基於 XML 與動態代理人技術之跨企業電子商務交易系統

謝至斌
繪展科技
m9021018@mail.ndhu.edu.tw

吳秀陽*
國立東華大學資訊工程學系
showyang@csie.ndhu.edu.tw

摘要

傳統的電子化交易環境，EDI 系統往往是唯一的選擇。EDI 可以完成電子化文件，表單傳送等，但是建構系統的花費甚高，而且是屬於封閉式的交易環境。隨著網際網路的盛行，任何企業只要用低廉的價格便可以與全世界相連接，因此也開始了新一代網際網路電子商務的構想。為了建構跨企業間的商務環境，我們將整個跨企業的交易行為模化為一個工作流程（Workflow），並且基於 WfMC 的建議，設計一個視覺化的工作流程描述工具 AC-Diagram 來具體化交易流程。為了有效的執行跨企業流程，我們使用 XML 來作為交易環境中主要描述語言並建構一套基於動態代理人（Dynamic Agent）技術所研發出來的跨企業代理人交易系統 (Cross-enterprise Agent Transaction System, CATS)。在系統中設計使用了交易運作、交易回復、以及後交易回復等演算法來進行跨企業交易，並利用動態型代理人所具有的彈性及智慧，搭配 XML 可延伸化特性，讓代理人能夠在分散式的環境下，正確、有效率的完成使用者所交付的交易。

關鍵詞：電子商務交易、跨企業工作流、動態代理人、可延伸式標記語言

一、前言

在過去，企業如果需要建構一套電子化的商務環境，EDI（Electric Data Interchange）系統往往是唯一的方法。EDI 可以完成電子化的文件，表單傳送。但是建構一套 EDI 系統的花費甚高，而且 EDI 屬於封閉式的交易環境。如此一來便大大的提高了企業商務電子化的門檻。隨著網際網路的盛行，任何企業只要用低廉的價格便可以與全世界相連接，因此我們利用便捷的網際網路來作為跨企業電子商務的介質，所有的企業可以利用網際網路來互相連接交換資訊。為了能夠保證企業間可以便利的交換各種格式的資料，我們利用 XML 技術來進行系統的開發。並且為了保障商務活動可以正確進行，我們設計一個交易處理系統來幫助交易正確的進行。為了建構跨企業的商務環境，我們將整個跨企業交易行為模化為一個工作流程（Workflow），並且基於 WfMC[11] 的建議，設計一個視覺化且具備正規語意的工作流程描述工具—活動控制圖（Activity-Control Diagram，簡稱 AC Diagram）[12] ，來具體化交易的流程。在本文中我們進一步建構一套基於動態代理人（Dynamic Agent）技術所設計出來的跨企業代理人交易系統 (Cross-enterprise Agent Transaction System，簡稱 CATS)。先依據 AC Diagram 的正規語意將原本複雜的交易流程切割成子流程，然後根據子流程特性分配給動態代理人進行分散式執行。為了維持交易特性，我們設計了交易運作、交易錯誤處理及回復、以及後交易回復等演算法來進行跨企業交易，並利用動態型代理人所具有的彈性及智慧，搭配 XML 可延伸化特性，讓代理人能夠在分散式的環境下，正確、有效率的完成使用者所交付的交易。

二、相關研究

傳統的電子商務作法，是各公司擁有自己的資訊系統，彼此互不相容。EDI（電子資料交換）在可管理的電子商務過程中被普遍使用，使不同的數據標準可在電腦間做正確的電子資料轉換，為目前存在於電子商務中普遍的作法。但 EDI 實作方式是在一個複雜的技術環境中與最初的實體投資元件做連結（例如，使用昂貴又複雜的資料傳輸系統的加值網路），多數的中小企業皆無法從以 EDI 為基礎的商業過程中獲得效率提昇的利益。此外，因許多不同的特殊行業的 EDI 標準，使得他們之間難以轉換（如 ANSI X12、UN/EDIFACT、ODETTE 等等）[13] 。因為這樣的理由，EDI 在各行各業中的上下游廠商的結合上相當不便。所以只有不到百分之二或三的公司企業會廣泛的使用 EDI。以 XML 為基礎的電子商務技術，則以相當驚人的速度發展，舉凡在企業中供應鍊管理、客戶服務支援、資源整合和物流管理、...等應用，都有 XML 相關標準與解決方案不斷被提出[8] [10] 。

Workflow[11] 是一個由一連串具有連續性動作的工作所組成的工作流程，根據 WfMC[11] 的定義，”Workflow 是一個可預先定義其程序與規則的自動化文件、資訊交換與活動的流程系統”。這個系統將會有一個輔助的工具來幫助管理者，利用一些簡單的函式來達到商業管理的目的。工作流程

是一個以命令式的方式去建立 B2B 的交易，並且整合組織內部的資訊系統與組織對外的資訊系統，使其能夠合作並且自動的完成一個大的型交易，同時處理這個交易裡許多瑣碎的事情甚至進一步的可以去對工作流程做最佳化為目標。一直以來 workflow 多用於文件管理的應用上，而其概念也漸漸有被使用於電子商務相關的研究議題。

透過自動化的商業流程的系統將可以為企業帶來更多的益處，例如以最佳化的方式排除許多不必要的步驟。來改善效率；透過標準化的工作方法和有效的審查每一個程序的工作機制，來改善商業程序的管理；透過過去的紀錄與處理過的程序將客戶服務緊密的結合在一起，甚至於預測下次交易的可能性與時間；利用其流動的特性，讓商業程序更加簡化；提供更佳的彈性來控制或修改流程。

動態代理人(Dynamic-Agents[1])能夠依照所面臨的問題特性，使用不同的知識來解決問題，因此可以針對許多不同種類的問題設計解決的知識，並賦予動態代理人來執行與處理我們所面臨的問題，這樣的設計可以大幅提高代理人程式的重複利用性(Reusibility)。我們即利用了合作型代理人及動態代理人的技術來完成跨企業平台電子交易。

三、活動控制圖(Activity-Control Diagram)

我們的商務模式化與 XML 工作流自動化架構如圖 1 所示[12] 。最底層的部分，是可執行的 XML 工作流規範。中間層則是我們所發展的視覺化工具「活動控制圖」(Activity-Control Diagram, AC Diagram)，其中包含了活動圖件(Activity Notations)和控制圖件(Control Notations)。使用者可以直接運用，或是透過工作流樣板(Workflow Patterns)來進行商務模式化。我們在本節中對 AC Diagram 做概念性介紹，其詳細內容和正規語意，請參考[12] 。

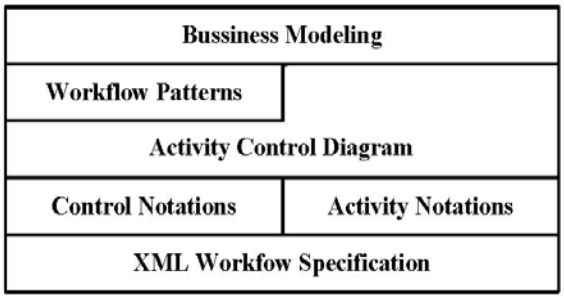

圖 1 商務模式化與工作流自動化架構[12]

一個完整的 AC Diagram 包含一組活動圖件、一組控制圖件、和一組關係(Relationships)來表示活動與控制之間的流程與相互關係。一個 AC Diagram 是一個 bipartite graph，亦即活動圖件必須連結控制圖件，反之亦然。

活動圖件用來描述商務活動。我們採用如圖 2 所示的 generic activities 而非明白的列出各種可能的商務活動，目的是讓商務描述更有彈性。其中 START 和 END 是用來啟動和結束一個商務程序。DISPLAY 是用來顯示執行狀態與程序結果。RETRIEVAL 和 INFORMATION 均為單向資訊傳輸，分別是由商務夥伴到使用者，以及反向傳輸；而 EXCHANGE 則表示雙向資訊交換。INTEGRATION 代表多方資訊結合；TRANSFORMATION 則是對資訊進行轉換。DECISION 用來在諸多可行方案中決定下一步驟。MODULE 是一個具備完整介面的流程模組，本身可以是另一個 AC Diagram，方便使用者進行模組化設計。CUSTOM 是保留給使用者定義客製模組。INVOKE 則是呼叫一個客製模組或是商務夥伴所提供的可呼叫應用程式。每一個活動圖件均有對應的輸入控制、輸出控制、以及參數，是活動行為與資訊傳輸之主要依據。另外每個活動皆訂有 timeout，是系統等待活動執行的時間限制。

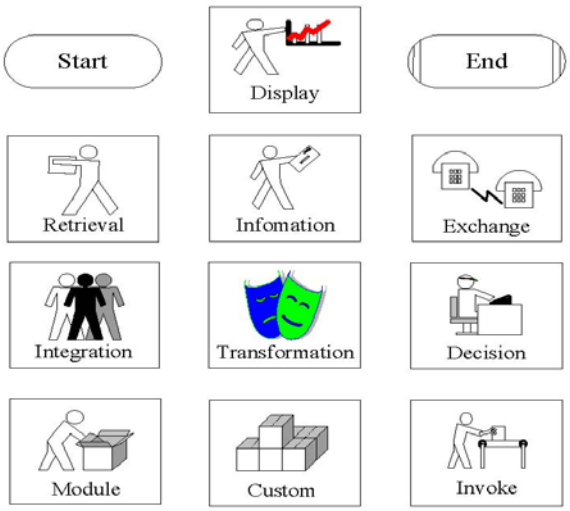

圖 2 活動圖件

大部分現存的流程描述工具只有通用的活動圖示，而未提供與上述活動圖件類似的機制。明確訂出活動的類別、語意性質、特別是資訊傳輸流向，有助於使用者的商務模式化過程，以及系統的自動化分析和處理。此外，我們仍然保留 CUSTOM 圖件做為通用活動描述。

另一方面，圖 3 所顯示的控制圖件則是用來描述各種流程定義中所需要用到的控制結構，可以充分的規範商務活動之間的複雜關係。其中 SEQUENCE 代表一個活動緊接著另一個活動的循序關係。OR 表示數個方案，可以任選其中之一來繼續進行。若是先前選擇的方案失敗，可以回到此點另循他途。相反的，AND 則是所有方案皆必須進行。XOR 與 OR 類似，但只能擇一而行，不同

方案之間彼此具有互斥性。PRIORITY 也和 OR 類似，但是不同方案之間的優先順序不同，必須先嚐試進行高優先方案，若是失敗了才進行低優先方案。COUNT 是在所有方案選擇固定數目同時進行，如果部份已經選擇的方案失敗，仍有未選擇方案可滿足數目限制，仍可以回到此點選擇這些方案。CHOICE 代表單一條件式的方案選擇；CONDITION 則是多條件式的選擇，只要符合者皆須進行。除了 SEQUENCE 之外，上述其餘七個控制圖示皆屬於「切分結構」(split structures)，因為皆是將單一流程切分為多個流程。接下來的四個圖示則是所謂的「結合結構」(join structures)，因為是將多個流程合併成一個流程，兼具同步意義。OR_JOIN 代表任何一個成功的輸入就可以啟動輸出，如果失敗，可以等待其他成功輸入。AND_JOIN 則是必須等待所有的輸入皆成功到達，才能啟動輸出。COUNT_JOIN 是滿足固定數目的成功輸入即可啟動輸出。PRIORITY_JOIN 代表必須依照優先順序等待輸入，只有當所有高優先順序輸入失敗，才能讓成功的低優先順序活動來啟動輸出。最後 LOOP 代表依條件而決定的重複性程序。

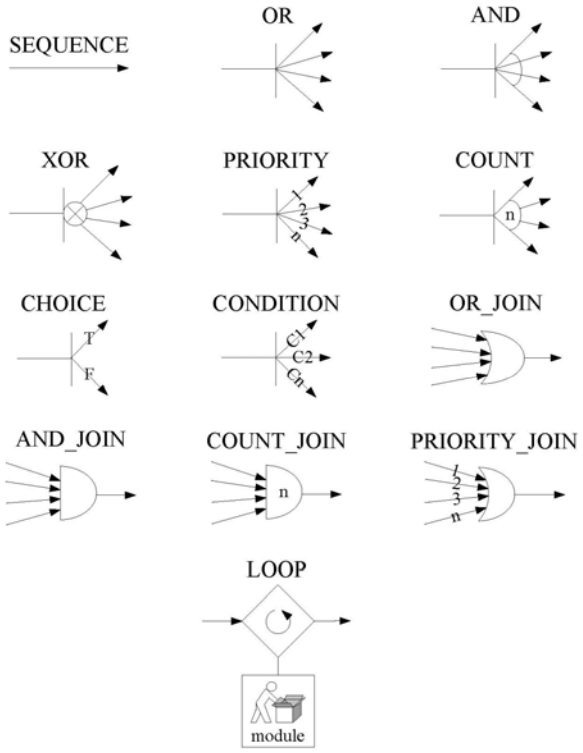

圖 3 控制圖件

圖 4 是一個旅行規劃之 AC Diagram 範例，表示安排一個商務旅行必須同時安排機票和旅館，二者皆成功才能成行，任何一個失敗都會導致取消整個旅行安排。AC Diagram 的設計，著眼於直覺化和視覺化，具備高度彈性同時語意明確，讓非資訊專業的商務使用者也能運用自如。

在使用者設計完成一個商務程序之 AC Diagram 之後，我們的商務模式化系統(請參考[12])可以依據正規語意，自動的將 AC Diagram 轉換成 XML 工作流規範(XML workflow specification)，成為本文所要討論的跨企業代理人交易系統之輸入。交易系統的任務，正是將此一商務流程在網際網路上順利執行，同時保證流程執行的交易特性。

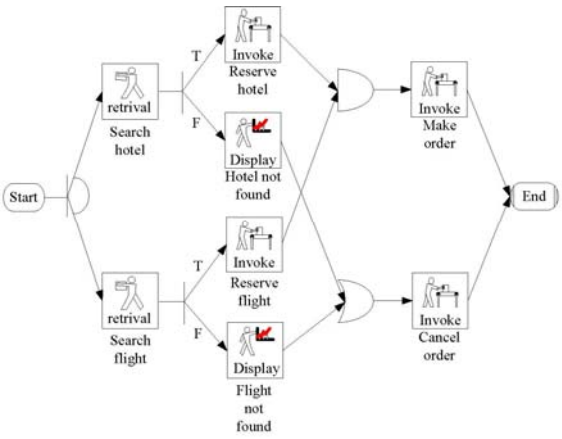

圖 4 旅行規劃之 AC Diagram

四、跨企業代理人交易系統

為了執行跨企業商務，我們使用了多重代理人技術來設計跨企業交易系統，其架構如圖 5 所示。在系統開始執行流程內容之前，使用者利用我們所提供的 XML／跨企業交易轉換器（XML Workflow to Cross Enterprise Transactions Translator）來將設計好的活動控制圖予以規劃並使用 Enterprise Information Manager 中的商業邏輯以及夥伴資訊相關等資料，進行一些必要的流程切割等動作，將原先的活動控制圖轉換成為跨企業的交易性流程描述。隨後交易流程描述規範繼續由 CATS 系統進行執行。在 CATS 中具有三大子系統。交易管理員（Transaction Manager、TM）負責整個交易活動的進行，例如交易的執行（Execute）、捨棄（Abort）、回復（Rollback）與其結果的傳遞、回報等細節。交易管理員在進行交易執行時必須與代理人管理員（Agent Manager、AM）進行密切的合作，代理人管理員負責管理代理人程式的起始（Initialize）、結束（Terminate）、以及狀態控管（Monitor）等動作。代理人管理員將無時無刻的控管著代理人池（Agent Pool）中的代理人，以確保整體系統的穩定性。而系統的主角就是一個個存在於代理人池中，稱之為泛用交易代理人（Generic Transactional Agent、GTA）的代理人程式。泛用交易代理人根據流程中每一項活動與控制的內容，執行細部的交易動作。接下來我們將進行每一個子系統的詳述。

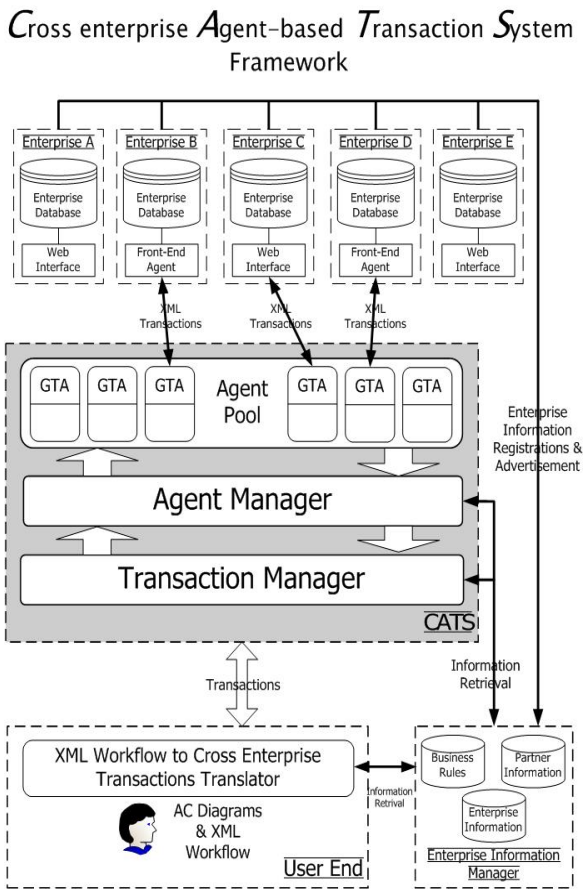

圖 5 CATS 跨企業代理人交易系統架構

我們將整個活動控制圖流程拆解成數個子流程(Subflow)，這些分解完成的子流程是 CATS 系統處理的基本單位。系統將依照子流程的特性，分配代理人來負責處理。一個子流程為一個以上的活動(Activity)或一連串的活動—流程控制—活動(Activity—Control—Activity)單元所構成。一個子流程必由活動作為起始，而最終必須由一個活動作為結束。一個活動控制圖必定至少含有一個子流程；而子流程的長度並不固定，極端的情況下一個子流程可能就代表了整個活動控制圖，或者也有可能僅僅只包含一個活動。所有子流程將繼承原活動控制圖的循序性，因為這些子流程都是從原工作流程切割出來的部分，因此子流程中的活動圖件及控制圖件當然保有與原流程同樣的順序性，切割的過程中不會破壞原有流程特性。故我們可以保證子流程執行時候整體行為的模式會遵循原流程的設計邏輯。此外每個子流程為系統處理的基本單位，因此當設計代理人程式的時候將不會太過於複雜，並可提高每個代理人的使用率，進而提升整體的執行效率。最後，流程開始執行之時我們也可以判斷那些子流程是需要被執行，那些不需要被執行，藉以減少冗餘的執行以及資源的浪費。

(一) 流程切割演算法

當一個大流程被切割成數個子流程的時候，我們賦予這些子流程一些特性，以保留原活動控制圖所具有的處理特性。因為一個活動控制圖中，控制元件細述了它的流程進行，所以我們將控制依其特性分成三大類，分別在下列各段中描述。

一般類(General Class)：包含控制圖件 SEQUENCE 和 LOOP。一般類流程控制圖件都具有單一輸出/入的特性(Single Input Single Output，SISO)，當我們在建立子流程的時候，若遇到這種控制圖件所接連的活動圖件都可以直接加入到正在建立的的子流程路徑中。

切分類(Split Class)：包含控制圖件：OR、AND、XOR、PRIORITY、COUNT、CHOICE、CONDITION。切分類流程控制圖件的特性是單一輸入多重輸出(Single Input Multiple Outputs，SIMO)，當一個子流程在建立的時候到遇到此類控制的時候，則必須選擇一個適當的輸出路徑作為正在建立的子流程的執行路徑，而選擇出來後剩下的輸出則必須要根據控制圖件的特性，建立具有該特性的新的子流程。因此具有 n 個輸出的切分類控制必須建立出 n - 1 個新的子流程出來。AND 連接的所有的活動圖件具有同步執行的特性，故系統須盡可能將這些活動於同一時間內執行。處理時候系統可利用隨機判斷的方式或經驗法則，挑選出一個 AND 連接的活動來作為接續，成為現階段正在建立中之子流程的執行路徑。其他剩下的輸出活動會再度行成一個個新的子流程，這些新的子流程我們會把他們全部加到一個稱之為同步執行子流程佇列(Concurrent Subflow Queue、CSQ)中，當親流程(Parent flow、代表可以觸發這些子流程群的上一層流程)開始進行 AND 流程控制時，就會呼叫系統執行在 CSQ 中的對應子流程群。 CATS 系統會優先指派代理人來負責執行這些子流程；而原先包含 AND 流程控制的子流程則會依照該它的原始特性（由該子流程的引導控制決定）加到對應的選擇性子流程佇列(Alternative Subflow Queue、ASQ)或是 CSQ 中。OR 的基本處理法則與處理 AND 控制的相似。不同的是，OR 所輸出到的活動圖件並不具有同步執行的特性，系統執行這類控制的時候會自行判斷要啟動幾個對應的子流程。因此一旦方向確定以後，OR 所產生出新的子流程群將存放在 ASQ 中以便執行。XOR 與 OR 類似，但只能選擇單一輸出，其他的輸出有互斥的特性，所以剩下的輸出活動所產生的新子流程則會被置放在互斥性子流程佇列(Exclusive Subflow Queue、ESQ)中。PRIORITY 必須遵照優先權選擇輸出活動時候。因此只需根據使用者定義的次序來選擇流程進行方向即可，而方向選定後的後續處理方式與 OR 控制相同。COUNT 與 AND 類似，只是同時啟動的子流程依據數目而定。CHOICE 在執行期間一旦決定輸出走向以後，另外一個輸出的方向將永不執行，故 CHOICE 的輸出有互斥的特性。當 CHOICE 選擇完成輸出活動後，剩下的輸出活動所產生的新子流程則會被置放在互斥性子流程佇列中，執行期間

內除非有 CHOICE 流程控制判定需要執行 ESQ 中的流程，否則在 ESQ 中流程將不會被系統執行。最後，CONDITION 則是條件式的 AND，處理方式與 AND 類似。

結合類 (Join Class)：包含 AND_JOIN、OR_JOIN、PRIORITY_JOIN、COUNT_JOIN。結合類圖件的特性是多重輸入單一輸出(Multiple Inputs Single Output，MISO)。此類控制圖件主要在進行流程的彙整，完成彙整就會根據條件判定是否繼續進行或回復。其處理方式皆類似，首先檢查在所有已經建立完成的子流程中（包括 ASQ、CSQ、ESQ）是否有存在一個子流程已經包含了該控制圖件，若有，則結束掉正在建立的流程。若無就將該控制圖件加入到正在建立的子流程中，並且繼續由該控制的輸出方向建立流程的下一步。

了解控制圖件的性質後，我們可以知道在建立子流程的時候必須準備好三個佇列：選擇性子流程佇列、同步子流程佇列、及互斥性子流程佇列。所有切割建立出來的子流程都會被置於 ASQ、CSQ 或 ESQ 其中一個。切割演算法的詳細流程如圖 6 所示，基本上是依照圖論中的深度優先拜訪的方式來進行，遇到不同的控制圖件，則依據其語意，對流程做不同之處理，產生新的子流程。圖 4 之旅行規劃 AC Diagram，其切割之過程、切出之子流程、和各子流程所屬佇列結果如圖 7 所示。

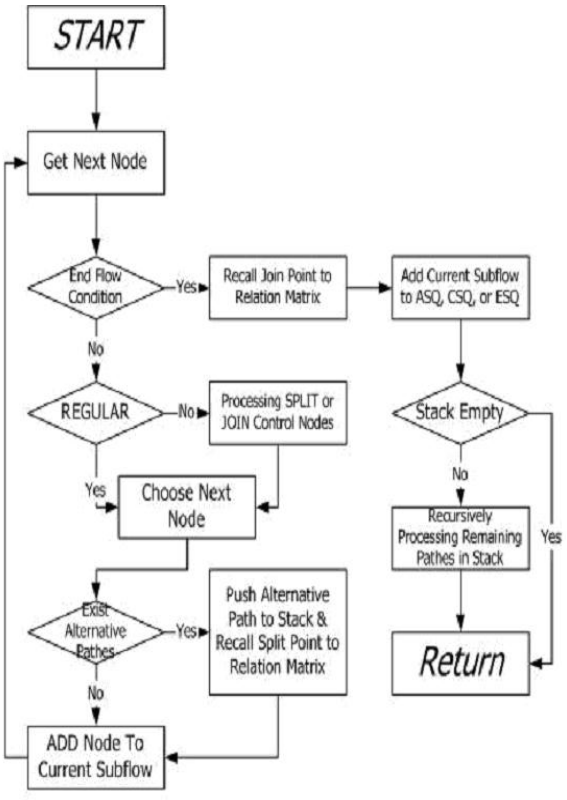

圖 6 流程切割演算法

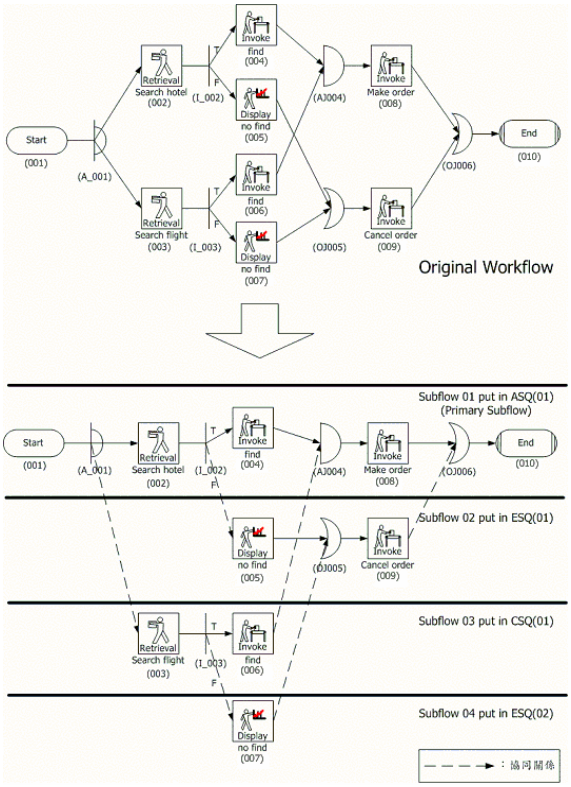

圖 7 流程切割演算法範例

我們的切割演算法及子流程具有下列特性。首先切割過程並不會破壞或顛倒兩兩元件之間的順序，即使兩兩子流程間有關係，我們也使用了關係矩陣的方法把二者間的因果關係紀錄下來。因此系統在執行流程的時候必定是依照原流程的語意來執行流程，不會有問題。其次，由於演算法是依照深度優先的方式來拜訪，因此我們可以保證原始活動控制圖必定會被完整的切割為子流程集合，且不會有任何的活動或是控制被遺忘掉。每一個活動圖件或控制圖件在確定所屬子流程之後立即加以標示，所以必定會屬於唯一的一個子流程，如此可以保證執行的時候不會發生同一活動重複執行的情形。最後，每一個可以分支的控制都會建立出新的子流程，不同的子流程若無因果關係即可同時執行，在系統資源允許的情況之下，系統可以針對流程執行最佳化，達成可最大同步化（Maximal Parallelism）之特性。

(二) 多重代理人工作流程引擎

經過切割以後的子流程集合就是我們要執行的對象，一旦這些子流程中有某個流程經過複雜的運作而到 END 以後，全部的流程便結束。在執行流程的時候，為了能夠並列執行這些流程，並且應付各種不一樣的錯誤情況，我們導入多代理人的技術並使用交易管理系統來管理子流程的執行。有了管理系統，我們就可以對子流程進行執行（Execution）、交付（Commit）、回復（Rollback）等常見的交易運作，讓工作流程具有 ACID 的交易

特性，在發生錯誤的時候也可以適當的處理。

交易管理系統

我們將活動控制圖切割出來的每個子流程當成是一份交易的手稿（Transaction Script）並且交付系統執行。交易管理系統就是為了能夠讓每一個流程的執行具有交易的特性而設計。由圖 8 可以看出我們交易管理系統的組織架構。最底下的部分是一個 XML 子流程的接收端，接收並剖析流程切割完畢之後每一個子流程的 XML 交易描述手稿。接下來的核心執行器會執行剖析完的交易手稿。系統中有一個部分專門跟代理人管理系統溝通，這的部分接收來自代理人的狀態資訊，當有子流程出問題必須回報核心執行器的時候，溝通回報器會回報核心控制器作判斷。核心判斷器將會根據交易錯誤處理器之結果讓代理人執行新的流程或回復流程，我們將設計一系列的交易演算法來讓流程可以順利進行。

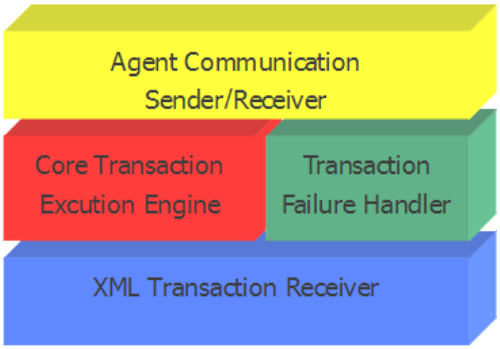

圖 8 交易管理系統架構圖

交易執行演算法

交易執行方法如圖 9 所示。系統首先從 ASQ 中挑出主要子流程來執行，當流程執行到活動圖件的時候，系統就會根據活動內容去執行。當執行到控制元件時便會判斷如何進行流程。若是 AND 控制，則系統會將 CSQ 中的對應子流程同步執行；若是 OR 控制，則系統會根據目前可用的資源來判斷，若資源充裕，則同步執行在 ASQ 中相對應的子流程，若資源不足，則系統會保留無法執行的流程，待有資源可用的時候再執行；若是 PRIORITY 控制，則系統依照定義的優先權順序來執行相對應的子流程，其他則是資源多寡予以同步執行或保留；而 CHOICE 或 CONDITION 判斷會根據之前活動所得到的成果，判斷流程方向，若需要執行到 ESQ 中相對應的子流程，則系統就必須將 ESQ 中的子流程載入執行。若是 AND_JOIN 圖件則代表著流程進行到該處必須等候所有的輸入流程都完成後才可繼續。若超出等候的限制時間，則系統必須要開始對流程進行回復；而 COUNT_JOIN 只要在限制時間內達到一定數量的成功流程輸入就可以繼續下去，若否則進行回復。OR_JOIN 只要有任何一個成功的流程輸入，就可以繼續進行下去，但若有兩個成功的流程先後抵達 OR_JOIN 則先到的流程先進行，剩下的流程則會等待，但萬一成功的流程發生錯誤回到 OR_JOIN 的時候，那麼在等候中的流程們就會依照先後的順序來執行。執行 PRIORITY_JOIN 時候系統必須依照優先權的方式來進行輸入資料判定，讓高優先權的成功流程輸入先繼續進行下去，一旦流程發生錯誤回到 PRIORITY_JOIN 的時候，系統會依據優先權的先後次序來決定誰先繼續下去。此外 LOOP 的處理方式較為特殊，LOOP 控制主要的功能是將我們的活動控制圖中有循環的部分包成一個控制。LOOP 執行的時候會另外再執行新的活動控制圖模組，當這個模組執行結束以後再用執行結果與限制條件比較，直到滿足給定的條件才可繼續進行。因此系統在執行 LOOP 控制的時候會暫時的將原先的流程停止，然後去執行 LOOP 所包含的活動控制模組，當執行結束後會進行判斷，直到滿足使用者給定的條件以後，原先的流程才會繼續進行下去。

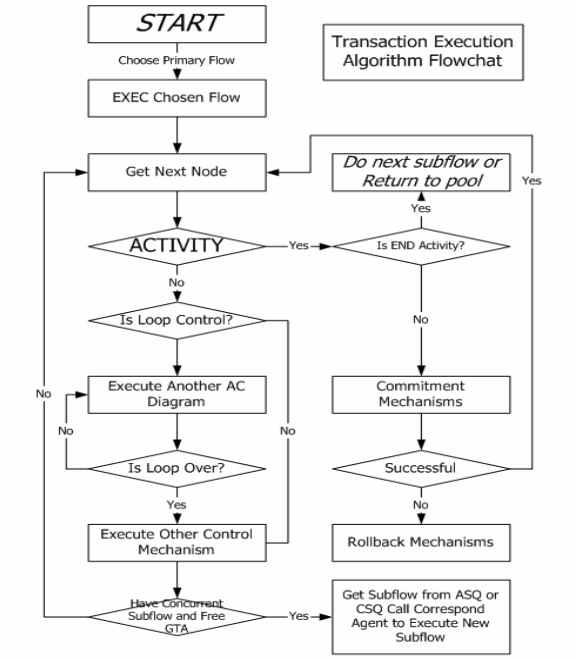

圖 9 交易執行演算法

交易錯誤處理及回復演算法

流程運作發生錯誤的時候，我們必須要能夠儘早將錯誤的部分予以回復，以免造不當的交易執行結果。處理錯誤的方式，首先偵測哪一個流程的活動或控制出錯，並且開始回復。從我們可以從圖 10 知道，系統由流程發生錯誤的位置開始，對錯誤的活動或控制進行回復(Rollback)。我們假設每一個活動必須具有相對應的回復機制，而系統作活動回復的時候就是使用相對應的回復機制來完成。接下來循著錯誤流程進行的方向反向執行（虛

線的方向)，每次執行到活動圖件的時候便執行相對應的回復機制將完成的結果撤回。當反向執行到切分類控制的時後便會查看是否有可以繼續進行的替代流程。可以繼續進行的替代流程有三種，第一種是尚未執行的流程。第二種是已經執行完成的流程，若有流程已經執行完成，則系統可以直接將完成的結果加入，並從已完成流程的路徑繼續做下去。最後一種是正在執行中的流程，當有這種流程的時候，系統會等待該流程成功的結束以後，按照處理已完成的流程的方法來繼續進行。如果系統偵測到有可以繼續執行的替代流程，那麼正在回復的流程會轉向去執行替代流程（粗黑線的方向）；如果沒有可替代的流程的，那麼我們就會繼續回復，直到流程的起點便結束，而當一個流程的主要子流程被回復到起點活動（也就是 START 活動）的時候，我們整個流程將會宣告執行失敗而停止。如果當我們的反向執行到達一個結合類的控制時候(如圖 11)，我們會根據該控制的特性來判斷，舉例而言，如果是一個 AND_JOIN 或 COUNT_JOIN，那我們會繼續對他的所有成功的輸入流程進行回復；如果是 OR_JOIN 或 PRIORITY_JOIN 則會根據選擇策略來決定哪一個流程可以繼續進行下去，而原先完成的流程則繼續進行回復動作。完整的交易回復演算法流程如圖 12。此外 LOOP 控制的回復方式也需要特別處理。因為 LOOP 具有執行其他流程的特性，因此當進行回復的時候，系統會先根據先前的結果判斷需要對 LOOP 所包含的活動控制圖模組進行何種的回復方式，接下來便開始逐一的對該活動控制圖模組進行回復動作。

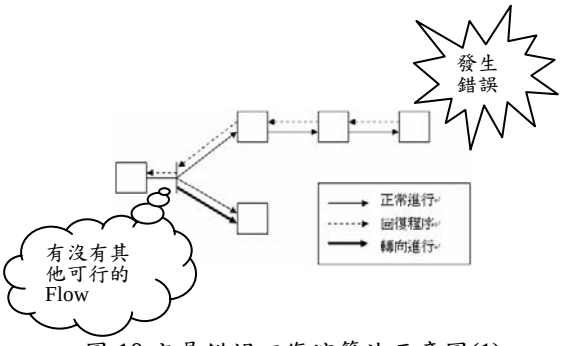

圖 10 交易錯誤回復演算法示意圖(1)

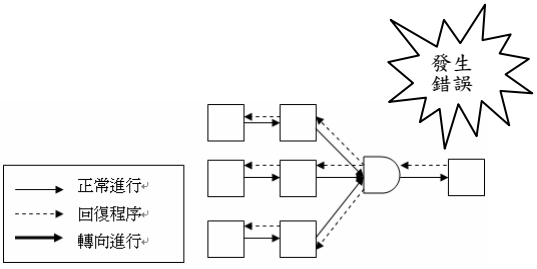

圖 11 交易錯誤回復演算法示意圖(2)

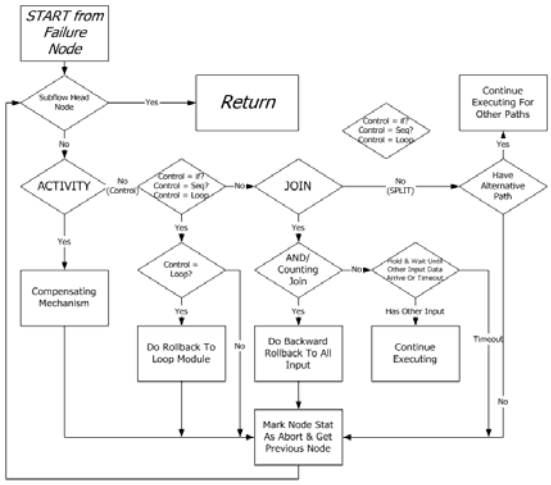

圖 12 交易回復演算法

交易結束後回復演算法

一旦有一個子流程到達 END 活動或是回復到了 START 活動以後，便代表著所有工作需要停止下來，並將結果回報給使用者。但因為並非所有流程皆是我們真正想要的結果，因此冗餘結果必須予以回復。後回復演算法(Post Rollback Algorithm)就是設計來將不需要的部分清除。如所圖 13 示，演算法由 END 活動開始，以廣度優先方式，向 START 活動的方向逐一回溯，找尋與 END 相連接的 Frontier 推展開來，並檢查已進行的活動是否為真正結果所需，若否則將該活動予以回復。當回復至 START 活動，就可以保證不會有冗餘的執行結果。

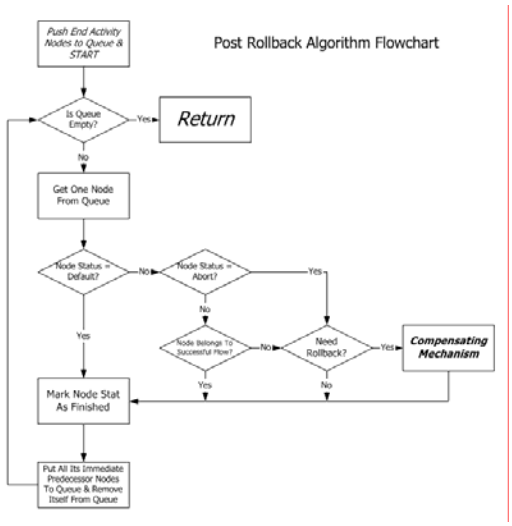

圖 13 後交易回復演算法

交易演算法特性

我們將每一個子流程視為一個延伸交易來進

行處理，滿足交易所應具有的四大特性—ACID 特性。原子性(Atomic)：子流程在執行的時候，可以被我們所指派的代理人程式完整的做完。並且當任何一個環節出問題的時候，錯誤的子流程可以被回復，也可以回溯到流程中某一個分支點繼續進行。除了保留 All or None 的特性之外，也具有彈性讓系統將可以動態的改變流程的執行，讓該流程依然可以完成而不至於忽略掉可能成功的交易結果。一慣性(Consistency)：當代理人在處理流程必定遵照該流程所定義的方式逐步完成每一個動作，如果沒有錯誤發生，代理人無法隨意的去更換流程中所定義的實行順序，因此一個流程不論是執行的先後順序如何，其結果都將一致。隔離性(Isolation)：兩兩活動圖件間只在有先後關係（happened-before）的情況下，兩個活動才會具有資料傳遞的情形發生，因此基本上只要不具先後關係的兩個活動都應該是獨立而不會互相干擾的兩個執行個體。因此在兩兩子流程之間除了觸發（由切分類控制所引起）的以及匯整（由結合類控制所引起）關係外，彼此的活動進行都不會影響到不同的子流程中的活動的進行。永續性(Durability)：當一個子流程已經執行完成它所定義的所有活動以及控制之後，這個子流程所完成的結果將會被系統持續保持下來，除非是錯誤發生而導致回復，或是後回復動作時被回復，我們的結果都不可能被取消。ACID 的特性可以保證流程的執行結果一定是正確可信任的。

代理人管理系統

代理人管理系統是根據 FIPA (Foundation for Intelligent Physical Agents)的建議架構所設計的[5] 。透 FIPA 的代理人建議架構，代理人們可以相互合作完成任務，而且不同的代理人平台也可以代理人溝通語言(Agent Communication Language、ACL)來溝通，讓我們的代理人系統可以跟其他代理人系統相互合作，達成一個分散式的開放平台。代理人程式是一個具有智慧的程式，它可以根據所處的環境及所面臨的狀況採取反應，用於分散式的交易環境可以更有彈性的執行分散式的交易。代理人本身的架構則是採用由 Q. Chen、P. Chundi、Umesh Dayal、M. Hsu 等人首先提出來的動態代理人技術[3] ，基本想法是將代理人的智慧與基本能力分開來，一個動態代理人具備動態行為能力(Dynamic Behavior)、分散式環境通訊能力(Distributed Communcation)、移動性(Mobility)、資源以及知識管理(Resource & Knowdege Management)。最重要的是可以動態載入知識或行為的能力，因此可在商務流程執行中，依據不同的任務，使用不同的行為，亦可能在執行期間不斷的載入並卸除一些行為能力來應付各種不同的狀況。我們設計了泛用交易代理人(Generic Transaction Agent、GTA)來執行任務。泛用交易代理人是繼承動態代理人的概念，並加入交易執行能力的設計而得。在圖 14 架構中可看到泛用交易代理人具有 Base Communication Facility 來和網際網絡或代理人管理系統進行溝通。而 Flow Excution Engine 則是交易執行核心，這裡面包含了我們所設計的交易執行演算法，用來執行流程所指定的交易。Exception Handler 專責處理各種例外狀況的發生，包含流程進行時候每一個活動或控制執行時候所發生的例外以及代理人程式的例外等等。Agent Cooperation Facility 處理泛用交易代理人與各種其他代理人合作的進行。當系統開始執行流程的時候，泛用交易代理人會逐一剖析我們所設計的 XML 交易流程資訊中每一個活動及控制，此時 Dynamic Behavior Loader 就會根據使用者在活動或控制中用 XML 所定義的相關資訊來查詢對應的行為以及知識，讓 Dynamic Behavior Loader 將查詢出來的行為藉由網路載入到 Dynamic Behavior & Domain Knowledge，然後再用這個行為來執行，完成流程活動所需指定的工作。例如當系統與 IBM 公司以及 SUN 公司進行交易的流程的時候，泛用代理人首先根據與 IBM 公司交易的活動內容向知識系統查詢相對應的行為，接下來便將這個行為載入，並且根據活動內容的參數、條件、以及限制等相關資訊來進行交易。當與 IBM 公司交易完成以後，便會得到一個交易結果，接下來當泛用交易代理人要跟 SUN 公司交易的時候，則會將先前所載入之行為（與 IBM 交易的行為）先移除，接下來利用與 SUN 公司交易的活動內容向知識系統查詢相對應的行為，然後載入行為，執行交易。

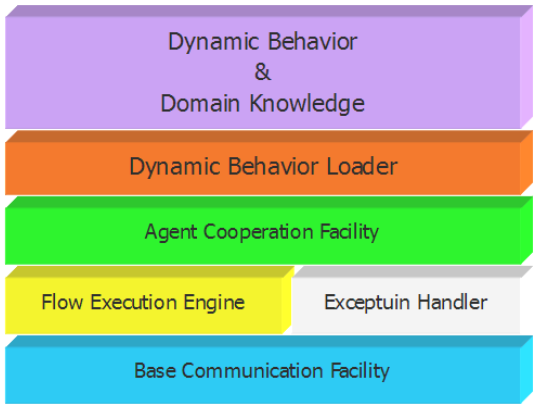

圖 14 泛用交易代理人架構

代理人分配方法

當一個活動控制圖被我們切割成為數個子流程後，交易管理系統會根據代理人管理系統所具有的代理人資源來分配子流程給代理人們去進行運作。我們在此也提出數個分配代理人的概念性方法提供系統的實作參考。

靜態代理人預留分配法是先根據專案內容向代理人管理系統要求代理人資源，此時代理人管理系統會檢查現有代理人是否充足、整體系統的負載

是否過高等因素來決定是否接受交易管理系統的要求，如果現有代理人不夠充足，則回應代理人不足的訊息及目前可用的代理人數量，此時交易管理系統可以決定是否要接受不足額的代理人來進行流程運作或是將這些結果交由使用者來判斷。除了讓使用者去指定代理人數量的方法外，交易代理系統也可以進行流程需求的評估。交易代理系統可依據一些因素來決定代理人需要的數量，例如時間、總體流程所需要耗費的系統資源數量等。靜態代理人預留分配法可以保證每個進入執行的流程都會具有足夠的代理人來進行運作。但是這種方法需要先對代理人管理系統進行要求，必要的時候必須計算每個流程的負載量來決定要求代理人的多寡。因此面對一些執行期間特殊的錯誤時候（例如代理人程式錯誤而當掉的時候），處理起來會較為複雜，也較沒有彈性。

相較於彈性較差的靜態代理人預留分配法而言，動態代理人分配法的代理人分配將是動態的進行而非靜態的配置。因此代理人系統隨時都有不同的代理人被分派出去，也隨時有不同的代理人被歸還到系統中。最簡單方式就是需求式代理人分配法，也就是讓負責流程的各個交易管理系統根據實際需求配置代理人來進行任務，用完後隨即更還給代理人管理系統，以便其他的交易管理系統使用。需求式分派法有幾個基本原則要注意，首先必須先分配給位於 ASQ 中的主要子流程代理人來執行。其次，由於 CSQ 具有同步執行的特性，因此在 CSQ 中的子流程有較高的優先權與同一流程的其他子流程競爭代理人資源，而 ASQ 中的子流程除主要子流程之外其優先權都較 CSQ 中得子流程為低。最後 ESQ 中的子流程並不需要額外的執行代理人，只需要讓原先執行該流程的代理人去選擇流程的執行方向即可。動態分配法的好處是方法直觀，彈性大，不需要預先執行評估運算，而且可以在代理人使用完後馬上歸還系統，讓系統隨時都可以有代理人資源可以用。但是此的缺點在於執行期間若系統負載過重，流程可能有要不到代理人的問題，如此會大大的降低執行流程的效率，並且有可能間接導致流程執行的失敗。

由於靜態預先配置法與動態分配法各有優缺點，因此可以截長補短採用混合式的分配法，在執行一個流程之前，先用簡單快速的評估法預測需求，接下來執行期間讓交易管理系統依照動態分配法的方式向代理人管理系統要求配置代理人，如此可兼顧效能與彈性。

代理人知識系統及呼叫介面

知識系統是由許多的行為(Behavior)組成，行為可以讓代理人具有執行更多不同任務的能力。另外我們提出通用代理人介面，讓使用者及協力開發者一同參與設計行為。這些介面包含：

- **參數傳遞介面 (Set_Parameter)**：利用 Set_Parameter 介面來將活動所設定的設定參數傳遞到行為中。
- **執行介面(Action)**：使用執行介面讓代理人真正的去執行動作。
- **取得結果介面(Get_Result)**：當執行完成之後，代理人可透過取得結果介面來得到執行後的結果。

以上所提供的三個介面為最主要的行為呼叫介面，更完備的呼叫介面則超出本文討論範圍。

(三) 其他子系統概述

在 CATS 系統架構中，還有許多其他的子系統負責不同的工作。企業資訊管理系統可以提供系統每個企業的基本資料讓系統在作切割流程、執行期間交易決策、以及交易時所需要利用的知識等相關資訊。企業資訊管理系統包含幾個主要部分：

- **Business Rules**：記載著一些由使用者建立的商務活動規則。提供系統在與其他企業交易時的參考。當系統在處理商務契約的時候，便可以引用這些規則來判斷。此外也作為提供流程切割演算法執行時候的切割準則。
- **Enterprise Information**：記載著企業的內部資訊，由使用者負責維護。當與其他企業交換資訊的時候，這部分的資訊就會被擷取出來。Enterprise Information 僅允許由內部存取以確保企業的重要資訊不會隨意的外流。
- **Partner Information**：記載著企業夥伴的相關資訊，其來源可由使用者建立或由夥伴企業來向我們註冊。在進行交易時後，如果有新的資料則系統會更新舊有資料。

五、系統實作與效能分析

(一) 發展環境介紹

我們選擇開發的語言是 SUN Microsystem 所發展的 JAVA 語言。並用 EMORPHIA[4] 所提供的一套 Component-based 的開發工具 FIPA-OS[6] 來進行實作。FIPA-OS 提供一套完整的代理人應用程式介面（Agent API）及完善的代理人平台讓我們來使用。XML 相關文件處理則是利用 Apache Group 所發展出來的 Xerces2 Java Parser[1] 。

(二) 系統實作介紹

我們將每一個活動圖件包成活動單元（Activity_Unit），活動單元包含三個屬性值（Attibute），分別是時間統計屬性（Time_Statistics）、輸入屬性（Input）、輸出屬性（Output）。時間統計屬性記載著到子流程目前為止，所有活動及流程控制累積的最大上限時間，這個屬性可以幫助我們在評估系統的負載，可以作為

分配代理人時候的參考。輸入及輸出兩個屬性會紀錄一個活動的輸入控制 ID 以及輸出控制 ID，讓我們可以在執行活動的時候知道因果關係。而流程控制圖件則包成控制單元（Control_Unit），控制單元結構中只有時間統計屬性記載著流程到目前為止累積的最大上限時間。

最後，切割完的每一個子流程寫成 XML 格式交易描述包覆在 <Transaction></Transaction> 中。<Transaction>有兩的屬性值， Queue_Type 屬性標明了這個子流程是哪種類型的流程，而 Total_Time 則紀錄整個流程的最大限制執行時間。所有完成的子流程都會被紀錄到一個專案資料檔中，而紀錄關係矩陣資訊的檔案也會被寫到該專案資料檔中，當要執行一項工作流程的時候，只需將該專案交給交易管理系統去運作即可。

我們將交易管理系統設計成一個交易管理代理人，並且用 FIPA-OS 來將它實做出來。並且設計一個如圖 15 所示可以清楚顯示及控管整個交易進行的 GUI 介面來。

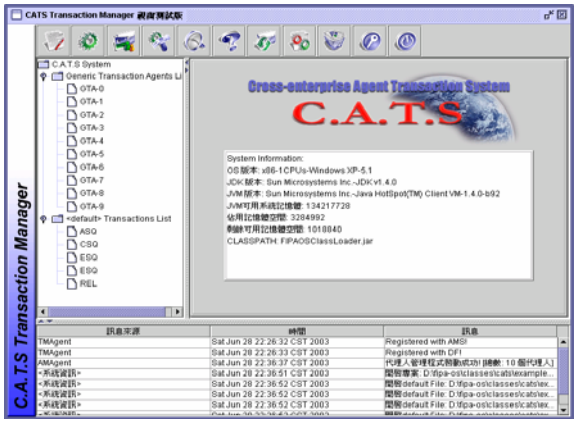

圖 15 CATS 交易代理人管理介面

我們利用 FIPA-OS[6] 來實作我們的泛用交易代理人，並且遵循 FIPA[5] 所公定的規格書為標準來設計。我們的泛用交易代理人實作了 FIPA-OS 所提供的 Agent Shell，將每一個流程的執行視為一個 Task，利用 Task Manager 來負責傳遞訊息到交易管理系統及代理人管理系統。而代理人之間則是利用 ACL 的請求協定(Request-Protocol)來進行溝通。泛用代理人的流程執行由活動執行器以及流程控制器來進行。活動執行器由 ActivityProc 類別來實作(圖 16)，我們將每一種活動都設計成不同的活動類別。每一個活動敘述都會經由對應的活動處理類別剖析，並且處理活動參數的設定以及對應行為的載入。而流程控制器由 ControlProc 類別來實作圖 17，每一個流程控制的相關運作邏輯都放置在同一個類別中，一旦我們要進行流程控制的時候只要將輸入資料放入該類別中便可，流程控制器將會判斷出流程應該進行的方向。

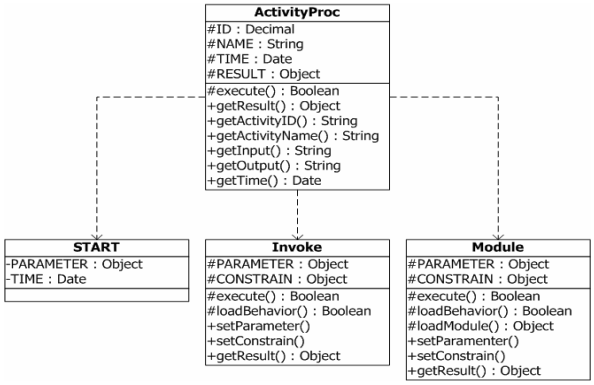

圖 16 ActivityProc 的 Class Diagram

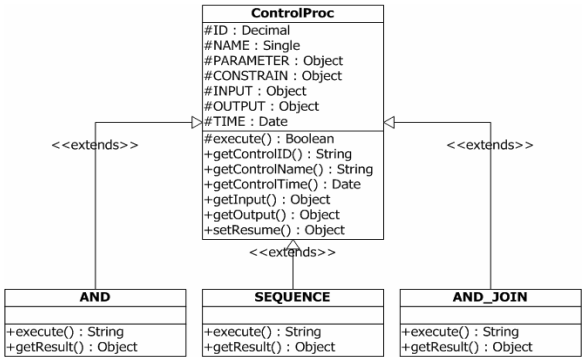

圖 17 ControlProc 的 Class Diagram

(三) 效能分析

首先要針對系統是否能正常執行完成使用者所設計的活動控制圖。我們的測試平台是 Intem Pentium4™ 1.8G, 512 MB RAM 系統是 Windows® XP Professional SP-1，SUN Java™ SDK，使用 FIPA-OS v2.1.0 作為 Agent Platform。我們利用如圖 18，經過標示的旅行代理業的範例來測試。在此範例中，最需要執行的流程為包含 Search Hotel 的主要子流程及包含 Search Flight 置於 CSQ 中第一個流程（由 003 為起點所形成的流程），只要這兩個流程成功，則我們就可以下訂單（ID=009 的 Make Order）。執行的結果如圖 19 與圖 20 所示，可以知道執行結果的正確性。

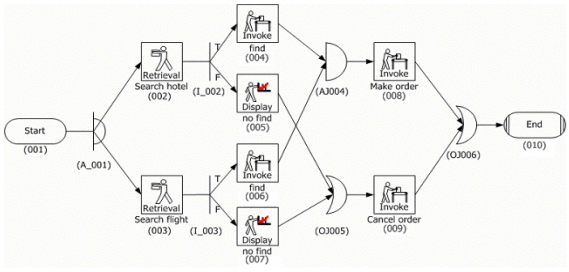

圖 18 旅行代理業範例

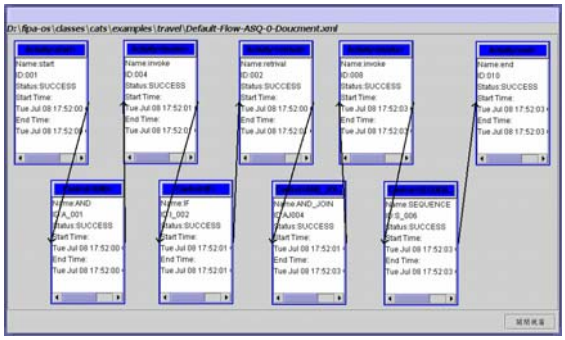

圖 19 主要子流程執行結果

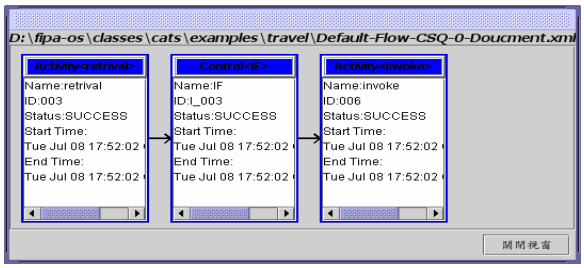

圖 20 CSQ 執行結果

接下來針對系統進行錯誤回復測試。當 ID 為 005 的 Invoke 活動產生執行錯誤，流程會由失敗位置開始進行回復，當回復到 A_001 的 AND 控制時，系統將等待或執行以 003 為起點的流程，若執行完畢，那整個流程就算完成。如果失敗，系統將繼續回復直到 001 後便會結束執行。執行結果如圖 21 和圖 22，顯示系統可以正確的回復錯誤的交易。

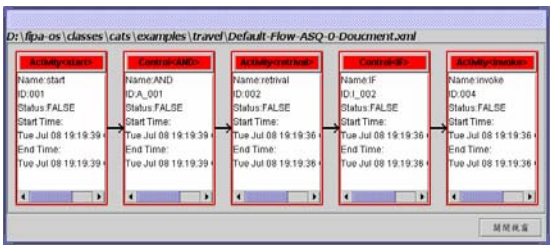

圖 21 ID 005 之活動錯誤回復後的結果

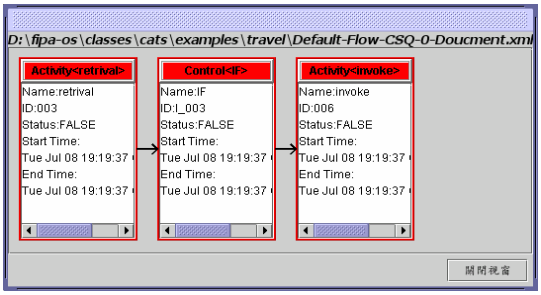

圖 22 CSQ 中錯誤流程回復結果

對於執行效率的測試，我們假設系統執行的時間應該與需要執行的活動及控制的總數量成正比，因此如果活動以及控制的數量增加，則系統的執行時間應該成線性比例的成長，而不能呈現指數成長趨勢。我們設計一個活動控制圖流程，並且設法延伸這個流程的長度，並固定代理人的數量來進行實驗。實驗的結果如圖 23 所示，當流程的長度變長了以後，整體執行時間隨之增加，而增加的幅度經由圖表的推算得知略成線性成長，與預期的結果一致。但值得注意的一點，由圖中可以看到活動的執行佔了大多數的時間，而控制所佔的比例較少。但是隨著活動的增加，雖然活動及控制的執行時間增加了，但系統額外的負載時間也同時增加，這並不尋常。經過討論和分析，我們初步認為改善活動搜尋演算法以及降低系統呼叫，應該可以提升效能。

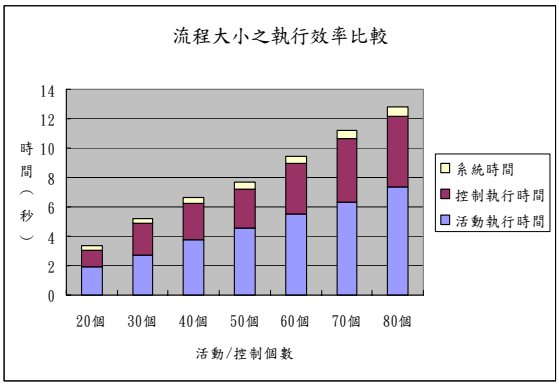

圖 23 活動控制個數與執行時間的影響

我們利用一個複雜度高的活動控制圖來測試代理人個數對執行時間的影響。從圖 24 可以看到，當可執行的代理人數量較少時，系統可能會執行較慢，而當代理人數量增加的時候，系統執行效率逐漸提升。當可用代理人數達到流程執行所需的最大數量之後，執行的效能便不會增加。

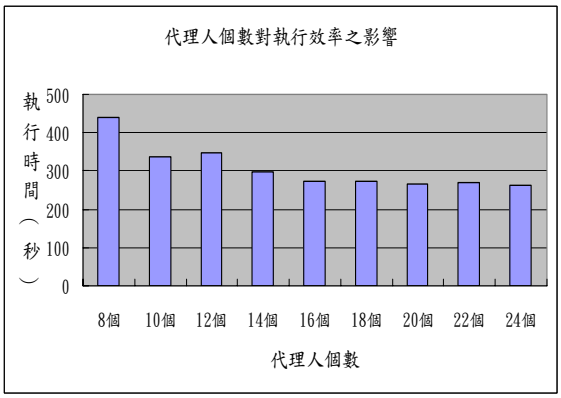

圖 24 代理人個數對系統執行之效率

六、結論與未來展望

在本篇論文中，我們將複雜的商務流程利用活動控制圖來呈現，而且設計了活動控制圖的分析方法，讓活動控制圖可以正確的描述商務活動。我們也提出了一套基於動態代理人技術的多代理人交易平台來進行流程的操作。並設計了流程切割演算法，適度的將原先的活動控制圖切割成較小的子流程單元。融合了傳統的交易觀念，我們設計了流程執行演算法，錯誤回復演算法及流程結束後回復演算法來處理流程的執行，讓流程執行具 ACID 特性。最後，我們利用動態代理人的概念設計了泛用交易代理人，並且設計代理人管理系統來負責代理人的一切狀態控管。利用泛用代理人的技術，代理人具有高度彈性來處理各種不同的任務，並提高每個代理人的使用率。我們相信我們所發展出來的一系列結果以及系統都可以很快速的應用在真正的商務活動上，並且應該可以獲得不錯的效率。

目前我們所設計的系統已經可以正確運行，而系統未來發展的重點與研究目標，第一是加強活動控制圖的正確性分析，研究有效的演算法來幫助分析流程的語意，並且建立錯誤語意的樣板的資料庫。第二是研究並設計資源評估演算法來幫助我們執行一些系統效能分析。第三是改善系統執行效率，包括減少系統呼叫以及改善搜尋演算法。第四是提供更好的知識系統／行為的設計介面，提供協力設計者開發知識行為的依據。最後我們希望擴展系統架構，使成為網路服務（Web Service）以進一步提高開放性與互通性。

七、致謝

本研究部分由國科會計劃 NSC91-2213-E-259-017 支持，僅此致謝。

八、參考文獻

[1] Apache. XML Project Home at: http://xml.apache.org/

[2] Q. Chen, P. Chundi, Umesh Dayal, M. Hsu, "Dynamic-Agents", *International Journal on Cooperative Information Systems*, 1999.

[3] Q Chen, U Dayal, M Hsu, ML Griss, “Dynamic-Agents, Workflow and XML for E-Commerce Automation”, *EC-Web*, 2000.

[4] EMORPHIA. Hompage at: http://www.emorphia.com/research/about.htm

[5] FIPA Offical Home at: http://www.fipa.org/

[6] FIPA-OS Project Home at: http://sourceforge.net/projects/fipa-os/

[7] RH Guttman, AG Moukas, P Maes, “Agent-mediated electronic commerce: a survey”, *Knowledge Engineering Review*, 1998.

[8] H Li, “XML and Industrial Standards for Electronic Commerce”, *Knowledge and Information Systems*, 2000.

[9] RJ Glushko, JM Tenenbaum, B Meltzer, “An XML framework for agent-based E-commerce”, *Communications of the ACM*, 1999.

[10] Solomon H. Simon, *XML: Ecommerce Solutions for Business and IT Managers*, McGraw-Hill, Inc. 2001.

[11] WfMC, The Workflow Management Coalition (http:// www.**wfmc**.org)

[12] Wu, Shiow-yang and Kuo-Chang Lin. “Cross Enterprise Business Modeling with AC Diagrams and Workflow Patterns”. *IEEE CEC 2005: 7th International IEEE Conference on E-Commerce Technology*, Munich, Germany, July 19-22, 2005.

[13] XMLchinese.com. Available online at: http://www.XMLchinese.com