last child of PNML

Algorithm	processActivitiesConvert(N, PNML)
	{
	if (N.getNodeName() == "sequence")
	{
	Nn = PNML.getLastChild();
	Create a place Ss as the start of sequence;
	if (validate(Nn))
	{
	if (Nn == "place")
	{
	Create place P2;
	PNML.add(P2);
	Create arc link from Nn to P2;
	Create arc link from P2 to Ss;
	}
	else
	{
	Create arc link from Nn to Ss;
	}
	}
	PNML.add(Ss);
	Parse(N.getNodeList, PNML);

```
                Create a place Se as the end of sequence;
                Create an arc to link from Ss to the first node of N;
                Create an arc to link from the last node in PNML to Se;
                PNML.add(Se);
            }
            else if (N.getNodeName() == "flow")
            {
                Nn = PNML.getLastChild();
                Create a transition T1 as the start of flow;
                PNML.add(T1);
                Create a transition Tn as the end of flow;
                Create arc link from T1 to Nn;
                NL = N.getNodeList();
                For each node of NL
                {
                    if (node.getNodeName() == "links")
                    {
                        Get child node named link and get attribute name of link L;
                        LinkMap.createIndex(L);
                    }
                    Create a place P1 as the start of branch;
                    Create an arc to link from T1 to P1;
                    Parse(node, PNML);
                    Create a place Pn as the end of branch;
                    Nnn = PNML.getLastChild();
                    Create arc link from Nnn to Pn;
                    Create arc link from Pn to Tn;
                }
                PNML.add(Tn);
```

	} }

在上述的過程中，都是將 Link 先註冊以及紀錄，最後一次處理。針對 LinkMap 中的每一個項目，取得 Source 及 Target，針對 Link 建立狀態並且將 Source 及 Target 間相互連結。見表 3-24 所示。

表 3-24 連結處理演算法範例表

Function	**Link**
Description	Input: LinkMap, link relations keeped in this map Variable: Source, source node of each link item Target, target node of each link item

Algorithm	Link()
	{
	For each item of LinkMap
	{
	Source = item.source;
	Target = item.target;
	if (validate(Source) & validate(Target))
	{
	Create place P named item.key;
	Create arc link from Source to P;
	Create arc link from P to Target;
	PNML.add(P);
	}
	}
	}

3.2.3 分析網路死結模組

在 BPEL-PN 引擎中，另一項功能是偵測派翠網路的死結，此為引擎中另一個獨立的模組，其模組架構圖如圖 3-22 所示。

首先先將派翠網路檔案送入 Node Inflator 中進行處理，如第二章所提及，派翠網路本身並無階層性或關聯性，也就是派翠網路是照著指定的屬性設定而連結起來，這對於死結偵測並無幫助，故先依派翠網路文件中之敘述，建立起所有應建立的節點(Node)，而節點分為兩種形式，分別是狀態式節點以及轉移式節點。而產生的節點暫時並無關聯性，並分散於記憶體中，再進入下一階段處理。

經過 Node Inflator 處理後的結果，接下來送入 Process Manager，Process Manager 會將散落在記憶體中的節點，進行關聯與連結，而其關聯性是依派翠網路中，定義之連線而決定。此外，會依照每個節點的輸入數和輸出數，進行判斷與連結，找到起始點(Begin Node)及結束點，起始點為含有標記之節點，並且只有一個輸入點進行權杖歸還，而結束點則是有一輸出的連線，連回開始點連結的地方，形成一個大的環形結構，在決定開始及結束點後，以此為基準找出中間相連的基本流程(Basic Process)。

具有關聯性的節點列表，會由 Process Manager 送入虛擬流程(pseudo process; PSP)的管理模組 PSP Candidate Manager 進行檢測。所有的 PSP Candidate，開始節點都必須是大於單一的輸出，結束點都須為至少兩項輸入，藉由這樣的原則找出所有的候選(Candidate)路徑。再藉由虛擬流程判斷的規則，進行虛擬流程路徑的篩選，找出符合原則的虛擬流程，藉由這些虛擬流程來進行最後的死結偵測。

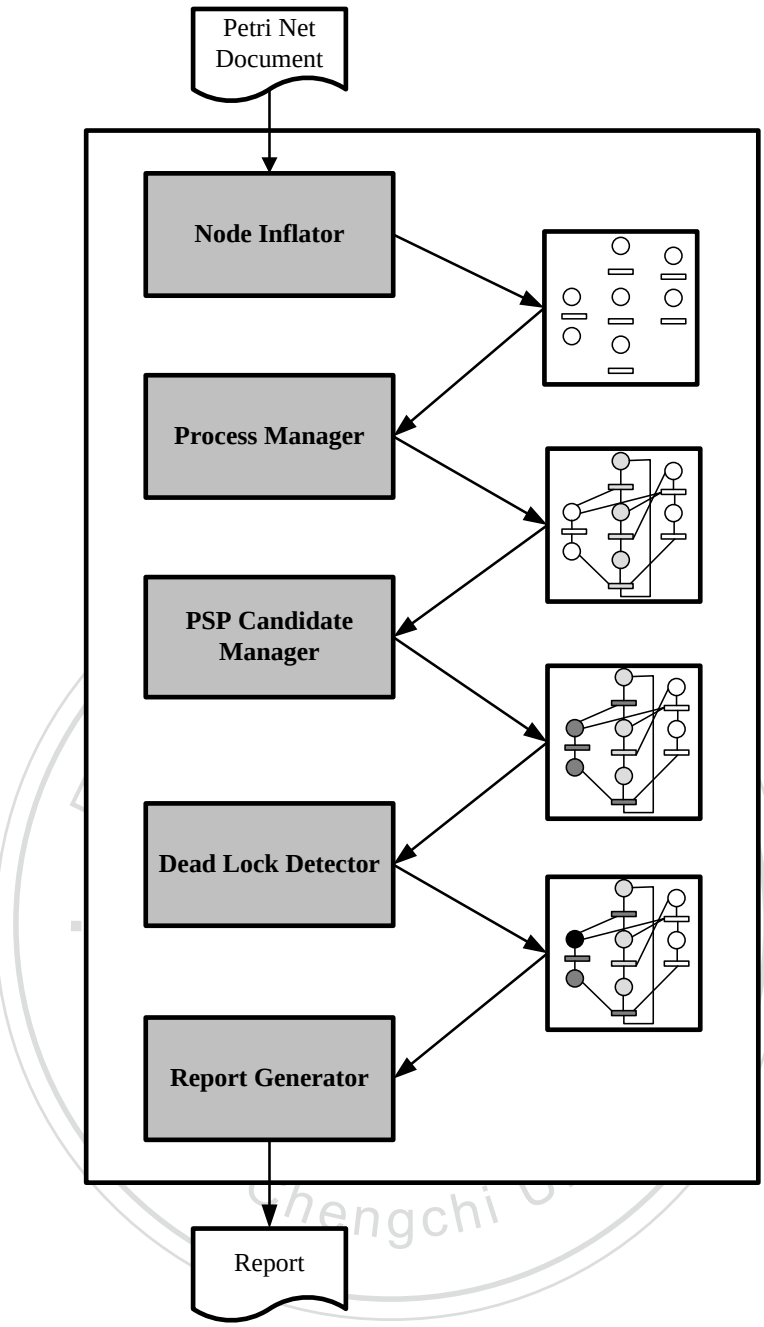

圖 3-22 網路死結分析模組架構圖

篩選後的虛擬流程以及節點列表，則會送入 Dead Lock Detector 中進行死結偵測，死結偵測的原則將透過針織技術來分析。最後死結偵測的結果，會藉由 Report Generator 產生報表告，以呈現網路狀態與特徵。

3.3 針織技術

派翠網路領域中的針織技術(Knitting technique)是種簡單、有強大能力的技術，而且可以整合在系統管理工具中，自動地判斷與分析出網路狀態。它組成的派翠網路，是一種同步選擇(Synchronized Choice; SC)網路。如果一個設計的派翠網路不是同步選擇網路，那麼它很有可能會出現死結或無界限的設計錯誤。而且，同步選擇網路的分析，一般需要多項式時間。但其應用在簡單且基礎的死結結構上，有不錯表現，且同時考量了網路結構與可以達到的狀態。

大型系統所建構的派翠網路模型，所遭遇到的問題就是複雜度很高的可達性分析。而針織技術的發展(Chao, 1992; Chao, 1994a, b, e, f; Chao, 1995; Chao, 1997a; Chao, 2001a, b)找到了一些簡單而有效的規則，可以分析合成的派翠網路所需的條件與屬性。在早期的編織技術，只處理兩個轉移之間(Transition to Transition; TT)的循序或並行處理，和兩個狀態之間(Place to Place; PP)的循序或互斥程序，所合成的網路，但是，在某些限制條件下，分析是很困難的。例如，在兩個轉移之間不允許其他路徑的出現，或產生兩個互斥且循序進行的流程(Zhou, 1991)，使得這兩個轉移從並行的型態，變成為循序的型態，導致轉換結果與原始狀態的情境無法對應符合。

文獻中常見的網路分析規則有很多，例如由上往下(top-down)或是由下往上(bottom-up) (Chen, Tsai and Chao, 1993; Datta and Ghosh, 1984; Silva,1985, Zhou, 1989)，對等實體生成法(peer entity generation) (Ramamoorthy, 1985)和針織法(Chao, 1999; Chao et al. 1994a, b, c; Chao and Wang, 1997)，本研究將針對針織法進行探討與應用。

接下來先針對分析過程中需要參考的定理，從相關文獻中整理出 16 個相關定義，其基本定義如下：

定義 1：同步距離(synchronic distance)，在 t_1 和 t_2 兩個轉移之間，在單一個派翠網路中，N 定義為 $d_{1,2} = \text{Max}\{|\sigma(t_1) - \sigma(t_2)|\}$，其中 σ 表示觸發的順序，且 $\sigma(t)$表示 σ 發生的時間值。d(W, V)是 W 集合中，轉移的最大觸發值，V 集合是沒有被觸發的轉移。

在圖 3-23 中，圖(a)表示一個條件狀況良好的派翠網路；圖(b)表示 PT 生成網路[p_3 t_3 p_4 t_5]；圖(c)表示加入一個調節迴路(regulation circuit, RC) [t_4 p_7 t_3 p_6]後，再進行同步，就會轉變成一個條件狀況良好的派翠網路。

在圖 3-23(c)中，$d_{3,4} = 1$，因為t_d^3可以比t_d^4多被觸發一次。

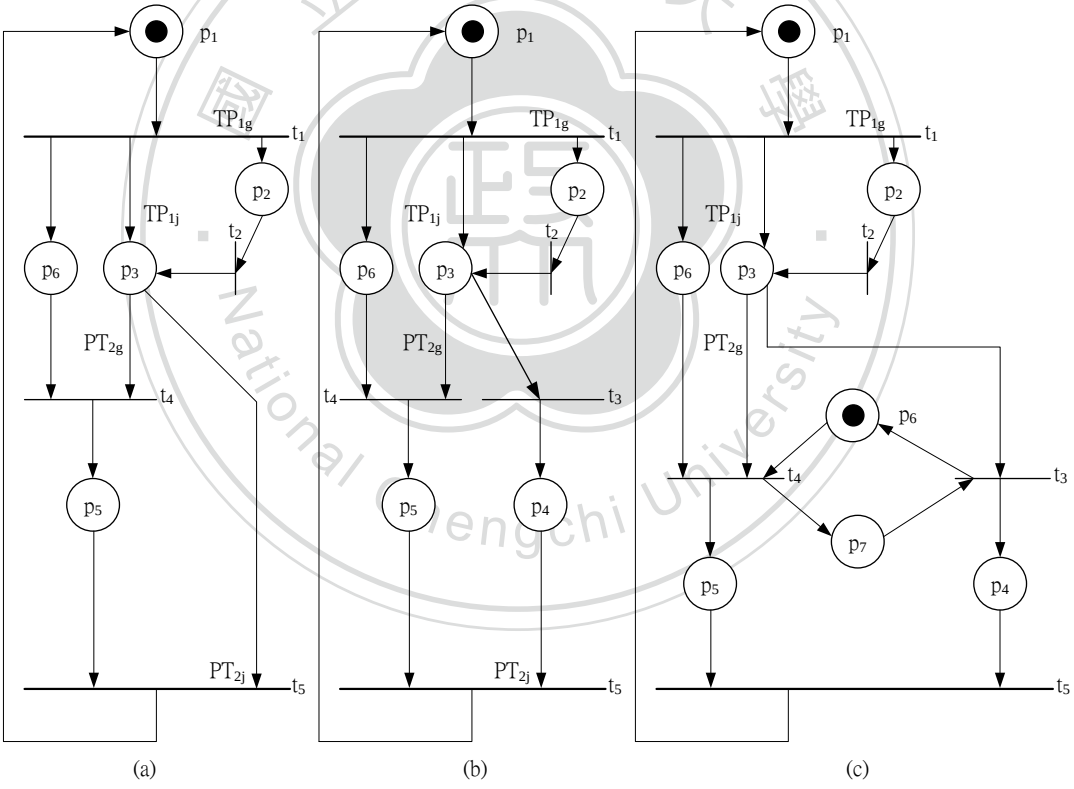

圖 3-23 不同觸發狀況的派翠網路圖

定義 2：起始位置(Home place, p_h)，t_α 是一個輸入的狀態，也是一個派翠網路觸發的最原始起始點。

定義 3：基本流程(basic process)在派翠網路中，這是一個很特別的一個路徑，其中(1) $\forall t \in T$，$|{}^\bullet t| = |t^\bullet| = 1$，且$\forall p \in P$，$|{}^\bullet p| = |p^\bullet| = 1$；(2)起始位置 p_h 是帶有有標記的。

圖 3-23 (a)中，p_1 是一個起始位置，而[p_1 t_1 p_3 t_4 p_5 t_5 p_1]是一個基本流程，這是組合的第一步驟，若一個條件狀況良好的派翠網路中，能從它的基本流程開始，接下來的分析操作，就不需要太多的步驟，可以編織出大且複雜的派翠網路。在編織技術中基本的元素，就是派翠網路的虛擬流程接下來，將定義虛擬流程。

定義 4：虛擬流程 Π。在派翠網路中，一個有方向性的基本路徑上，其任何一個節點(包含轉移或狀態)，除了它的兩個終端節點，只有一個輸入節點和輸出節點；起始節點被定義為生成點 g_Π，它呈現的是一個分岔的狀態，結束節點是一個從分岔後再結合起來的節點，即稱為 j_Π。

圖3-23(a)中，路徑$\Pi_1 = [t_1\ p_2\ t_2\ p_3]$，這段路徑就是一個虛擬的流程，可以用寫成$g_{\Pi 1} = t_1$，$j_{\Pi 1} = p_3$。接下來，將定義另一種特別的虛擬流程稱為虛擬PSP。

定義5：虛擬PSP(virtual PSP, VP)。簡單來說，它是兩個節點的虛擬流程。圖3-23 (a)中，路徑[p_3 t_5]就是一個虛擬的PSP。

定義6：在虛擬流程中Π_1，…， Π_i，其首要(prime) t_s(或是p_s)，可以是一個轉移(或是狀態)，假設t(或p)是有方向性的路徑，開始轉移(或狀態)的節點，Γ_1(包含Π_1)，…， Γ_i(包含Π_i)，除了轉移(或是狀態)之外，其他節點是不共用的。

編織技術的合成規則，其之間的關係是取決於相關的虛擬流程。接下來將定義兩個虛擬流程之間的結構關係。

定義7：Π_1和Π_2兩個虛擬流程之間的結構關係，其中Π_1和Π_2的關係有下列三種定義：

(1)互斥(Exclusive)：若Π_1和Π_2是互斥關係(縮寫為'EX')，標記為$\Pi_1 \mid \Pi_2$，

假設∃ Π_1和Π_2的首要p_s。

(2) 並行(Concurrent)：Π_1和Π_2是相互並行的(縮寫為‘CN’)，標記為$\Pi_1 \parallel \Pi_2$，假設(a)∃ Π_1和Π_2的首要t_s，或是(b)Π_1和Π_2是分別是在兩個不同的子網路中，是(a)或(b)其中的一種。

(3) 循序(Sequential)：假設Π_1和Π_2既不是互斥也不是並行，假設(a)$j_{\Pi 1} = g_{\Pi 2}$或(b)$\forall p \in \Gamma$，是(a)或(b)其中一種，從Π_1到Π_2的路徑中，$p \neq p_h$，則在循序的狀況下，Π_1比Π_2早時(sequentially earlier) (縮寫為‘SE’) ，標記為$\Pi_1 \rightarrow \Pi_2$，而當循序的狀況時Π_2比Π_1晚時(sequentially later)(縮寫為‘SL’)，標記為$\Pi_2 \leftarrow \Pi_1$。

圖 3-24 是 TT 和 PP 的規則範例，規則 TT.3 其中必須同時考慮 TP(例如$[t_2\ p_4]$)和 PT(例如$[p_5\ t_7]$)的生成；圖(a)表示途中可以發現幾條 TP 生成的路徑$[t_6\ p_4]$，且當 $t_2 \mid t_6$ 時，規則 TT.3 必須考慮所有在 LEX 的轉移，例如 $\text{LEX}(t_2, p_4) = \{t_2, t_6\}$。圖(b)表示$\forall p \in \text{LCN}(\text{TP}_{1g}, \text{PT}_{2g})$ (= $\{p_5, p_8\}$)，必須有一條 VPT 從 p_5 到 PT_{2j} 符合 PP.2 的每一個規則，否則將會造成 p_s 變成標記數量無上限的狀況；圖(c)表示，過程中必須考慮所有的 $t(p_f)^\bullet$，圖示是一個反例，途中沒有 p_v 給 $t_8 \in (p_f = p_8)^\bullet$。

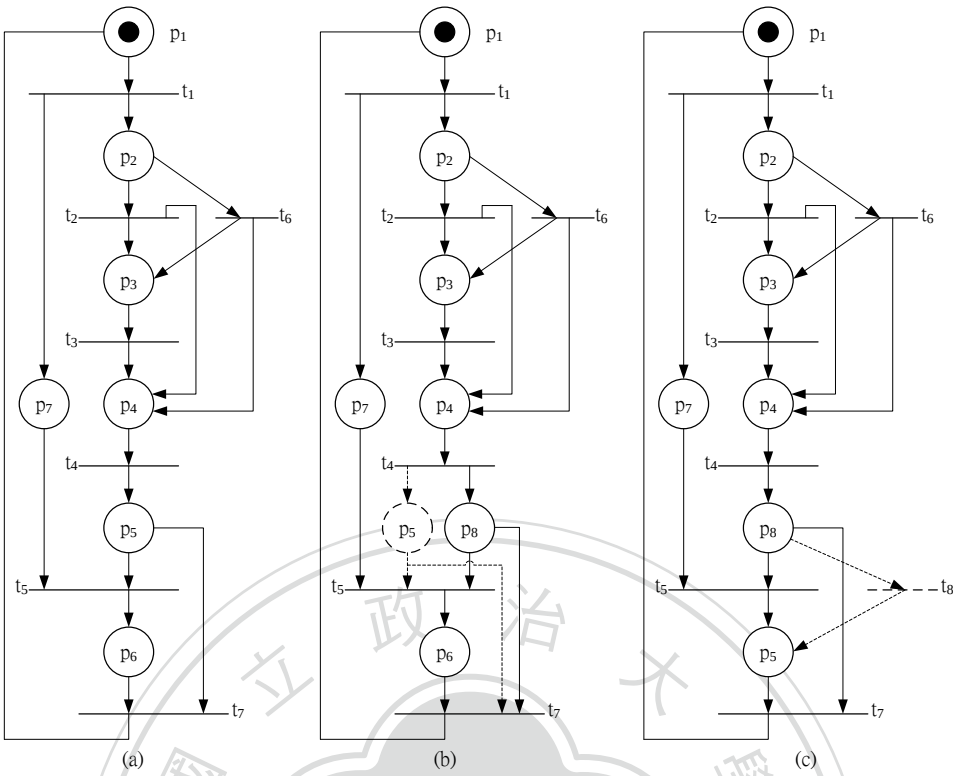

圖 3-24 TT 和 PP 規則範例圖

定義8：當PSP Π_i在PP迴圈當中時，假設$\exists\ \Pi$，則Π o Π_i。

要特別說明的是Π_1和Π_2也相互形成迴圈，標記為Π_1 o Π_2，假設$\Pi_1 \rightarrow \Pi_2$或是$\Pi_2 \rightarrow \Pi_1$其中任何一種狀況，在這情境中，派翠網路是用來描述的模型，並假設它是不斷地循環操作。

在圖3-24(a)中，[t_1 p_7 t_5]和[t_1 p_2]共用了同樣的首要$t_s = t_1$，也同時是相互並行的。

在圖3-25(a)中，$\Pi_1 = [p_4\ t_3]$，且$\Pi_2 = [p_4\ t_4]$，p_4是首要的p_s，且$\Pi_1 \mid \Pi_2$。

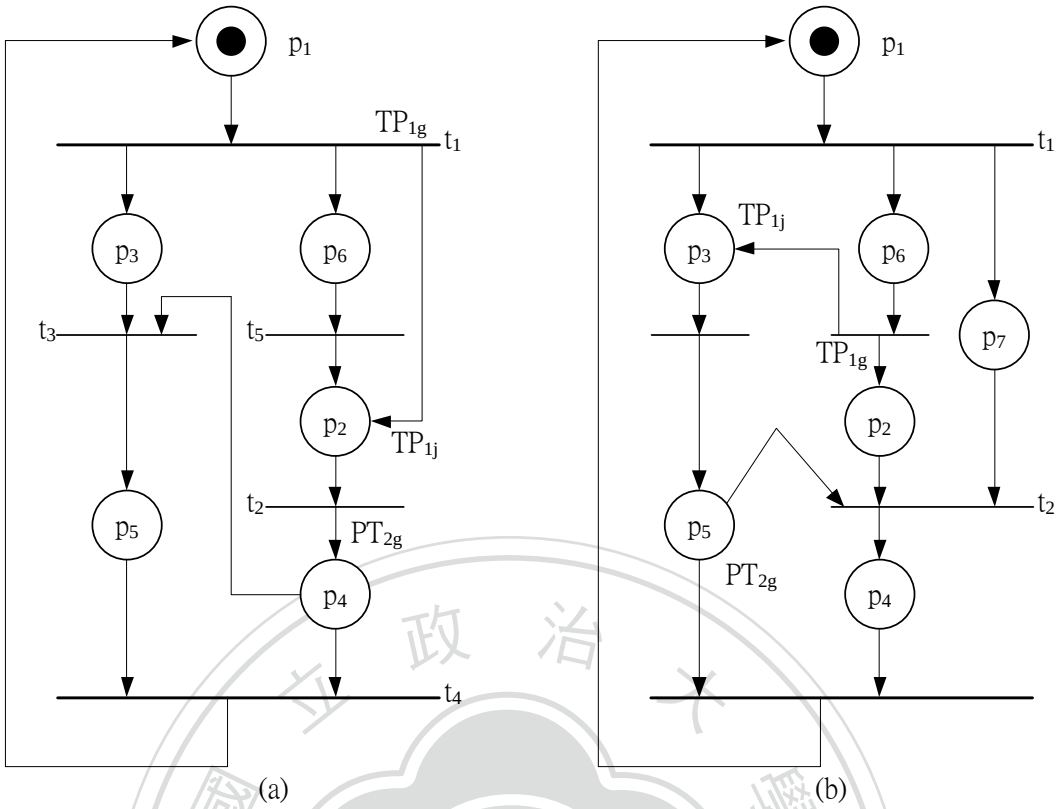

圖 3-25 TP 生成的特殊關係‘SQ’和‘CN’範例圖

在圖 3-25 中，圖(a)表示 TP_{1j} ‘SQ’ TP_{1g}；圖(b)表示 TP_{1j} ‘CN’ TP_{1g}。

在圖3-26(a)中，$\Pi_1 = [t_4\ p_7\ t_6\ p_1\ t_1]$且$\Pi_2 = [t_1\ p_2]$，這些都是在同一個基本迴圈中，其中也包含了$\Pi_1$和$\Pi_2$；因此$\Pi_1 \rightarrow \Pi_2$。

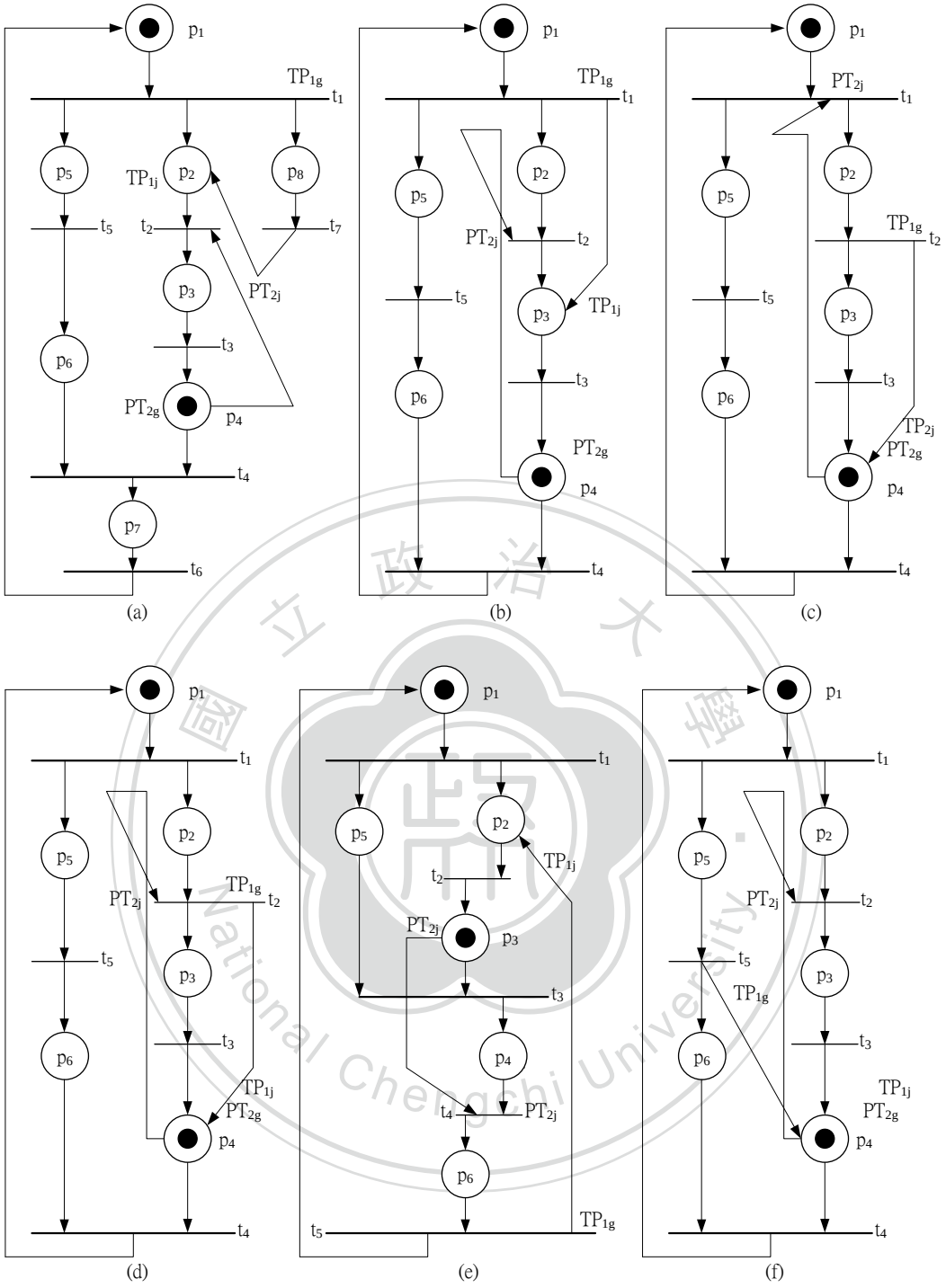

圖 3-26 TP與PT生成的特殊情況範例圖

在圖 3-26 中，圖(a)表示 PT_{2j} 'SL' TP_{1g} 且 PT_{2j} 'SE' PT_{2g}，這是反向生成；圖(b)表示當 t_1 被觸發後，TP 路徑的標記，干擾了 PT 路徑，$TP_{1g} = t_1$ 到 $TP_{1j} = p_3$；圖(c)表示 PT_{2j} 'SE' PT_{2g}，這也是反向生成。$\forall p_v (= p_6)$，$\neg(p_v \parallel P\ T_{2j})$因為 $p_6 \leftarrow t_1$，這是一個條件狀況良好的網路。圖(d)表示 PT_{2j} 'SE' PT_{2g}，這是一個反向生成，

$p_v = p_6$，$(p_v \parallel PT_{2j} = t_2)$。最初的標記是在 p_4，當 t_4 被觸發後 p_4 的標記會被帶走，這時候 t_2 會陷入死結，無法再提供標記到 p_4；圖(e)表示另一個反向生成 TPPT，其 $PT_{2j} \rightarrow TP_{1g}$，這情況網路是活的，在$\neg(p_v \parallel PT_{2j})$。$p_v = p_5$，$PT_{2j} = t_4$ 的條件下，在 t_5 被觸發後，會回補一個標記到 p_3；圖(f)表示另一個反向生成 TPPT，$PT_{2j} \rightarrow TP_{2g}$ 和 $PT_{1j} \parallel TP_{2g}$，與圖(d)不同的是，這情況網路也是活的$(p_v \parallel PT_{2g})$，$p_v = p_6$，$PT_{2j} = t_2$ 在 t_5 被觸發後，會回補一個標記到 p_4。

在圖3-27 (a)中，轉移t_2和t_7是互斥的，但狀態p_2和p_5是相互並行的。

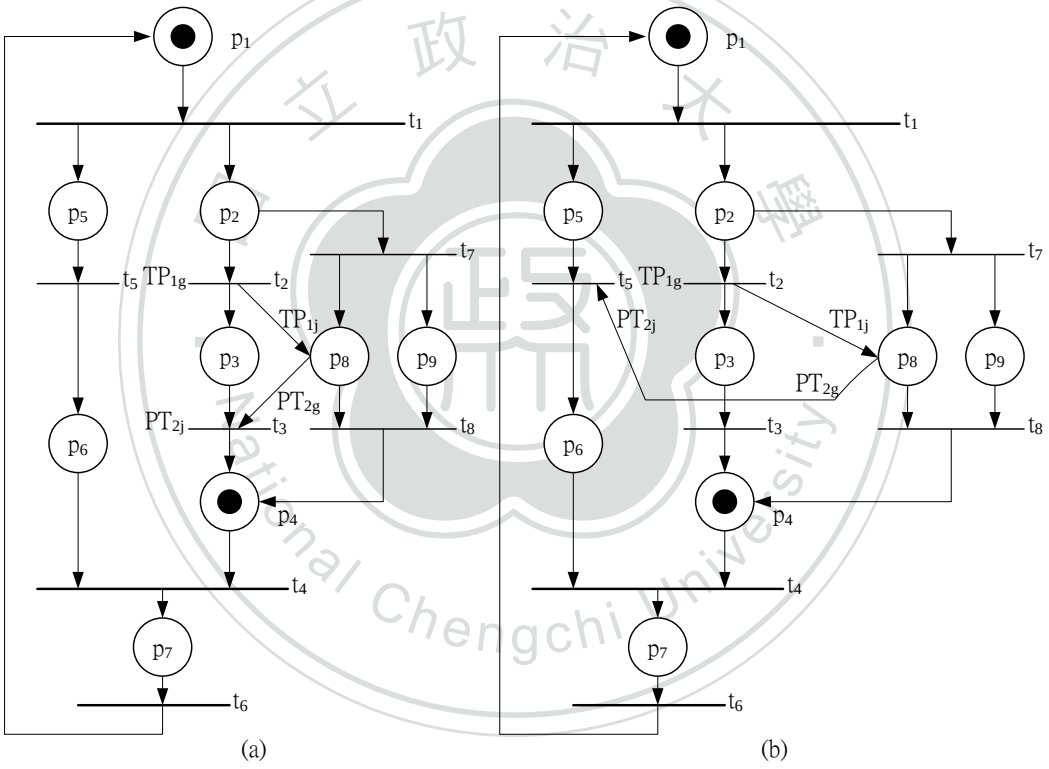

圖 3-27 TPPT 正向與反向生成圖

在圖 3-27 中，圖(a)表示正向的 TPPT 生成，TP_{1g} 'EX' TP_{1j}，PT_{2j} 'SL' TP_{1g}；圖(b)表示反向的 TPPT 生成 TP_{1g} 'EX' TP_{1j}，PT_{2j} 'CN' TP_{1g}。當 t_2 沒有被觸發時，$PT_{1j} = p_8$，就不會有標記進入，會造成死結。

定義9：假設，當兩個節點都在同一個虛擬流程中，它們兩個是互相相等的；當

兩個節點不在同一個虛擬流程中，它們兩個關係如下一個定義。

定義10：若Π_i與Π_k是局部互斥集(local exclusive set, LEX)關係，X_{ik}是所有虛擬流程的最大集合(基本基數的集合)，它們是互斥或等於或是與Π_i互斥，但與Π_k又不互斥。

即$X_{ik} = LEX(\Pi_i, \Pi_k) = \{\Pi_z \mid \Pi_z = \Pi_i$或是$\Pi_z \mid \Pi_i, \Pi\neg(\Pi_z \mid \Pi_k), \forall\, \Pi_{z1}, \Pi_{z2} \in X_{ik}, \Pi_{z1} \mid \Pi_{z2}\}$，所以$X_{ik}$和$X_{ki}$是互斥的。

定義11：Π_i局部並行集(local concurrent, LCN)與Π_k的關係，C_{ik}是所有虛擬流程最大集合(基本基數的集合)，它們是並行、等於或是與Π_i並行，但與Π_k又不並行。

即$C_{ik} = LCN(\Pi_i, \Pi_k) = \{\Pi_z \mid \Pi_z = \Pi_i$或是$\Pi_z \mid \Pi_i, \Pi\neg(\Pi_z \mid \Pi_k), \forall\, \Pi_{z1}, \Pi_{z2} \in C_{ik}, \Pi_{z1} \mid \Pi_{z2}\}$，所以$C_{ik}$和$C_{ki}$是並行的。

圖3-24(a)中，$X_{13} = \{\Pi_1 = [p_2\ t_2\ p_3], \Pi_2 = [p_2\ t_6\ p_3]\}$，此外$X_{31} = \{\Pi_3 = [p_3\ t_3\ p_4\ t_4\ p_5]\}$。

圖3-24(b)中，$C_{46} = \{\Pi_4 = [t_4\ p_5\ t_5], \Pi_5 = [t_4\ p_8\ t_5]\}$此外$C_{64} = \{\Pi_6 = [t_5\ p_6\ t_7\ p_1\ t_1]\}$。其中$C_{ik}$和$X_{ik}$可能不唯一。$C_{ik}(X_{ik})$的所有並行或互斥的虛擬路徑，必須在PP或TT的路徑中，否則可能會發生死結或是無限標記的問題。下面兩組是相對的情形：

(*LEX*, *TT*, ||)和(*LCN*, *PP*, |)。

同樣地，$LEX(t_a, t_b)$ $(LCN(p_a, p_b))$是一組轉移或狀態，而不是虛擬路徑。假設$G(J)$表示一組Π_g或Π_j，應用在單一個TT或PP規則裡，為了避免標記無限狀況，就必須有新的路徑，從虛擬路徑的每個X_{gj}分別連到每個X_{jg}。

定義12：單純生成(Pure Generation, PG)。在虛擬路徑中產生一個路徑，這個基本行程可以織成一個大的網路。

定義13：互動式生成(Interactive Generation, IG)在兩個虛擬路徑中，產生一個路徑，TT或PP的生成路徑，必須在兩個轉移或狀態之間，從岔開點t_g或p_g開始，到t_j或p_j結合點結束。

單純生成的流程，是沒有與其他個體互動的，他是獨立生成作業的，如果生成的兩端點是轉移或是狀態，它就是一組TT或PP生成，對應的路徑就是TT或PP路徑，如圖3-24(a) TT路徑為[t_1 p_7 t_5]，PP路徑為[p_2 t_6 p_4]。

TT和PP規則中，又區分為兩種型態，正向(forward)和反向(backward)生成，其定義如下：

定義14：假設，$n_g \rightarrow n_j$由前向後生成，稱為正向(或前向)生成，其它為反向生成。

圖3-23(a)中，[t_1 p_2]路徑是正向生成，圖3-26(a)中的[p_4 t_2]路徑，是反向生成路徑。

定義 15：TP_i，PT_i，或 PSP_i，(i = 1, 2)：i 有兩種狀態，其中 i = 1 表示獨立生成，當 i=2 時為伴隨生成。

TP_{ig}和 TP_{ij}：其中 g 和 j 分別表示 TP_i 的生成點(Generation Point)和接合點(Joint Point)。

PT_{ig}和 PT_{ij}：其中 g 和 j 分別表示 PT_i 的生成點和接合點。

Π_{gi}和Π_{ji}：其中g和j分別表示TP_i或PT_i中虛擬路徑的生成點和接合點。

在TPPT規則下，有許多細節問題，需要再分項討論，這些規則是除了PT和TP的路徑相關之外，必須符合派翠網路的基本邏輯外，還可以分為以下子問題：

(1)TP_{1j}與TP_{1g}的相對位置考量。

(2)PT_{2g}和PT_{2j}如何選擇。

接下來，本文將討論如何配置TP_{1j}、PT_{2g}和PT_{2j}。

TP_{1j}的規則，對於TP_{1j}對應到TP_{1g}，並有沒有特別限制。大致上有兩種情況：(1)“SQ”(圖3-25(a))或“CN”(圖3-25(b))和(2)“EX”(圖3-27(a))。

PT_{2g}的規則，有兩種PT路徑的應用，一個是利用PT路徑，來吸收多餘的標記，從TP路徑來的，如圖2-25(a)所示。另一種是有PT路徑的情況來約束TP路徑，這樣就不會注入額外的標記，範例如圖3-28所示。在這兩種情況下，定義為PT_{2g} “SQ” TP_{1j}。

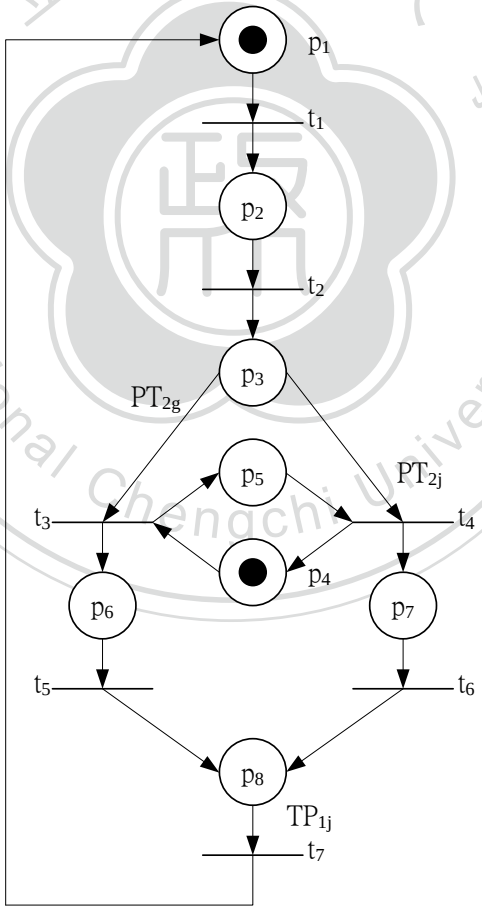

圖 3-28 PT_{2g} ‘SQ’ TP_{1j}範例圖

此外，PT_{2g}一定是一個p_e，其定義如下。

定義16：$p_f \leftrightarrow \boldsymbol{TP}_{1j}$或$p_f = \boldsymbol{TP}_{1j}$，$\forall t \in (p_f)^{\bullet}$，$\exists\ p_v \in {}^{\bullet}t$，$p_v \neq p_f$，$\boldsymbol{TP}_{1j} \parallel p_v$，$p_v$和$p_f$兩個都是在反向的TT生成中，也有可能不在其中，$p_v$在TPPT生成中，不是一個$p_e$。$\overline{p_e}$是所有$p_e$中，最接近$p_f$的$p_e$，標示為$p_e \rightarrow p_f$。

3.4 編織的規則

編織法在派翠網路分析中，是一個有效率的方法，其主要是使用TT和PP編織的規則，更詳細的部份包括狀態的標記限制、活性與可逆性已有文獻(Chen, 1993)證明，因此本研究引用Chao(2006)的TT、PP和TPPT規則，即可分析轉換後之網路，但在網路服務轉換後的派翠網路中，有許多不會發生的情況，甚至是不允許發生的情況，主要是因為BPEL的規範，對流程控制較嚴謹，因此本文將融入BPEL相關規則，並調整TT、PP和TPPT規則，使轉換與判斷過程中，除了可以確保網路分析結果之正確性外，另一方面可加快分析判斷的速度。為確保在執行時期的安全考量，若由BPEL轉換生成之派翠網路，需增加TT.5的判斷，以防止執行流程進行時，不正常跳出或結束，造成無法預期的狀況發生。

(1) TT的規則：

在派翠網路中的路徑，從$t_g \in \Pi_g$到$t_j \in \Pi_j$，

TT.1　假設$t_g \mid t_j$，在一個環狀結構中，則不是TT.1，搜尋其它的Π。

TT.2　假設$t_g \leftarrow t_j$或$t_g = t_j$，則必須有一個標記，在路徑的狀態中。

假設$\Pi_g = \Pi_j$，則不是TT.2，搜尋其它的Π。

TT.3

TT.3.1　搜尋X_{gj}中，每個Π_g中的轉移t_g的TP路徑，連接到狀態p_k的路徑。

TT.3.2　搜尋X_{jg}中，每個Π_j中的虛擬PT路徑，從到狀態p_j到轉移t_j的路徑。

TT.4　假設沒有從t_g到t_j的路徑，或是它們都在同一個迴圈中，如果有，找出從t_g'到t_j'的TT路徑，且其在路徑中的順序是t_g, t_j, t_g'和t_j'。

TT.5　從t_g到t_j，途中若狀態或轉移有分支，其分支最終必須限制在t_g到t_j的節點(狀態或轉移)上，並於報告文件中加註。

(2) PP的規則：

在派翠網路中的路徑，從$p_g \in \Pi_g$到$p_j \in \Pi_j$，

PP.1　假設$p_g \parallel p_j$，該情況則不是PP.1，搜尋其它的Π。

PP.2

PP.2.1　在C_{gj}中尋找所有的Π_j路徑，從轉移t_k到狀態p_j的TP路徑。

PP.2.2　在C_{jg}中尋找所有的Π_g路徑，從狀態p_g到轉移t_g的虛擬PT路徑。

PP.3　BPEL生成之網路需增加判斷

PP.3.1　在C_{gj}中尋找所有的Π_j路徑，從轉移t_k到狀態p_j的TP路徑，途中若狀態或轉移有分支，其分支最終必須連到p_j。

PP.3.2　在C_{jg}中尋找所有的Π_g路徑，從狀態p_g到轉移t_g的虛擬PT路徑，途中若狀態或轉移有分支，其分支最終必須連到t_g。

TT路徑中p_g(或是p_j)的輸出(或輸入)就是生成點或是接合點。

(3) TPPT的規則：

若BPEL生成之網路判斷結果有TPPT生成，需特別檢查。假設在一個TP生成環境下，路徑的尋找規則如下：

TPPT.1　TP若且唯若PT且反之亦然。

TPPT.2　PT_{2g}是一個TP路徑的p_e，且(PT_{2j} ‘EX’ TP_{1g})。

TPPT.3　假設PT_{2j} ‘SQ’ PT_{2g}，則PT_{2j} ‘SE’或‘SL’對應到TP_{1g}和PT_{2g}，或是PT_{2j} = TP_{1g}。

TPPT.4　假設PT_{2g} → TP_{1j}或PT_{2g} = TP_{1j}，則PT_{2j}中，所有從PT_{2g}輸出的轉移，不包括PT_{2j}，與TP_{1g}須在同一個路徑上，對到一個反向的TT路徑。

TPPT.5　假設TP_{2g}是一個起始位置，則適用規則(2)或是(3)，如果他是一個反向(或正向)的生成，在起始位置需有標記。

TPPT.6　若PT_{2g}對一個反向生成來說，若不是起始位置，則在PT_{2g}需要有一個標記。

第四章　系統實作與結果

4.1 WSBPEL 轉換派翠網路

本節將以範例實際測試 WSBPEL 轉派翠網路，轉換方式是使用本研究所開發之 BPEL-PN 引擎，在輸入 BPEL 原始檔案後，將其轉換成派翠網路結構，轉換後如要觀察網路結構圖，可透過 PIPE 軟體輔助，開啟後可以觀察到整個派翠網路的整個節點和架構。之後可透過引擎中的分析系統，觀察該網路之活性，也可透過分析軟體，找出該網路之主要路徑、主要獨立路徑及網路關鍵的生成點與結合點。

4.2 派翠網路範例

範例一：測試個案修改自 WSBPEL 2.0 規格書中，所展示的的訂單處理範例，此訂單處理網路服務，在收到顧客的訂單之後，啟動三個並行執行的活動，其最終之目的，是計算訂單的最終價格。但計算過程中，價格卻和選擇的貨運方式、訂單的生產排程、訂單的物品裝載有交叉的關聯。範例中的三個活動雖然可以並行處理，但其中有資料與控制的相依問題，例如，需要有貨運送的價格，才能決定最終的發票價格，以及需要有送貨的相關資訊，才能決定最後的訂單生產排程，流程內容如圖 4-1 所示。

圖 4-1 中的虛線代表執行序，大圈圈表示平行處理活動的範圍，實線代表平行處理活動時的同步活動。圖 4-2 中是以圖 4-1 活動內容為主，在取得其程式碼後，修改帶入本研究所開發之 BPEL-PN 引擎，轉換後會得到 PNML 檔案，再將 PNML 檔案透過解析軟體，例如常見的 PIPE 軟體，就可以得到圖 4-2 的訂單處理流程之派翠網路圖。

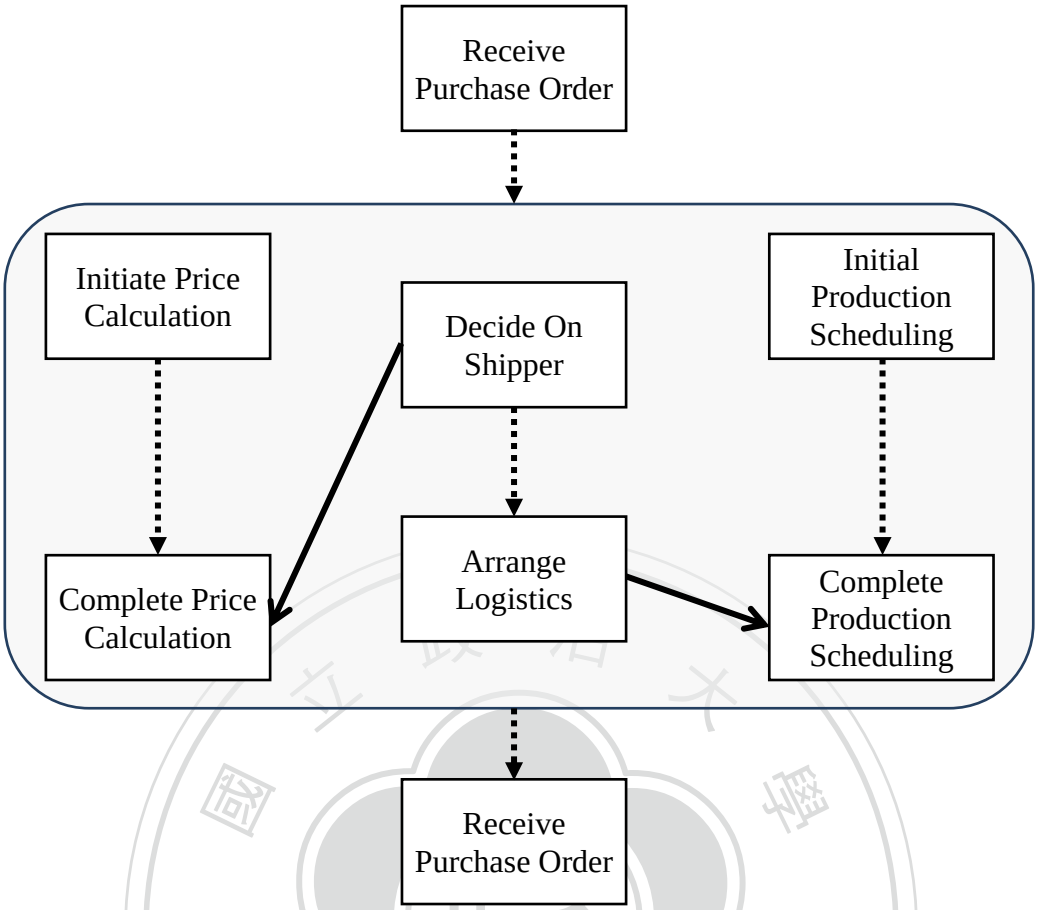

圖 4-1 訂單處理流程圖

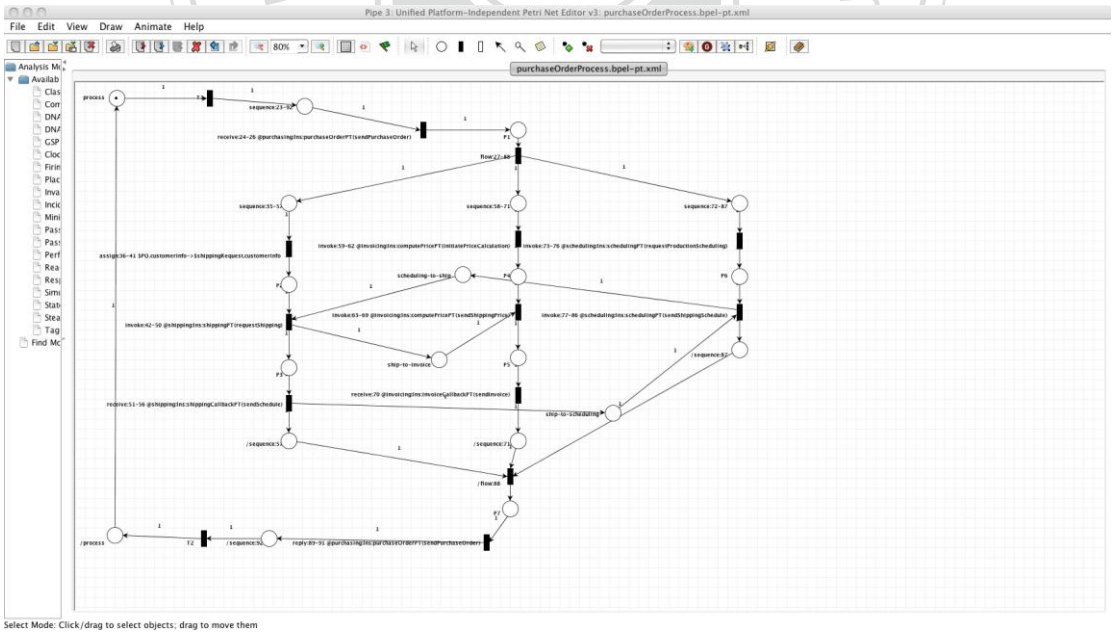

圖 4-2 訂單處理流程之派翠網路圖

以圖 4-1 活動內容為主，在轉換成派翠網路後後，帶入 BPEL-PN 引擎後，進行網路分析，就可以得到圖 4-3 的訂單處理流程之分析報告圖。在報告書中會顯示在途中的基本流程有哪些，並對應到原始程式碼的行號；所出現的虛擬路徑位置，與對應到原始碼的行號，最主要的是有偵測到死結情形，發生死結的位置出現在 p_6 節點，對照到報表顯示，在程式第 77-86 行 PT 連結處，執行的是 sendShipingSchedule，在交互對照情況下，可以容易地發現問題點，和程式流程中的所在位置。

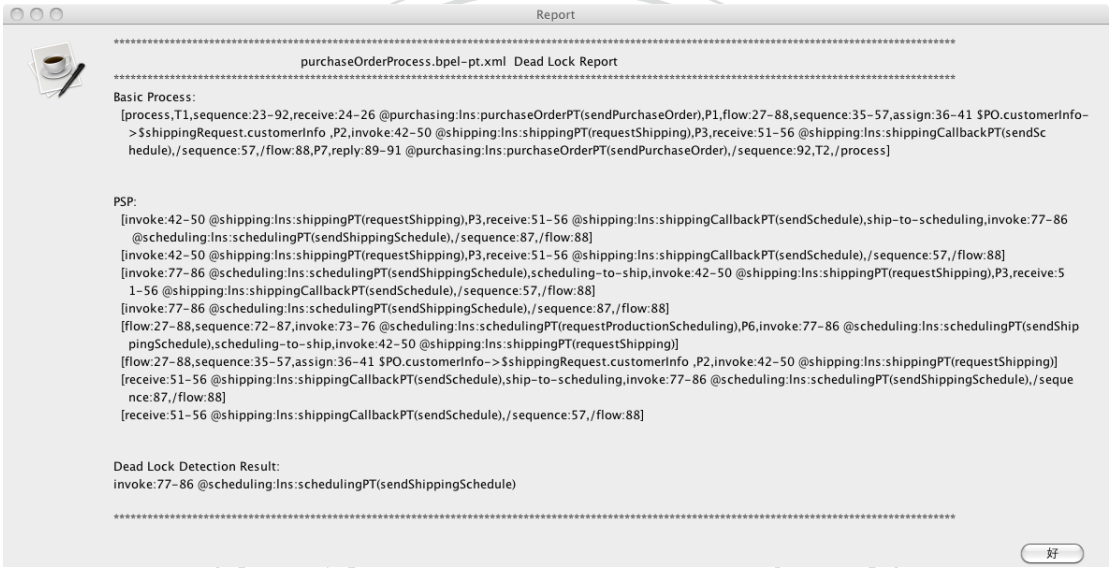

圖 4-3 訂單處理流程之分析報告圖

範例二：在 Collaxa 網站中，有一組合式網路服務流程範例，其中有貸款取得流程，本研究以該範例進行測試，圖 4-4 為貸款取得的流程圖。

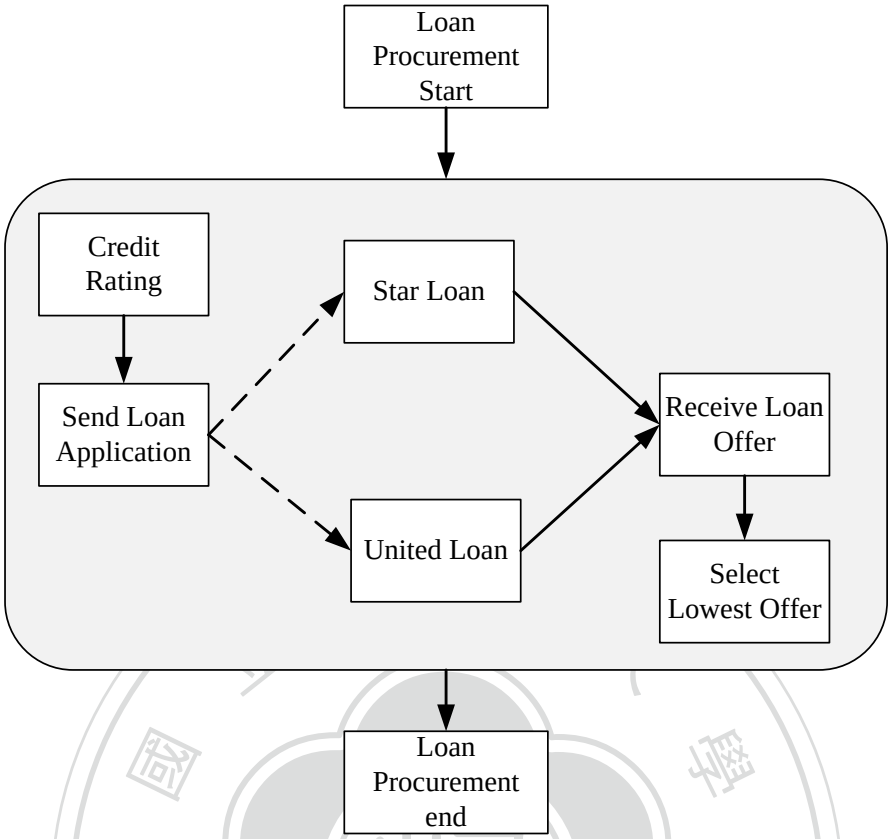

圖 4-4 貸款取得流程圖

在流程中首先進行信用評等(Credit Rating)，評等結果通過後，則送出貸款申請(Send Loan Application)，分別送給 Star Loan 以及 United Loan 兩單位；待兩單位處理完畢後，會收到貸款的處理結果(Receive Loan Offer)，報價最低的為最後選擇(Select Lowest Offer)，此為貸款取得流程的概述，圖 4-4 中虛線為平行處理的活動。表 4-1 為本範例之程式碼。

表 4-1 貸款申請範例程式碼範例表

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="LoanFlow" targetNamespace="http://samples.cxdn.com"
suppressJoinFailure="yes" xmlns:tns="http://samples.cxdn.com"
xmlns:services="http://services.cxdn.com"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
```

```
<sequence>
    <receive name="receiveInput" partnerLink="client"
portType="tns:LoadFlow" operation="initiate" variable="input"
createInstance="yes"/>
    <sequence>
        <assign>
            <copy>
                <from variable="input" part="payload"
query="/loanApplication/SSN"/>
                <to variable="crInput" part="payload" query="/ssn"/>
            </copy>
        </assign>
        <invoke name="invokeCR" partnerLink="creditRatingService"
portType="services:CreditRatingService" operation="process"
inputVariable="crInput" outputVariable="crOutput"/>
        <assign>
            <copy>
                <from variable="crOutput" part="payload"
query="/rating"/>
                <to variable="input" part="payload"
query="/loanApplication/creditRating"/>
            </copy>
        </assign>
    </sequence>
    <sequence>
        <assign>
            <copy>
                <from variable="input" part="payload"/>
                <to variable="loanApplication" part="payload"/>
            </copy>
        </assign>
        <flow>
```

```
                    <sequence>
                        <invoke name="invokeUnitedLoan"
partnerLink="UnitedLoadService" portType="services:LoadService"
operation="initiate" inputVariable="loadApplication"/>
                        <receive name="receive_invokeUnitedLoan"
partnerLink="UnitedLoanService" portType="services:LoanServiceCallback"
operation="onResult" variable="loanOffer1"/>
                    </sequence>
                    <sequence>
                        <invoke name="invokeStarLoan"
partnerLink="StarLoanService" portType="services:LoanService"
operation="initiate" inputVariable="loanApplication"/>
                        <receive name="receive_invokeStarLoan"
partnerLink="StarLoanService" portType="services:LoanServiceCallback"
operation="onResult" variable="loanOffer2"/>
                    </sequence>
                </flow>
            </sequence>
            <switch>
                <case
condition="bpws:getVariableData('loanOffer1','payload','/loanOffer/APR')&gt;
bpws:getVariableData('loanOffer2','payload','/loadOffer/APR')">
                    <assign>
                        <copy>
                            <from variable="loanOffer2" part="payload"/>
                            <to variable="selectedLoanOffer2" part="payload"/>
                        </copy>
                    </assign>
                </case>
                <otherwise>
                    <assign>
                        <copy>
```

```
                    <from variable="loanOffer1" part="payload"/>
                    <to variable="selectedLoanOffer1" part="payload"/>
                </copy>
            </assign>
        </otherwise>
    </switch>
    <invoke name="replyOutput" partnerLink="client"
portType="tns:LoanFlowCallback" operation="onResult"
inputVariable="selecredLoanOffer"></invoke>
  </sequence>
</process>
```

經由本研究的轉換處理過程，即可將上述 WSBPEL 2.0 程式碼轉換成派翠網路的格式，針對表 4-1 之貸款流程的範例程式碼為樣本，進行轉換後由 Pipe 3.0 開啟 PNML 檔案，便會得到圖 4-5 之貸款取得流程之派翠網路模型圖結果。

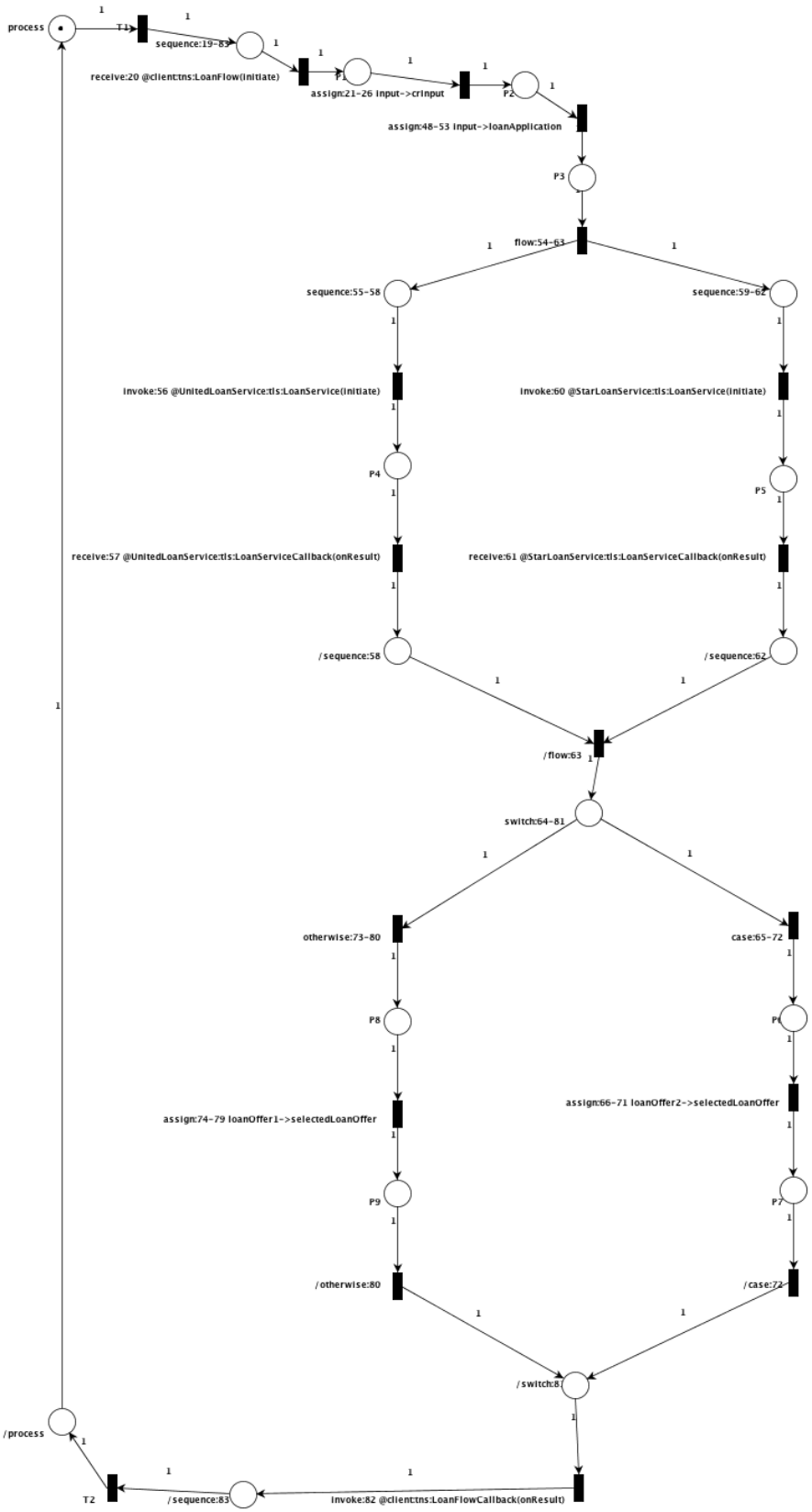

圖 4-5 貸款取得流程之派翠網路模型圖

進行完 WSBPEL 轉派翠網路後，產生的派翠網路檔，可再藉由引擎的死結偵測功能進行死結的檢測，偵測的結果報表，如圖 4-6 所示。過程中未發現死結，從開始到結束共 27 個基本流程節點，其中有一 TT 和 PP 流程，分別是 54-63 的 flow 和 64-81 的 switch 流程，TT_{1g} 是 flow: 54-63，TT_{1j} 是/flow: 63，PP_{2g} 是 switch: 64-81，PP_{2j} 是/switch: 81，最後是列出所有虛擬流程。

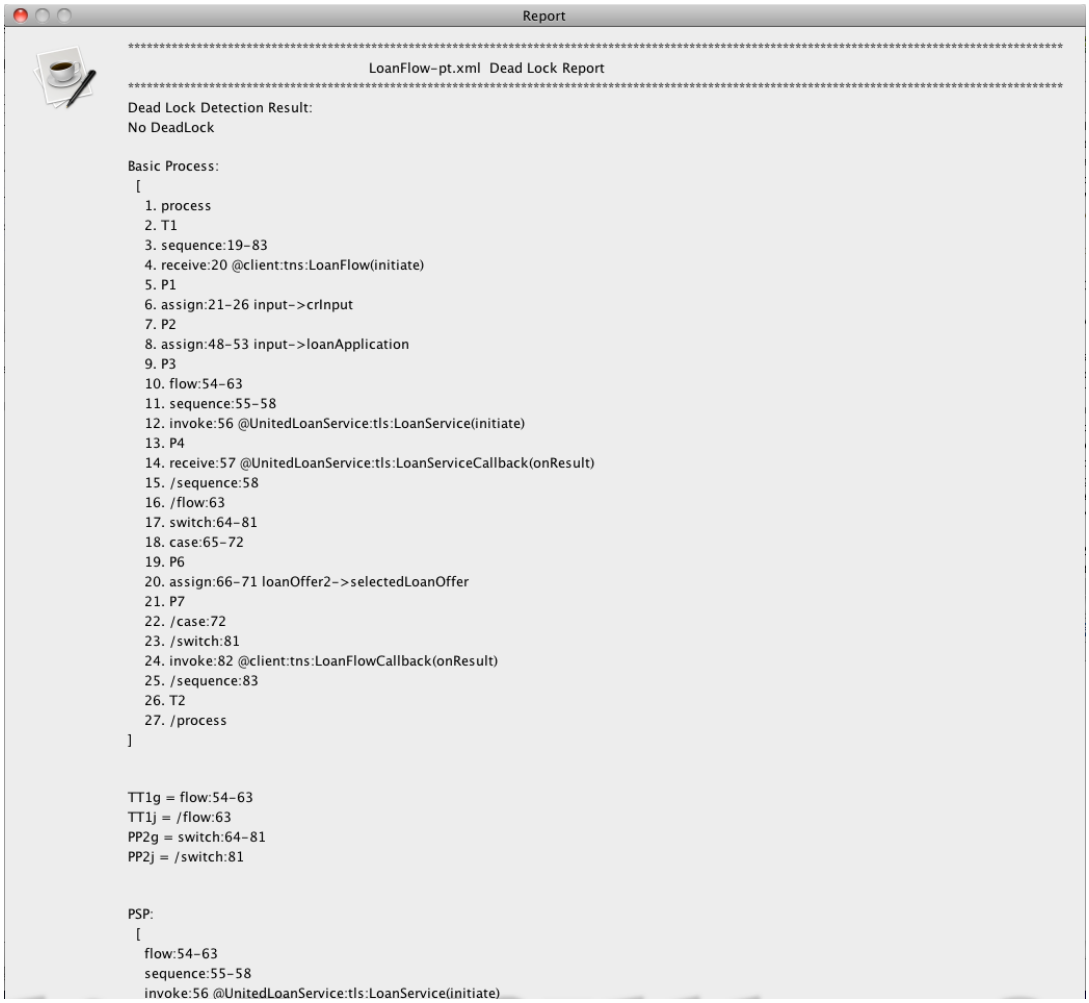

圖 4-6 貸款取得流程死結偵測報表圖

此範例為官方原始版本，內容為 WSBPEL 1.0 版，但此版本中的部份程式碼在進入 WSBPEL 2.0 版後，有些指令已停用或更改，例如 switch case 等等，因此將程式碼，修改成符合 WSBPEL 2.0 版之敘述，如表 4-2 所示為 WSBPEL 1.0 原始程式碼。

表 4-2 貸款取得流程 WSBPEL 1.0 程式碼

```
<switch>
        <case
condition="bpws:getVariableData('loanOffer1','parameters','result/APR') >
bpws:getVariableData('loanOffer2','parameters','result/APR') ">
            <assign>
                <copy>
                <from variable="loanOffer2" part="parameters"
query="result"/>
                <to variable="selectedLoanOffer" part="parameters"
                    query="/onLoanFlowResult/result"/>
                </copy>
            </assign>
        </case>
        <otherwise>
            <assign>
                <copy>
                <from variable="loanOffer1" part="parameters"
query="result"/>
                <to variable="selectedLoanOffer" part="parameters"
                    query="/onLoanFlowResult/result"/>
                </copy>
            </assign>
        </otherwise>
    </switch>
```

本研究之引擎以 WSBPEL 2.0 為主，但仍兼容 WSBPEL 1.0 版本的轉換。但此版本中的部份程式碼在進入 WSBPEL 2.0 版後，有些指令已停用或更改，如 switch case 等等，因此將程式碼中的 switch case 程式碼，修改成符合 WSBPEL 2.0 版之 if 敘述，結果如表 4-3 貸款取得流程 WSBPEL 2.0 程式碼之內容所示。

表 4-3 貸款取得流程 WSBPEL 2.0 程式碼

```
<if>

                <condition>bpws:getVariableData('loanOffer1','parameters','result/
APR') > bpws:getVariableData('loanOffer2','parameters','result/APR') </condition>
                    <assign>
                <copy>
                <from variable="loanOffer2" part="parameters" query="result"/>
                <to variable="selectedLoanOffer" part="parameters"
                    query="/onLoanFlowResult/result"/>
                </copy>
            </assign>
                    <else>
                        <assign>
                    <copy>
                    <from variable="loanOffer1" part="parameters"
query="result"/>
                    <to variable="selectedLoanOffer" part="parameters"
                        query="/onLoanFlowResult/result"/>
                    </copy>
                </assign>
                    </else>
                 </if>
```

經過版本更新為 WSBPEL 2.0 版後，新版的貸款取得流程轉換為派翠網路之結果，如圖 4-7 所示。

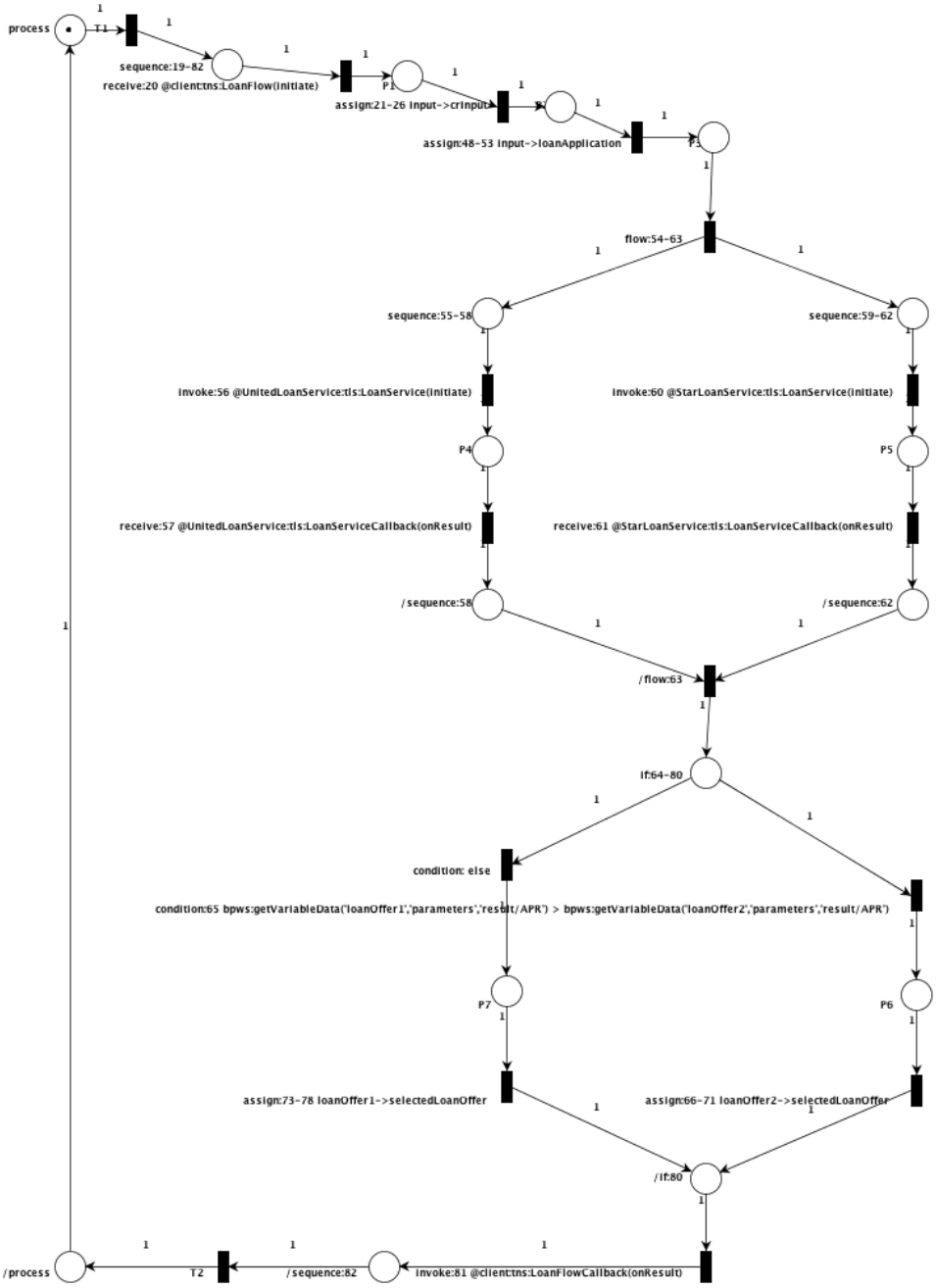

圖 4-7 WSBPEL 2.0 版貸款取得流程之派翠網路模型圖

經由本研究之引擎，將 WSBPEL 2.0 轉換為派翠網路之結果後，可再送入引擎中另一個模組，繼續進行死結偵測，偵測後產生之報表，如圖 4-8 所示。過程中仍未發現死結，從開始到結束共 25 個基本流程節點，其中有一 TT 和 PP 流程，分別是 54-63 的 flow 和 64-80 的 switch 流程，TT_{1g} 是 flow: 54-63，TT_{1j} 是/flow: 63，PP_{2g} 是 if: 64-80，PP_{2j} 是/if: 80，最後是列出所有虛擬流程。

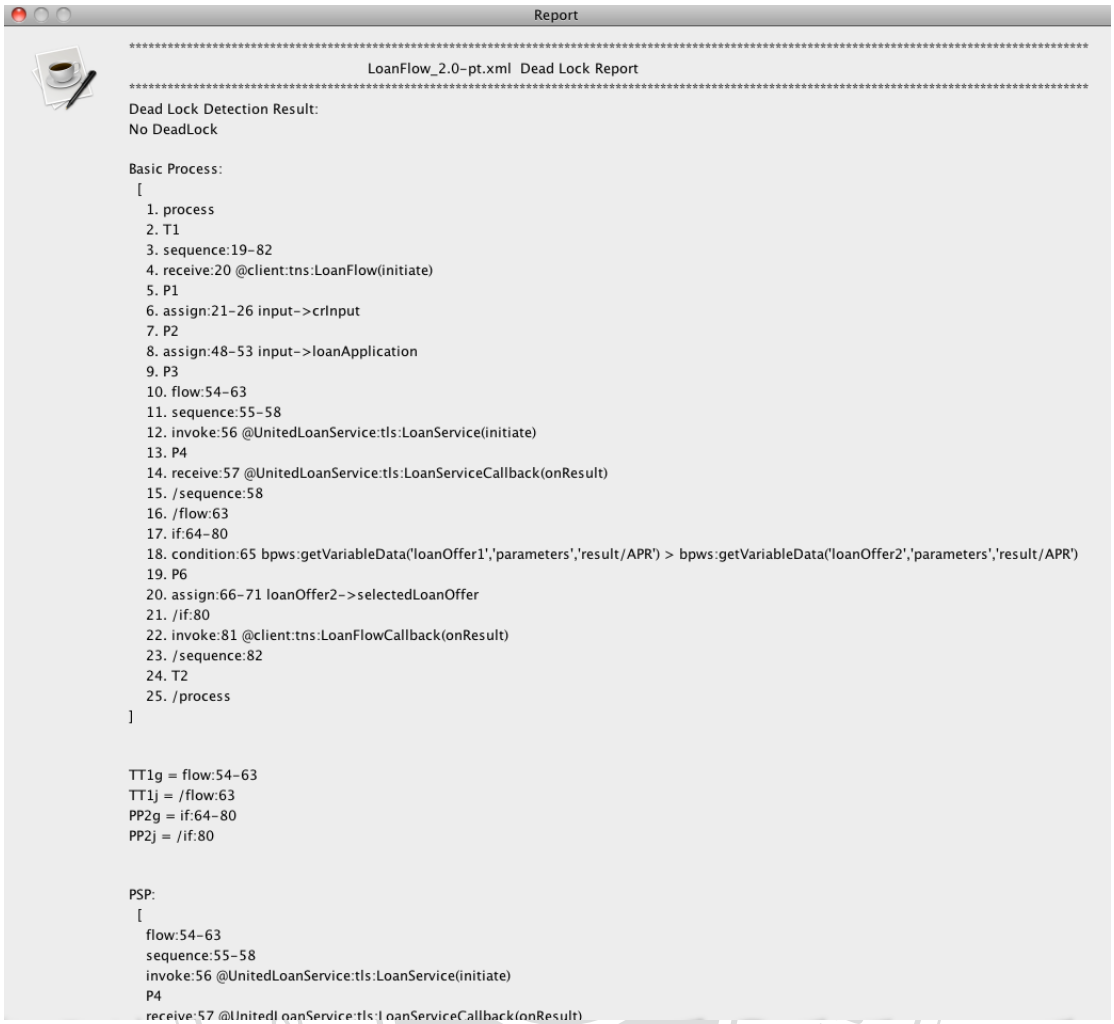

圖 4-8 WSBPEL 2.0 版貸款取得流程死結偵測報表圖

第五章　結論與建議

5.1 研究結論

全球化的經濟體系已逐漸形成，產業結構必須因應外界環境而有所改變，才能確保企業的優勢與地位；現今的商業流程與服務日趨複雜，客戶要求多元化的情況下，必須大量依賴資訊系統才能處理，為了滿足各項需求，系統功能也變得更加複雜，使得系統開發的成本提高，管理與維護的工作也更加困難。

結合雲端環境的網路服務，是目前解決系統整合最好的方法，在服務導向的架構環境中，網路服務的項目快速增加，組合式的網路服務，也成為系統發展的趨勢，提供了更多樣化的系統功能。雖然組合出來的服務樣式變化多且有彈性，但也造成維運上很大的負擔，除了管理的問題之外，因資源或服務共用的情形增加，同時也提升死結出現的機會，所以自動化的判斷分析機制，成為不可缺乏的工具。

死結的發生，對於商務系統來說，會造成相當大的影響及損失，對於企業形象，更會造成嚴重的傷害，因此，在服務組合形成後，如何快速地判斷網路活性，是相當重要的，因此，本研究建構了一個從服務流程組合，轉換成派翠網路，再針對派翠網路分析，最後進行活性的判斷，有效地解決系統整合所產生的問題。

5.2 研究貢獻

雲端環境中存在著大量的網路服務與組合應用，這些複雜的組合應用，本研究透過 WSBPEL 的語法轉換，可以快速地將系統流程，轉換成派翠網路進行分析與判斷。

本研究提供幾項重要貢獻如下：

(1) BPEL 是網路服務流程管理與控制的主要核心，本研究將新版的WSBPEL 語法，進行轉換與分析的實作，使網路服務流程可以自動轉換、檢驗與分析。

(2) 企業中資訊系統的流程，會隨時因應外界環境的快速變化，進行調整或修改，調整後的結果可即時輸入本研究之系統，透過引擎處理，可以快速地轉換與分析流程。

(3) 本研究考量引擎未來擴充性，因此將引擎結構模組化，除此之外，也將引擎執行方式，以網路服務形式提供服務，使系統能更容易調整與擴充，在需要的情況下，還可加入不同條件規則的模組，提供更多元化的分析與判斷功能。

(4) 本研究的實作部份，是透過網路服務的形式提供服務，在結合雲端環境下，可將本引擎快速地整合在其它的系統中，立即提供服務。

(5) 經過引擎轉換處理後，系統會提供網路服務分析的報表，以描述網路環境中，有哪些特殊的網路組合結構，是需要特別注意的，例如：基本流程、虛擬流程、虛擬流程之間的互動結構關係、局部關係或特殊生成結構，在瞭解網路的結構後，可協助管理者調整流程。

(6) 網路服務流程轉換後的結果，可透過視覺化方式呈現，再與系統產生的分析報表交叉比對，管理者可以更輕易地瞭解系統流程與結構，對於日後流程的調整，亦能更快速精準。

(7) 在死結情形發生時，產生的報表可提供死結位置與鄰近節點的關係，並可以透過圖像註記，反向找出程式碼位置，系統還會依據判斷的規則，提供建議處理的方式。透過原始的 WSBPEL 程式碼、報表與派翠網路對

照資料，能快速瞭解，找出問題點，對系統的管理人員有很大的幫助。

(8) 特殊的派翠網路結構，在過去文獻中，大多以人工或半自動方式找出，非常的費時與費力，且在互相影響的狀況下，例如循序、互斥或是並行結構關係，分析與判斷更加困難，透過系統化的分析，可以自動且快速地找出結果。

5.3 未來發展

本研究提出雲端系統服務流程，轉換之規則與分析，並建置實作系統，未來還有許多發展方向，歸納為下列幾項：

(1) 結合語意網路，當輸出、輸入之語意相符時，即可相互連結的原則，可自動組合連結，產生更多的服務組合，這些快速生成的網路服務，元件數量很多，結構相當複雜，此時亦可利用本研究的引擎，運用自動化分析功能，加速網路的分析。

(2) 本架構在擴充後，可於動態執行的環境中，提供即時的監控。

(3) 在雲端運算中，各個網路服務的負載會隨時改變，未來可結合伺服器的負載變動，動態調整服務環境，當負載減少時，減少服務資源；負載增加時則配置更多的虛擬機器，以提供更快送、穩定的服務。

(4) 結合不同分析方法，針對分析結果進行比較與驗證。

參考文獻

1. Andrews, T., F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, "BPEL4WS V1.1 specification," http://public.dhe.ibm.com/software/dw/specs/ws-bpel/ws-bpel.pdf, 2003.

2. Business Process Modeling Language (BPEL), http://www.ebpml.org/bpml.htm, Access time: Nov 2010.

3. Berners-lee, T., 1998, "What the Semantic Web can represent," http://www.w3.org/DesignIssues/RDFnot.html, Access time: Nov 2010.

4. Berners-Lee, T., "Semantic Web - XML2000," http://www.w3.org/2000/talks/1206-xm2k-tbl/sidr1-0.html, 2000.

5. Gruber, T., "What is an Ontology?" http://www-ksl.stanford.edu/kst/what-is-anontology.html, 2000.

6. IBM, "IBM cloud computing," http://www.ibm.com/cloud-computing/us/en/, Access time: Nov 2010.

7. Lehmann, M., "Web Services Composition," http://www.isys.uni-klu.ac.at/ISYS/Courses/03SS/S_DKE/lehmann.ppt, 2003.

8. Snell, J., "The Web services insider, Part4: Introducing the Web Services Flow Language," http://www-106.ibm.com/developerworks/webservices/library/ws-ref4/, Jun 2001.

9. W3C, "Web Service Choreography Interface(WSCI)," http://www.w3c.org/TR/wsci/, Access time: Nov 2010.

10. WFMC, "XML Processing Description Language(XPDL)," http://www.wfmc.org/standards/XPDL.htm, Access time: Nov 2010.

11. Arkin, A., S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, S. Commerce, C. K. Liu, S. Thatte, P. Yendluri and A. Yiu, "Web Services Business Process Execution Language Version 2.0," 2005.

12. Arpinar I. B., R. Zhang, B. Aleman-Meza and A. Maduko, "Ontology-driven Web services composition platform," ISeB, 2005, pp.175-199.

13. Bada, M., R. Stevens, C. A. Goble, Y. Gil, M. Ashburner, J. A. Blake, J. M.

Cherry, M. Harris and S. Lewis, “A short Study on the Success of the Gene Ontology,” Web Semantics: Science, Services and Agents on the World Wide Web 1, 2004, pp. 235-240.

14. Benatallah, B., M. Dumas, M. C. Fauvet and F. A. Rabhi, “Towards Patterns of Web Services Composition,” *Patterns and skeletons for parallel and distributed computing*, 2003a, pp. 265-296.

15. Benatallah, B., Q. Z. Sheng and M. Dumas, “The Self-Serv Environment for Web Services Composition,” *IEEE Internet Computing*, Vol. 7, No. 1, Jan/Feb 2003b, pp. 40-48.

16. Berardi, D., D. Calvanese, G. D. Giacomo, M. Lenzerini and M. Mecella, “A foundational vision of E-Services,” *In Proceedings of the Workshop on Web Service, E-Business, the Semantic Web (WES’) held in conjunction with the 15th Conference on Advanced Information Systems Engineering*, Klagenfurt / Velden, Austria, 2003.

17. Bertoli, P., M. Pistore and P. Traverso, “Automated Composition of Web Services by Planning in Asynchronous Domains,” *In Proc. ICAPS’05*, 2005.

18. Casati, F., S. Ilnicki, L. Jin, V. Krishnamoorthy, M. C. Shan, “eFlow: a Platform for Developing and Managing Composite e-Services,” *Proceedings Academia/Industry Working Conference on Research Challenges*, Apr 2000, pp. 341-348.

19. Chandrasekaran, S., J. A. Miller, G. A. Silver, I. B. Arpinar and A.P. Sheth, “Performance Analysis and Simulation of Composite Web Services,” *The International Journal of Electronic Commerce and Business Media* (EM), Vol. 13, No. 2, June 2003, pp. 120-132.

20. Chao, D. Y. and D. T. Wang, “A Reduction algorithm of Petri net, ” *Proc. Int'l Comp Symp (ICS 92’)*, Taichung, Taiwan, Dec. 13-15, 1992, pp. 16-23.

21. Chao, D. Y., M. C. Zhou and D. T. Wang, “Extending Knitting Technique to Petri net Synthesis of Automated Manufacturing Systems,” *The Computer Journal*, Oxford University Press, Vol. 37, No. 1, Jan. 1994a, pp. 67-76.

22. Chao, D. Y. and D. T. Wang, “A Synthesis Technique of General Petri nets,” *Journal of Systems Integration*, Vol. 4, No. 1, Feb. 1994b, pp. 67-102.

23. Chao, D. Y. and D. T. Wang, “An Interactive Tool for Design, Simulation, Verification, and Synthesis of Protocols,” *Software-Practice and Experience*, Vol.

24, 1994c, pp. 747-783.

24. Chao, D. Y. and D. T. Wang, “Knitting Technique with TP-PT Generations for Petri net Synthesis,” Technical Report No. CIS-94-45, Dept. of Computer and Information Science, New Jersey Institute of Technology, 1994d.

25. Chao, D. Y. and D. T. Wang, “Petri Net Synthesis and Synchronization Using Knitting Technique,” *IEEE Int’l Conf. SMC*, San Antonio, TX, October2-5 1994e, pp. 652-657.

26. Chao, D. Y. and D. T. Wang, “The Knitting Technique and Its Application to Communication Protocol Synthesis,” *MASCOTS'94*, Durham, NC, Jan. 31 - Feb. 2, 1994f, pp. 234-238.

27. Chao, D. Y. and D. T. Wang, “XPN-FMS: A Modeling and Simulation Software for FMS Using Petri nets and X window,” *International Journal of Flexible Manufacturing Systems*, Vol. 7, No. 4, October 1995, pp.339-360.

28. Chao, D. Y. and D. T. Wang, “Knitting Technique and Structural Matrix for Deadlock Analysis and Synthesis of Petri Nets with Sequential Exclusion,” *MIS Review*, Vol. 7, December 1997a, pp.45-85.

29. Chao, D. Y. and D. T. Wang, “Two Theoretical and Practical Aspects of Knitting Technique Invariants and a New Class of Petri Net, ” *IEEE Transactions on System*, Man, and Cybernatics, Vol. 27, 1997b, pp. 962-977.

30. Chao, D. Y., “Petrinetsyn the Sisand Synchronization Using Knitting Technique,” *Journal of Information Science and Engineering*, Vol. 15, 1999, pp. 543-568.

31. Chao, D. Y., “A Computer Aided Design Technique for Flexible Manufacturing Systems Synthesis Utilizing Petri Nets,” *Computer-Aided Design, Engineering, and Manufacturing: Techniques and Applications*, Volume III, Operational Methods in Computer Aided Design, CRC Press, 2001a, pp. 8.1-8.64.

32. Chao, D. Y. and J. A. Nicdao, “Liveness for Synchronized Choice Petri Nets,” *Computer Journal* (British Computer Society), Vol. 44, No. 1, 2001b, pp. 124-136.

33. Chen, Y., W. T. Tsai and D. Y. Chao, “Dependency Analysis a Compositional Technique for Building Large Petri Net,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, 1993, pp. 414-426.

34. Cherbakov L., G. Galambos, R. Harishankar, S. Kalyana and G. Rackham, “Impact of Service Orientation at the Business Level,” *IBM Systems Journal*, Vol.

44, No. 4, 2005, pp. 653-669.

35. Chiu, H. S. and C. Yang, "Beyond e-Commerce Software Quality: Web Services Effectiveness," Proceeding of Second Asia-Pacific Conference on Quality Software, 2001, pp. 397-405.

36. Curbera, F., Y. Gol and, J. Klein, F. Leymann, D. Roller, S. Thatte and S. Weerawarana, "Business Process Execution Language for Web Services, Version 1.0," 2002, pp. 80.

37. Curbera, F., M. Duftler, R. Khalaf, W. Nagy, N. Mukhi and S. Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing*, Vol. 6, No. 2, Mar/Apr 2002, pp. 86-93.

38. Datta, A. and S. Ghosh, "Synthesis of a Class of Deadlock-free Petri Nets," *Journal of ACM*, Vol. 31, 1984, pp. 486-506.

39. Fensel, D., "The Semantic Web and Its Language," *IEEE Intelligent Systems*, 2000, pp. 67-73.

40. Foster, H., S. Uchitel, J. Magee and J. Kramer, "Model-based, Verification of Web Service Compositions," presented at *Eighteenth IEEE International Conference on Automated Software Engineering* (ASE), Montreal, Canada, 2003.

41. Fu, X., T. Bultan and J. Su, "Analysis of Interacting BPEL Web Services," *In Proc. WWW'04*, 2004.

42. Grau, B. C., B. Parsia and E. Sirin, "Combining OWL ontologies using E-Connections," *Web Semantics: Science, Services and Agents on the World Wide Web*, 2006, pp. 40-59.

43. Guarino, N., "Formal Ontology and Information Systems," *Proc. Of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98*. Trento, Italy, Amsterdam, ISO Press, 6-8 June 1998, pp. 3-15.

44. Hendler, J., "Agents and the Semantic Web," *IEEE Intelligent Systems*, 2001, pp. 30-37.

45. Horrocks, I. and P. F. Patel-Schneider, "Three Theses of Representation in the Semantic Web," *ACM*, 2003, pp. 39-47.

46. Jian, Y. and M. P. Papazoglou, "Web Component: A Substrate for Web Services Reuse and Composition," *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, May 2002, pp. 21-36.

47. Juric, M. B. and M. Krizevnik, “WS-BPEL 2.0 for SOA Composite Applications with Oracle SOA Suite 11g,” 2010, pp. 178.

48. Li, M., P. van Santen, D. W. Walker, O. F. Rana and M. A. Baker, “SGrid: a Service-Oriented Model for the Semantic Grid,” *Future Generation Computer Systems*, 2004, pp. 7-18.

49. Limthanmaphon B. and Y. Zhang, “Web Service Composition with Case-based Reasoning,” *In Proceedings of the 14th Australasian database conference*, Adelaide, Australia, 2003, pp.201-208.

50. Maamar, Z., S. K. Mostefaoui and H. Yahyaoui, “Toward an Agent-based and Context-Oriented Approach for Web Services Composition,” *IEEE Transact Knowledge Data Engineering*, Vol. 17, No. 5, 2005, pp. 686-697.

51. Maedche, A., B. Motik, L. Stojanovic, R. Studer and R. Volz, “Ontology for Enterprise Knowledge Management,” *IEEE Intelligent Systems*, Vol. 18, 2003, pp. 26-33.

52. Marton, A., G. Piccinelli and C. Turfin, “Service Provision and Composition in Virtual Business Communities,” *Symposium on Reliable Distributed Systems*, Oct 1999, pp. 336-341.

53. Matskin, M. and J. Rao, “Value-Added Web Services Composition Using Automatic Program Synthesis,” Springer-Verlag Berlin Heidelberg, LNCS 2512, 2002, pp. 213-224.

54. Medjahed, B., B. Benatallah, A. Bouguettaya, A. H. H. Ngu and A. K. Elmagarmid, “Business-to-Business Interactions: Issues and Enabling Technologies,” *The VLDB Journal*, Vol. 12, No. 1, May 2003, pp. 59-85.

55. Murata, T., “Petri Nets: Properties, Analysis and Applications,” *Proceedings of the IEEE*, Vol. 77, No. 4, Apr 1989, pp. 541-580.

56. Ramamoorthy, C. V., S. T. Dong and Y. Usuda, “An Implementation of an Automated Protocol Synthesizer (APS) and Its Application to the X.21 Protocol,” *IEEE Transactions on Software Engineering*, Vol. 11, 1985, pp. 886-908.

57. Sensoy, M. and P. Yolum, “A Context-Aware Approach for Service Selection Using Ontologies,” *AAMAS’06*, Hakodate, Hokkaido, Japan, May 8-12 2006, pp. 931-938.

58. Silva, M., “Las Redes de Petri: en la Automática y la Informática,” Editorial AC, Madrid, 1985.

59. Rao, J. and X. Su, “A Survey of Automated Web Service Composition Methods,” *Semantic Web Services and Web Process Composition*, Vol. 3387, 2005, pp. 43-54.

60. van der Aalst, W. M. P., “Three Good reasons for Using a Petri net based Workflow Management System,” presented at Proceedings of the *International Working Conference on Information and Process Integration in Enterprises* (IPIC’96), 1996, pp. 179-20.

61. van der Aalst, W. M. P., A. H. M. ter Hofstede, B. Kiepuszewski and A.P. Barros, “Workflow Patterns,” Distributed and Parallel Databases Vol. 14, No. 1, July 2003, pp.5-51.

62. van der Aalst, W. M. P., M. Dumas, and A. H. M. ter Hofstede, “Web Service Composition Languages: Old Wine in New Bottles?,” presented at Euromicro Conference, 2003, Proceedings. 29th, 2003.

63. van der Aalst, W. M. P., “Don’t Go with the Flow: Web Services Composition Standards Exposed,” *IEEE Intelligent Systems*, Jan/Feb, 2003.

64. van der Meer, D., A. Datta, K. Dutta, H. Thomas, K. Ramamritham and S. B. Navathe, “FUSION: A System Allowing Dynamic Web Services Composition and Automatic Execution,” *IEEE International conference on E-Commerce* (CEC), 2003, pp. 399-404.

65. Wang, Y. and E. Stroulia, “Semantic Structure Matching for Assessing Web-Service Similarity,” *Proceedings of the First International Conference on Service Oriented Computing*, Trento, Italy, 2003.

66. Wang, F. H. and H. M. Shao, “Effective Personalized Recommendation Based on Time-Framed Navigation Clustering and Association Mining,” *Expert Systems with Applications*, Vol. 27, No. 3, 2004, pp. 365-377.

67. Younasa, M., K. M. Chaob, and C. Laing, “Composition of Mismatched Web Servicesin Distributed Service Oriented Design Activities,” *Advanced Engineering Informatics*, Vol. 19, No. 2, 2005, pp. 143-153.

68. Zeng, L., B. Benatallah, H. Lei, A. H. H. Ngu, D. Flaxer and H. Chang, “Flexible Composition of Enterprise Web Services,” *The International Journal of Electronic Commerce and Business Media*, Vol. 13, No. 2, Jun 2003, pp. 141-152.

69. Zeng, L., B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam and H. Chang,

"QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, Vol. 30, No. 5, May 2004, pp. 311-327.

70. Zhou, M. C., F. DiCesare and A. A. Dosrochers, "A Top-Down Modular Approach to Synthesis of Petri Net Models for Manufacturing Systems," *Proc. of IEEE Robotics and Automation Conference*, cottsdale, AZ, 1989, pp. 534-539.

71. Zhou, M. C. and F. DiCesare, "Parallel and Sequential Mutual Exclusions for Petri Net Modeling for Manufacturing Systems with Shared Resources," *IEEE Trans. on Robotics and Automation*, Vol. 7, No. 4, 1991, pp. 515-527.

72. Zhou, M. C. and K. Venkatesh, "Modeling, Simulation and Control of Flexible Manufacturing System: a Petri Net Approach," World Scientific, pp. 41-42, 1998.

73. 李志偉，以 Petri Net 為基礎的網路服務組合箝制驗證及簡化方法，中原大學資訊管理學系碩士論文，2004。

74. 高慶霖，以派翠網路偵測網路服務流程之死結，中正大學資訊管理研究所碩士論文，2004。

75. 葉俊仁，企業競合上上策：若即若離 企業合夥關係新主張：Loosely Couple，資訊與電腦，(264)，2002，pp. 78-83。