國 立 交 通 大 學

資 訊 工 程 學 系

碩 士 論 文

軟體代理人開發工具之設計

The Design of A Software

Agent Development Tool

研 究 生 ：黃 鼎 新

指導教授 ：鍾 乾 癸教授

中華民國九十三年七月

軟體代理人開發工具之設計

The Design of A Software Agent Development Tool

研究生：黃鼎新　　Student：Ding-Hsin Huang

指導教授：鍾乾癸　　Advisor：Chyan-Goei Chung

國立交通大學

資訊工程學系

碩士論文

A Thesis

Submitted to Department of Computer Science and Information Engineering College of Electrical Engineering and Computer Science National Chiao Tung University in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science and Information Engineering

July 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年七月

軟體代理人系統開發工具之設計

研究生：黃鼎新　　指導教授：鍾乾癸

國立交通大學

資訊工程研究所

摘要

軟體代理人是單一獨立程式，可不需外部的操控即可自主執行以達成目標任務，並可藉由代理人間之訊息傳遞合作達成共同的目標 [2]。由於代理人具有自主性，移動性，合作性，及學習性等獨特特性，因此已被專注應用於多執行緒的大型分散式系統，如資料搜集、網路管理、網路交易等等系統 [13][17]。爲加速代理人程式之開發，已有許多軟體代理人開發工具問世 [1][8][18]，然而這些工具均著重於代理人程式間類別的發展，缺乏代理人系統架構分析到代理人程式執行驗証的整套工具。因此本研究提出一套漸進式發展的代理人系統開發工具，以支援軟體代理人系統之設計、製作及驗証。

在軟體代理人設計階段，此工具可支援使用者定義軟體代理人系統的系統架構，包括代理人的目標、工作、代理人間關係、行動性、及學習性；在製作階段，此工具自動依各代理人特性與關係，自動提供樣程式供設計人選擇填入，以節省程式撰寫時間；在驗証階段，本工具支援使用者定義代理人系統之佈局與各代理人啓動之順序，以驗証代理人系統程式之正確性。

本工具架構於 IBM 分司 Aglets 軟體代理人平台系統上 [1][19]，利用此工具確可彌補 Aglets 系統在系統架構與系統驗証能力之不足，並可提升代理人程式發展效率與品質。

The Design of A Software Agent Development Tool

Student : Ding Hsin Huang Adviser: Chyan-Goei Chung

Institute of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
Nation Chiao Tung University

Abstract

Software agent is a program that has its own goal and execution thread. Software agent can act as a human-being that has the ability to work against any trouble and cooperate with other agents to achieve its goal. Software agent has four characteristics: autonomous, collaboration, learning human behaviors, and executing tasks anywhere in network by mobility. Therefore, software agent programs have been applied to developing large-scale multi-thread distributed system, such as data collection, network management, and network transaction. To accelerate the development of software agent-based system, there have been some development tools. However, these agent development tools just focused on coding level by providing agent classes to support user to write agent source code. There are no agent development tools to give an overall support from agent system analysis, agent behavior design, to agent system execution and validation. This thesis has design an agent system development tool that supports incremental development process and help user to analysis, design, and validate agent programs.

At agent system design phase, our tool can support users to define agent system architecture, including the goal of agent, the tasks of agent, the relations between agents, mobility of agent, and learning ability of agent. At design and implementation phase, our tool will provide proper template code for user to apply. At validation phase, our tool supports user to define agent system deployment model and execute agent programs in order to validate the execution results.

Our tool, which is based on IBM Aglets System, provides more services including architecture analysis tool and system validation tool. Users are able to increase agent system development performance and quality with our tool.

目錄

圖目錄

第一章 緒論

1.1 研究背景與動機

過去三十年來，隨著軟體系統越趨龐大，軟體系統中存在越來越多複雜的互動，尤其在多執緒的分散式系統中，更是充斥許多執行時期才決定的互動動作，而每個互動的動作又是分別由不同元件的執行緒負責，互動的過程中又包括複雜的溝通協定。要設計如此複雜的大型系統相當困難，但是在現實生活中，這類複雜的大型系統卻又十分常見。因此在過去的二十年裡，有許多的研究開始設法解決這個問題，試圖找出更好的塑模技術與開發工具，而在 80 年代發產出的軟體代理人概念，則被認爲是目前最適合的解答 [2][3][4]。

軟體代理人的概念迷人之處在於其自主性 (Autonomous)，此特性定義代理人爲目標導向且單獨執行的個體，在執行期間不需要受到外部的操作，而能主動完成目標。對於開發人員而言，軟體代理人是非常好的塑模工具 [6][14][15]，因爲代理人的自主性概念將程式元件視爲主動執行且在執行中能自行解決問題的個體，而這個概念對於分析系統的開發人員來說非常自然。

軟體代理人爲了能夠儘量不受外部指揮而自行達成目標，代理人具有幾種能力：”自主性”、”學習能力”、與”合作能力”。其中自主性表示代理人能主動執行，不需管理人員介入即能朝目標進行；學習能力則是指代理人能在執行期間習得經驗與知識，並能在後續執行過程中加以運用；合作能力則是指代理人可藉由和其他代理人交換訊息來獲得彼此所需的資料以達成目標。因此，若開發人員使用軟體代理人來開發系統，會使得傳統功能導向的運作模式變成目標導向的運作模式。

隨著網路的發達，大型的軟體系統通當會將子系統分散在數個不同的網路節點上，而子系統之間的互動則需透過網路傳遞訊息來達成，在此系統架構下，若子系統間的訊息資料交換量非常龐大，將會佔用許多網路頻寬。爲減少子系統之間大量資料傳遞，行動程式的概念應運而生，能將運算資料的程式碼送至資料所在地執行，執行完成後的結果則傳送回原子系統，以節省大量的網路頻寬，提升系統效能，如圖 1-1 所示。

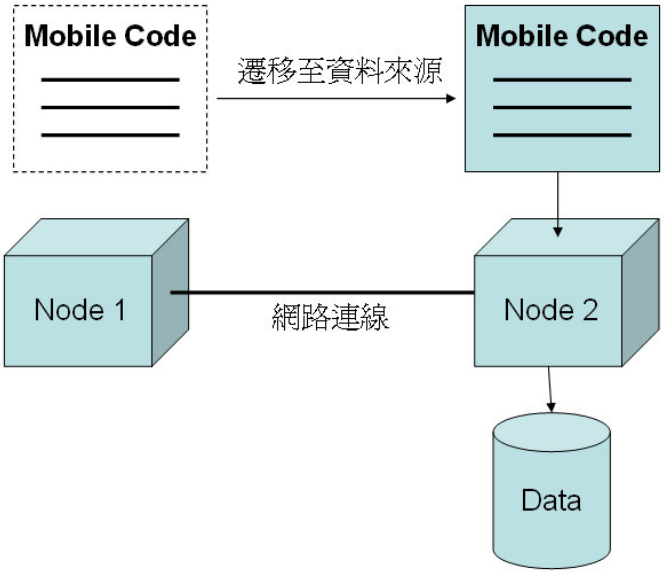

圖 1-1 行動程式的概念

軟體代理人若搭配行動程式的構想後，軟體代理人跳脫在單一主機上執行的限制，代理人功能大幅提升，具備行動力的代理人勢必取代原本的行動程式，因此，”行動力” 也成爲軟體代理人應有的特性之一。

爲了發展軟體代理人程式，現今已有許多的軟體代理人開發工具，其中較爲著名的包括 IBM 公司所開發的 Aglets [1][19]、ObjectSpace 公司所開發的 Voyager [18]、Reticular System 公司所開發的 AgentBuilder [8][9]。但是這些代理人開發工具均只提供使用者開發單一代理人程式所需之工具類別，沒有提供使

用者從最上層的代理人系統架構開始分析的工具，因此使用者無法作複雜代理人系統之完整規劃及設計，甚至利用開發工具做一致性檢查，因此，對於軟體代理人系統的設計師而言，迫切需求一套支援漸進式的軟體代理人系統開發的工具。

要設計一套軟體代理人開發工具，首先需了解軟體代理人程式與傳統程式的差異以及軟體代理人系統的開發流程。因此本論文首先分析單一軟體代理人程式的結構，以及代理人特性對程式結構之影響。對於多軟體代理人之系統，則分析軟體代理人之間的合作模式，規納出供需關係 (Provider-Requestor) 與主從關係 (Master-Slave)，代理人系統是以供需關係聯絡各個主要的子系統，每個子系統內則是由主從關係組合而成的樹狀階層架構，利用供需與主從關係，使用者可以定義出代理人系統之架構。每個代理人溝通、遷移、與學習能力又會因爲不同的系統架構與執行環境有不同的執行模式，本論文藉由分析達成各項能力的設計與執行模式而制訂程式樣板，以增進使用者的開發效率。

本研究提出 AgentIDE，以幫助設計師來幫助使用者開發軟體代理人系統之用，它提供使用者三個階段的支援：系統架構分析、代理人設計與實作、及系統執行驗証。在系統架構分析階段，AgentIDE 提供圖形化的介面，並利用前述的供需關係與主從關係幫助使用者以漸進式的方式定義代理人系統之架構，並支援使用者定義各關係的模式與各代理人之特性，讓呈現出的系統架構圖更爲清楚；在代理人設計與實作階段，AgentIDE 依系統架構圖的定義來提供各代理人適當的程式碼樣板讓使用者套用，節省撰寫程式碼的時間；在執行驗証階段，AgentIDE 則提供使用者劇本描述工具，方便使用者執行與驗証軟體代理人系統。

1.2 章節介紹

本論文共分爲八章，除本章外，其他章節介紹如下：

第二章先簡介軟體代理人概念，介紹代理人特性與代理人系統之架構，並得出目標導向的軟體代理人系統開發流程。最後會分析與比較現今的軟體代理人開發工具，並提出代理人開發工具應俱備之功能需求。

第三章將提出一套軟體代理人開發工具，AgentIDE，以支援第二章所提之需求，並進一步設計其功能與系統架構。

第四章將針對 AgentIDE 中 ”系統架構編輯器” (Architecture Editor) 的功能規格、介面設計、系統架構設計、模組設計及所需的資料結構作詳細說明。

第五章則介紹 AgentIDE 中 ”程式編輯器” (Code Editor) 的功能規格與系統架構。

第六章則介紹 AgentIDE 中的 ”佈局管理器” (Deployment Manager) 的功能規格、系統架構以及所需之資料結構。

第七章將以一 ”競標系統” 爲範例來說明如何使用本工具及其執行之流程。

第八章爲本論文之結論及未來工作事項。

第二章 軟體代理人程式特性的分析

本章主要的目的是介紹軟體代理人的概念，並針對目前軟體代理人（Software Agent）開發工具的設計理念加以分析與探討。軟體代理人 (Software Agent) 爲新一代的程式開發技術，在 2.1 節中，將會針對軟體代理人的概念與技術做簡介。在 2.2 節，將介紹軟體代理人程式之結構以及軟體代理人各特性對程式結構之影響。2.4 節將更進一步介紹軟體代理人程式之系統架構。在 2.5 節裡，我們會綜合之前的分析並設計一個代理人程式之開發開發流程。在本章最後，我們會對現今之軟體代理人開發工具做分析與比較。

2.1 軟體代理人之特性

軟體代理人的概念來自於讓電腦程式可以像人類一般主動完成工作，而不需要人類在中間介入指揮。軟體代理人可以學習人類的行爲模式，並藉由軟體代理人互相合作交換資料來完成複雜的工作。隨著網路的快速發展，使得資訊大量湧現，導致人類在龐大的資訊中要尋找真正有用的資料變得非常困難。因此便有人提出軟體代理人 (Software Agent) 的概念－讓使用者將想要達成的目標以簡潔的方式告知軟體代理人，完全委托軟體代理人代爲執行，使用者則只需要等待軟體代理人的執行結果，無需煩惱軟體代理人完成目標的過程。如此一來，軟體代理人可以幫使用者節省很多的時間與精力。

軟體代理人的定義爲：若是一個電腦程式可以主動完成使用者委托的工作，而且完成的過程就如同使用者親自去完成的一樣，那麼我們就稱這樣的電腦程式是一個軟體代理人程式。

軟體代理人爲了要順利地完成使用者委托的工作，必須俱備下面五個特性：

1 自主性 (Autonomous)：

軟體代理人接受使用者委托後開始執行，在完成任務之前所做的任何動

作，全部由軟體代理人自行決定，不需要使用者介入。

2 合作 (Collaboration) ：

軟體代理人彼此之間可以互相溝通，藉由交換訊息以得到執行任務所需要的資料，另外，軟體代理人之間也可以要求對方的服務以完成任。

3 學習 (Learning) ：

軟體代理人要能以使用者的工作模式完成委托的任務，因此， 軟體代理人必需有學習使用者工作模式的能力。此外，軟體代理人藉由和使用者溝通，或是和其他軟體代理人合作，可以從中學習有用的經驗，或是修正某些錯誤的執行方式，當下次有類似的任務委派下來時，軟體代理便能依據經驗更快速地完成任務。

4 反應 (Reaction)

軟體代理人能夠在受到外部刺激時做出適當的反應，外部的刺激包含執行環境的改變以及外部傳來的訊息。

5 行動力 (Mobility)

軟體代理人爲了要完成任務需要充足的資料，當一個執行環境不能提供充足的資料給軟體代理人時，軟體代理人可主動遷移到別的執行環境去獲取更多的資料。

以上所述的軟體代理人的特性中，反應能力可以算是自動化能力的一種，所以目前研究中的軟體代理人的四個主要特性即爲自主性 (Autonomous) ，合作 (Collaboration)，學習 (Learning) ，行動力 (Mobility) 。當一個代理人同時兼具四個個重要特性時，我們稱之爲“聰明代理人”(Smart Agent)，如圖 2-1 所示。

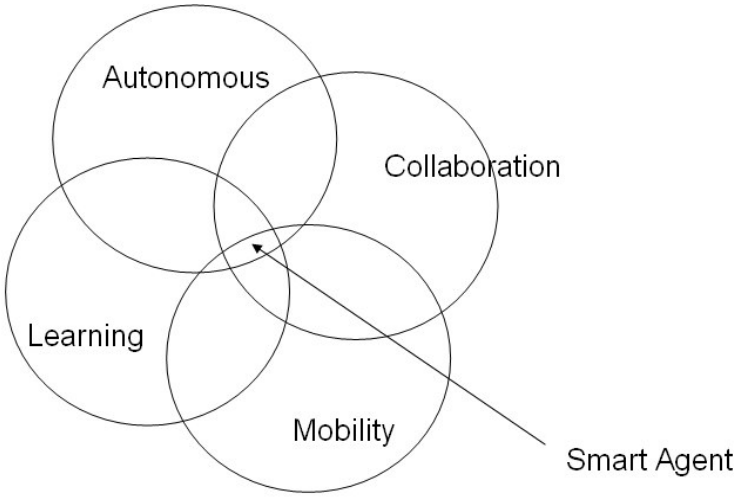

圖 2.1 Smart Agent

2.2 軟體代理人程式之結構

軟體代理人的概念類似於物件導向中的物件，擁有自己內部的方法和變數，因此目前的軟體代理人開發平台多是以物件導向的程式語言來實作 [1][2]，並且讓一支代理人程式對應到一個類別，這可以讓代理人程式繼承所有物件導向的優點。不過軟體代理人有自主性、行動力、合作等特性，使代理人程式在結構上需要有一些特殊的設計來支援。在本節中，我們會分析代理人的各個特性，其實作的方式，以及其對於代理人程式結構之影響。

2.2.1 自主性與代理人程式結構

自主性 (Autonomous) 是軟體代理人程式最重要的特性，以便代理人擁有自己的執行緒，可以單獨在任何的主機上執行。爲了要實現代理人的自主性，代理人必須有一個函式來當程式進入點，類似 C 語言程式中的 “main()” 函式。當使用者要執行一個代理人時，必須先把代理人產生出來得到一個代理人物件並去呼叫其程式進入點，代理人才會真正執行起來。代理人的程式進入點通常名命爲 “run”，爲了方便討論，我們在之後稱進入點函式爲代理人的啓動函式。

由於代理人一產生出來後，Agent Server 就會呼叫其啓動函式，因此代理人要做的工作就會安置在啓動函式之中，當使用者要開發一個代理人程式時，便要繼承代理人平台提供的代理人類別並覆寫啓動函式，如圖 2-2 所示。

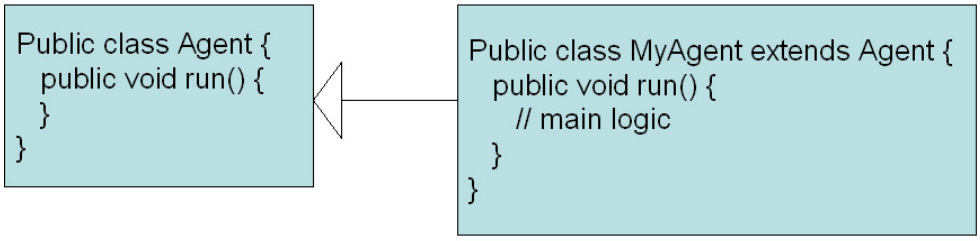

圖 2-2 繼承代理人類別並覆寫啓動函式

2.2.2 行動力與代理人程式結構

行動力 (Mobility) 讓軟體代理人能在執行時期遷移到他處，軟體代理人遷移的原因是爲了獲得更多資料以滿足目標。

軟體代理人在遷移後必需以在遷移之前的狀態下繼續執行，因此，代理人的遷移機制必須提供程式執行狀態儲存與載入的支援。實作軟體代理人平台的程式語言會影響程式執行狀態的儲存與載入方式，目前最常用之實作軟體代理人平台語言爲 Java。Java 套件提供了兩個對執行時期類別狀態儲存與載入的機制：物件序列化 (Object Serialization) 機制和 reflection 機制 [22]，物件序列化可以將系統執行時期物件內部的狀態儲存成檔案，而 reflection 機制則可以自物件狀態檔中讀出類別內部的變數值，如圖 2-3 所示。

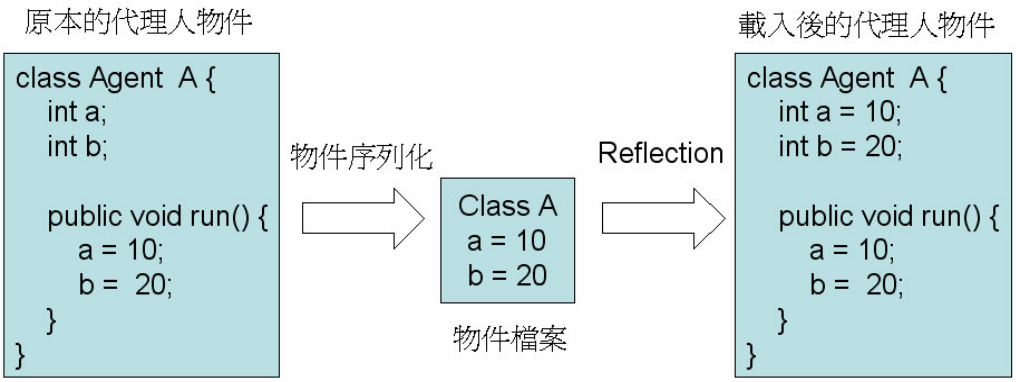

圖 2-3 Java 的物件列化機制與 Reflection 機制

由於 Java 所提供的物件序列化機制只會紀錄類別變數的值，但是沒有記錄類別在執行時期的執行堆疊 (Execution Stack) 內容，因此載入後的物件無法知道原本此物件執行到哪一個指令。若以軟體代理人來看，當軟體代理人下達遷移的指令之後，底層的 Agent Server 只會紀錄目前代理人物件內部的變數，當軟體代理人物件被送到目地主機，目地主機的 Agent Server 也只能將代理人物件內部的變數值載入回去，但無法讓代理人程式直接自遷移指令完成後的下一個指令開始執行。因此當代理人物件到達之後，底層的 Agent Server 將代理人物件的內部狀態載入完成後會直接呼叫代理人的啓動函式，讓代理人物件由此開始執行，如圖 2-4 所示，這種只儲存物件資料狀的的遷移方式稱爲 Weak Mobility。

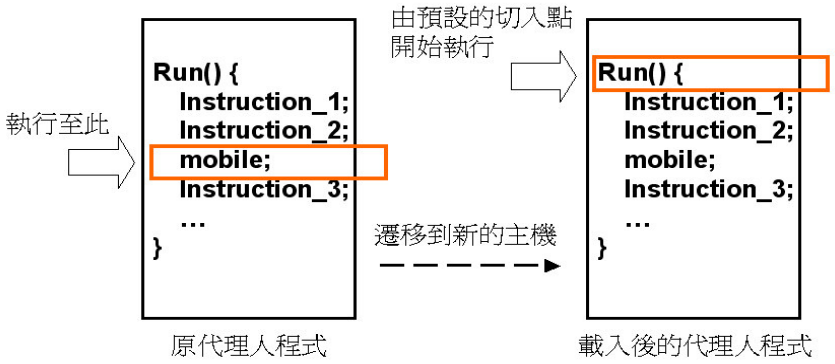

圖 2-4 Weak Mobility 的遷移機制

由於 Weak Mobility 遷移機制的影響，軟體代理人每次遷移到一個新的地方後必須要從啓動函式開始重新執行，因此，開發人員撰寫代理人程式時必須要在啓動函式中去處理狀態變化對程式內部邏輯的影響。一個有遷移動作的軟體代理人程式基本架構會如圖 2-5 所示，啓動函式一開始的主要邏輯會執行代理人在每一個遷移到的主機上要做的例行工作，當軟體代理人執行完主要邏輯後，軟體代理人會判斷本身的目標是否已經滿足，若是尚未滿足，表示代理人在這台主機上執行無法得到達成目標所需的全部資料，這時代理人便可利用遷移指令要求底層的 Agent Server 將自己送到指定的位置；若代理人執行完，發覺目標滿足，這時代理人就可以將結果回傳，或是要求 Agent Server 將自己解構。

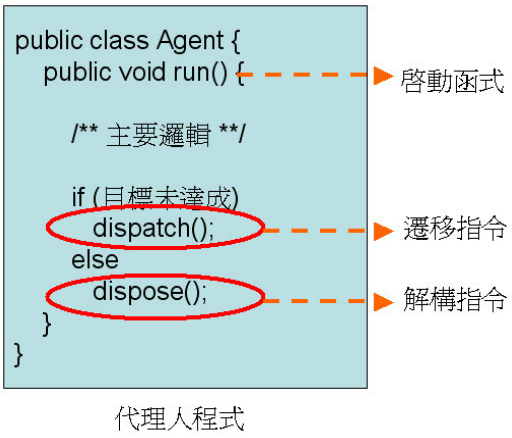

圖 2-5 軟體代理人程式的基本結構

若是要求使用者將所有的代理人邏輯都放進啓動函式中，代理人內部的邏輯會變得非常繁雜，因爲會需要許多的狀態判斷，例如，代理人內部的全域變數初始化動作若是放在啓動函式之內，而啓動函式在每次代理人遷移到一個新的主機上時就會被執行一次，那麼使用者就必須在初始化動作之前加上狀態判斷的式子，以免初始化動作被執行好幾次。爲了儘量避免這類的問題，除了原本代理人的啓動函式之外，軟體代理人平台另外設定數個其他的函式給代理人，這些函式都會有其呼叫的時機，讓使用者可以將部分程式邏輯放在其中以減少使用者在邏

輯裡做狀態判斷的次數。如表 2-6 所示，onCreation, run, onArrival 和 onDispatching，這些函式是軟體代理人平台設定的特殊函式，每個函式都有特定的呼叫時機，例如 onCreation 函式只會在代理人被產生時被呼叫一次，因此類似初始化的動作就可以直接擺在 onCreation 函式內；而 onArrival 函式則會在代理人遷移到主機之後被呼叫，因此使用者可以將每次遷移到主機後要做的動作放在這裡。

函式名稱	被呼叫的時機
onCreation	代理人被產生時, 在 run 之前
run	1　代理人被產生後 2　代理人遷移到新的主機之後
onArrival	代理人遷移到新的主機之後, 在 run 之前
onDispatching	代理人呼叫遷移指令之後,遷移動作執行之前
onDisposing	代理人被 Agent Sever 解構之前

圖 2-6 軟體代理人程式內部關於行動力的特殊函式

由表 2-1 的函式，我們可以畫出軟體代理人生命流程中各個階段呼叫函式以及其順序，如圖 2-7 所示。圖中的每個著色方塊都代表代理人內部的特殊函式，

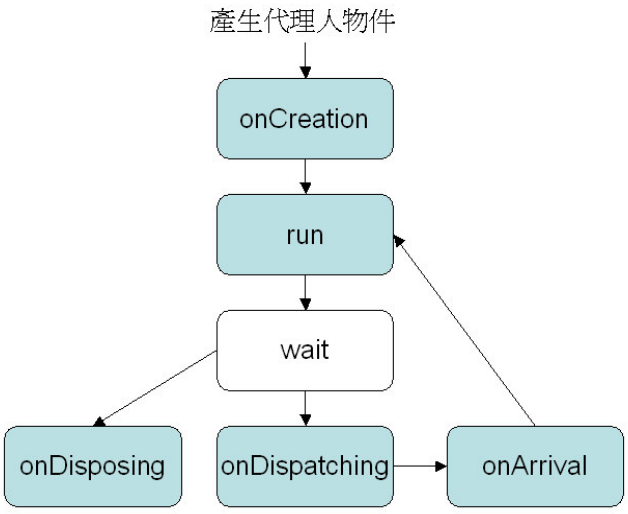

圖 2-7 軟體代理人的程式，於遷移的各個階段所呼叫的函式

圖中未著色方塊 “wait” 不是代理人的特殊函式，我們用它來表示代理人等待 Agent Server 指揮的狀態。當代理人發出遷移 (dispatch) 指令，這個指令實際上會傳到底層的 Agent Server，Agent Server 收到代理人的遷移要求後 Server 會暫停代理人的執行緒，這時代理人就會進入 “wait” 狀態，Server 接著會呼叫 onDispatching，待 onDispatching 函式執行完之後，Server 才將代理人物件序列化並傳到指定位址。

使用者設計有行動力的代理人時，一定會設定兩個部分：第一是代理人的旅程 (Itinerary)；第二則是代理人起始的地點。代理人的旅程 (Itinerary) 是一個代理人儲存在內部的變數，它專門用來儲存代理人在執行時期要遷移到的主機位址。Itinerary 一定是代理人的全域變數之一，因此代理人必須在前面提到的 onCreation 函式裡對這個變數做初始化。如圖 2-8 所示。

設定代理人的起始地點的目地是讓代理人被產生出來時不會執行到主要邏輯，而是等到了指定地點之後才開始執行，這在設計要求服務的代理人時最爲常見。解決的辦法有兩個：第一，讓代理人在 onCreation 函式最後下遷移指令，如圖 2-8 所示；第二，讓代理人的主要邏輯寫在除了 onCreation 和 run 的其他特殊函式中，如圖 2-9 所示，將代理人的主要邏輯放在 onArrival 函式中，如此

```
public class Agent {
  public void onCreation() {
    // initialize global variable...
    dispatch("atp://140.114.208.54");
  }
  public void run() {
    // main logic...
  }
}
```

圖 2-8 設定代理人的起始地點，辦法一

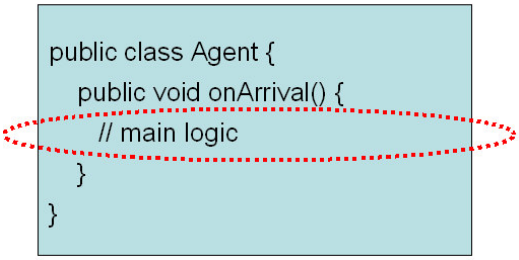

圖 2-9 設定代理人的起始地點，辦法二

代理人程式被產生出來時就不會執行到主要邏輯，而是每次代理人遷移到新的主機上時才會執行其在 onArrival 中的主要邏輯。

2.2.3 合作能力與代理人程式結構

合作能力，或者說是溝通能力，是代理人程式的另一個重要特性。軟體代理人在執行時期若是發現在主機的執行下沒有足夠的資料以滿足代理人的目標時，代理人除用行動力遷移到別的主機上去獲取資料外，也可以透過和其他代理人溝通來得到所需的資料，藉以完成目標。

由於軟體代理人行動力的影響，代理人在執行時不會有固定的位址，因此代理人之間不應該像傳統網路程式一般直接建立點對點的連線並透過連線收送訊息；相反的，代理人應該透過一個專責處理訊息的中繼站來收發訊息，代理人要發送訊息時只要將訊息交給中繼站並告知接收訊息的對象，中繼站就會負責將訊息送給對應的代理人 [1]。軟體代理人系統中 ，最理想的訊息中繼站莫過於 Agent Server，因爲 Agent Server 能夠控管所有於其上執行的代理人物件，當它收到屬於某個代理人的訊息時，Agent Server 可以通知代理人接收訊息。對於代理人本身而言，代理人只需要傳送訊息的對像以及接收訊息的方法即可 ，Agent Server 提供一個抽象的代理人介面 Proxy 來隱藏底層複雜的溝通動作。

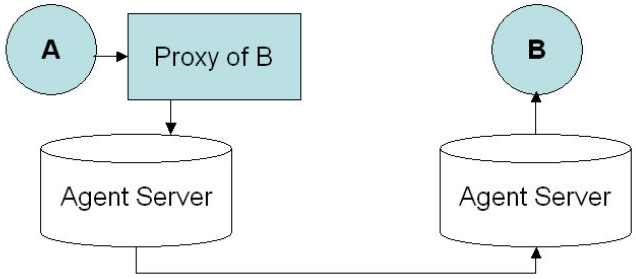

圖 2-10 軟體代理人以 Proxy 當作統一的溝通介面

代理人利用訊息處理函式 (Message handler) 來處理訊息。當 Agent Server 收到送給代理人的訊息之後，Agent Server 會偵測代理人的狀態，當代理人處於 wait 的狀態時 ，表示代理人現在沒有在執行並且可以處理訊息 ，這時 Agent Server 將訊息當成參數去呼叫代理人的 Message handler，如此，代理人便能用 Message handler 來處理送來的訊息。我們可以將 Message handler 加入 Agent 的週期圖內，如圖 2-11 所示，圖中的 “handleMessage” 函式即爲代理人的訊息處理函式。

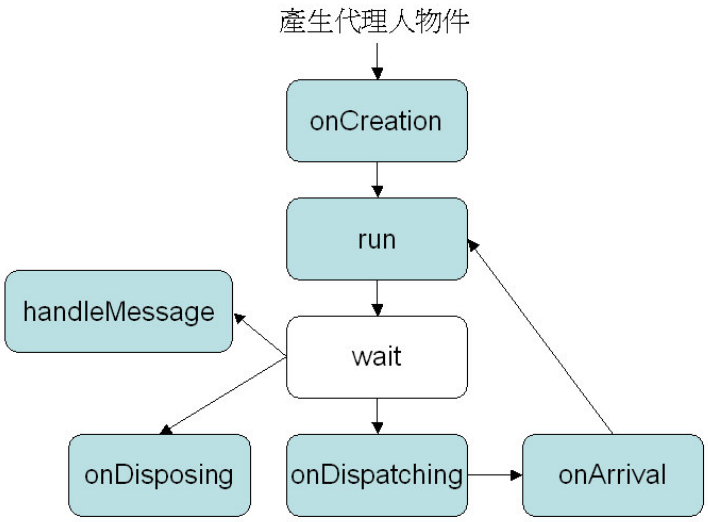

圖 2-11 軟體代理人的程式結構

軟體代理人透過 Agent Server 來溝還的另一個好處是讓代理人能夠非同步地收送訊息，減少代理人等訊息的時間。當有訊息傳送給代理人時，Agent Server 會將訊息先儲存起來，等到代理人物件空閒時 (即處於 wait 狀態) 才通知代理人來收。發送訊息的代理人也同樣可以在送完訊息後不等回覆訊息 (Reply) 直接往下執行，回覆訊息會先傳給 Agent Server，直到代理人向 Agent Server 要回覆訊息時，Agent Server 才需要將回覆物件回傳給代理人。

我們將代理人訊息發送與接收的方式整理如下：

1 訊息發送方

> Synchronous Send / Get Reply

◦ Synchronous Send 是軟體代理人用來傳送訊息的一種方法，當代理人使用這個方法傳送訊息時，代理人的執行緒會停在這個指令，直到對方傳送 Reply 回來，執行緒才會繼續執行。

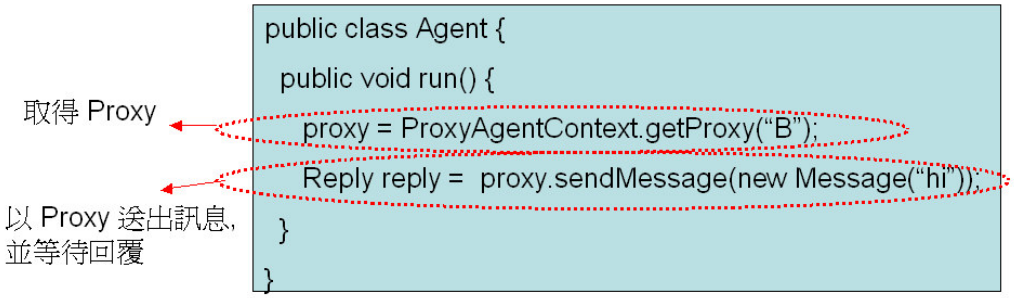

圖 2-12 Synchronous Send / Get Reply

> Asynchronous Send / Asynchronous Get Reply

◦ Asynchronous Send 和 Synchronous Send 剛好相反，當發送方用 Asynchronous Send 發送訊息後可以直接往下執行，直到代理人想要收 Reply 時才會主動向 Server 要。

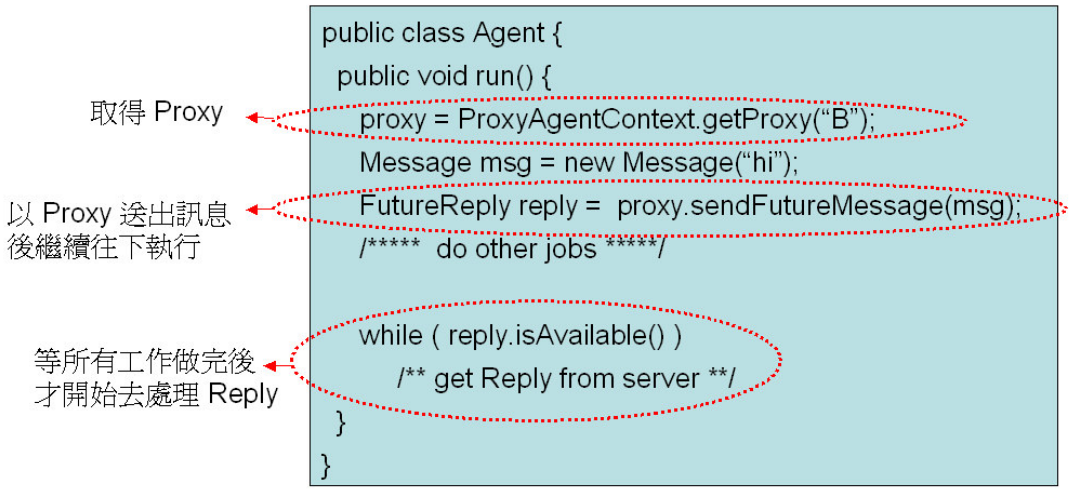

圖 2-13 Asynchronous Send / Asynchronous Get Reply

2　訊息接收方：

> Synchronized Message Handling　(程式碼如圖 2-14)

訊息處理函式在預設情況下是同步函式，在同一個時間內只能被呼叫一次。代理人一次只能處理一個訊息，如圖 2-15 所示，訊息"one"、"two"、"three"依次送進來，代理人也只能依次處理。

```
public class Agent {
  public boolean handleMessage(Message msg) {
    if (msg.sameKind("one"))  {
      // do something…
      return true;
    } else if (msg.sameKind("two")) {
      // starting…
      return true;
    } else if (msg.sameKind("three")) {
      // do something
      return true;
    }
  }
}
```

圖 2-14 Synchronous Message Handling

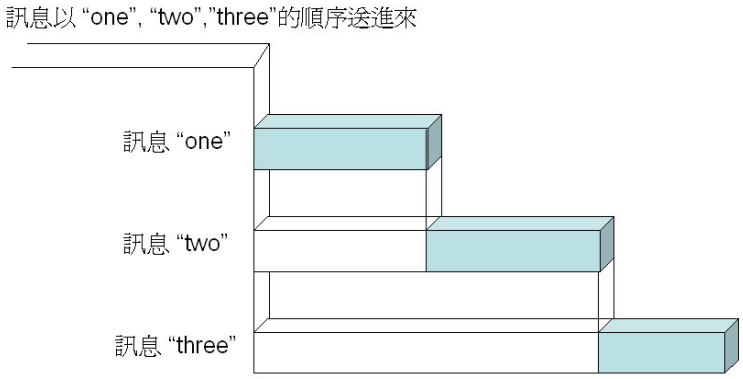

圖 2-15 Synchronous Message Handling

> Parallel Message Handling　(程式碼如圖 2-16)

若接收訊息的代理人需要能平行處理每一個傳送進來的訊息時，接收方可以用 exitMonitor 指令設定它的訊息處理函爲平行處理模式。當有訊息傳送進來時，接收方會開另一個執行緒去呼叫訊息處理函式來處理送進來的訊息物件，如圖 2-17。

exitMonitor 指令讓代理人可以產生另一個執行緒來處理別的訊息

```
public class Agent {
  public boolean handleMessage(Message msg) {
    if (msg.sameKind(“one”))  {
      // do something…
      return true;
    } else if (msg.sameKind(“two”)) {
      // starting…
      getMessageManager().exitMonitor();
      // continuing
      return true;
    } else if (msg.sameKind(“three”)) {
      // do something
      return true;
    }
  }
}
```

圖 2-16 Parallel Message Handling

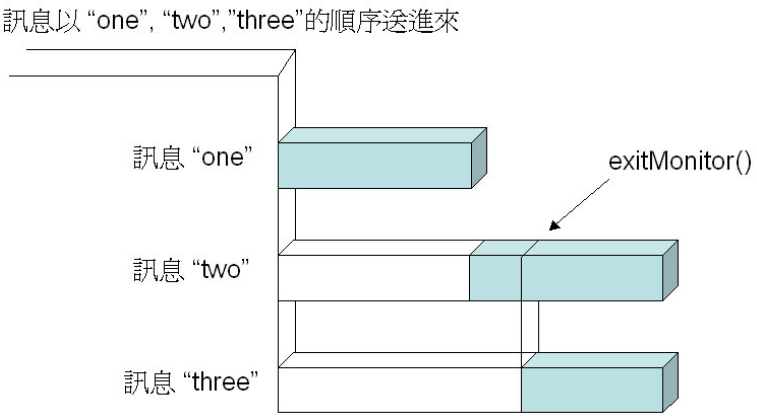

圖 2-17 軟體代理人平行處理訊息的方式

2.2.4 學習力與代理人程式結構之影響

軟體代理人學習能力 (Learning) 的定義是：軟體代理人可以學習到新的知識，並且能夠運用知識加速自己的執行，所以軟體代理人有沒有學習力特性，我們可以從軟體代理人的知識庫 (Knowledge Base) 類型來判斷，通常知識庫有兩種，一種是固定式 (Fixed – Knowledge base)，也就是知識庫在初始化完成後就不會再更動其中的內容；另一種是變動式 (Changeable – Knowledge base)，知識庫會隨著情況不同而可能有所改變。所以若是使用者在描述代理人時設定軟體代理人有固定式知識庫 (Fixed - Knowledge base) 時，我們就可以據以推斷這個軟體代理人並不會學習到新的知識而只是使用知識庫裡的知識。若使用者設定 軟體代理人有一個變動式的知識庫，那麼這個知識庫就有可能會擴充，同時表示，代理人有學習的能力。

知識庫除了固定式以及變動式兩種型態之外，我們依據知識庫跟代理人之間的關係再分成兩類：

1. 知識庫是代理人程式內部的一個資料結構，此時軟體代理人是這個知識庫的擁有者 (Owner)，其他的代理人沒有辦法直接使用他的知

識庫，如圖 2-16 左半邊所示。

2. 知識庫是代理人程式外部的物件，存在於代理人程所執行的環境之下，可能的型式爲檔案、資料庫。此時軟體代理人跟這個知識庫的關係是 ”連線使用關係" (Connect)：代理人必須先跟知識庫建立連線，然後才能取得資料。這表示軟體代理人雖然使用這個知識庫但是並非擁有它，知識庫可能被其他的軟體代理人使用，如圖 2-18 右半邊所示。

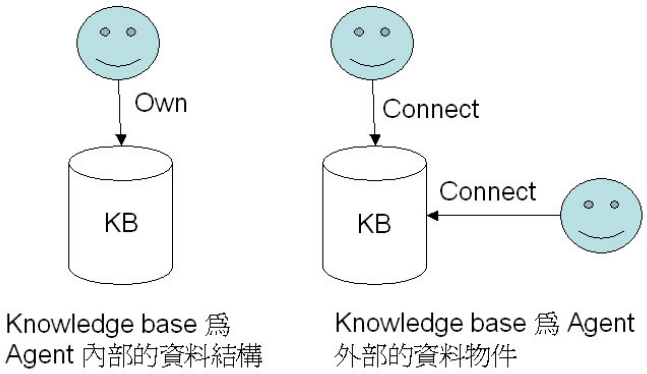

圖 2-18 知識庫與代理人之間的關係

2.3 軟體代理人之系統架構

軟體代理人系統指的是以軟體代理人爲單位組合而成的系統，系統中的代理人會利用兩種合作的關係以達成系統目標，兩種合作關係分別爲：供需關係 (Provider - Requestor)以及主從關係 (Master – slave)。在本節中，將會介紹代理人之間的兩種互動關係，以及代理人系統如何組成。

軟體代理人系統可以分爲單一代理人系統 (Single Agent System) 以及多代理人系統 (Multi – Agent System)。單一代理人系統是指，當使用者需求較爲簡單時，系統只需要一個代理人便可達到使用者的需求。不過在大多數的情況下，代理人系統的需求通常十分複雜，開發人員必須以多個代理人程式合作來達成，這

種以多個代理人合作來組成的系統，我們稱爲多代理人系統 (Multi – Agent System)，如圖 2-19 所示。

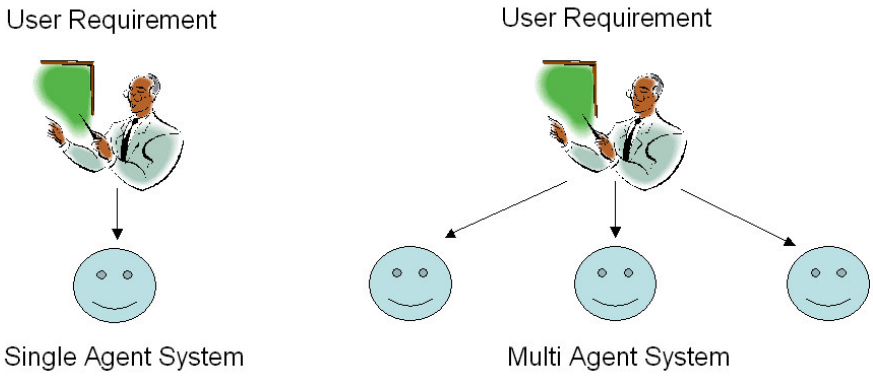

圖 2-19 單一代理人系統與多代理人系統

多代理人系統中的代理人由於無法獨自完成系統的需求，在執行時期必然需要合作溝通以完成工作，這時代理人之間就會產生互動關係。若代理人在執行時期，發覺需要和其他代理人做訊息溝通，我們稱訊息溝通兩端的代理人關係爲供需關係 (Provider – Requestor)，在關係中採取主動的代理人我們稱爲 Requestor Agent，採取被動的代理人則稱爲 Provider Agent。

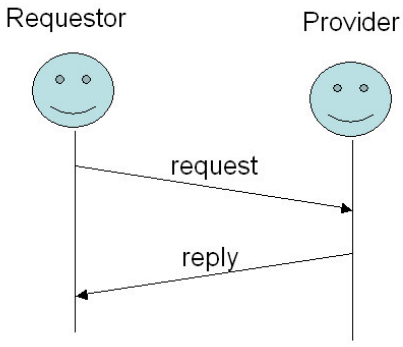

圖 2-20 供需關係 (Provider - Requestor)

軟體代理人之間的另一種關係爲主從關係 (Master – Slave)。當代理人本身的目標過於複雜，不適合獨自完成時，使用者可以讓代理人產生子代理人，並將

原本應該由代理人來處理的工作轉由子代理人來處理。在主從關係中產生其他代理人來執行的代理人稱爲 Master Agent，而被產生出來的代理人則爲 Slave Agent。

主從關係的好處在於讓 Master Agent 可以在執行時期動態地產生 Slave Agent 起來執行工作，這跟傳統的函式呼叫的運作觀念十分類似。不過，由於每一個 Slave 都是擁有獨立執行緒的個體，Master Agent 產生完 Slave Agent 之後就可以繼續做自己的工作，這可以節省呼叫函式時必須要等函式回傳才能繼續執行的等待時間；另外，每一個 Slave 都是一個代理人，可以利用代理人的特性完成高難度的工作，這些都是傳統函呼叫能力所不及的。

主從關係就語意上可以看成：”Master Agent 產生 Slave Agent，讓 Slave Agent 幫 Master 完成目標” 如圖 2-21 所示，這表示原本應該由 Master Agent 自己完成的工作，現在轉由 Slave Agent 代爲完成，同時這也表示 Slave Agent 的目標是上層代理人的子目標。所以我們可以把 Master Agent 和其所產生的 Slave Agent 視爲一個工作群組 (Group)，這個群組的階層中，上層代理人的目標會包括下層代理人的目標。由於主從關係工作群組中的 Slave Agent 也可以再往下產生其他的代理人，因此我們可以得到一個階層式的群組架構。如圖 2-22。

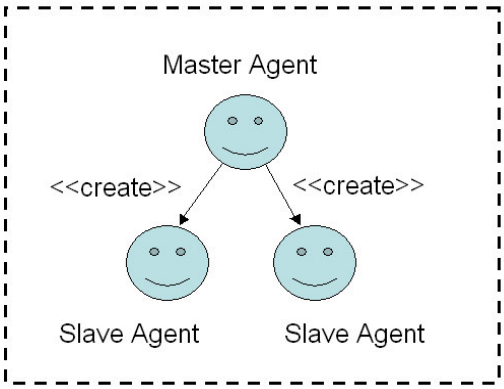

圖 2-21 主從關係造成的工作群組

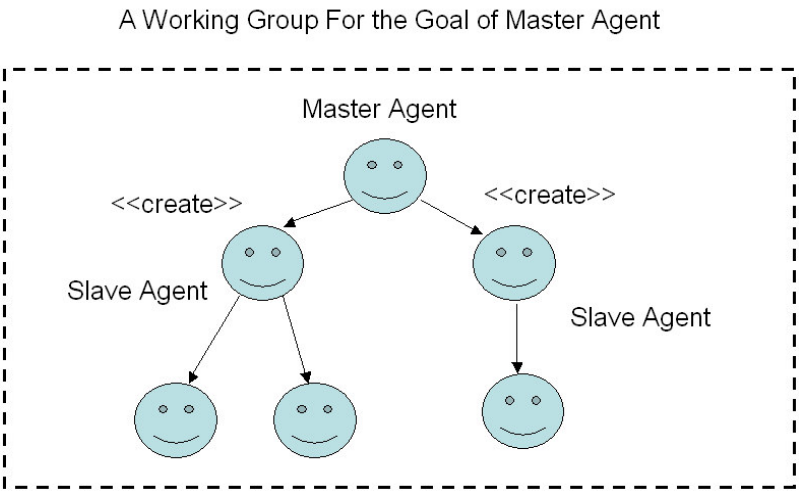

圖 2-22 主從關係造成的階層

若將軟體代理人系統中以主從關係所構成的代理人群組都找出來，我們會發現系統架構會如圖 2-23 所示，由數個代理人群組構成，代理人群組之間則由供需關係相連。總合前面所述的供需關係、主從關係、以及代理人群組的觀念，我們就可以表示出多代理人系統的整體架構。代理人群組內可以是單一代理人，或是由主從關係建構出的階層群組。群組之間的關係則爲供需關係。群組間的每一修供需關係都有其實際負責的代理人，但是就群組的觀點來看時，可以把群組間的供需關係都視爲群組間裡最頂層的 Master Agent 跟 Master Agent 的溝通，只是實際運作時 Master Agent 將溝通的動作交給 Slave Agent 去處理而已。實際的代理人系統架構會如圖 2-24 所示。

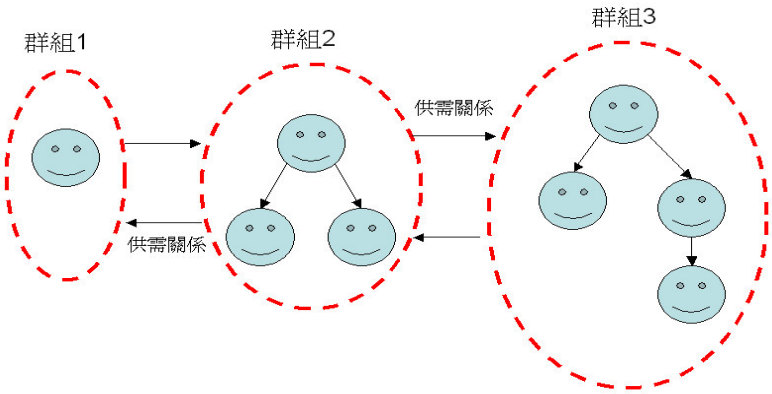

圖 2-23 以代理人群組來看代理人系統架構

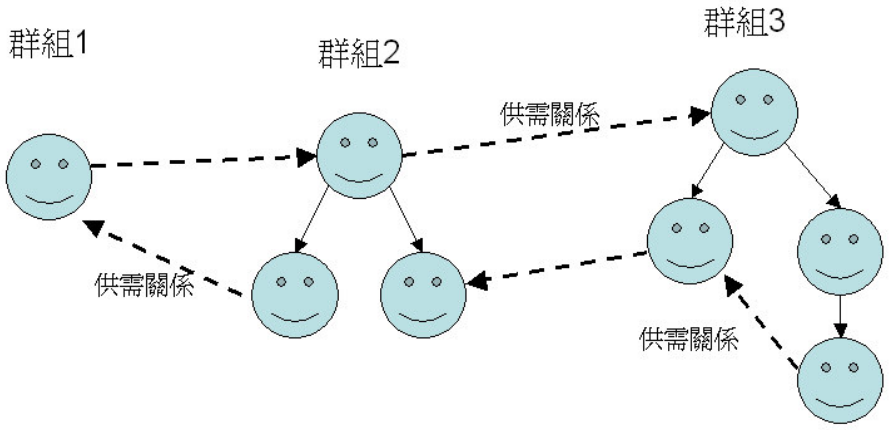

圖 2-24 實際的代理人系統架構

2.4 軟體代理人系統之開發方式分析

開發流程的主要目的在於提供開發人員一套有系統的方法分析系統的架構[10]，本節中將依據 2.3 節分析之軟體代理人系統架構，找出符合軟體代理人系統的漸進式開發流程，並依此提出軟體代理人開發工具應具備之需求。

由於軟體代理人系統架構可以用代理人群組以及代理人群組之間的供需關係來描述。因此我們將代理人系統的型態分成單一代理人群組構成的系統，以及由多個代理人群組組合而成的系統，並由這兩個型態切入來討論軟體代理人系統

的分析方法。

單一群組構成之代理人系統架構是一個樹狀圖，由頂層代理人爲根節點，逐層往下發展，以目標的觀念來看，下層代理人的目標都是其上層代理人的子目標，或者說上層代理人的目標會包含下層代理人的目標，如圖 2-25 所示。由於將上層代理人的工作分給下層代理人之後，上層代理人的目標並不會改變，有改變的是上層代理人的工作內容，因此整個系統中的代理人都是爲了達成群組頂層代理人的目標，所以當最頂層代理人 (Top Agent) 的目標確定後，整個代理人群組的目標也跟著確定。

由於代理人的目標會有"向下包含"的特性，使用者在分析單一群組的代理人系統時，應該是以 Top-Down 的順序來分析。使用者必須先設定頂層代理人的目標並評估是否要設計下層代理人幫頂層代理人分擔工作。而由頂層代理人產生出的下層代理人還可以再往下生出新的代理人，造成一個子群組，如圖 2-26 所示。

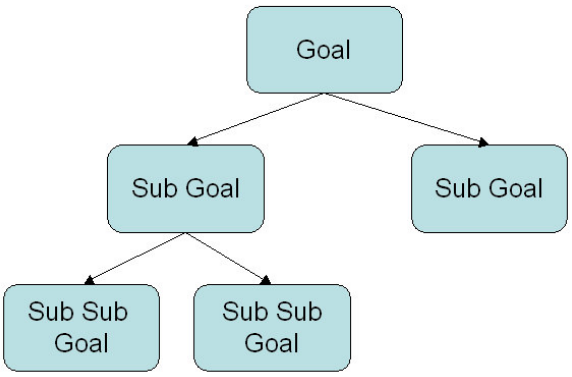

圖 2-25 下層代理人的目標是上層代理人的子目標

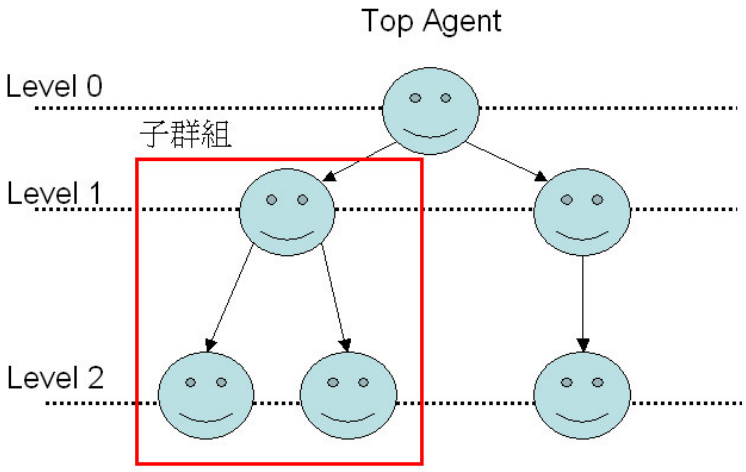

圖 2-26 以下層代理人爲根節點的子群組

同一層的代理人可能會有供需關係，此供需關係可以被視爲以此供需關係兩端代理人爲根節點的兩個子代理人群組間的供需關係，如圖 2-27 所示。因此開發人員每往下生出一層代理人階層，就必須去設定該層代理人之間的供需關係。另外，由於下層代理人可能會分擔原本屬上層代理人的供需動作，因此當使用者爲代理人產生下層代理人時必須要決定是否要把屬於上層的供需關係交給下層來處理。

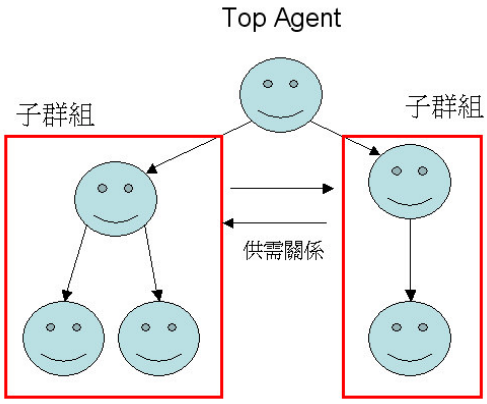

圖 2-27 代理人子群組之間的供需關係

軟體代理人系統若是由多個代理人群組組合而成，就表示此代理人系統一開

始就將系統的需求分給數個軟體代理人分別處理，我們可以將這些處於系統中最上層的代理人稱爲 Top Level Agent。和單一代理人群組不同之處在於，多群組的代理人系統一開始就有數個 Top-Level Agent，而單一代理人群組的系統則只會有一個 Top-Level Agent，如圖 2-28 所示。所以開發人員分析多群組的代理人系統時，產生第一層代理人後就必須定義第一層代理人之間的供需關係。接下來的分析動作則和單一代理人群組分析方式相同。

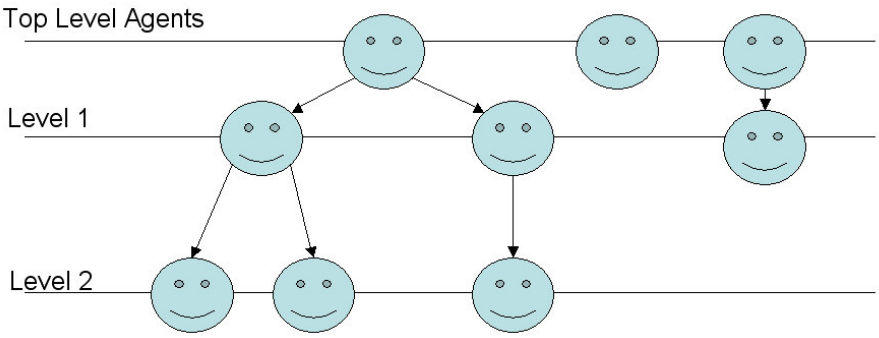

圖 2-28 多群組之代理人系統

經由上述的討論，我們發現最適合軟體代理人系統的開發流程應該由上到下，以階層爲單位逐步分析每個代理人的目標，設定同層代理人之間的供需關係，之後評估代理人目標的複雜度並決定是否要設計下一層代理人來分擔工作。我們將這個分析流程條列如下：

Step 1 定義軟體代理人系統的系統目標。

Step 2 分析系統目標，決定要把系統目標分成幾個子目標，分別交由系統最上層的代理人(Top Level Agents)來負責。

Step 3 根據每個最上層代理人的目標定義代理人之間的供需關係。

Step 4 分析每個代理人所負責的目標，決定是否需要再往下產生新的代理人來分擔工作。如果不需要往下產生新的代理人，以這個代理人爲根節點的代理人群組就分析完成。

Step 5 如果往下分割出數個子目標，那麼開發人員必須產生新的代理人去

負責。

Step 6　檢視每個新生的代理人，分析達成目標的過程中需不需要自其他主機或是其他代理人獲得資料，並決定取得資料的方式。

Step 7　檢視新生代理人是否會取代原代理人的溝通關係，視情況加以調整。回到 Step 4。

經由上述的分析方式，開發人員可以循序地建構出軟體代理人系統的架構，這個架構包含了系統中的所有代理人、代理人的目標，以及代理人之間的關係。而此基礎的代理人系統結構已經分配好所有代理人應負責的工作，開發人員便可以針對每個代理人內部的工作做細部分析與設計。此階段爲軟體代理人開發流程中的 Architecture Analysis Phase。

由於軟體代理人程式的性質跟物件導向的類別很相近，因此代理人程式都是由物件導向程式實作，其中每一個代理人會對應一個類別，讓代理人程式能繼承所有物件導向程式的優點。所以依前述的部驟確定了每個代理人的目標後，相當於此代理人程式所對應的類別功能已確定。開發人員的任務就是將代理人程式的抽象目標轉換成實際的類別。代理人目標可以細分爲數個'工作，而每個代理人的工作則可以視爲代理人內部的私有方法 (method)的 組合。將方法再細部分析下去則會發現代理人在內部函式運作的途中會分析出設計類別 (Design Class) 來幫助代理人完成工作，因此開發人員必須接著分析每一個代理人內部方法要包含的類別，以及這些類別的關係，其步驟如圖 2-29 所示。

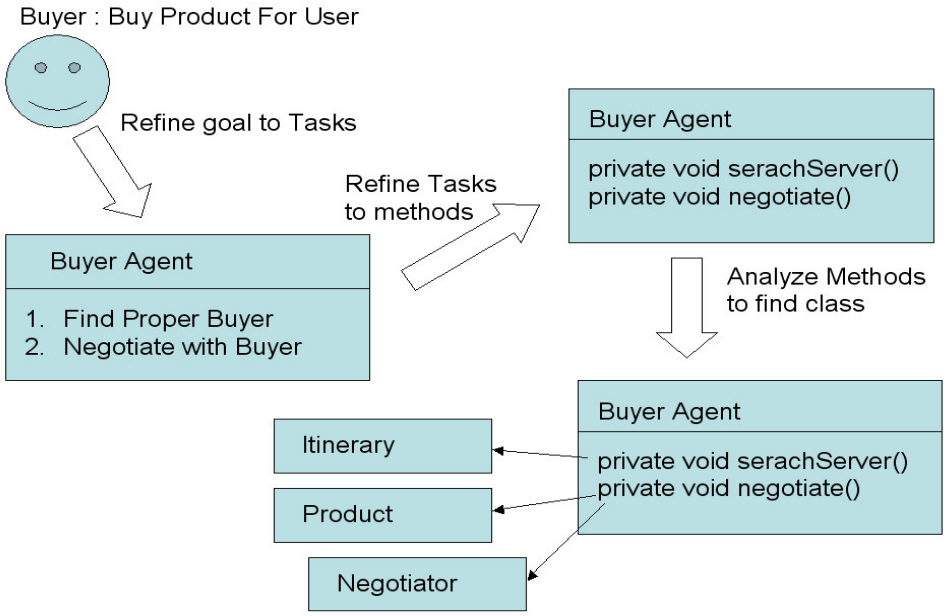

圖 2-29 代理人目標的分析流程

總合本節所描述的步驟，可以爲軟體代理人系統整理出一個漸進式的開發流程，如圖 2-30 所示，其中在 Architecture Analysis Phase 裡定義開發人員分析軟體代理人系統架構的流程，在 Agent Class Analysis Phase 中，開發人員已經定完代理人系統的基本結構並開始逐一分析每個代理人所負責的工作，最後在 Agent Task Analysis 階段，開發人員會分析出代理人內部的方法、變數、以及方法會使用到的工具類別。

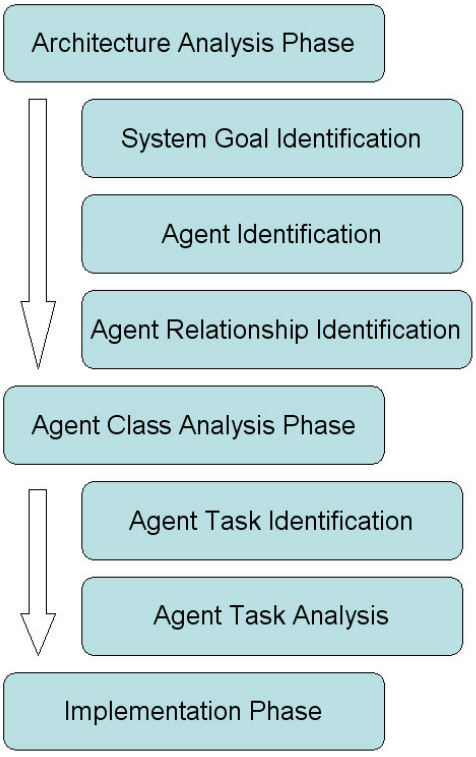

圖 2-30 軟體代理人系統的開發流程

經由上述的討論可以發現，軟體代理人系統開發流程是以系統的目標爲起點，以 Top-Down 方式分析系統。因此一套可以支援軟體代理人系統開發流程的開發工具，應該具備下列五項需求：

1 Agent-Based Programming

一個代理人開發工具必須讓使用者以代理人爲單位來建構系統，在此比較中，上述的三個代理人開發工具都可以提供這樣的能力。

2 Support Agent Characteristics

一個專爲軟體代理人設計的開發工具應該要支援軟體代理人的各項特性。但是 AgentBuilder 開發工具並沒有支援當前使用最普遍的行動力技術，這也是現今許多其他開發工具共有的問題。以目前的環境來說，一個沒有行動力的代理人其執行緒將會被迫侷限在單一主機上，大大減少

代理人在執行時期爲了達成目標而可以採取的策略。

3 Friendly User Interface

爲了幫助使用者更方便，更容易開發出代理人系統，開發工具應該不只提供工具類別，而應該提供使用者介面方便使用者操作，以更高階的方式來撰寫程式。開發工具並應同時整合執行的平台方式便用者在撰寫完程式碼後能夠立即執行驗証。

4 Incremental Development Process

開發工具應該提供使用者類似 2.4 節中所討論的漸進式開發流程，幫助使用者做系統架構分析、代理人分析、代理人內部使用之類別分析、以及最後的執行驗証。

5 Portable Code

代理人開發工具應該要以一個跨平台的語言來當作使用者撰寫的代理程式之語言，因爲只有在跨平台的基礎下，軟體代理人才能更自由的使用行動力在網路節點上遷移。

2.5 現今軟體代理人開發工具之介紹

現今的軟體代理人開發工具都面臨沒有公認標準可遵循的困境，雖然軟體代理人的概念非常強大，但是由於各個代理人開發平台並沒有統一的規格，導致現今的軟體代理人開發工具都各自著重在代理人某些特性的開發。在 2.5.1 節中將介紹目前三個較爲完整的代理人開發工具，並在 2.5.2 節中總結其優缺點。

2.5.1 軟體代理人開發工具之分析介紹

(1) IBM Aglets

IBM Aglets 套件是由 IBM 公司以 Java 所實作的軟體代理人開發套件[1][19]。Aglets 套件最主要的目地是提供使用者一個支援自主性、溝通能

力、以及行動力的基礎工具類別，方便使用者撰寫代理人程式。由於 Aglet 套件是以 Java 語言來實作的，所以其行動力的型式是 Weak Mobility。Aglets 套件中最重要的兩個類別分別是 Aglet 和 AgletContext。其中 Aglet 類別即爲最軟體代理人的基礎類別，使用者開發代理程式時都必須繼承 Aglet 類別並覆寫其中的方法。AgletContext 類別代表代理人執行時期底層的 Agent Server，它負責控管代理人程式的生命週期、幫代理人程式收送訊息、依據代理人程式的需求將代理人程式移動到指定的位址。程式設計師不需要自己撰寫 Agent Server，只需要會在代理人程式碼中利用 Agent Server 提供的服務來達成任務。

IBM Aglets 的程式結構十分類似於 2.2 節所介紹的軟體代理人程式結構，代理人類別內部由數個特殊函式組成，其中啓動函式 run() 爲代理人程式的進入點。每當代理人被產生，或是遷移到一個新的主機時，都會執行啓動函式。代理人可以在自己的程式內部下指令要求底層的代理人將自己傳送到別的主機來執行，如此代理人程式便能在執行時期主動遷移的動作。IBM Aglets 的訊息溝通方式是靠訊息處理函式來完成，在 Aglet 類別裡已經定義了一個名爲 handleMessage() 的訊息處理函式，當有訊息傳給代理人時，底層的 Agent Server 會先將此訊息儲存起來，等到代理人物件沒有在執行動作時呼叫代理人的訊息處理函式來處理訊息。IBM Aglets 的代理人程式架構如圖 2-31 所示。

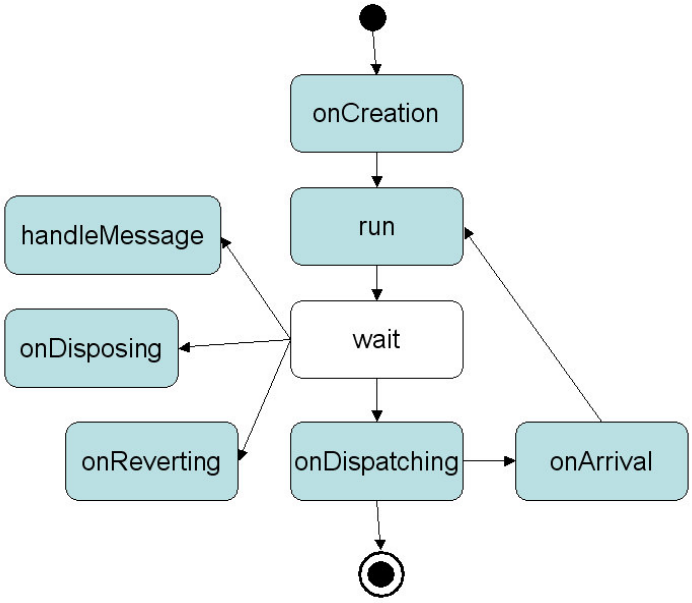

圖 2-31 IBM Aglet 的代理人程式結構

IBM Aglets 套件最終的目標是希望能讓使用者擺脫傳統由單一程式來啓動功能，而改由 “Agent-Based” 的設計模式，也就是整個系統全部是由主動的代理人物件以及代理人平台組成。

(2) ObejctSpace Voyager

由 ObjectSace 公司所開發出來的 Voyager 套件是專門爲了開發分散式系統而設計 [18]。Voyager 也是以 Java 語言實作而成，繼承 Java 程式跨平台的特性。Voyager 最強調的功能是 “Remote Object Computing”。由於一般在寫物件導向程式時，所有的物件都被限制在本地端的機器上執行，Voyager 提供讓使用者可以在遠端產生物件、控制遠端物件、以及和遠端物件的工具類別，使用者只要會使用 Voyager 提供的工具類別，就可以非當輕易地做出 “Remote Object Computing”的功能。使用遠端物件的好

處跟使用具有行動力的軟體代理人很類係，都可以幫助使用者有效的節省頻寬，提高系統效率。

Voyager 並不是以開發軟體代理人爲主要目的，因此 Voyager 並沒有像 IBM Aglets 一樣對代理人程式內部結構有嚴謹的設計，對於 Voyager 來說代理人物件跟一般的物件沒有區別，只是代理人物件會在程式碼內部指定每當這個物件被遠端產生時必須呼叫的方法，這個方法類似於 IBM Aglets 中的啓動函式。Voyager 裡的代理人物件互相要溝通時則是一律利用類似 RMI 的方式，以方法呼叫來溝通。

(3) Reticular System AgentBuilder

AgentBuilder 是由 Reticular System 公司所研發出來的智慧型代理人 (Intelligent Agent) 開發工具。AgentBuilder 是目前擁有圖形化使用者介面中功能較完整代理人開發工具 [8][9] 。

在 AgentBuilder 的設計裡，所有的代理人都是以 Rule – Based 方式來控制其內部的行爲，每個代理人都有一個循環週期 (Agent Cycle) 。一開始代理人會先將自己內部變數的狀態做初始化，接著在每個週期內代理人會檢查 Rule Base 中有沒有符合目前執行情況的規則可以套用，用果有，那麼代理人就會套用這條規則並且執行該規則內所描述的動作；如果沒有可套用的規則，那麼代理人就會進入下一個循環週期並重覆上述的動作。圖 2-32 即表示 AgentBuilder 所定義的智慧型代理人模型。

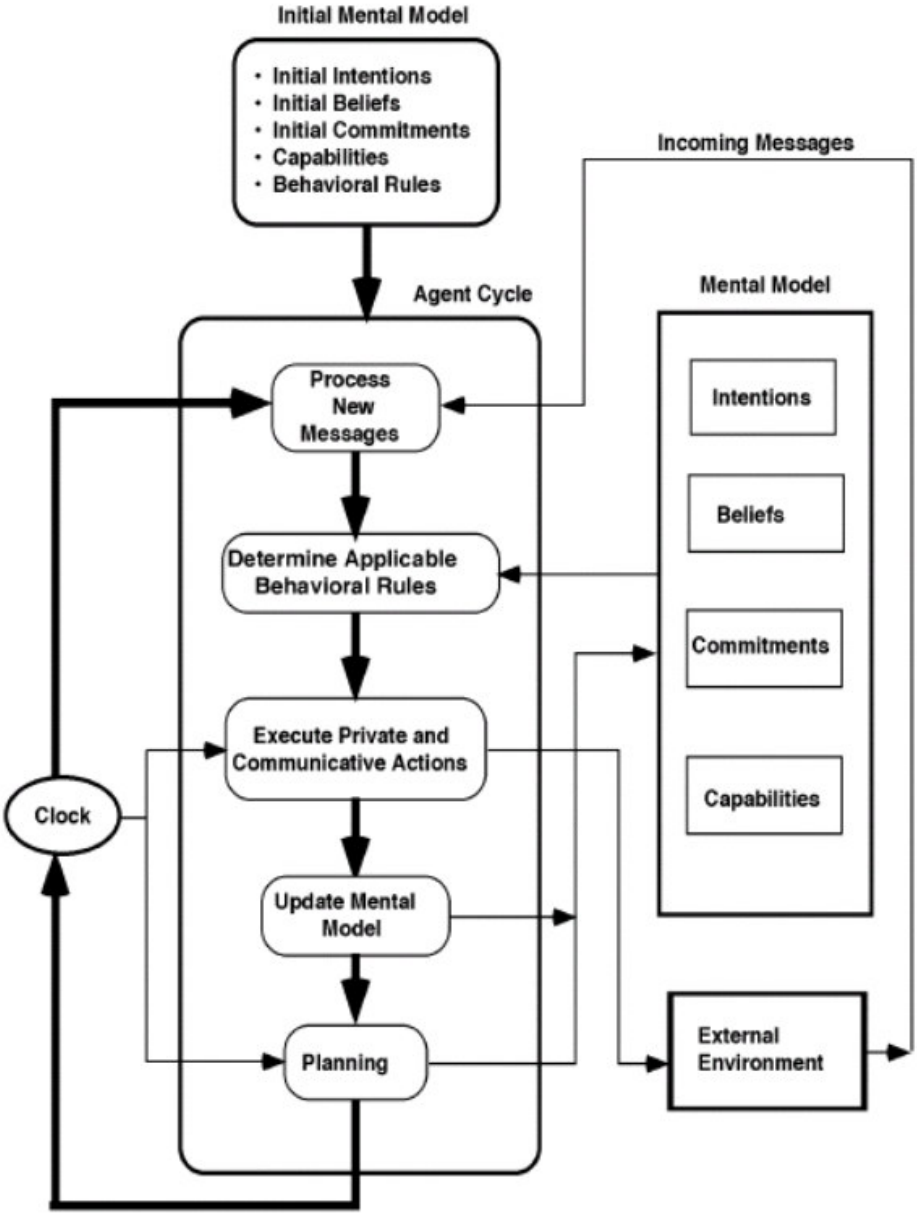

圖 2-32 AgentBuilder 的代理人生命週期

由於 AgentBuilder 將代理人設計成 Rule – Based 的架構，使用者在利用 AgentBuilder 來設計代理人程式時也必須以行爲規則來描述，所以如何讓使用者可以方便地撰寫代理人的行爲規則就成了 AgentBuilder 最重要的任務。除了設計規則之外，每個代理人都有“Mental Model”，如圖 2-32 所示，”Mental Model”其實相當於類別內部的變數，代表著代理人目前執行

狀態，行爲規則內的判斷條件通常也都是依”Mental Model”裡的值。所以在定義行爲規則之前使用者在定義代理人時，AgentBuilder 也會要求使用者去定義代理人的 “Mental Model”。

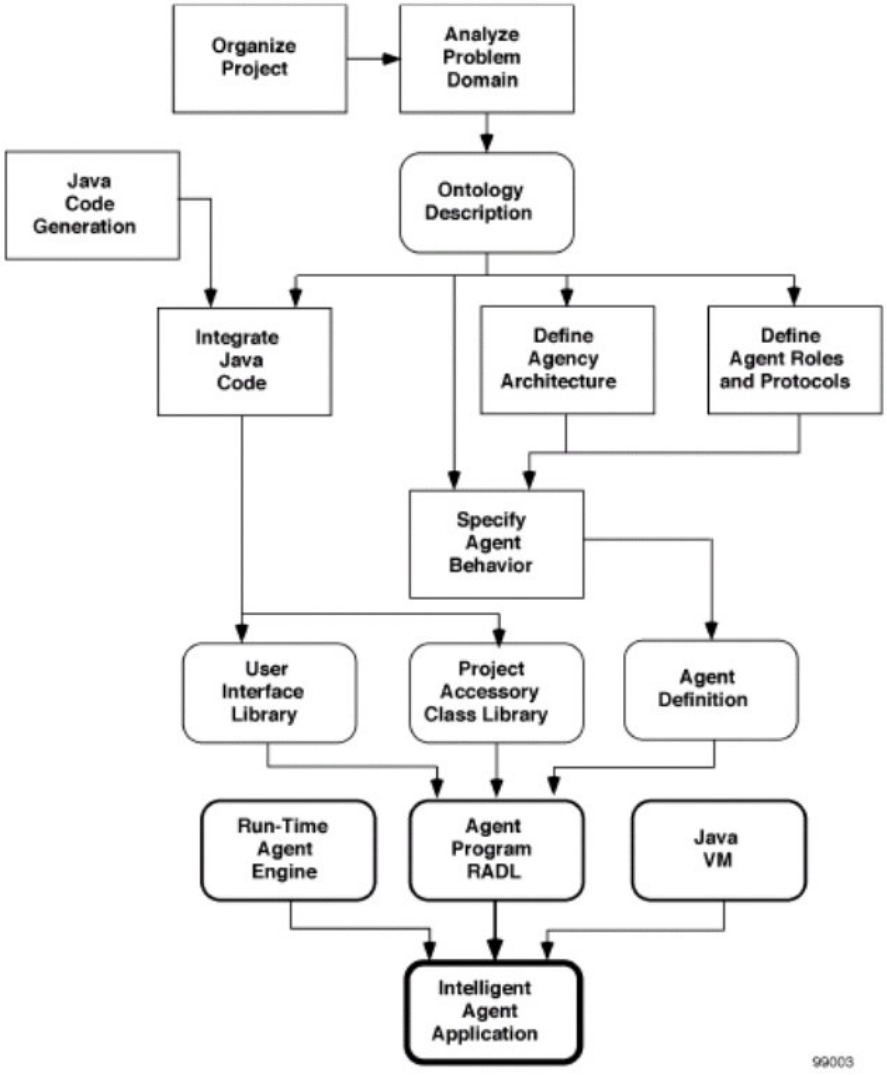

圖 2-33 AgentBuilder 提供給使用者的開發流程

圖 2-33 定義了 AgentBuidler 的使用流程，其中幾個主要流程要做的工作描述如下

1 Organize Project：AgentBuilder 提供使用者管理專案內容的介面。專案內容包括各個代理人的定義與其內容以及由外部載入的類別。

2 Analyze Problem Domain & Ontology Description：AgentBuilder 認爲一個代理人系統會專注於解決某個領域 (domain) 下的問題，而每個領域會有一些特殊的類別專門處理這個領域的動作，或是專門儲存屬於這個類別的資料。當使用者要撰寫這個領域的代理人程式時就會使用到些跟這個領域相關係的類別。因此 AgentBuilder 讓使用者可以在撰寫代理人程式之前先定義出 所有跟領域相關聯的類別架構 (Domain Ontology) 。

3 Define Agency Architectures：在 Domain analysis 之後，使用者可以開始把系統功能分配到一個或多個代理人上，在這個階段裡 AgentBuilder 讓使用者定義代理人，並且讓用者定義代理人之間的互動過程。

4 Specify Agent Behavior：使用者定義各個代理人之後，接著就要定義代理人的行爲，AgentBuilder 提供 Rule Editor 給使用者，讓使用者可以替代理人產行爲規則。

5 Create Agent Application：當使用者描述完一個代理人的行爲規則後，代理人可以利用 AgentBuilder 提供的 Agent Engine 去執行代理人程式碼。

2.5.2 軟體代理人開發工具之比較

分析 2.5.1 節所提到的三個工具時，可以 2.4 節所提的五個軟體代理人開發工具的需求來討論。

1 Agent-Based Programming

軟體代理人開發工具應該以代理人爲單位來開發，IBM Aglets、Voyager、AgentBuilder 都符合此需求。

2 Support Agent Characteristics

軟體代理人的特性有自主性、合作能力、學習能力、以及行動力，因此軟體代理人系統開發工具必須要支援代理人的四個特性。2.5.1 節中所

提到的三個代理人開發具中，AgentBuilder 在設計時則沒有考慮到行動力的特性，因此，AgentBuilder 無法支援所有的代理人特性。

3 Friendly User Interface

一個好的軟體代理人開發工具應該要提供圖形化的介面來幫助使用者，加快使用者開發系統的速度。在 2.5.1 節討論的三個工具中，由於 IBM Aglets 和 ObjectSpace Voyager 只支援開發軟體代理人所需的基礎類別，因此沒有提供親切的使用者介面。

4 Incremental Development Process

代理人開發工具應該要支援漸進式的開發方式，引導使用者從代理人系統的分析、代理人工作分析、代理人程式邏輯設計、到代理人執行驗証，以 Top – Down 的方式逐步完成這些階段。這是 2.5.1 節所提到的三個代理人開發工具都缺乏的。

5 Portable Code

代理人開發工具應該以一個套件支援豐富、技術成熟、且跨平台的語言來當作其開發的語言，這可以讓開發出來的代理人系統更容易使用。在 2.5.1 節所述的三種開發工具中，AgentBuilder 是以內建的 Rule Editor 讓使用者編輯，其最後所得出的程式碼則是只有在 AgentBuilder 上才能執行，這會限制其所開發之代理人能力。

總合上述的比較可以得到圖 2-34 之結果，分析此表會發覺，現今的軟體代理人開發工具都沒有辦法滿足我們提出的五個軟體代理人開發工具需求，因此我們想要設計一個滿足這五個需求的軟體代理人開發工具來幫助使用者。

工具	Agent-Based Programming	Support Agent Characteristic	Friendly UI	Incremental Development Process	Adaptable
IBM Aglet	O	O	X	X	O
ObjectSpace Voyager	O	O	X	X	O
Reticular System AgentBuilder	O	No support for mobility	O	X	X

圖 2-34 代理人開發工具之比較

第三章 軟體代理人系統開發工具之設計

根據第二章所分析的軟體代理人系統開發具之五個需求：Agent-Based Programming、Support Agent Characteristic、Friendly User Interface、Incremental Development Process、以及 Portable Code，本研究擬開發一個符合這些需求的軟體代理人開發工具 AgentIDE (Agent Integrated Development Environment)，本章介紹其功能規格與系統架構，首先在 3.1 節將提出 AgentIDE 之功能規格；3.2 節則會分析 AgentIDE 之系統架構。

3.1 軟體代理人開發工具 AgentIDE 之功能規格

由第二章之軟體代理人開發流程可知，軟體代理人系統是以 Top-Down 由上而下滾動式 (Incremental) 的方式設計，開發人員從軟體代理人系統的需求開始分析，將系統需求視爲代理人系統的目標，並依照目標的複雜度來決定是否將此目標再往下細分；系統目標分出來的子目標都由一支軟體代理人程式來負責，並須依照主目標與子目標與子目標及子目標間的合作關係來決定代理人之間的供需關係。如此循環切割，直至每一代理人的目標均可容易轉換成程式爲止。當一代理人目標被細分爲數個子目標時，需分析此代理人原有的供需關係，並決定這些供需關係動作是否要交由下一層的子代理人來負責。當開發人員決定了軟體代理系統之架構後，才針對每一個軟體代理人依其目標決定負責之任務，依任務設計代理人程式之內部邏輯，再撰寫代理人程式碼。

依此漸進式的軟體代理人系統開發流程，支援此軟體代理人系統開發之工具 AgentIDE 應具備下列的功能：

1 支援使用者以圖形化方式定義軟體代理人系統架構，包括系統內所有的軟體代理人的目標、以及軟體代理人之間的關係，包括供需關係與主從關係，且用不同符號來表示。代理人間的主從與供需關係，使設計者可

輕易了解軟體代理人系統的整體架構。

2 支援使用者依代理人的目標定義代理人所應包含之任務 (Task)，並設定各任務所需的代理人特性，包括行動力特性與學習力特性。如圖 3-1 所示。

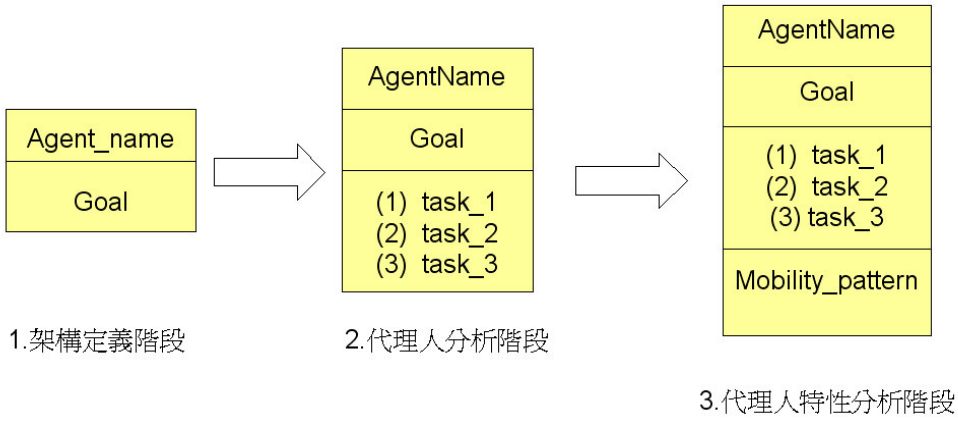

圖 3-1 代理人於系統架構圖上的精鍊 (Refine) 過程

3 支援使用者程式碼編寫工具。除程式編輯功能外，會依系統架構中代理人之間的關係與代理人特性的定義，自動提供樣板程式供使用者選用。例如，代理人被定義有行動力特性時，自動提供行動力樣板給使用者套用。

4 提供代理人程式執行環境，幫助代理人程式之驗証。

3.2 AgentIDE 之系統架構

圖 3-2 爲 AgnetIDE 之系統架構圖。由於 AgentIDE 要提供使用者從代理人架構分析設計到程式碼撰寫各個階段的輔助工具，因此，AgentIDE 的系統將包含下列的子系統：

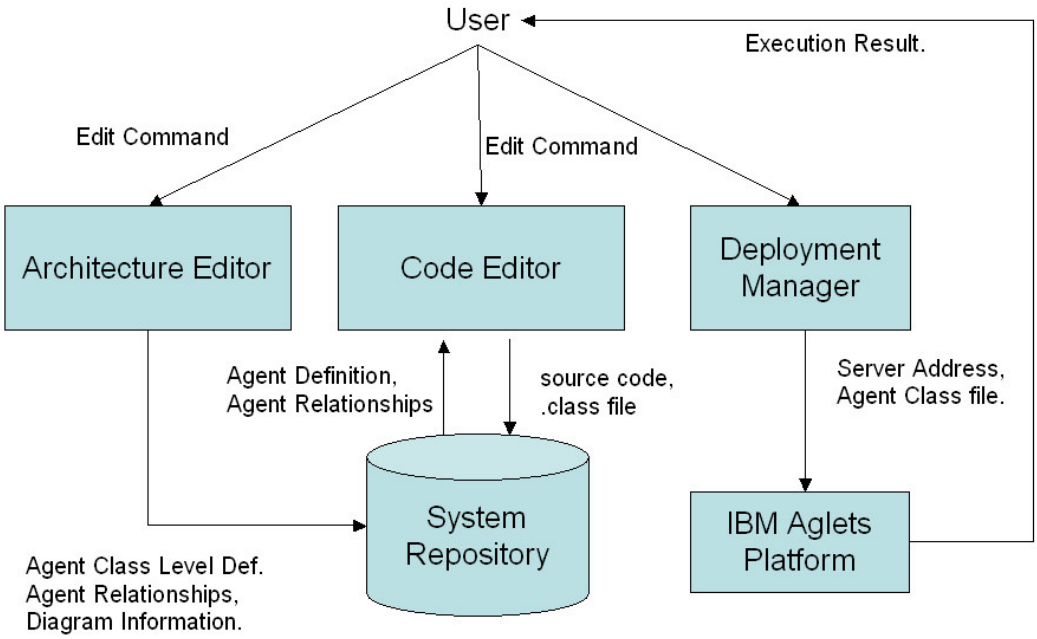

圖 3-2 軟體代理人開發工具，AgemtIDE 架構圖

1 **系統架構編輯器 (Architecture Editor)**：此編輯器提供圖形介面讓使用者定義代理人的名稱與其目標的描述，及定義關係連線，包括供需關係與主從關係。架構編器還支援使用者依代理人目標來定義其實際之任務與代理人類別的介面，並讓使用者依據代理人任務的需求來定義代理人行動力、學習力等特性。一個代理人會對應到一個類別程式檔，此檔的檔名跟代理人的名稱相同，因此使用者設定代理人名稱時，編輯器會檢查使用者設定的代理人名稱必須符合命名規則。

2 **程式碼編輯器 (Code Editor)**：提供使用者一個文字編輯器讓使用者撰寫代理人程式碼。此工具讀入系統架構圖中的代理人設定，預先產生代理人類別的樣板程式碼，包括代理人類別的變數宣告與方法宣告。程式碼編輯器會依代理人系統架構圖中的設定提供使用者代理人特性的樣板套

用。代理人之特性樣板包含：溝通樣板、行動力樣板。

3 **佈局管理器 (Deployment Manager)**：使用者撰寫完代理人之程式碼後，佈局管理器可以幫使用者執行代理人程式。使用者可利用佈局管理器來設定要執行之代理人程式、代理人伺服器、以及代理人程式要在哪個代理代理人伺服器上被執行起來。由於佈局管理器可以讓使用者一次執行多個代理人，有時候爲了滿足某些劇本 (Scenario) 的要求，代理人執行時必須有先後順序之分，因此佈局管理器還可以設定代理人程式執行的先後順序。

4 **IBM Aglets Platform**：AgentIDE 使用 IBM Aglets 的套件，提供使用者開發軟體代理人程式所需的基礎工具類別。選擇以 IBM Aglets 做爲底層平台的原因有二：

(1) IBM Agelts 爲 Java-Based 的軟體代理人開發套件，符合 2.5 節所提，代理人開發工具應該以物件導向、技術發展成熟、套件支援豐富、跨平台的程言來實作的需求。

(2) IBM Aglets 爲開放原始碼的專案，目前已到達 2.0.2 穩定版，且提供完整的文件說明，這讓我們可以依 AgentIDE 之需求修改底層的軟體代理人平台，對代理人系統有更多自主權。

綜合以上所述 AgentIDE 之輔助，開發人員可以利用系統架構編輯器來定義軟體代理人系統之架構，定義出系統中每個代理人的目標、代理人的任務、代理人之間的關係、代理人類別的介面、與代理人使用到的工具類別。

代理人之系統架構定義完後，使用者可利用程式碼編輯器開始撰寫一代理人內部的邏輯。程式碼編輯會讀入系統架構圖中的代理人資料，依據代理人的類別

定義產生類別樣板，包括類別變數與方法的宣告，藉以節省使用者的時間。使用者在撰寫代理人程式碼時可以套用編輯器提供的溝通樣板與學習力樣板；寫完代理人程式的主要邏輯，執行驗証無誤後，再套用行動力樣板。如此可幫助使用者漸進式的撰寫代理人程式碼，減少發生錯誤的機率。

使用者撰寫完代理人程式碼後，可利用佈局管理器設定要執行起來的代理人伺服器，並設定每個伺服器上要執行的代理人程式。佈局管理器會爲每個代理人伺服器產生操縱台 (Console) 視窗，讓使用者可以觀察代理人的執行結果。

我們可由第二章所得之代理人開發流程來看 AgentIDE 對流程中各個活動的支援，如圖 3-3 所示。

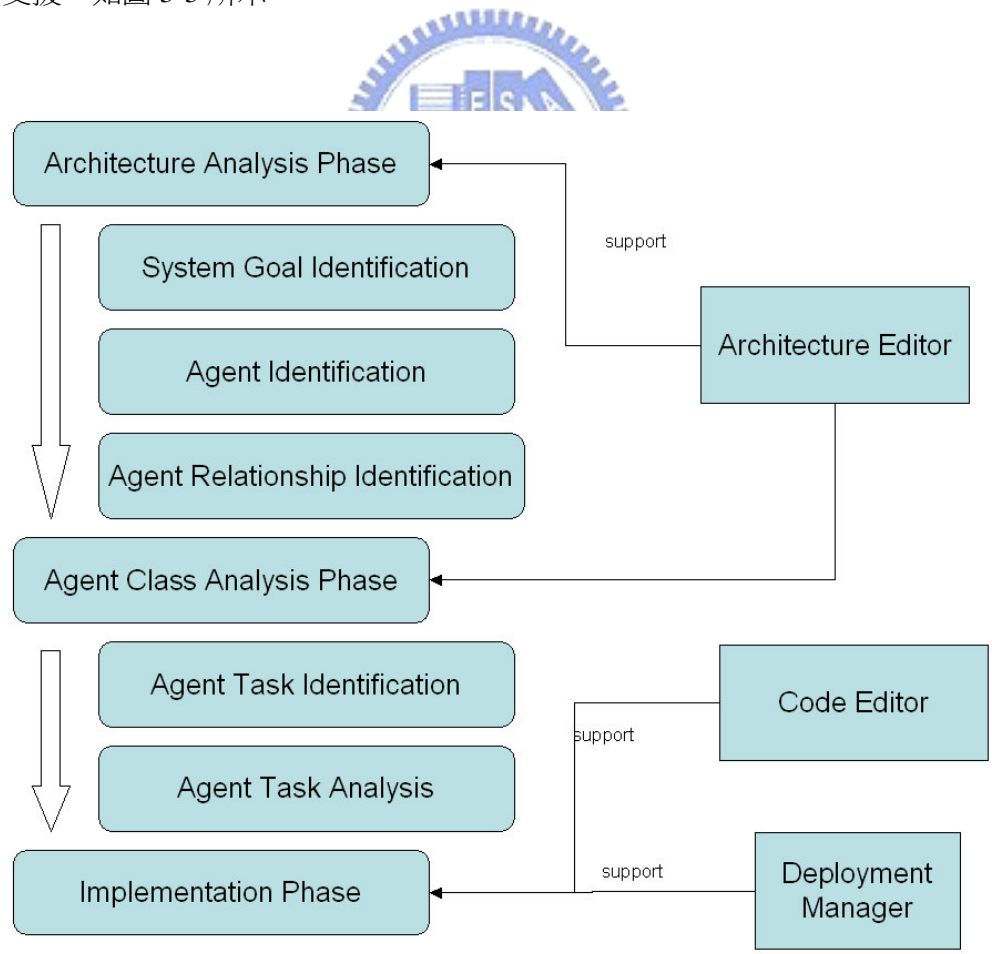

圖 3-3 AgentIDE 對軟體代理人開發流程各階段的支援

因此，由以上所述之流程，可以確定在 3.1 節所提的軟體代理人開發工具之需求能獲得滿足，並確定 AgentIDE 可以幫助使用者以 Top-Down 的方式定義出軟體代理人系統，並提供程式樣板幫助使用者撰寫代理人程式碼。AgentIDE 的設計可以幫助使用者能更有系統且更快速地發展軟體代理人程式。於第四章和第五章中，將針對 AgentIDE 主要的三個模組：系統架構編輯器、程式輯編器、佈局管理器分別做更詳細的設計。

第四章 系統架構編輯器之設計與實作

4.1 系統架構編輯器之功能需求

本工具的主要目的是幫助使用者以圖形化的方式定義代理人系統之架構。系統架構編輯器須支援使用者定義代理人名稱及其目標、定義各代理人爲達成目標所需執行之任務。使用者可依代理人任務之需要，決定是否要與其他的代理人產生供需關係；當代理人任務過於複雜時，使用者可利用主從關係產生新的子代理人來分擔工作。當整個代理人系統架構定義完之後，系統架構編輯器讓使用者進一步設定各個代理人間的關係模式及各代理人之行動力與學習力特性。

爲達成上述之需求，系統架構編輯器需俱備之功能如下：

1 支援圖形介面的編輯器，供使用者繪出代理人系統之架構圖，如圖 4-1 所示。

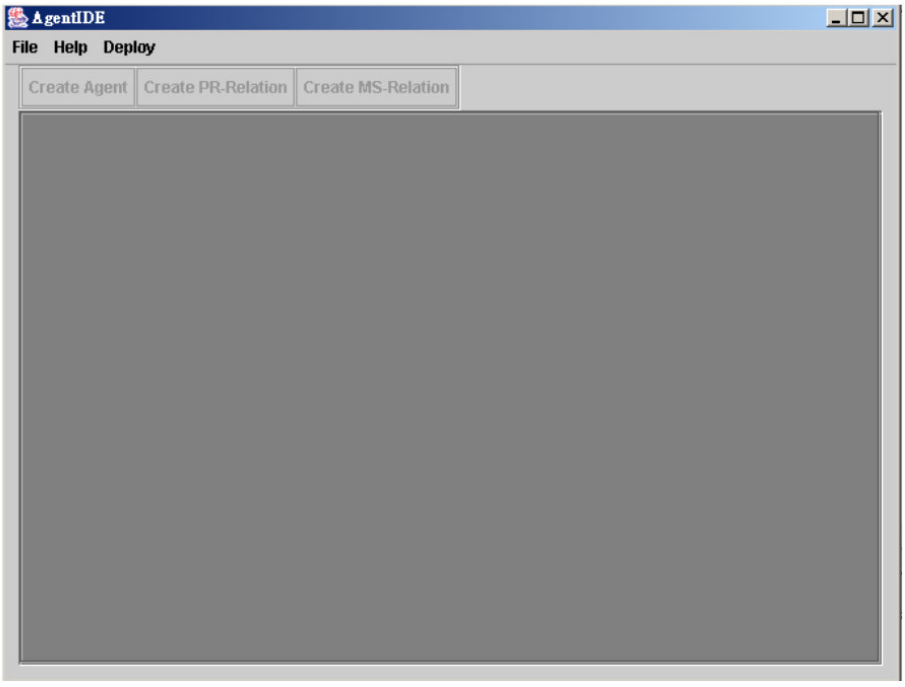

圖 4-1 系統架構編輯器介面

2　支援使用者定義代理人、設定其名稱與目標描述 (Goal Description)，並於架構圖中產生代理人圖示，如圖 4-2 所示，使用者在架構編輯器的工具列中，點選“產生新代理人”按鈕，然後在編輯器面版上的空白位置

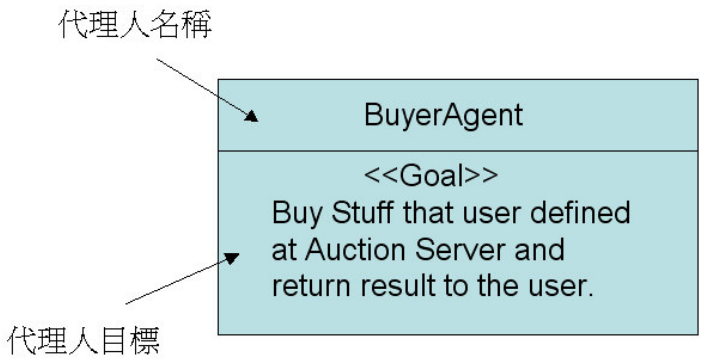

圖 4-2 代理人設定視窗與產生之圖示

點選滑鼠，架構編器會跳出代理人之設定視窗，如圖 4-3 所示，要求使用者填寫代理人之名稱與目標之描述。若代理人名稱重複定義或代理人名稱不符合類別命名規則，系統自動顯示警告訊息。

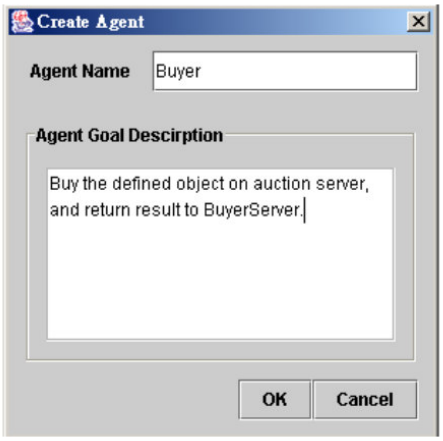

圖 4-3 產生代理人的設定視窗

3　使用者用滑鼠右鍵點選代理人圖示，編輯器會跳出編輯選單，如圖 4-4

所示，使用者點選其中的“Task Specification”選項後，編輯器會顯示代理人之任務編輯視窗，如圖 4-5，供使用者設定代理人之任務，圖 4-6 爲設定完成後的圖示。

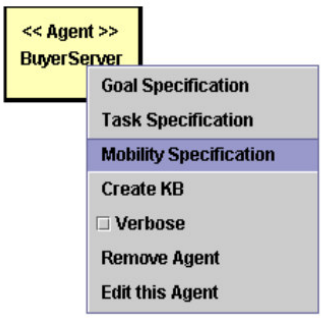

圖 4-4 編輯選單

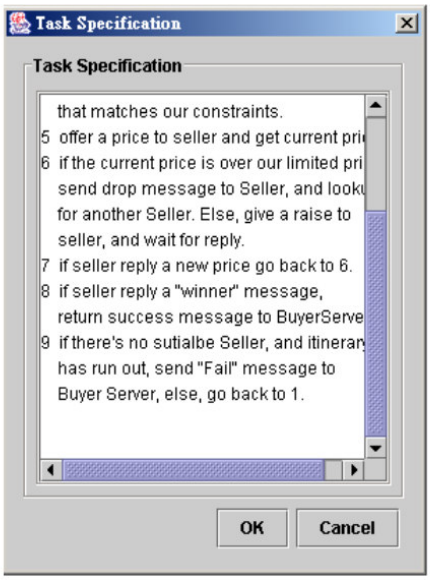

圖 4-5 代理人任務編輯視窗

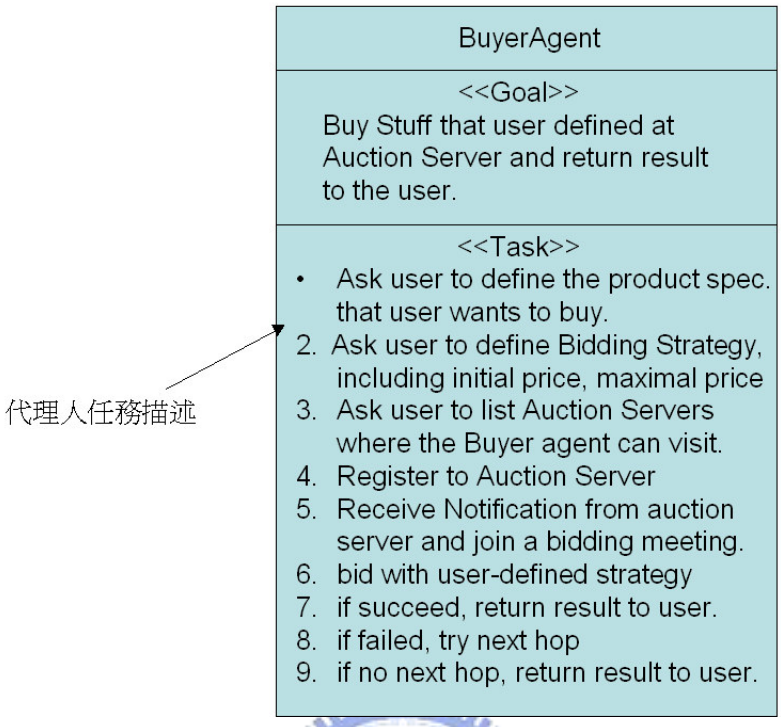

圖 4-6 代理人任務設定完後之圖示

4 支援使用者依代理人的任務建立供需關係 (Provider – Requestor)，並以符號表示代理人之間訊息傳遞的方向。使用者可點選架構編輯器工具列上的“供需關係”按鈕，以叫出供需關係產生視窗來設定供需關係兩端之代理人。由於使用者在完成整個系統架構圖後必須再針對每個供需關係做模式設定，因此新建立而未被設定的關係線會以紅色來表示。

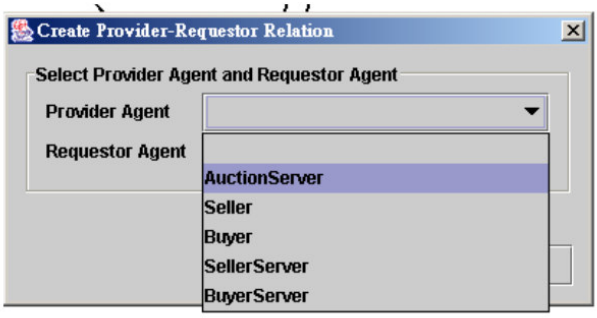

圖 4-7 供需關係設定視窗

5　支援使用者在代理人之間建立主從關係 (Master – Slave)，如圖 4-8 所示。使用者可點選架構編輯器工具列上的“主從關係”按鈕，系統會跳出主從關係之設定視窗，如圖 4-9 所示，供使用者設定主從關係兩端之代理人。

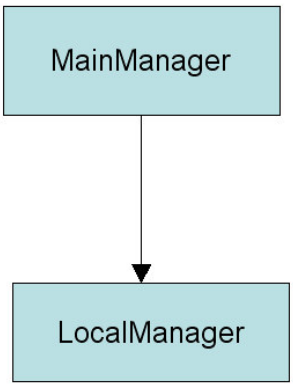

圖 4-8 主從關係圖示

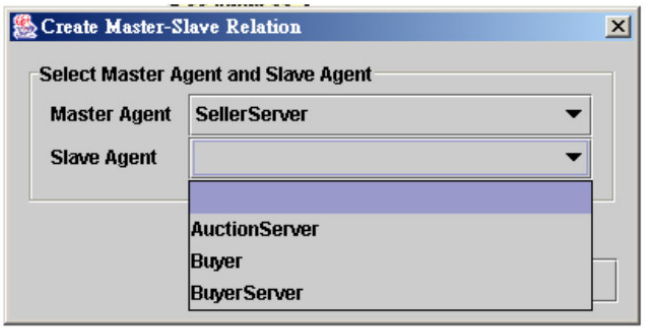

圖 4-9 主從關係建立視窗

6　支援使用者進一步設定供需關係的屬性，使用者用滑鼠雙擊系統架構圖中的關係線可叫出供需關係設定視窗，如圖 4-10，使用者可在此設定供需關係的屬性，包括”**多重性**”與”**傳送方式**”。供需關係的”**多重性**”分爲“**1 對 1**”、”**1 對多**”、”**多對 1**” 和 “**多對多**”，使用者可利用供需關係設定視窗中組合選單來選擇適當的型態。

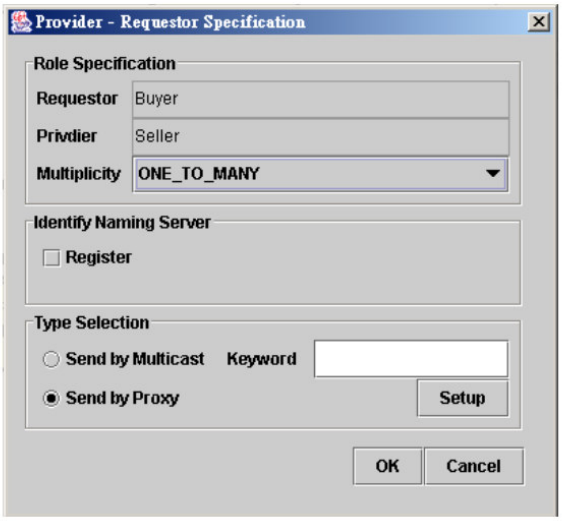

圖 4-10 供需關係設定視窗

供需關係設定視窗中的 “Identify Naming Server” 欄位，則是讓使用者指定此供需關係是否包含跟 Naming Server 的註冊的動作。由於代理人沒有類似 IP Address 的靜態位址，而是以執行時期才產生的 Proxy 物件來溝通，因此代理人若要在一台伺服器上被其他代理人找到，此代理人必定要先註冊，將自己的 Proxy 物件交給註冊對象，而註冊的對象會是另一個代理人或是底層的代理人伺服器，我們稱提供註冊與查詢服務的代理人爲 Naming Server。在系統架構編輯器中要指定一個代理人爲另一個代理人的 Naming Server 的方法，便是在其間的供需關係設定視窗中勾選 “Register”選項。如此，系統架構編輯器便會認定此供需關係中的 Provider 爲 Requestor 的 Naming Server，如圖 4-11 所示。

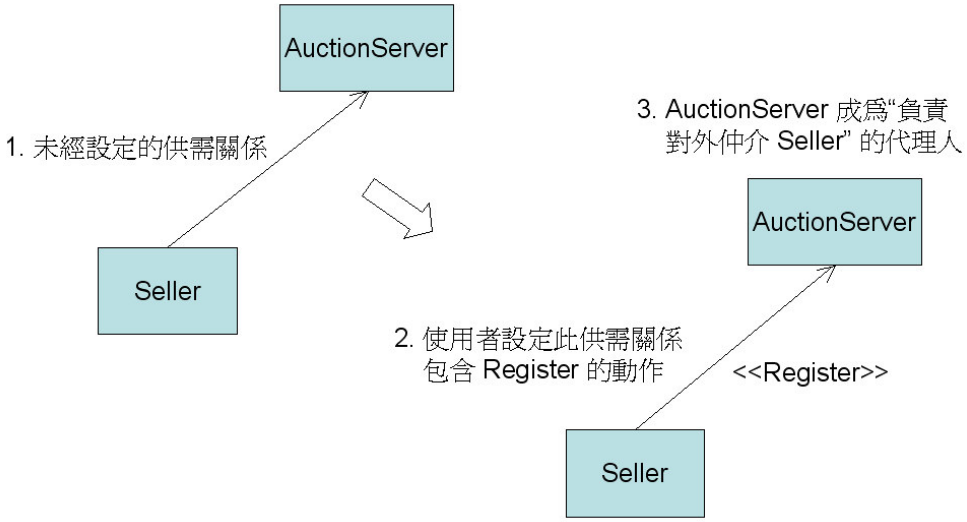

圖 4-11 指定其他代理人爲自己的 Naming Server

以圖 4-11 爲例，只要使用者設定 AuctionServer 爲 Seller 的 Naming Server，則系統中的其他代理人便可以透過 AuctionServer 的仲介服務以得到 Seller 的 Proxy。Naming Server 提供查詢 (Lookup) 以及主動告知 (Tell)的功能，如圖 4-12 所示，Auction Server 爲 Seller 的 Naming Server，因此它可提供 Buyer 來查詢，或是主動告知 Buyer。

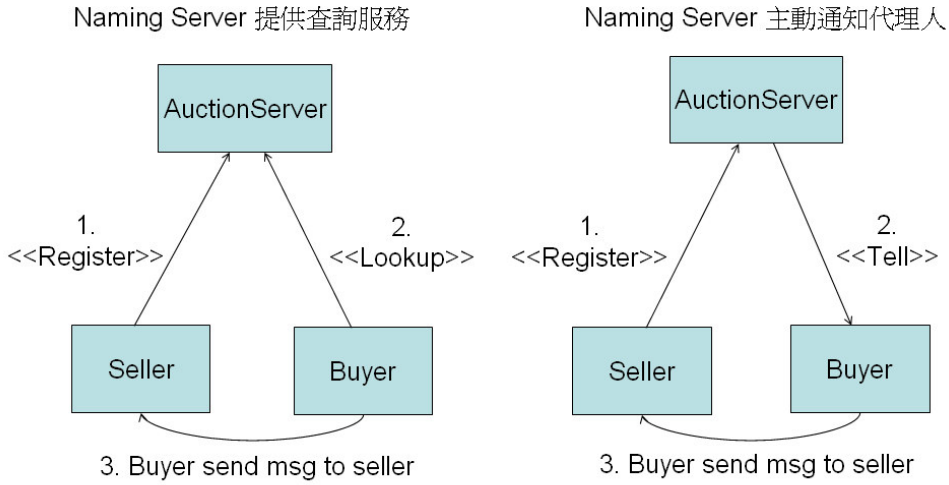

圖 4-12 Naming Server 提供 "Lookup" 與 "Tell" 的服務

Requestor 要取得 Provider 的 Proxy 物件，除了透過 Provider 的 Naming Server，還可以藉由自己的產生者 (Master) 來得知，如圖 4-13 所示。這表示上層的產生者雖有溝通對象的 Proxy，但是卻將此訊息傳送的任務交給子代理人來處理。

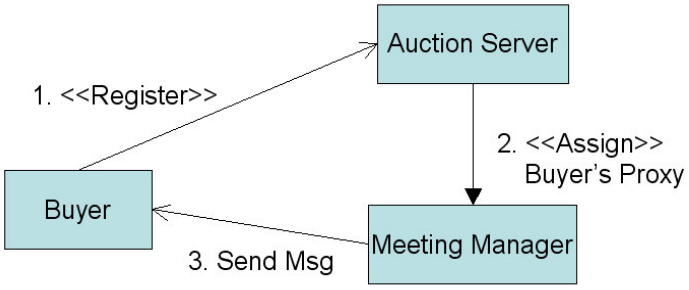

圖 4-13 Meeting Manager 透過產生它的
Auction Server 取得 Buy 的 Proxy

在供需關係設定視窗中，使用者需選擇供需關係的"傳送方式"，包括："**Send by Multicast**" 以及 "**Send By Proxy**"。當代理人利用 Multicast 的方式來傳送訊息時，代理人不需要取得對方的 Proxy，但是 Multicast 的收送兩端必須在同一台主機上才能完成。

若使用者設定供需關係型態爲 "**Send By Proxy**"，則表示此供需關係是藉由代理人的 Proxy 物件來溝通，此時使用者必須設定 Requestor 取得 Provider Proxy 的方式。由之前的分析，我們可將 Requestor 要取得 Provider Proxy 的方法規納爲下列七種：

1 **By Agent Server Support**：Requestor 以關鍵字向底層的 Agent Server 註冊，而 Provider 則利用此關鍵字向 Agent Server 查詢。

2 **Master Assigned**：Requestor 的 Master 以初始化變數告知 Provider 的 Proxy。系統架構編輯提供此選項的先決條件爲 Requestor 有產生者，其 Requestor 的上層代理人 (Ancestor Agents) 中必須有指向 Provider 的供需關係或是跟 Provider 的 Naming Server 有供需關係，如此 Requestor 的上層代理人才有可能得到 Provider 的 Proxy 物件，並傳給 Requestor。

3 **Use Multicast to Ask Provider**：Requestor 利用 Multicast 直接詢問 Provider。

4 **Requestor Create Provider**：Requestor 和 Provider 之間存在主從關係，致使 Requestor 可在產生 Provider 後直接擁有其 Proxy 物件。

5 **Provider has register to Requestor**：Provider 已經將自己的 Proxy 告知 Requestor，使 Requestor 可直接跟 Provider 溝通。

6 **By Ask Naming Server**：Requestor 詢問 Provider 的 Naming Server。系統架構編輯器提供此選項的條件爲 Requestor 存在一條指向 Naming Server 的供需關係。

7 **Naming Server Tell**：Provider 的 Naming Server 主動通知 Requestor。系統架構編輯器提供此選項的條件爲 Naming Server 存在一條指向 Requestor 的供需關係。

上述的七個選項中，部分選項有限制，供需關係的設定視窗會根據各選項的限制，提供使用者可能的選項，讓使用者選擇 Requestor 取得 Provider Proxy 的方式，如圖 4-14 所示。藉由讓使用者設定供需關係的機制，系統架構編輯器可以協助使用者檢查各個關係是否都能如預期般建立。最後，每個設定完成的關係線會由原本的紅色轉爲黑色，表示此關係已設定完成。

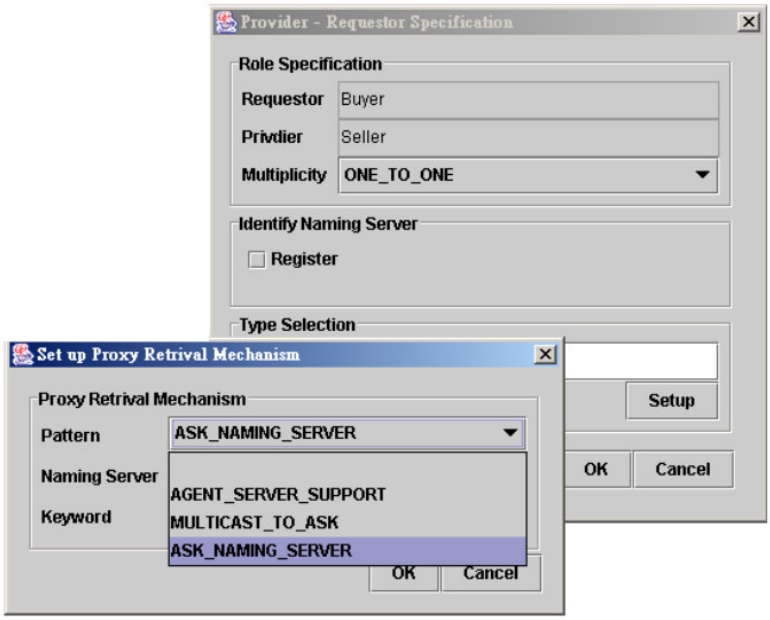

圖 4-14 設定 Requestor 取得 Proxy 的方式

7 支援使用者進一步設定主從關係的多重性，使用者可用滑鼠雙擊架構圖中的主從關係連線叫出設定視窗，如圖 4-15，主從關係的多重性選項只

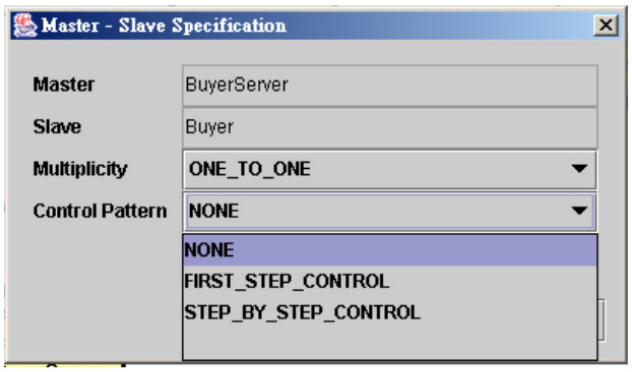

圖 4-15 主從關係設定視窗

包含 “1 對 1” 和 ”1 對多”，因爲一個代理人物件只會由一個父代理人產生一次。主從關係設定視窗中的 “Control Pattern”選項是讓使用者選

擇 Master 對 Slave 的控制方式，共有三種選項：

1 None：Master 產生 Slave 後不會控制代理人。

2 First Step Control：子代理人產生出來後，沒有要在本地端主機執行任務，而由父代理人將其遷移到指定之位置才開始執 行主要邏輯。

3 Step By Step Control：子代理人本身不會主要遷移，而是由其父代理人控制代理人每次要遷移的位址。

主從關係設定視窗中會紀錄 Master 代理人要在初始化 Slave 代理人時傳進的 Proxy，當使用者設定供需關係時，系統編輯器會利用此紀錄幫助使用者尋找是否 Requestor 的產生者 (Master) 會將 Provider 的 Proxy 傳給 Requestor。

8 當代理人系統架構定義完，各個關係的模式也設定完之後，系統架構編輯器會讓使用者設定代理人特性，包括 “行動力” 與 “學習能力” 兩個特性。

(1) 設定 “行動力”：

爲了在靜態的架構圖中表示代理人的行動力，我們將代理人使用行動力的模式做出分類，讓使用者可直接選取。行動力模式的分類如下；

1 Itinerary Pattern：代理人將利用串列型態的變數 (Link-List or Array) 儲存要經過的網路節點位址，這種變數我們稱爲旅程變數 (Itinerary)。代理人在每次執行完主要邏輯之後，便要求代理人伺服器將自己遷移到旅程變數串列中的下一個網路節點，如此循環到走完所有節點，或代理人目標滿足爲止。採用 Itinerary Pattern 的代理人必然會在初始化階段設定旅程變數，並在每次主要邏輯執行完之後遷移

到下個主機位址，儘管代理人可能在執行主要邏輯時動態修改旅程變數，Itinerary Pattern 並不受影響。

2 One Shot Pattern：若代理人僅須主動遷移一次到特定位址去執行，則稱此代理人的行動力模式爲 “One Shot Pattern”。此模式有兩種做法；第一，代理人在初始化函式 (onCreation) 的最後一步執行遷移指令；第二，代理人在主要邏輯中以條件式包住遷移指令，預設情況將此條件式之狀變數設爲 “true”，當代理人遷過後便將此條件之狀態變數設爲 ”false”，如此可保證代理人只會遷移一次。

3 Dynamic Decision Pattern；若代理人沒有旅程變數，而且在生命週期中會不斷依主要邏輯執行之結果做遷移，則稱此種行動力之模式爲 Dynamic Decision Pattern。

上述遷移模式包括 Itinerary Pattern、One Shot Pattern、Dynamic Decision Pattern，其中 Itinerary Pattern 和 Dynamic Decision Pattern 和主從關係模式裡所設定的 “One Shot Pattern”可以共用。當使用者以滑鼠右鍵點選代理人圖示時，會跳出 Pop-Up Menu，點選其中的 “Mobility Setup” 後，系統會顯示行動力模式設定視窗，如圖 4-16，使用者可在此選擇代理人的行動力模式，設定好的模式會顯示在代理人圖示中，如圖 4-17 所示。

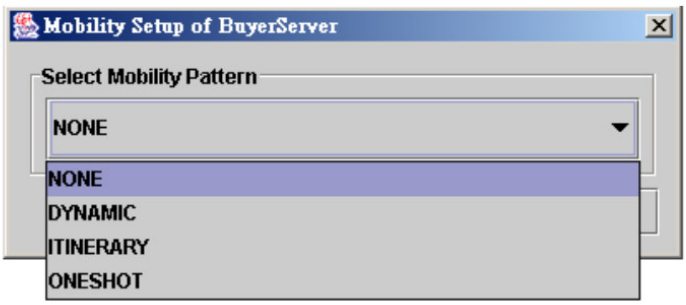

圖 4-16 行動力設定視窗

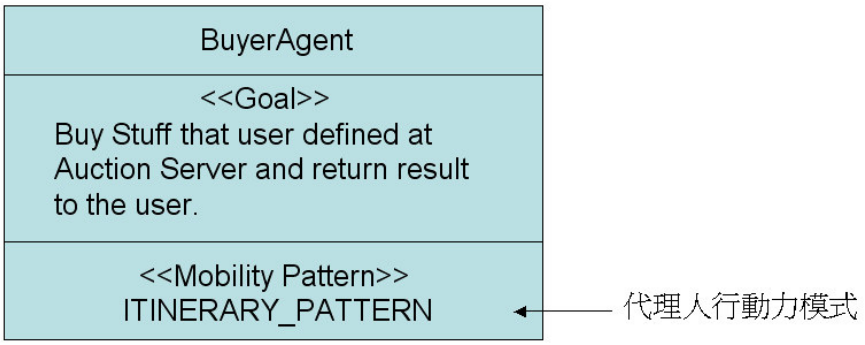

圖 4-17 代理人行動力之圖示

(2) 設定 “學習能力”：一個有學習能力的代理人必然包含一個知識庫來儲存習得之知識。代理人使用知識庫的過程分爲兩個步驟，第一是先取得知識庫的存取管道，第二步透過存取介面來存取知識庫。因此系統架構編輯器支援使用者設定代理人的知識庫，並將知識庫以圖示表現於代理人系統架構圖中，如圖 4-18 所示。知識庫的設定包含：

1 知識庫的名稱

2 知識庫的儲存媒體：資料庫、檔案、記憶體。

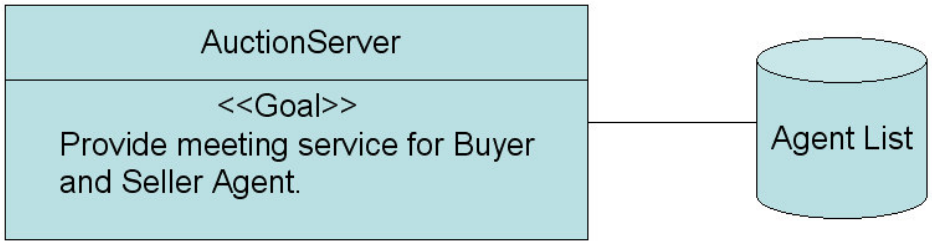

圖 4-18 代理人的知識庫之圖示

9　提供修改、刪除代理人相關設定與代理人之間的關係的功能。

10　提供使用者可調整的圖形面板，包括代理人圖示可依滑鼠拖曳而改變位置與大小。

4.2 系統編輯器之系統架構

圖 4-19 顯示本系統的系統架構圖，其中方塊圖示表示此系統的主要模組，而圖柱體則表示外部儲存媒體，實心箭頭表示資料流的傳遞。在各個模組中，編輯面板 (Editor Panel) 負責呈現系統架構圖之外觀與接收來自使用者之滑鼠事件輸入；代理人建構器 (Agent Constructor) 負責建立代理人之基本資料，包括代理人之名稱與目標描述；代理人關係建構器 (Agent Relation Constructor) 負責建立代理人之間的關係。以下對各個模組做介紹。

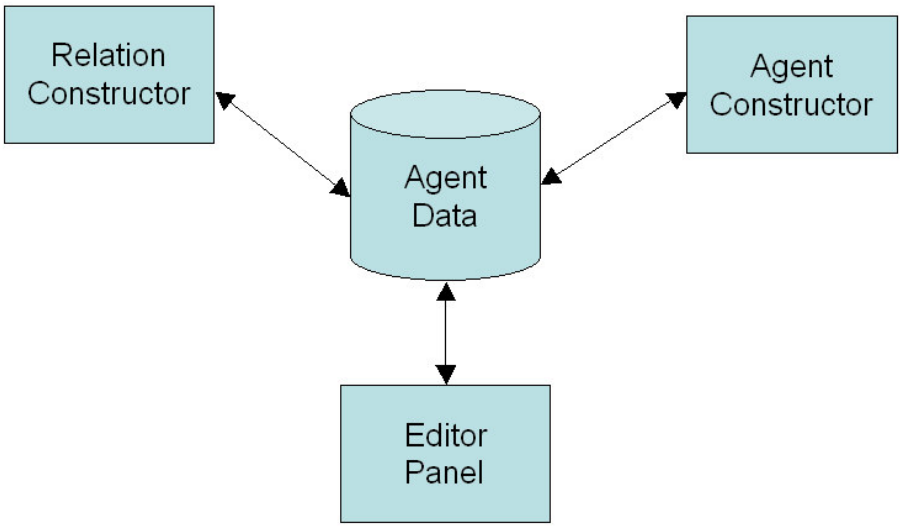

圖 4-19 系統編輯器的系統架構

1　編輯面板模組

編輯面板模組負責管理面版中各個圖示物件、記錄圖示物件之大小與呈現之位置，並接收來來自鍵盤與滑鼠之輸入，提供使用者編輯架構圖之功能。繪圖面板可將使用者設定好之代理人與代理人關係以圖示呈現於畫面中，並可讓使用者

調整面板中的各個圖示之位置與大小。當使用者利用代理人建構器或是代理人關係建構器改變代理人系統的資料時，編輯面板模組會將使用者的改變呈現出來。

2 代理人建構器

代理人建構器包含五個部分：

(1) 代理人基本資料建構器：負責在使用者產生代理人時，支援使用者設定代理人之名稱與目標描述，若發生代理人名稱重複定義的情況時，代理人基本資料建構器會顯示錯誤訊息。

(2) 代理人任務建構器：負責支援使用者設定代理人之任務，讓使用者詳細地描述代理人爲達成目標所需完成之工作。

(3) 代理人關係建構器：負責支援使者建立代理人之間主從與供需關係，以建立代理人系統架構。

(4) 代理人關係模式設定器：負責支援使用者定義代理人關係的模式，包括主從關係中父代理人操作子代理人的模式設定，與供需關係中需求方代理人如何取得供給方代理人的 Proxy 的設定。

(5) 代理人特性建構器：負責支援使用者設定代理人之特性，包括行動力特性建構器與知識庫建構器。

4.3 資料結構

系統編輯器係由 Java 1.4.1 語言實作，主要會使用到 Java 所提供之視窗程式套件以簡化實作之過程。圖 4-20 顯示此編輯器內部主要類別之關係圖。下面將介紹每個類別內部的資料結構。

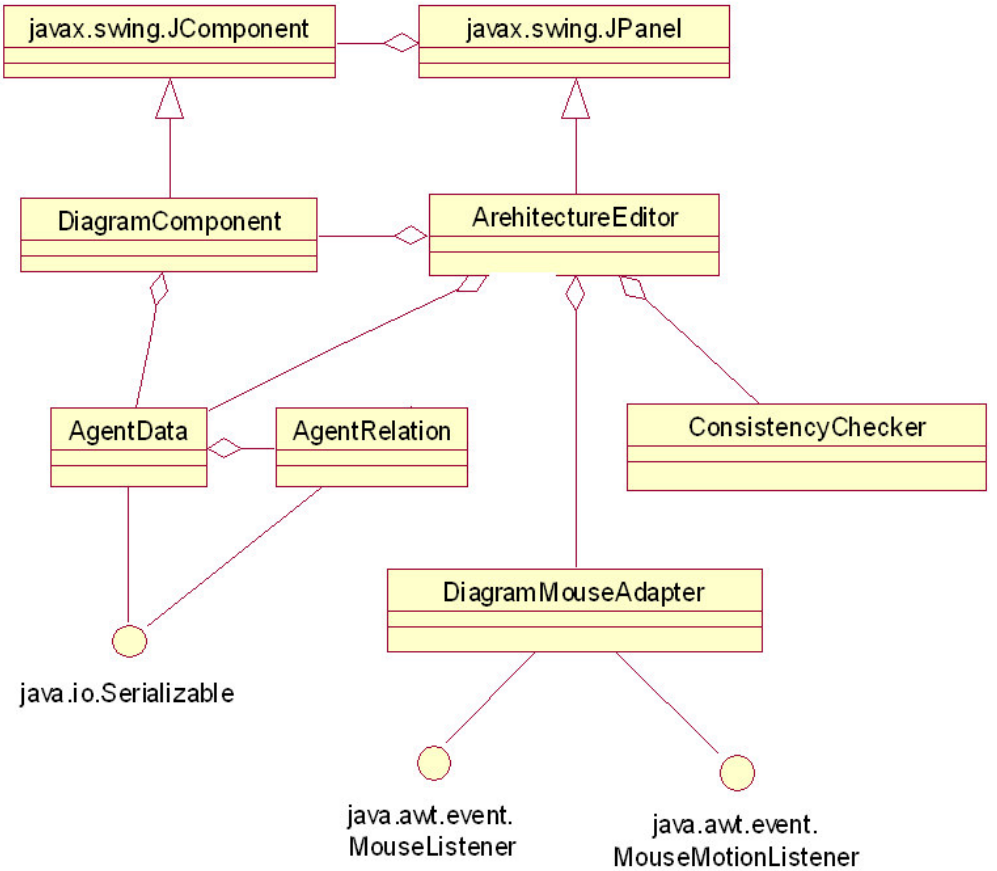

圖 4-20 系統架構編輯器實作之類別圖

(1) 類別“AgentData”：使用者以系統架構編輯器定義代理代理人時，編輯器會將使用者做的設定儲存於“AgentData”類別的物件中。AgentData 要記錄的包括

1 代理人類別名稱與目標描述。

2 代理人任務描述。

3 代理人的行動力模式。

4 代理人的學習力模式。

5 代理人往外相連的關係，包括供需關係與主從關係。

6 代理人圖示於編輯面板中的位置與大小。

(2) 類別 “AgentRelation”：AgentRelation 類別的物件負責儲存代理人之間的關係資訊。當使用者於編輯器面板中建立代理人的關係時，編輯器會產生 AgentRelation 物件來記錄代理人的關係。AgentRelation 要儲存的資料包括：

1 關係的模式 (供需關係或是主從關係)

2 關係的起始代理人

3 關係的終點代理人

(3) 類別 ”ArchitectureEditor”：ArchitectureEditor 類別繼承自 Java 的 JPanel 類別，並覆寫其 paintComponent(Graphics g) 方法。目的是利用 JPanle 類別的圖形元件容器功能來管理於其上所繪之元件 (代理人圖示)，而代理人圖示關係連線則由 ArchitectureEditor 負責於呈現面板中繪出，當使用者以滑鼠拖曳代理人圖示時，Architecture Editor 必須自動更新目前的代理人圖示位置與代理人關係連線。Arechitecture Editor 要儲存的資料包括：

1 代理人圖示物件與其對應之代理人資料物件

2 代理人關係物件

3 編輯器狀態

(4) 類別 “DiagramComponent”：DiagramComponent ,類別繼承自 java 的 JComponent 類別，目的是要利用 JComponent 類別的所提供之圖形元件功能。DiagrramComponent 類別覆寫 JComponent 類別的 paintCompoent(Graphcis g)方法，於其中加作繪出代理人圖示之程式碼。每個 DiagramComponent 會對應一個 AgentData 物件，當使用者利用編輯面板修改代理人資料後，DiagramComponent 會在重繪時更新圖示的內容。

我們將系統編輯器內部各類別之資料結構詳列於下：

1 AgentData：此類別負責儲存代理人相關之資料

```
public class AgentData {
    String agentName;              // 代理人類別之名稱
    String goal;                   // 代理人類別之目標描述
    Vector taskList;               /* 代理人之任務，各任務之描述以
                                      TaskData 格式儲 */
    String activeMobilityPattern   // 代理人的主動遷移模式
    String passiveMobilityPattern  // 代理人的被動遷移模式
    Vector proxyData;              // 目前已知其 Proxy 的代理人
    int x;                         // 代理人圖示物件於編輯畫面中的 x 座標
    int y;                         // 代理人圖示物件於編輯畫面中的 y 座標
    int width;                     // 代理人圖物件之寬度
    int height;                    // 代理人圖示物件之高度
    Vector relationships;          // 以此代理人爲起點之關係，儲存之元素爲
                                   // 類別 AgentRelationship
    DiagramComponent component;    /* 此代理人資料物件所對應到的圖形
                                      元件 */
}
```

2 TaskData：此類別負責紀錄代理人之任務描述與任務所使用之類別變數與方法。

```
public class TaskData {
    String description;            // 任務之描述
}
```

3 MethodData：此類別專門儲存類別方法宣告之內容

```
public class MethodData {
    String methodName;          // 類別方法之名稱
    Hashtable args;             // 類別方法所擁有之變數
}
```

4 AgentRelationship：此類別負責記錄代理人之關係

```
public class AgentRelationship {
    // 定義供需關係
    public static int PROVIDER_REQUESTOR = 0;
    // 定義主從關係
    public static int MASTER_SLAVE = 1;
    // 定義 Multiplicity
    public static int ONE_TO_ONE = 0;
    public static int ONE_TO_MANY = 1;
    public static int MANY_TO_ONE = 2;
    public static int MANY_TO_MANY = 3;
    int type;                   // 關係之模式，(主從關係或供需關係)
    int multiplicity;           // 關係之多重性
    Boolean isRegister;         // 是否包含註冊之動作
    Boolean isLookup;           // 是否包含查詢之動作
    Boolean isTell              // 是否包含告知的動作
    int proxy_get_mechanism; // 取得 Provider Proxy 的模式
    AgentData startAgent;   // 發動關係的代理人
    AgentData endAgent;     // 接受關係的代理人
```

}

5 ArchitectureEditor：繼承 javax.swing.JPanel，負責呈現代理人系統架構之圖示，並提供使用者新增代理人，修改代理人，建立與移除代理人關係之功能。

```
public class ArchitectureEditor extends javax.swing.JPanel {
    // 定義 ArchitectureEditor “插入代理人”之工作狀態
    public static int INSERT_AGENT = 0;
    // 定義 ArchitectureEditor “建立主從關係”之工作狀態
    public static int CREATE_MS_RELATION = 1;
    // 定義 ArchitectureEditor “建立供需關係”之工作狀態
    public static int CREATE_PR_RELATION = 2;
    Vector agentList;          /* 儲存編輯器上所有代理人之資料，Vector
                                  儲存之元素爲類別 AgentData */
    Vector relationList;       /* 儲存編輯器上所有的關係資料，Vector
                                  儲存之元素爲 AgentRelationship 類別 */
    Hashtable lineAreas;       /* 記錄目前編輯上每一條關係線的形狀與
                                  關係線所對之 AgentRelationshep 物件，
                                  當使用者於編輯面板上點選關係線時，
                                  編輯器必須判斷使用者以滑鼠點選之位
                                  置是否落於目前之關係線上 */
}
```

6 類別 “DiagramComponent”：繼承自 javax.swing.JComponent，目的是利用 JComponent 類別支援的圖形元件狀態管理的功能，並於其上依資料描繪代理人之圖示。

```
public class DiagramComponent extends JComponent {
```

```
    AgentData agentData;        // 此圖示元件所對應之代理人實際資料
    boolean isSelected;         /* 記錄此圖示目前的選取狀態，若是，則
                                   繪出代理人被選取時之圖示 */
    boolean isClassShowed;      /* 記錄目前代理人圖形顯示的內容，若此
                                   變數爲 true，即表示代理人圖形應顯示
                                   類別內容，若否，則應顯示代理人之名
                                   稱 */
}
```

7 類別 “DiagramMouseAdapter”：實作 java.awt.event.MouseListener 與 java.awt.event.MouseMotionListener，此類別會安裝於每每代理人圖示類別上，其功用是專門處理來自使用者之滑鼠事件，包括使用者拖曳代理人

```
public class DiagramMouseAdapter implements MouseListener,
                                             MouseMotionListener {
    int original_x,             /* 當使用者以滑鼠點選代理人圖示
                                   時，記錄滑鼠點選之 x 軸座標 */
    int original_y;             /* 當使用者以滑鼠點選代理人圖示
                                   時，記錄滑鼠點選之 y 座軸座標 */
    int original_w,             /* 當使用者以滑鼠點選代理人圖示
                                   時，記錄此圖示之寬度 */
    int original_h;             /* 當使用者以滑鼠點選代理人圖示
                                   時，記錄此圖示之長度 */
}
```

第五章 程式編輯器之設計與實作

5.1 程式編輯器與佈局管理器之功能需求

程式編輯器之目的在於提供使用者一個文字編輯介面以撰寫代理人之程式碼。當使用者以系統架構編輯器定義完代理人系統的架構之後，可利用程式編輯器進一步設計與實作各個代理人類別。程式編輯器需利用系統架構圖中所得之資料提醒使用者此代理人需完成的工作，並提供此代理人所需的程式碼樣板讓使用者選取套用。

爲達成上述的功能需求，程式編輯器應俱備下列之功能：

11 支援使用者文字編輯介面以撰寫代理人之程式碼。使用者在系統架構圖中以滑鼠雙擊要進行程式碼編輯的代理人，即可進入此代理人的程式編輯介面，如圖 5-1 所示。AgentIDE 所提供的程式編輯器包含四個部分：

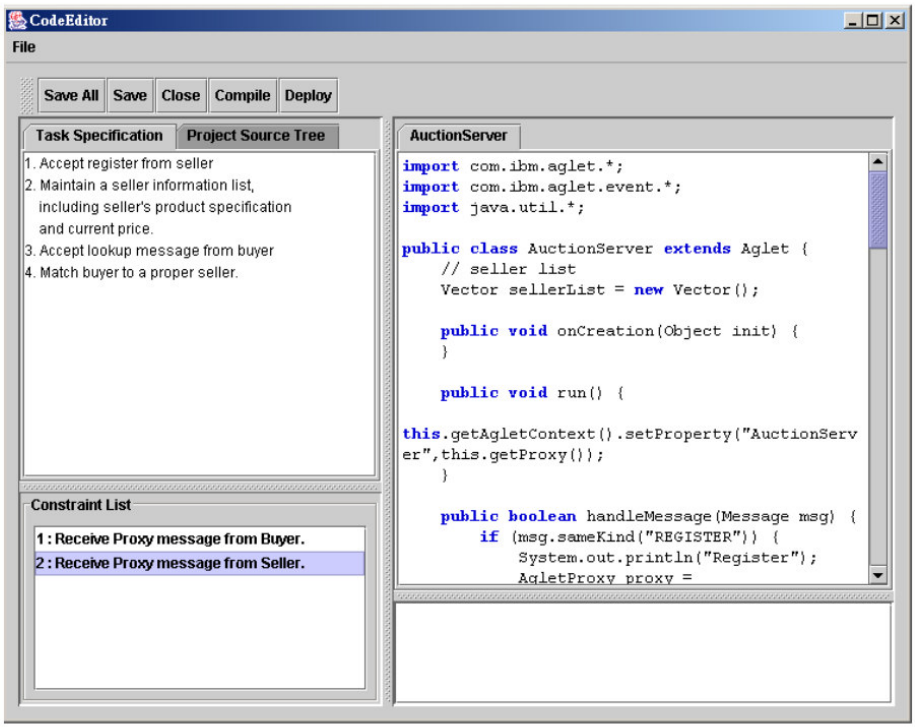

圖 5-1 程式碼編輯器介面

(1) 文字編輯面板 (Text Editor)：提供使用者撰寫程式碼之介面。並依使用者在系統架構圖中對於此代理人合作、遷移、與學習力的設定提供相對應的樣版程式碼。

(2) 任務描述面板 (Task Specification)：將使用者在系統架構圖中定義的任務列出，方便使用者於設計與實作代理人程式時對照參考。

(3) 特性需求面板 (Agent Constraints)：若於系統架構圖中，此此代理人有行動力、學習力、或是與其他代理人有關係，包括供需關係與主從關係，程式編輯器會依代理人所扮演的角色與擁有的特性在工作摘要上顯示提醒文字，例如此代理人必需處理來自某一代理人的訊息。

(4) 訊息輸出視窗 (Output Window)：當使用者撰寫完程式之後可利用編譯工具 (Compiler) 產生 .class 檔，若編譯時發生錯誤，程式編輯器則會將錯誤訊息顯示於訊息輸出視窗中。

12 程式編輯器需依使用者於系統架構圖中的設定，產生一分特性需求表以提醒使用者此代理人需完成之工作，代理人的特性需求列表內容應包括：

(1) 使用者於系統架構圖中所填寫的代理人任務描述 (Task Specification)，讓使用者方便對照並設計代理人的內部邏輯。

(2) 此代理人需處理哪些代理人所送來的訊息，以及處理的方式 (**Concurrent message handling** 或是 **Sequential Message handling**)。

(3) 此代理人需以何種型式發送訊息給另一個代理人 (**Send By Multicast** 或是 **Send By Proxy**)。

(4) 此代理人需產生的代理人。

(5) 此代理人需要套用的行動力模式。

(6) 此代理人需使用的知識庫，以及此知識庫的名稱與型式。

13 當使用者第一次編輯某一代理人之程式碼時，程式碼編輯器需在專案目錄下產生對應的類別原始檔，並讀入系統架構圖中的代理人資料，依代理人之規格幫使用者預先產生類別程式碼樣板，即類別宣告和代理人類別內部之方法宣告，代理人類別方法包括：

(i) onCreation(Object init)

(ii) run()

(iii) handleMessage(Message msg)

(iv) onArrival

(v) onDispatching

(vi) onDisposing

其中 onCreation()、run()、onDisposing() 函式是每個代理人都必需有的，因此程式編輯器會直接幫使用者產生。若此代理人於系統架構圖中有扮演供需關係中的 Provider 角色，則須俱備 handleMessage() 函式以處理訊息，因此編輯器會幫使用者自動產生此函式之宣告。而 orArrival() 與 onDispatching() 兩函式，編輯器則會依此代理人是否有行動力來決定是否幫使用者產生。

14 當使用者開始撰寫程式碼時，編輯器會依據系統架構圖中代理人的設定，提供使用者對應的程式碼樣板。例如在系統架構圖中，若一代理人與另一代理人間的供需關係是以 Multicast 的方式建立，則當使用者編寫此代理人之程式時，程式編輯器便可提供使用者 Multicast 的樣板程式碼以供套用。

程式碼樣板的目的是將使用者經常用到的特性之程式碼存於發展環境

中，並在需要時可直接套用，使用者只需要做些微的修改即能符合其需求。由於在軟體代理人程式中，我們發現許多代理人特性有固定的運作模式，也有對對應的程式碼撰寫模式，因此可以將這些運作模式轉換爲程式樣板，供使用者在撰寫程式碼時直接套用。使用者要套用樣板程式碼時，需先將輸入游標移至插入地點，並點選滑鼠右鍵以呼叫程式碼樣板選單，選單中會根據系統架構編輯時所得到的資料與溝通、遷移、學習模式的限制而列出目前此代理人可套用之樣板供使用者選擇，如圖 5-2 所示。使用者選定之後，樣板程式碼會在使用者指定

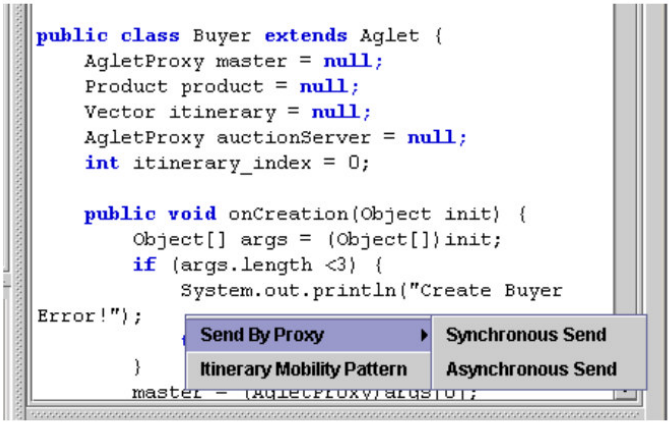

圖 5-2 於程式編輯介面叫出樣板選單

之位置插入。程式編輯器提供之樣板包括：

(1) 溝通樣板：若代理人在系統架構圖的供需關係中扮演 Requestor 的角色，表示代理人程式中需要主動發送訊息的程式碼，因此編輯器須提供代理人發送訊息的程式碼樣板，於使用者指定的位置插入。代理人傳送訊息的程式碼樣板包括：

(i) 同步傳送與接收訊息之樣板。程式編輯器會讓使用者設定要傳送之對像，並填入要傳送之物件，以及要接收結果之物件名稱，程式編輯器會在使用者指定之位置插入同步傳送與接

收訊息之程式碼樣板，如圖 5-3 所示。

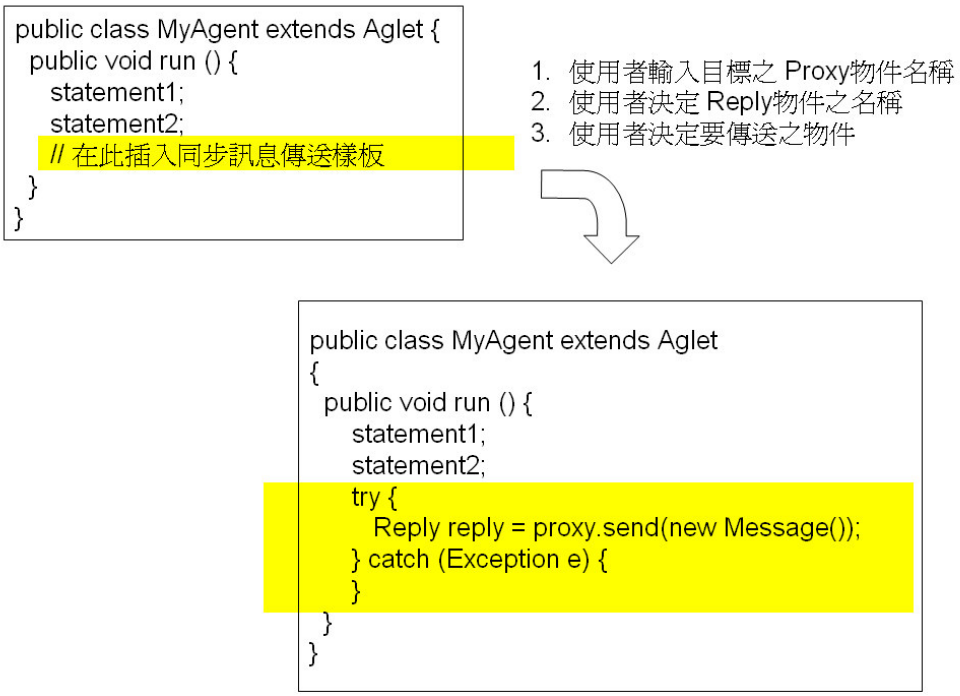

圖 5-3 同步傳送與接收訊息之樣板

(ii) 非同步傳送與接收訊息之樣板。程式編輯器會讓使用者設定要傳送之對像，並填入要傳送之物件，以及要接收之物件名稱，程式編輯器會在使用者指定之位置插入非同步傳送訊息之程式碼，如圖 5-4 所示。

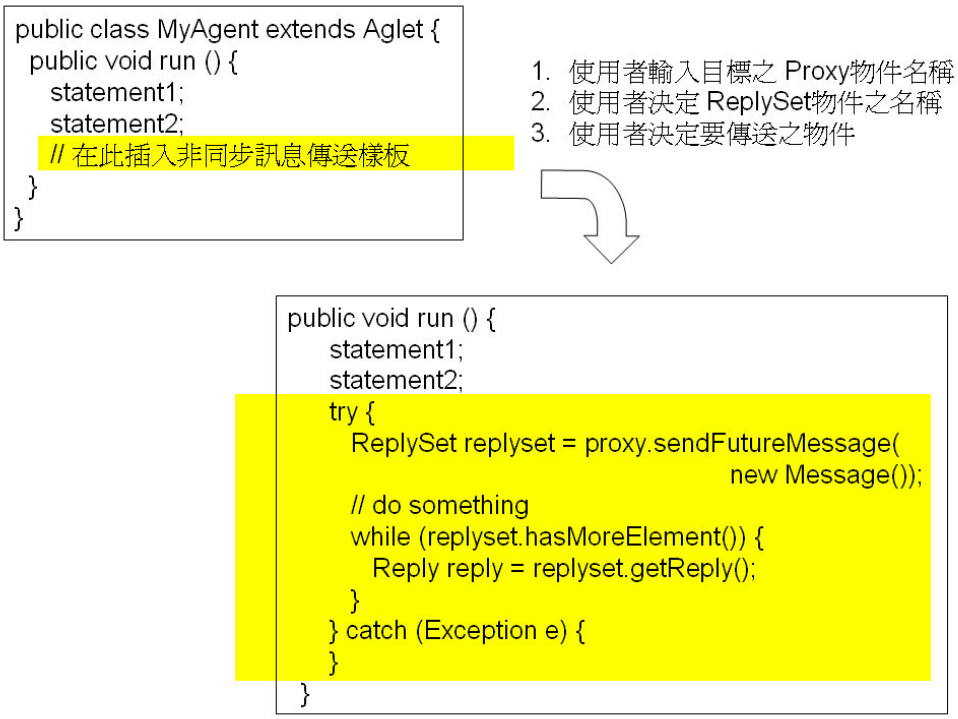

圖 5-4 非同步傳送與接收訊息之樣板

(iii) 群播訊息與接收群播回應之樣板。程式編輯器會讓使用者設定群播訊息的關鍵字，以及要接收回應之物件名稱，程式編輯器會在使用者指定之位置插入群播之程式碼，如圖 5-5 所示。

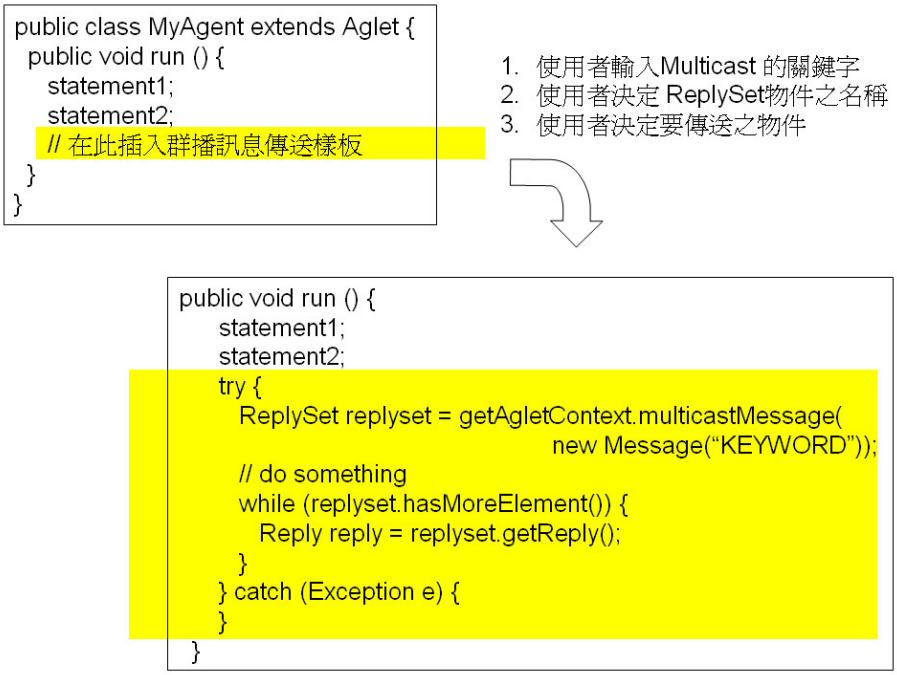

圖 5-5 群播訊息與接收群播回應之樣板

(2) 行動力模式：在系統架構編輯器中，使用者必須設定代理人的行動力模式，分爲主動遷移與被動遷移兩個部分。主動遷移是指代理人在自己的邏輯中下指令要遷移，包含 Itinerary Pattern、One Shot Pattern、Dynamic Generated Next hop Pattern 等三種型態。而被動遷移是指代理人由其父代理人控制要遷移的位址，包含 First Step Control 與 Step by Step Control 兩種型態。針對上述的各個型態，AgentIDE 的程式碼編輯都提供一分樣板程式碼，方便使用者套用，情細內容分述如下：

(i) Itinerary Pattern：表示此代理人需要一個記錄旅程的變數，並循序遷移至旅程變數中的各個節點。使用者要套用此樣板時，可在程式碼編輯介面上按下滑鼠右鍵，叫出“樣板套用選單”，如圖

5-4 所示，並選擇其中 ”Apply Itinerary Pattern” 選項。程式編輯器首先會幫使用者宣告一個全域的旅程變數，接著在代理人的 onCreation 函式中填入旅程設定之程式碼，最後在代理人主要邏輯後面加上 ”遷移至下一個旅程節點” 的程式碼，如圖 5-5 所示。套用完畢之後，程式編輯器會跳出詢問視窗，讓使用者選擇哪些原本在主要邏輯中的變數必須改爲代理人的全域變數，程式編輯器可以幫使用者做處理，如圖 5-6 所示。

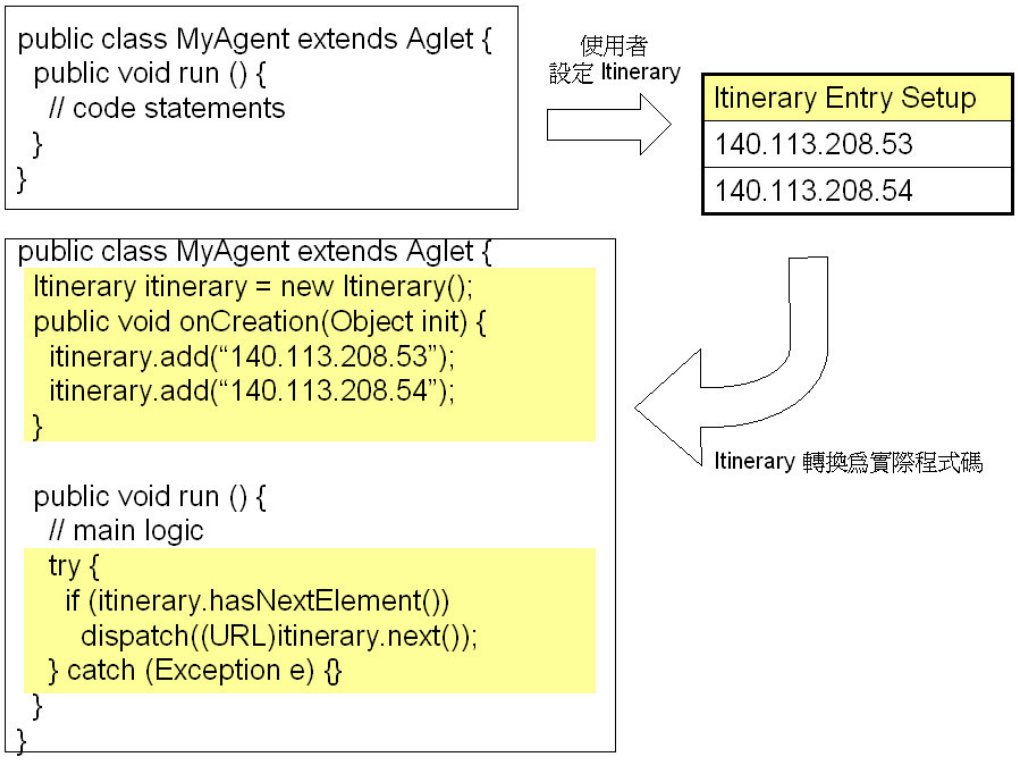

圖 5-5 套用 Itinerary Pattern 的樣板程式碼

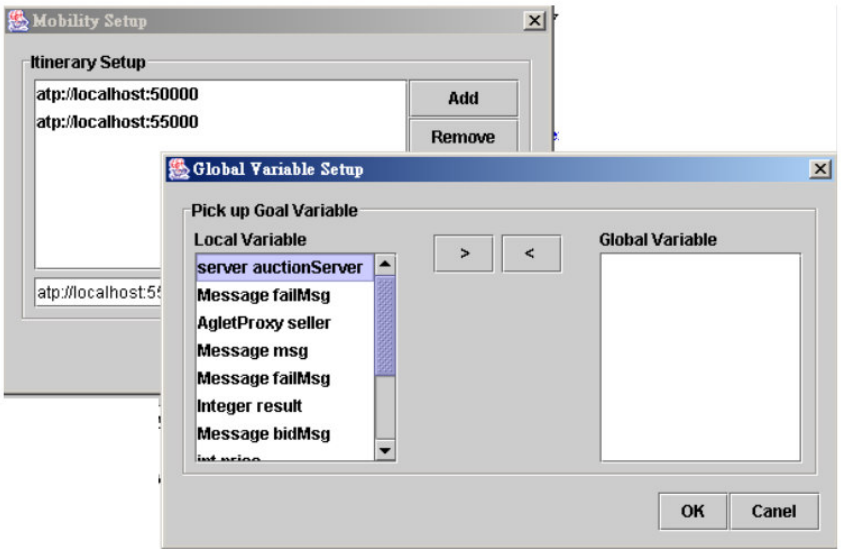

圖 5-6 全域變數的設定視窗

(ii) One Shot Pattern：若代理人要在執行主要邏輯之前主動遷移，則可套用此樣板，程式編輯器會詢問使用者要遷移的位址，並幫使用者在 onCreation 的最後加上遷移的程式碼，如此代理人便可以在執行完初始化動作之後、執行 run 函式的主要邏輯之前遷移到指定位址，如圖 5-7 所示。

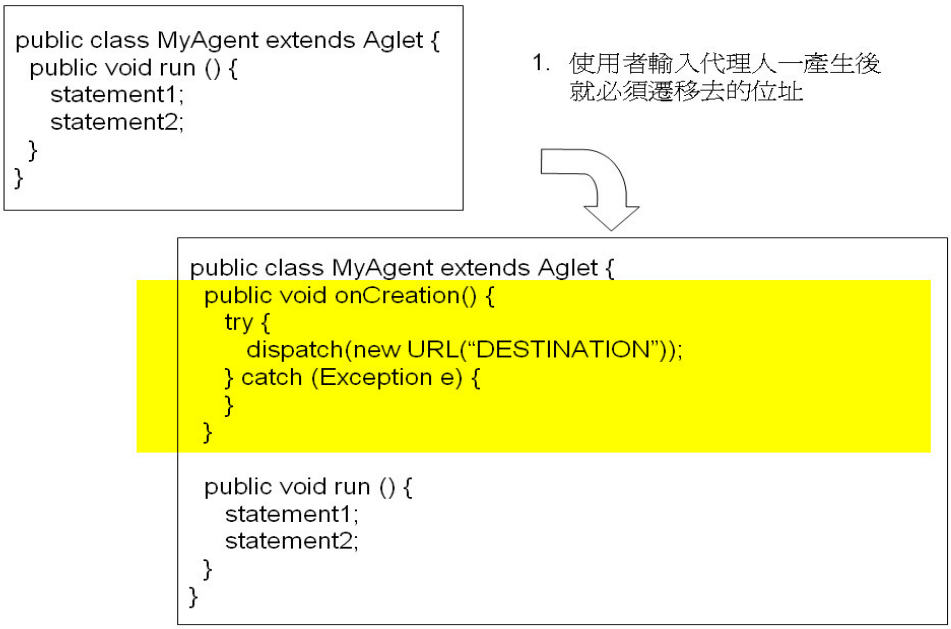

圖 5-7 套用 One Shot Pattern 的樣板程式碼

(iii) Dynamic Decision Pattern：若代理人是在執行主要邏輯時藉由運算而獲得下一個要遷移的主機位址，則此代理人的遷移模式我們歸類爲 Dynamic Decision Pattern。在此模式中，下一個要遷移的位址是由代理人於執行時期運算而得，完全由代理人的主要邏輯控制而無法預期，因此程式編輯器不提供此模式之樣板。

(3) 被動遷移樣板：主從關係中的父代理人可控制子代理人，並將子代理人遷移到指定的位址。父代理人控制子代理人遷移的兩種模式分別爲有兩種：

1 First Step Control Pattern：父代理人不希望子代理人在一開始產生的主機上執行主要邏輯，而是等父代理人將子代理人派出到指定位址後，子代理人才開始執行主要邏輯，此時子代理人可套用 First Step Control Pattern 的樣板，如圖 5-8 所示。

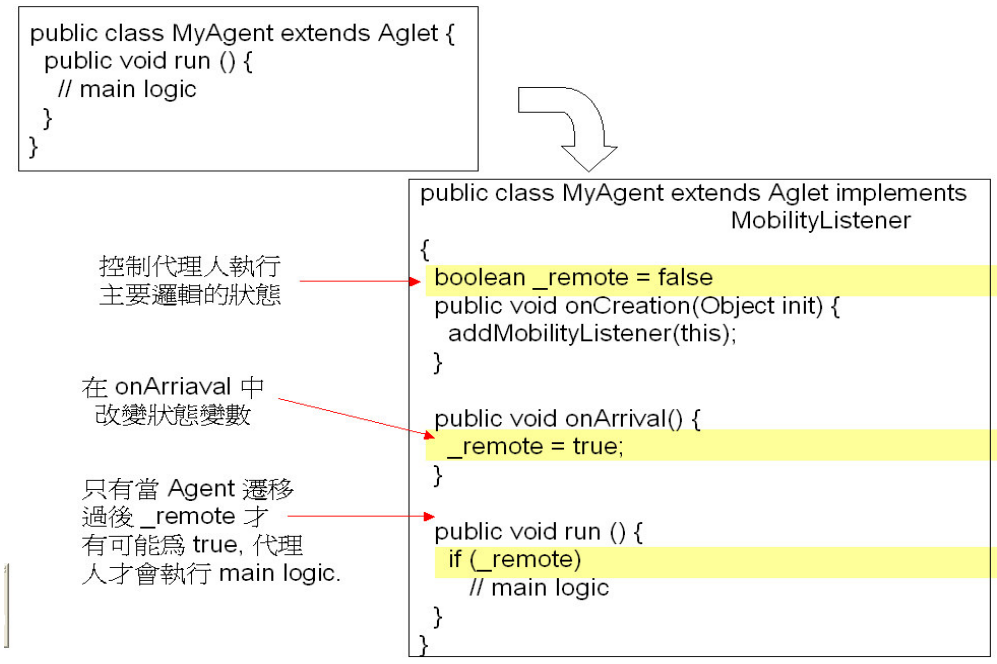

圖 5-8 套用 First Step Control Pattern 樣板程式碼

2 Step by Step Control Pattern：子代理人本身沒有遷移能力，而是由父代理人全權控制子代理人每次要遷移的地點，此時父代理人可套用 Step by Step Control Pattern 的樣板，如圖 5-9 所示。

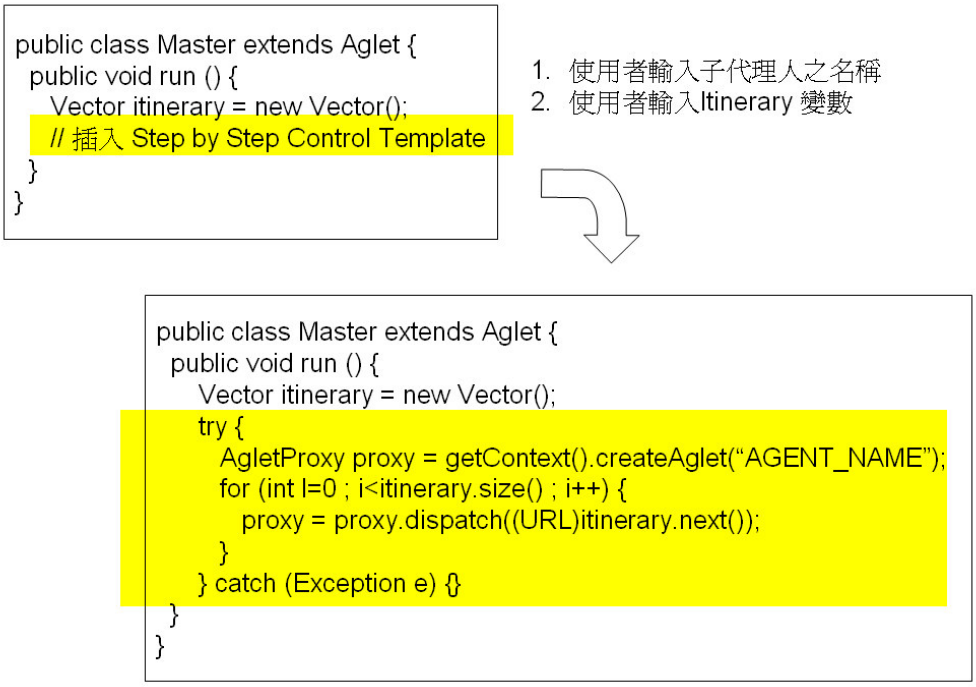

圖 5-9 Step by Step Control Pattern

15 當使用者撰寫完代理人之程式碼時，程式編輯器須支援程式碼編譯 (Compile) 之功能，並將編譯之結果顯示於訊息輸出視窗。

5.2 程式編輯器之系統架構

圖 5-10 爲程式編輯器之系統架構圖，共有三個主要模組，分別爲編輯器模組 (Editor Panel)、編譯器控制模組 (Compiler Controller)、任務規格面板模組 (Task Specification Panel)。以下對各個模組做詳細說明。

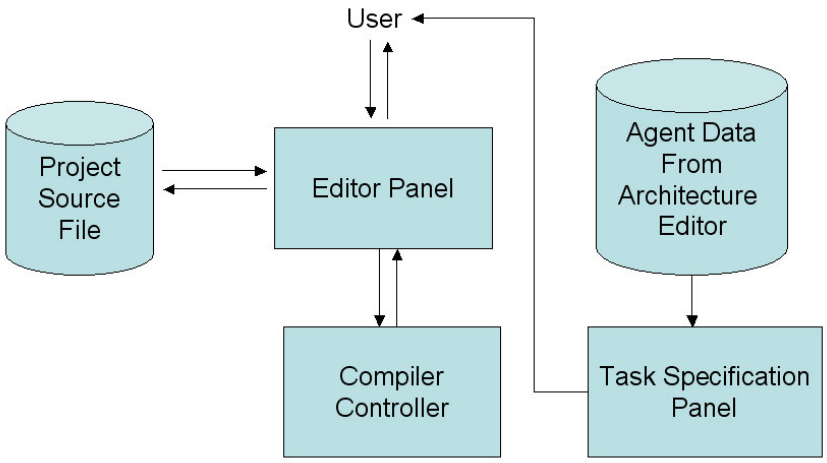

圖 5-10 程式編輯器之系統架構

1 編輯器模組 (Editor Panel)：此模組負責提供使用者文字編輯面板讓使用者針對設計與實作單一代理人類別，並依代理人系統架構圖之定義，預先幫代理人產生類別樣板程式碼，包括類別宣告，代理人方法宣告。當使用者撰寫程式碼時則提供此代理人類別所需之樣板程式碼讓使用者套用，以節省使用者時間。

2 編譯器控制模組 (Compiler Controller)：此模組負責連結代理人會使用到的外部程式套件，呼叫 Java 編譯器編譯代理人程式，並將產生的代理人類別檔統一安置於專案目錄之下。

3 任務規格面板 (Task Specification Panel)：此模組責責讀入系統架構圖的定義，將代理人的任務規格以及關係與特性設定讀出並顯示於面板上；其中代理人關係與特性的設定會轉為條列式需求規格，幫助使用者於撰寫程式碼時方便對照。

4 樣板程式碼規格檔：此資料檔包含於 “Project Source File” 中，其目的是以 XML 標籤來定義各代理人樣板包含的程式碼以及程式所需插入的位置。此規格檔中包含所有代理人可用之樣板程式模型，包括溝通樣板與行動力樣板。程式編輯器於啓動之前會讀入此檔，並將其中所記錄的樣板程式碼載入於 TemplateGenerator 類別中。樣板程式碼規格檔的格式爲：

```
<Template>
    <Name> 樣板名稱 </Name>
    <Insert>
        <Position> 程式碼插入位置 </Position>
        <code>
            程式碼 <args>
                        <Meaning> 變數的意義 </Meaning>
                        <Type> 變數的型態 </Type>
                    </args>
            程式碼; <br>
            程式碼; <br>
        </code>
    </Insert>
</Template>
```

5.3 資料結構

圖 5-11 爲程式編輯器的類別關係圖，下面將逐一介紹每個類別內部的資料結構。

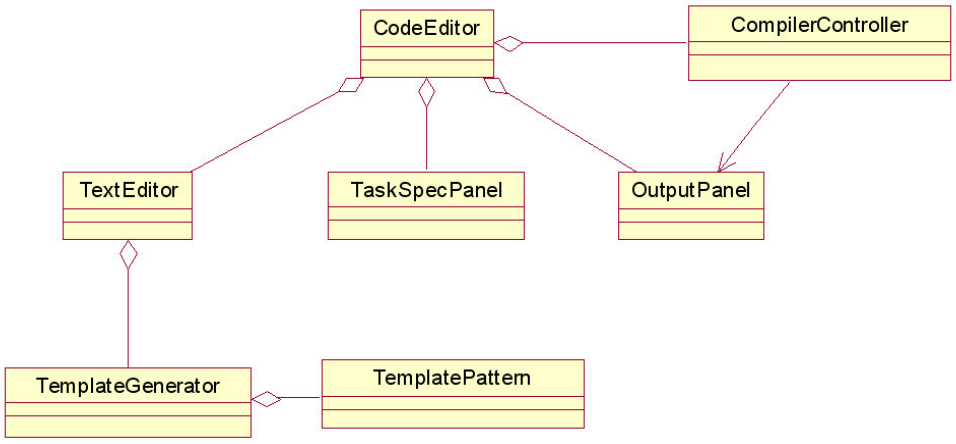

圖 5-11 程式編輯器內部之類別關係圖

(1) 類別 “CodeEditor”：此類別繼承自 java 的視窗元件，爲其任務面板，輸出面板，編輯面板的容器。此類別需記錄：

1 使用者目前編輯的代理人程式碼

2 使用者已開啓的代理人程式碼

(2) 類別 “TaskSpecPanel”：此類別負責讀入系統架構圖的定義，並將代理人之任務描述以及代理人的特性與關係的需求顯示於面板上。

(3) 類別 “TextEditor”：此類別提供使用者文字編輯介面，並依代理人系統架構中之定義幫使用者預先產生類別樣板程式碼，當使用者撰寫程式碼時，”TextEditor”會準備此代理人程式會用到的樣板程式碼以供套用。

(4) 類別 “OutPanel”：此類別負責呈現代理人程式的編譯訊息。

(5) 類別 “CompilerController”：此類別負責連結代理人程式會用到的所有其他類別檔案，呼叫編譯器編譯目前使用者編輯的代理人程式碼，並將編譯結果輸出於 “OutPanel”上。

(6) 類別 “TemplateGenerator”：此類別負責將使用者指定的樣板插入至目前使用者編輯的代理人程式碼中。

(7) 類別 “TemplatePattern”：此類別記錄一個特定樣板的模式，包括此樣板名稱、樣板需插入的位置、以及每個位置需插入之程式碼。

以下將各類別的資料結構詳列於下：

1 CodeEditor：

```
public class CodeEditor extends JPanel {
    Vector openedSource;              // 使用者已開啓的代理人程式碼
    TextEditor textEditor;            // 程式編輯工具
    TaskSpecPanel taskPanel;          /* 顯示代理人任務與特性需求列
                                         表的面板 */
    OutputPanel outputPanel;          // 顯示編譯結果之面板
    CompilerController compiler;      // 編譯器控制類別
}
```

2 TextEditor：

```
public class TextEditor extends JTextArea  {
    AgentData agentData;              // 目前編輯之代理人資料
    JPopUpMenu popUpMenu;             /* 編輯器需提供給此代理人的樣
                                         板選單， AgentData 的不同動
                                         態產生*/
    String tempCode;                  // 剪下或複製的程式片段
    Parser parser;                    /* 找出目前程式碼中之變數與
```

```
                                        代理人特殊函式之位置 */
    Vector variables;                   /* 記錄目前程式碼中之變數及其
                                        位置 */
    Vector functions;                   /* 記錄目前程式碼中代理人特殊
                                        函式之位置 */
    TemplateGenerator;                  // 幫助使用者產生樣板程式碼
}
```

3 TemplateGenerator：

```
public class TemplateGenerator {
    Hashtable hashTable;                /* 以樣板名稱對應其
                                        TemplatePattern 元件 */
}
```

4 TemplatePattern：

```
public class TemplatePattern {
    String patternName;                 // 此樣板之名稱
    Hashtable insertCode;               /* 以插入位置對應需插入之程式
                                        碼記錄 */
}
```

5 TaskSpecPanel：

```
public class TaskSpecPanel extends JPanel {
    AgentData agentData;                // 目前編輯的代理人資料
    JTextArea taskSpecArea;             // 顯示代理人任務描述的介面
    DefaultListModel constraintsListModel;    /* 目前編輯的代理人的特性
                                              需求，以 String 爲單位儲
                                              存 */
```

```
    JList constraintsList;                  // 顯示代理人特性列之介面
}
```

6 OutputPanel：

```
public class OutputPanel extends Jpanel {
    JTextArea output;                       // 顯示代理人程式編譯結果
}
```

7 CompilerController：

```
public class CompilerController {
    String compilerLocation;                // Java 編譯器所在位址
    Vector libs;                            /* 外部類別庫所在位址，以 String
                                               爲單位儲存 */
    String destinationLocation;             /* 編譯器產生之 .class 檔存放位
                                               址 */
}
```

第六章 佈局管理器之設計與實作

6.1 佈局管理器之功能需求

佈局管理器之目的在於幫助使用者執行代理人程式並驗証其結果。要執行代理人系統，使用者必須先將所需的代理人伺服器一一產生，再指定每個代理人伺服器上所要執行之代理人程式。代理人系統中各代理人雖可同時執行，但各代理人間的啓動執行是有順序性，例如要先執行接收端的代理人才能執行傳送端的代理人。佈局管理器之目的是提供使用者啓動多個代理人執行，並指定代理人間啓動之順序。爲了方便使用者不必每次執行都重新指定要產生之代理人伺服器與要產生之代理人及其順序，佈局管理器可將使用者執行的步驟儲存成一個劇本(Script File)，讓使用者下次執行時可以直接載入。

由於代理人程式必須在代理人伺服器上才能執行，且代理人程式之執行結果亦將顯示於代理人伺服器視窗，因此佈局管理器需支援使用者可在不同的位址上產上多個代理人伺服器，並將各代理人伺服器的輸出顯示於不同的面版上，以幫助使用者檢查代理人程式執行之結果。爲達成上述之功能，佈局管理器所需之功能需求如下：

1 支援使用者圖形化的介面來檢視並執行代理人系統，如圖 6-1 所示。

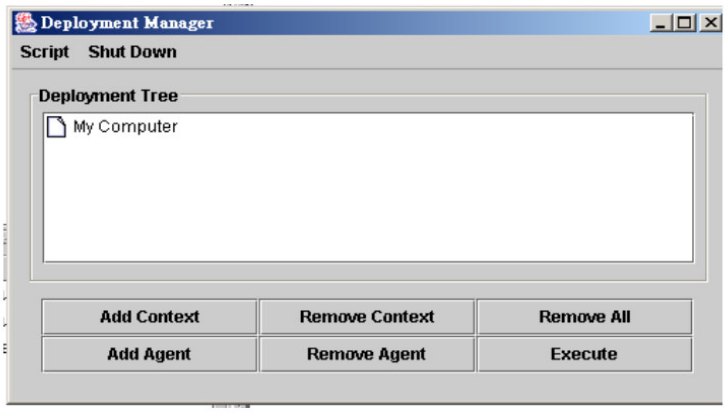

圖 6-1 佈局管理器介面

圖 6-1 的佈局檢視面版上顯示使用者目前設定的劇本，包括要產生的代理人伺服器與代理人物件。佈局管理器的工具列則提供使用者新增、移除代理人伺服器和代理人的功能。當使用者設定完執行劇本之後，可按下工具列的 “Execute” 按鈕要求佈局管理器依劇本開始執行。

2 支援使用者設定要執行之代理人伺服器。使用者可點選佈局管理器工具列中之 “Add Agent Context”按鈕，叫出代理人伺服器之設定視窗，如圖 6-2 所示，並設定伺服器之位址，佈局管理器若檢查出位址重複則顯示警告訊息。使用者設定完後，新的代理人伺服器會顯示於檢示面版中。

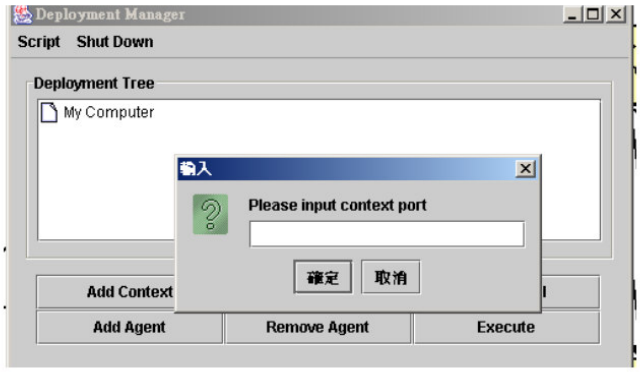

圖 6-2 產生代理人伺服器設定

3 支援使用者在已產生的代理人伺服器上執行指定的代理人程式。使用者先於檢視面版中選擇代理人伺服器，再點選佈局管理器工具列之 “Add Agent” 按鈕，叫出代理人物件產生視窗，如圖 6-3 所示。使用者須自專案目錄中指定要執行之代理人類別，選定後的代理人會在佈局的檢視面版中顯示，如圖 6-4 所示。

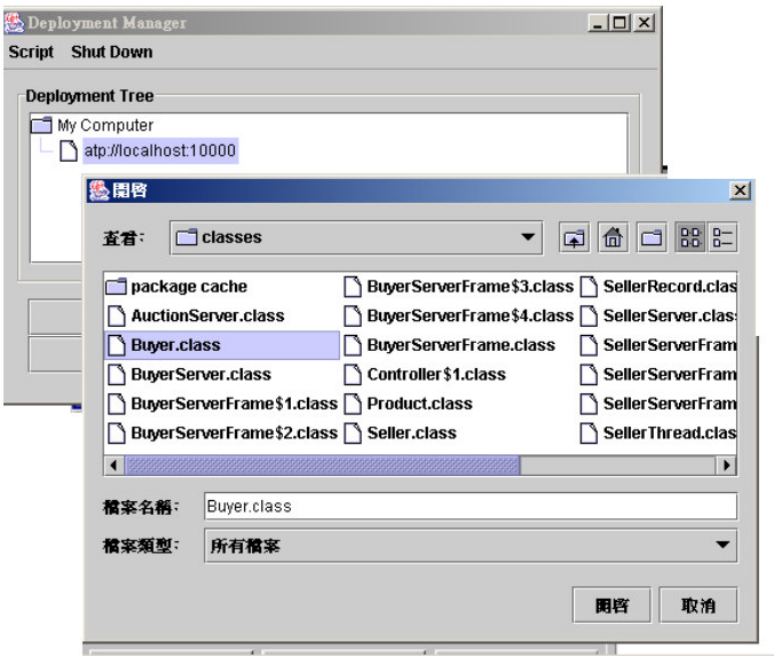

圖 6-3 代理人物件產生視窗

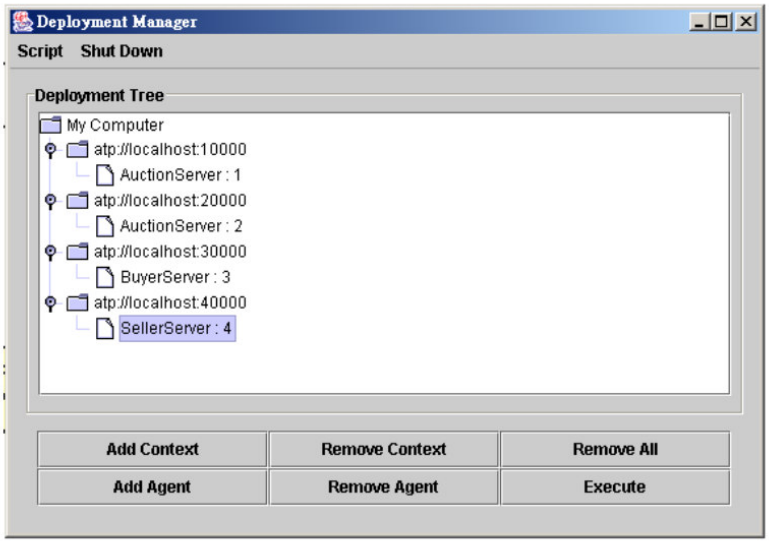

圖 6-4 代理人物件於佈局檢視面版中顯示

4 支援使用者依其劇本之需求在多個代理人伺服器上執行多個代理人物件，佈局管理器需支援使用者指定各代理人執行之順序，因此佈局管理器會記錄使用者新增代理人物件到佈局檢面版的順序，如圖 6-4 所示，

在實際執行時，佈局管理器便會依此順序產生代理人物件。

5 支援使用者儲存已安排好的劇本，方便使用者下次重新執行時可以直接載入劇本，而不用重頭設定代理人伺服器與代理人物件，如圖 6-5 所示。

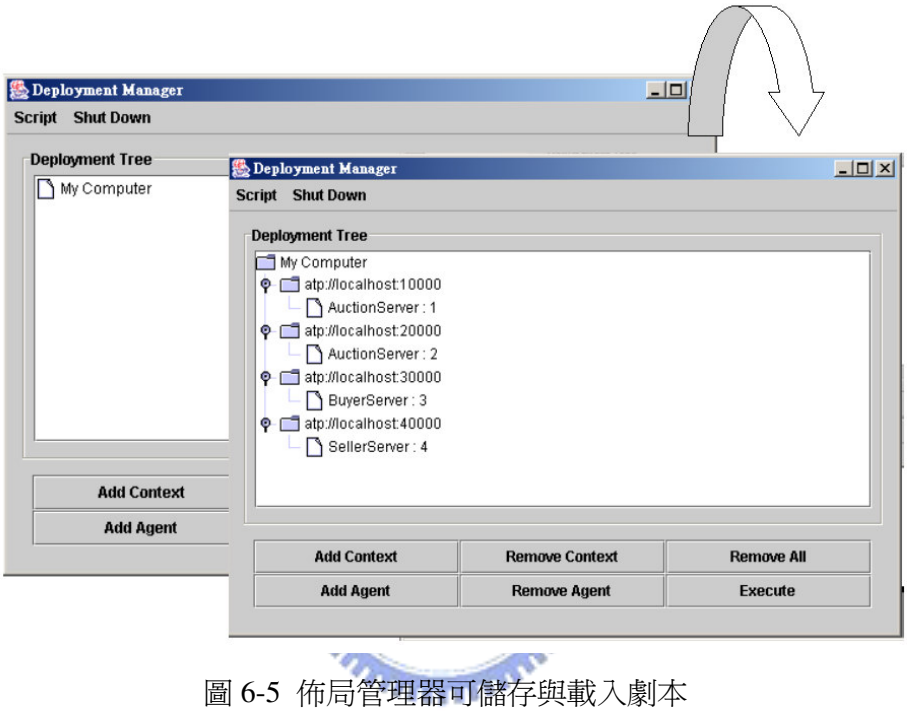

圖 6-5 佈局管理器可儲存與載入劇本

若佈局管理器載入劇本時發現劇本中所記載之檔案跟目前專案中的檔案不符合時，系統須顯示錯誤訊息。

6 幫助使用者做佈局的一致性檢查。當使用者在佈局檢視面版中規畫好此次要執行之代理人伺服器與代理人物件，並要求佈局管理器依照此佈局執行時，佈局管理器須將目前代理人物件分佈的情況跟系統架構圖做比對，找出目前的佈局中是否出現 “兩個需在同一台主機上建立供需關係的代理人” 被使用者佈局在不同代理人伺服器上，而兩個代理人又沒有行動力。若發現此種情形，則佈局管理器需顯示警告訊息。

7　每個代理人伺服器，都有獨立的結果顯示視窗，供使用者可以觀看各代理人於伺服器上執行之結果，如圖 6-6 所示。

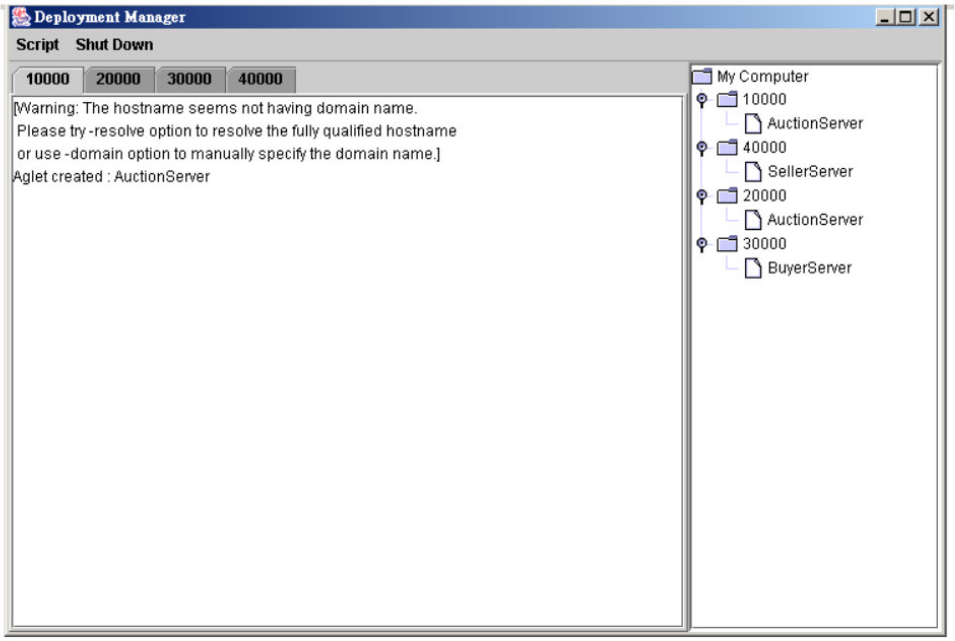

圖 6-6 佈局管理器執行使用者佈局之畫面

6.2 佈局管理器之系統架構

圖 6-7 顯示佈局管理器之系統架構圖，佈局管理器主要的兩個模組分別爲“Deployment Manager Panel”和“Agent Server”，其功能說明於下。

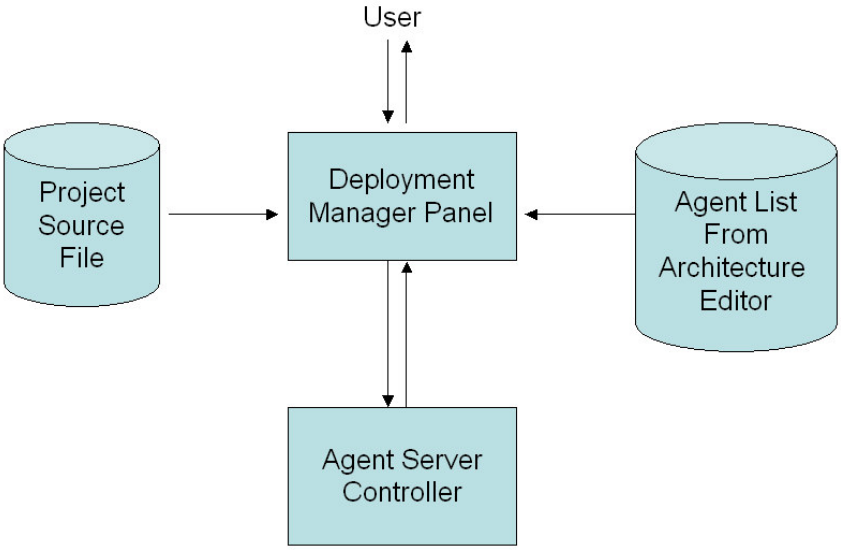

圖 6-7 佈局管理器之系統架構圖

1 Deployment Manager Panel：此模組負責提供使用者操作的介面，支援使用者以樹狀階層的方式規劃代理人系統執行時期之佈局，每個樹狀節點表示一個代理人伺服器，而代理人伺服器的子節點則表示要在此伺服器上執行之代理人。使用者要執行代理人程式之前，Deployment Manager Panel 將自動依系統架構圖的設定來比對佈局的正確性，若發生兩個只能在同一伺服器建立供需關係的代理人類別被使用者安置在不同的代理人伺服上，則 Deployment Manager Panel 會顯示警告訊息。Deployment Manager Panel 檢驗使用者佈局設定和系統架構圖一致性無誤後，便負責依使用者的佈局設定產生代理人伺服器與代理人物件，並將各個代理人伺服器之輸出顯示於面版上，供使用者檢驗程式執行之結果。

2 Agent Server Controller：此模組負責接受來自 Deployment Manager Panel 的指令操控代理人伺服器物件，包括 “產生代理人伺服器”、”在指定之代理人伺服器上產生代理人物件”、”終止並移除代理人伺服器”，Agent Server Controller 另外需負責將各代理人伺服器之執行結果轉到 Deployment Manager Panel 上，以方便使用者檢視執行結果。

6.3 資料結構

圖 6-8 爲佈局管理器的類別關係圖，下面將逐一介紹每個類別內部的資料結構。

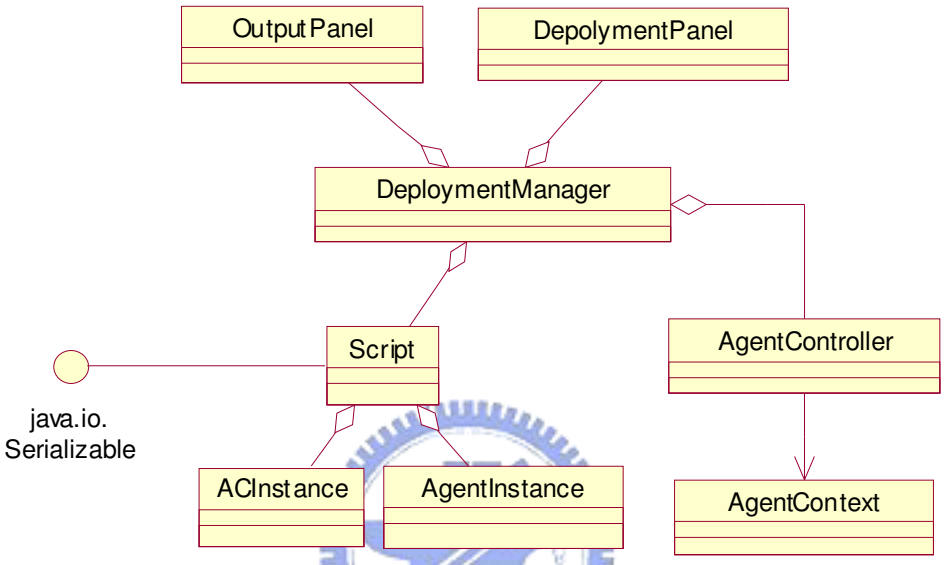

圖 6-8 佈局管理器內部之資料結構

(1) 類別 “ACInstance”：Agent Context Instance，記錄使用者佈局中的代理人伺服器資訊，包括：

1 代理人伺服器之名稱

2 代理人伺服器之位址

3 此代理人伺服器上所需執行之代理人物件

(2) 類別 “AgentInstance”：記錄使用者佈局中的代理人資訊，包括：

1 代理人類別

2 此代理人物件之順序號碼

(3) 類別 “Script”：利用 ACInstance 和 AgentInstance 兩個類別來儲存使用

者執行代理人程式之劇本。當使用者開始設定下一輪代理人系統執行時的佈局時，佈局管理器會自動產 “Script” 物件，用以記錄目前使用者在佈局檢視面版上安排之執行劇本，包括：

1 要執行的代理人伺服器及其位址

2 各代理人伺服器上要執行之代理人

3 代理人物件產生之順序

(4) 類別 “AgentController”：此類別依使用者設定之劇本來執行代理人程式，AgentController 會先啓動劇本中所有會用到的代理人伺服器於指定的位址，再依代理人物件產生順序逐一產生代理人。

(5) 類別 ”DeploymentManager”：此類別繼承自 Java 的視窗類別，屬於 UI 類別的容器類別，負責呈現其他的介面。

(6) 類別 “DeploymentPanel”：此類別提供使用者介面以設定每一輪要執行的代理人伺服器與代理人物件，並將佈局的資料儲存於 “Script” 物件中。

(7) 類別 “OutputPanel”：此類別負責呈現 Agent Context 的執行結果。

以下將各類別之資料結構詳列於下：

1 ACInstance：

```
public class ACInstance {
    String name;              //代理人伺服器之名稱
    int portNamber;           //代理人伺服器要啓動之埠號
    Vector agentList;         //於此代理人伺服器上執行之代理人物件
}
```

2 AgentInstance：

```
public class AgentInstance {
    String classType;          //此代理人物件之類別型態
    int sequenceNumber;        //此代理人被產生之順序
}
```

3 Script：

```
public class Script {
    Vector acList;             //此劇本中要執行之代理人伺服器
    Vector agentList;          //依執行順序儲存代理人物件
}
```

4 AgentController：

```
public class AgentController {
    Script script;             //AgentController 負責之劇本
}
```

第七章 使用範例

本章將以一個 ”簡化的競標系統” 當做實際範例，說明如何透過 AgentIDE 的系統架構編輯器來分析與設計代理人系統之架構、利用程式編輯器設計與實作代理人類別，並利用佈局理管理器驗証與執行代理人程式。

7.1 實例 – 簡化的網路競標系統

此網路競標系統主要是提供網路競標的功能，讓買方和賣方可以透過第三方的仲介而得以交換訊息。此系統之功能需求如下：

1. 競標系統需具備伺服端、買方、與賣方。
2. 買方的目標爲幫助使用者在價格上限內購得指定物品。
3. 賣方的目標爲幫助使用者在價格下限以上拍賣指定物品。
4. 競標系統伺服端可接收來自買方與賣方的登記參與競標。
5. 競標系統伺服端可幫助買方尋找適合的賣方，若找不到適合的賣方則通知買方；若找到適合的賣方則將賣方的資料交予買方，讓買賣雙方自行交易。
6. 買方和賣方交易期間，仍允許其他買方加入。
7. 買方和賣方交易期間，賣方將目前的競標價格告知所有買方，買方可依此價格決定繼續喊價或是放棄此交易，若沒有買方喊出更高價，則由上一輪喊出目前競標價格的代理人得標。
8. 買方或賣方完成交易後需回報使用者。

7.2 操作實例

使用者開啓 AgentIDE 後預設會出現系統架構編輯器視窗，使用者可利用此視窗上的 “File” 選單選擇開啓新專案，如圖 7-1 所示。

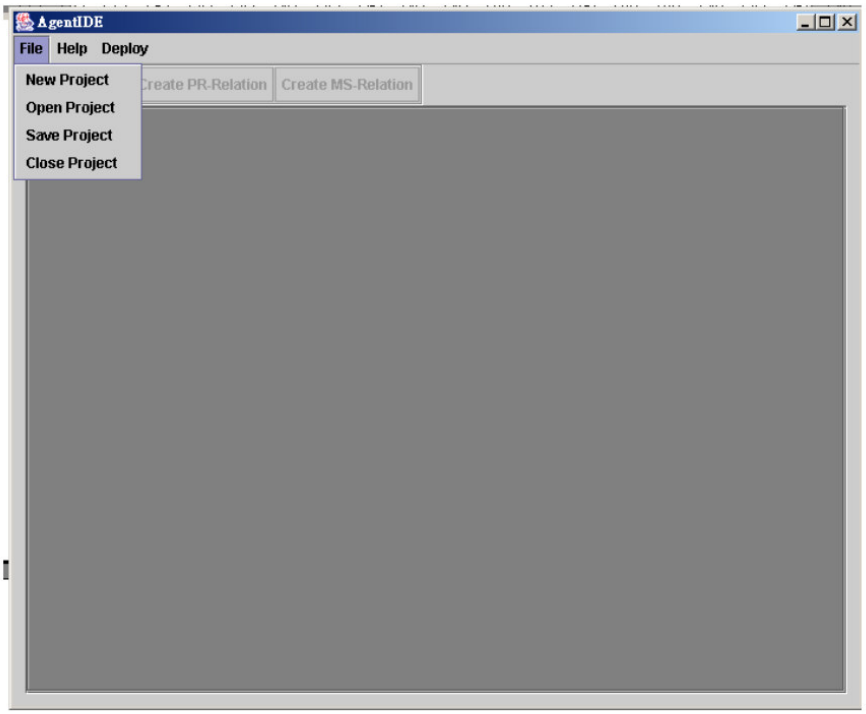

圖 7-1 利用選單選擇新專案

使用者產生新專案時可利用專案設定視窗以設定此專案資料夾之名稱，專案資料夾所在之位置，如圖 7-2 所示。

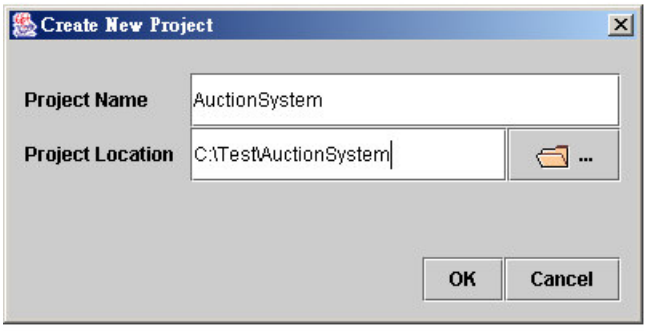

圖 7-2 專案設定視窗

當使用者設定完成專案後，便可利用系統架構編輯器來分析與定義代理人系

統之架構。由上述競標系統的需求來看，可以將競標系統分爲三個主要子系統，分別爲買方子系統、賣方子系統、及伺服端子系統。因此使用者可產生三個代理人：BuyerServer、SellerServer、AuctionServer，分別表示買方代理人、賣方代理人、與伺服端代理人，並設定各代理人之目標，圖 7-3 爲使用者產生 BuyerServer 代理人時的設定。

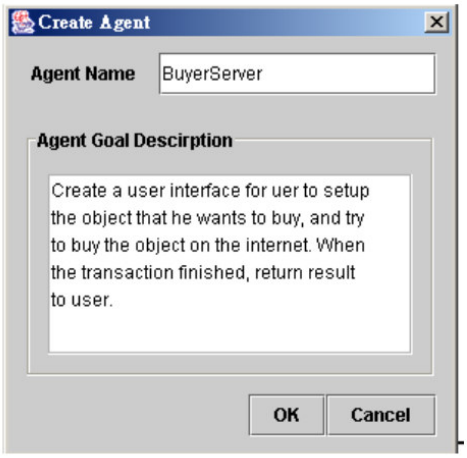

圖 7-3 產生代理人視窗

使用者可繼續分析各代理人之任務，並利用代理人任務設定視窗將任務之描述記錄下來，圖 7-4 爲協助使用者記錄 BuyerServer 代理人任務的視窗。

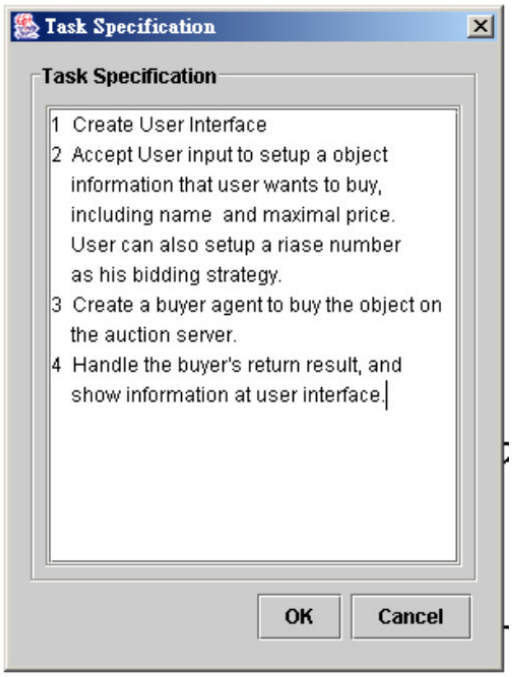

圖 7-4 定義代理人任務視窗

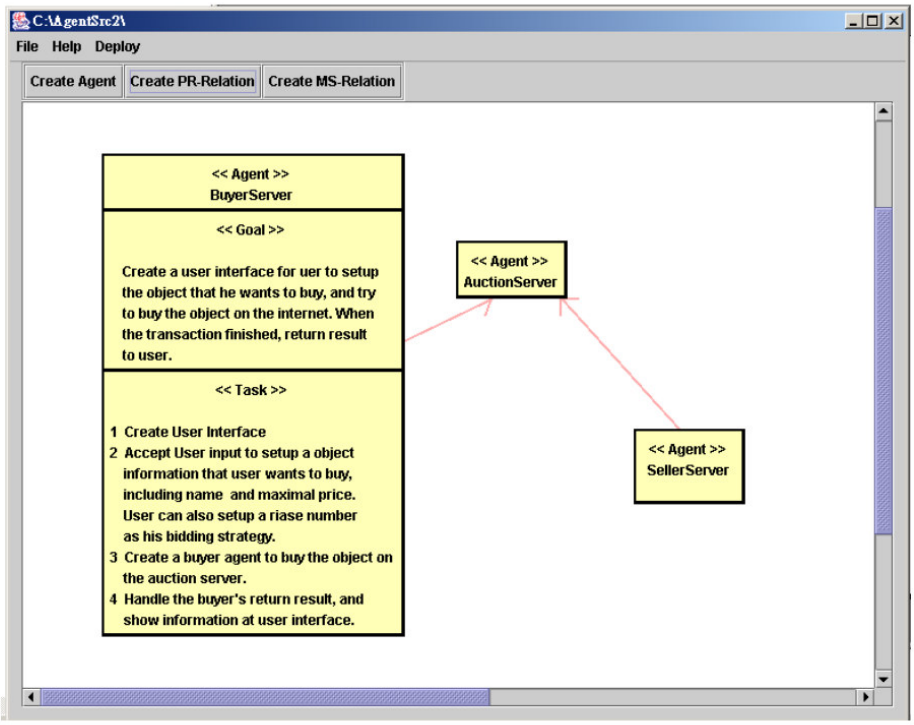

圖 7-5 展開後的代理人圖示

當代理人之任務定義完成後，使用者便可以根據代理人之任務決定代理人之間的供需關係，在此例中 SellerServer 須向 AuctionServer 登記，而 BuyerServer 則會詢問 AuctionServer 是否有適合的賣方存在，若有，則 BuyerServer 便可透過 AuctionServer 的仲介和 SellerServer 溝通。使用者可按下系統架構編輯器上的 “供需關係” 按鈕，叫出供需關係設定視窗來設定關係兩端之代理人，如圖 7-6 所示。

圖 7-6 定義供需關係

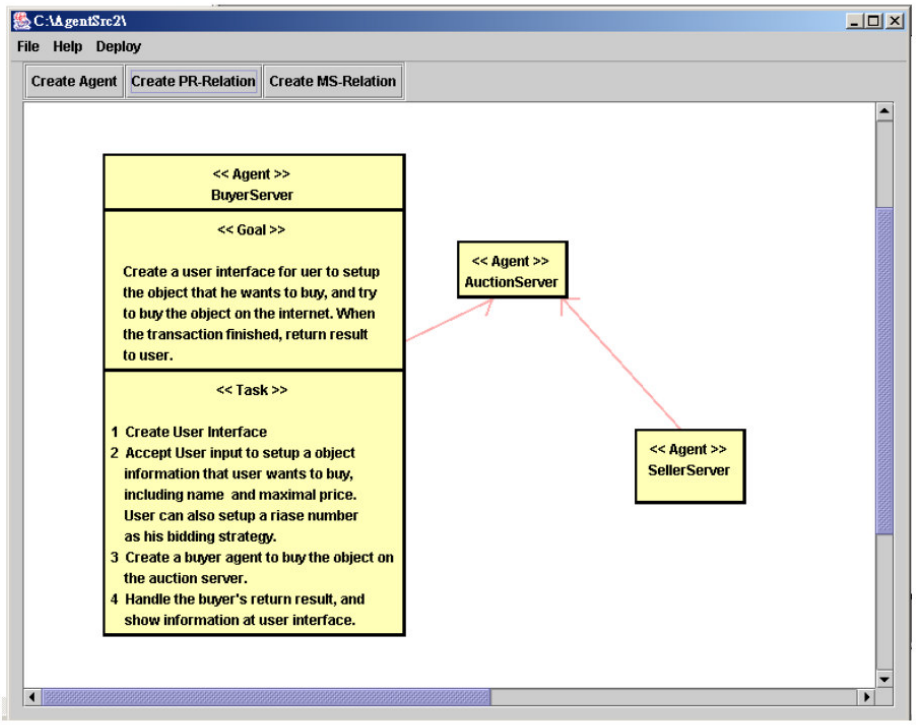

圖 7-7 定義 BuyerServer、SellerServer、和 AuctionServer 之間的關係

若代理人任務的複雜度過高，使用者可產生子代理人來分擔父代理人之工作。在此例中，BuyerServer 和 SellerServer 將實際在網路上買賣的工作交由其下的子代理人來做。使用者可利用主從關係來連結父代理人與子代理人。如圖 7-8，7-9 所示。

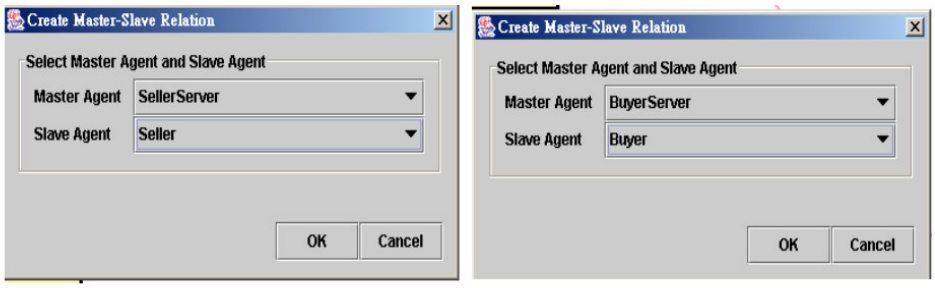

圖 7-8 爲 BuyerServer 和 SellerServer 連結其子代理人

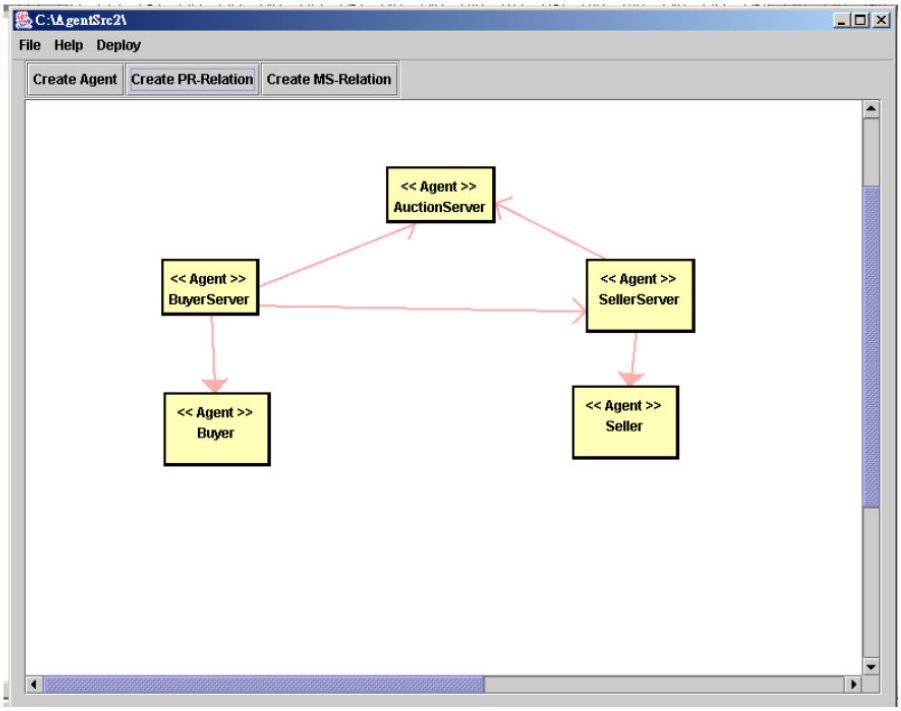

圖 7-9 客戶端代理人產生兩個子代理人

在產生子代理人之後，使用者必須重新檢視原代理人對外的關係連線，若發覺原本屬於客戶端代理人需負責的供需關係已轉由其子代理人負責時，使用者必須跟著修改關係連結。在此例中，BuyerServer 和 AuctionServer 間的溝通動作已交由 Buyer 執行；而 SellerServer 和 AuctionServer 間的溝通動作也改由 Seller 執行，如圖 7-10 所示。

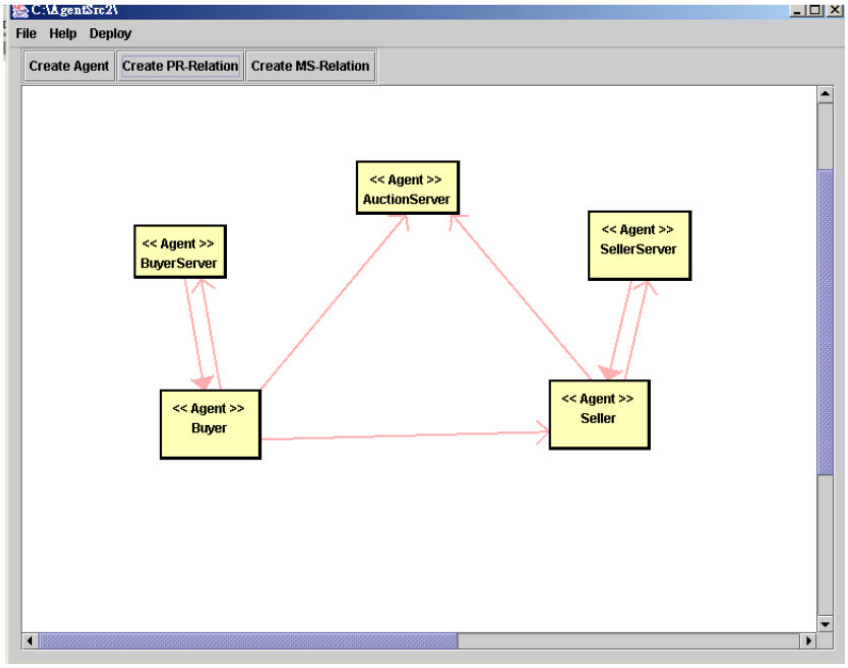

圖 7-10 更新關係後的系統架構圖

當整個基本的系統架構定義完之後，使用者便可開始設定代理人的關係模式。每一條設定好模式的關係連線會以不同顏色表示，藉以提醒使用者。其中，Seller 和 AuctionServer 的供需關係利用 “關鍵鍵字查詢” 方式建立，且通訊過程中需包括註冊之動作，其設定視窗如圖 7-11 所示。圖 7-12、7-13 分別表示 Buyer 和 AuctionServer 的供需關係設定視窗，以及 BuyerAgent 到 SellerAgent 的供需關係設定視窗。

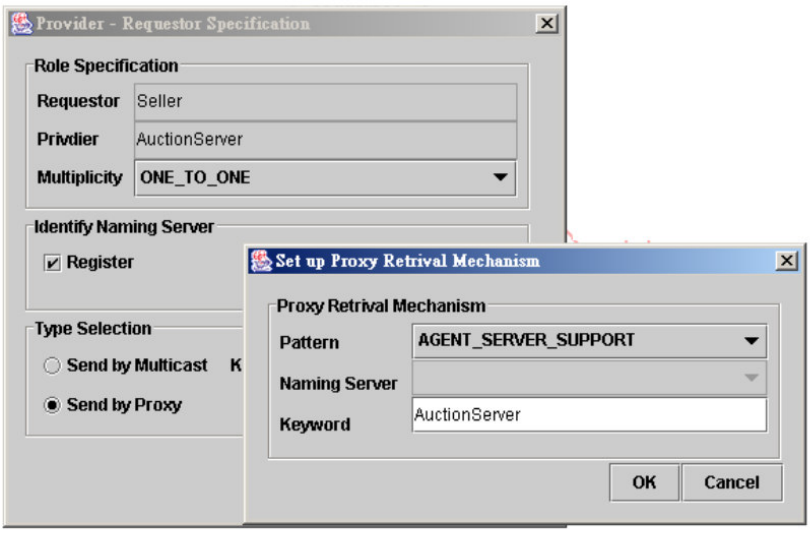

圖 7-11 SellerAgent 到伺服端代理人的供需關係設定視窗

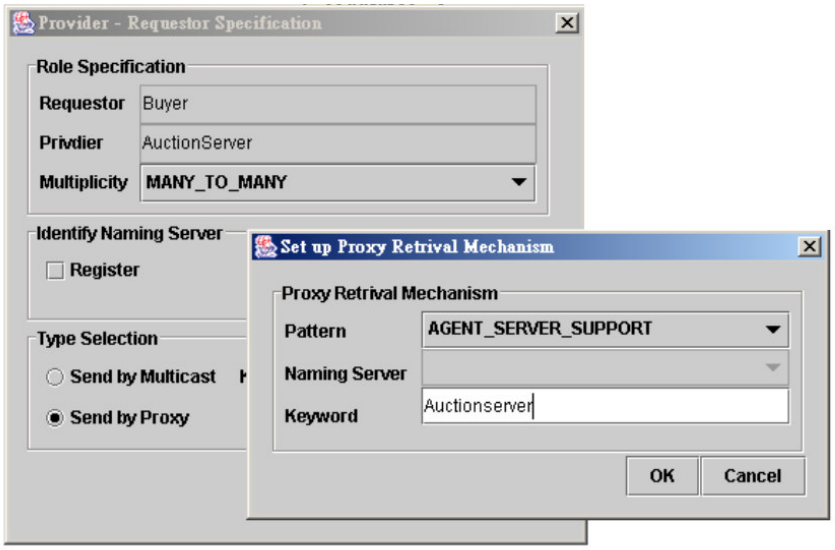

圖 7-12 Buyer 到 AuctionServer 的供需關係設定視窗

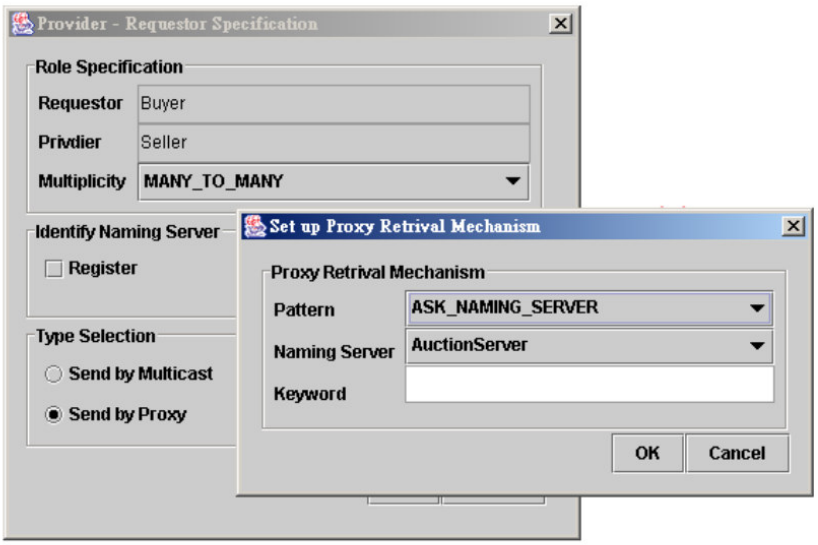

圖 7-13 Buyer 到 Seller 的供需關係設定視窗

當關係模式設定完成之後，使用者需設定各代理人的特性。在本例中，Buyer 和 Seller 均俱有遷移能力，可至各個主機上尋找 AuctionServer。代理人行動力設定視窗如圖 7-14 所示。

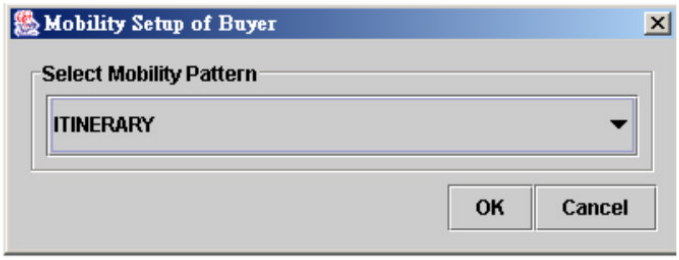

圖 7-14 代理人行動力設定視窗

當整個競標系統架構定義完成後，使用者可用滑鼠點選代理人以進入程式編輯器，程式編輯器會將系統架構中之資料載入，並預先幫使用者產生代理人類別程式樣板，如圖 7-15 所示。在使用者撰寫程式碼的過程中則可套用代理人特性樣板程式碼，程式碼會插入於使用者游標處，如圖 7-16 所示。

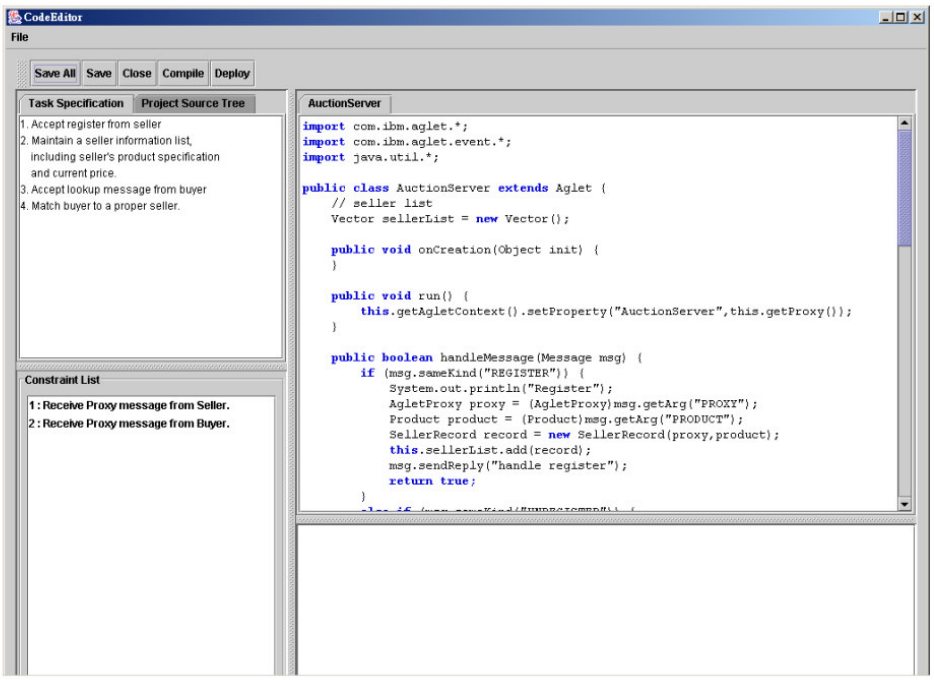

圖 7-15 以程式編輯器設計與實作 Buyer

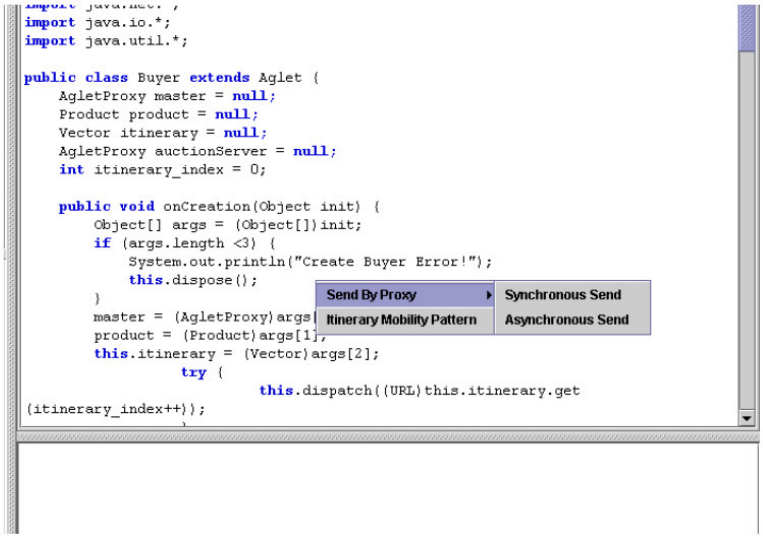

圖 7-16 撰寫程式碼時套用代理人特性的樣板程式碼

爲了執行與驗証代理人程式，使用者可利用程式編輯器工具列中的“執行”按鈕，以叫出佈局管理器，如圖 7-17 所示。

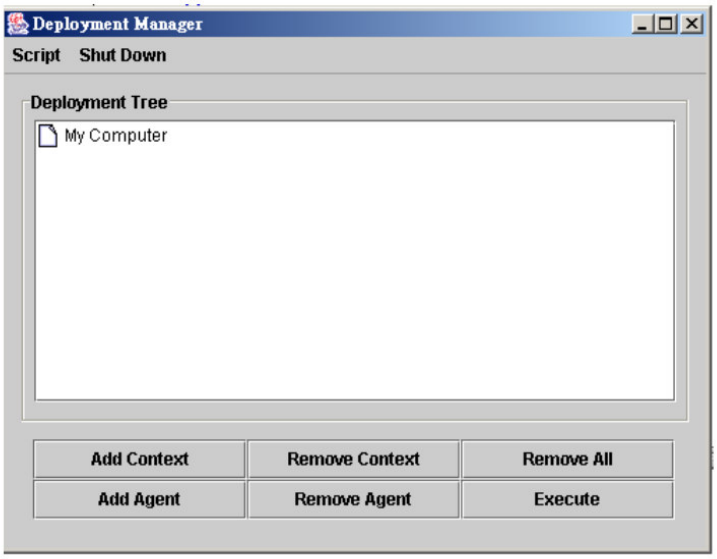

圖 7-17 佈局管理器

使用者,可利用佈局管理器一次啓動執行多個代理人程式，並指定各個代理人啓動之順序，如圖 7-18 所示。一旦使用者設定好劇本便可指示佈局管理器開始執行，佈局管理器會將使用者產生的所有的代理人伺服器的輸出分別顯示於不同面板供使用者檢視。如圖 7-19 所示。

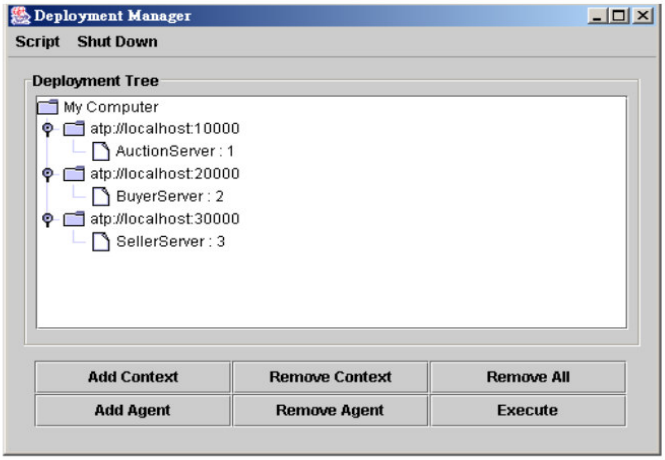

圖 7-18 佈局管理器的劇本設定

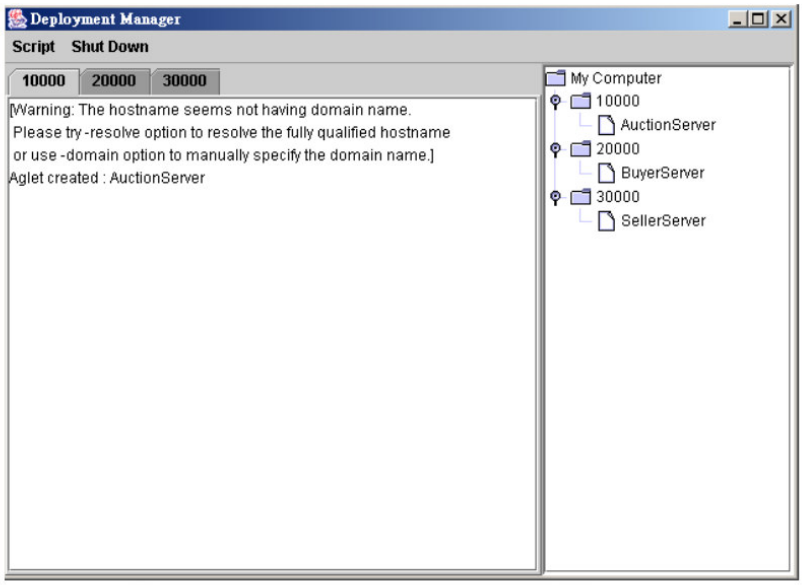

圖 7-19 執行佈局

在此例中，BuyerServer 和 SellerServer 會產生讓使用者設定買賣的介面，使用者設定完成之後，BuyerServer 和 SellerServer 會分別產生 Buyer 和 Seller 代理人到使用者指定的網路位址上尋找 AuctionServer 並進行交易。圖 7-20 爲 BuyerServer 執行時自動產生之介面。

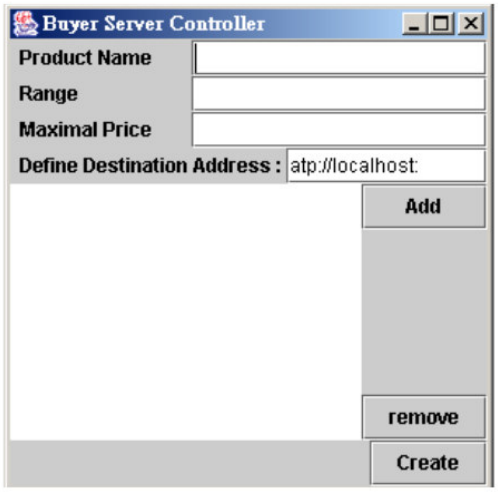

圖 7-20 BuyerServer 產生的設定介面

當使用者設定完要買的產品之後，按下 “OK” 鈕，則 BuyerServer 便會產生 Buyer 代理人，使用者可檢視 BuyerServer 所在的伺服器面版，確認是否有產生代理人的訊息，如圖 7-21。

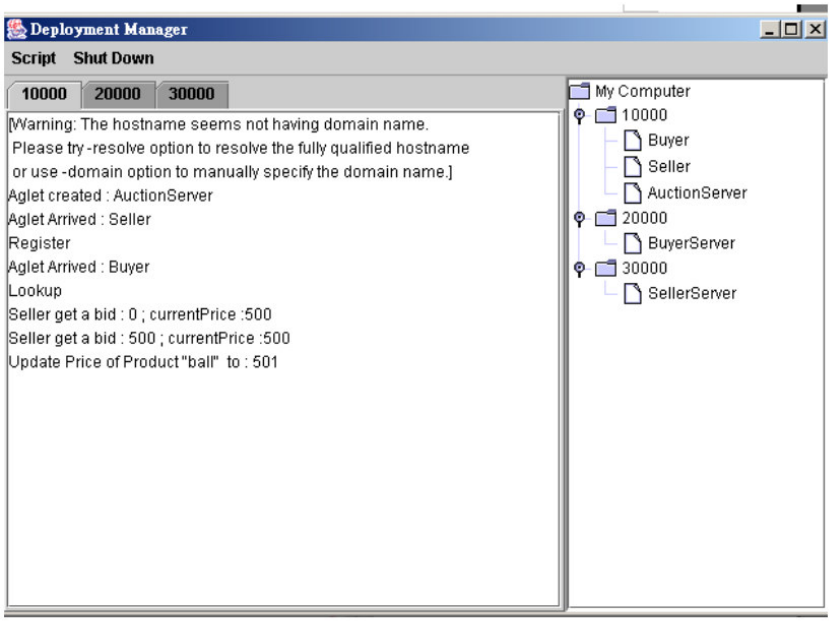

圖 7-21 Buyer 產生後遷移到 Context1 上尋找 AuctionServer

在此例中，Buyer 和 Seller 雙方可交易完成，並各自回報其主代理人，BuyerServer 和 SellerServer 會各自將交易成功的訊息印出，如圖 7-22 所示。

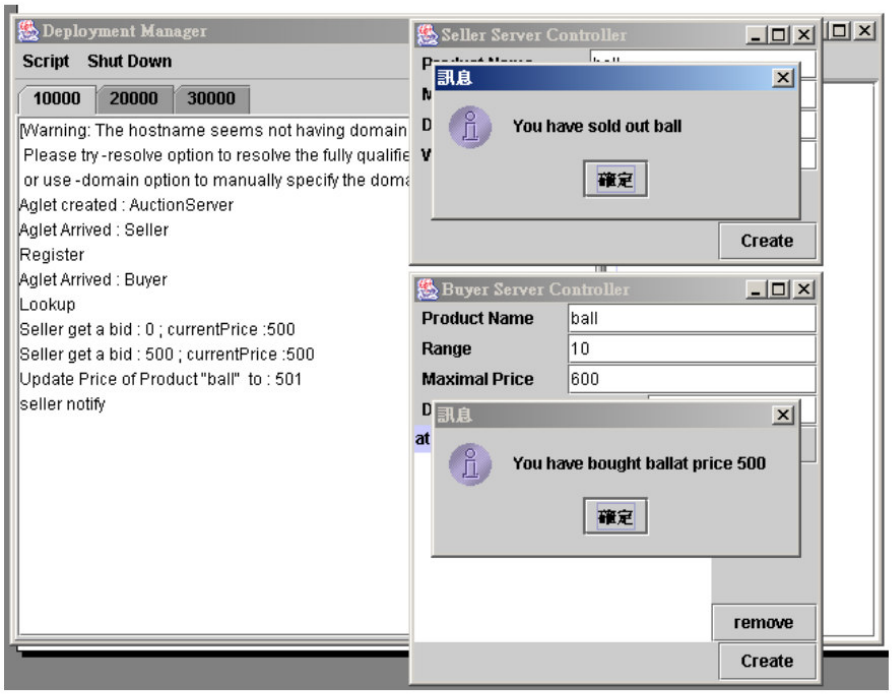

圖 7-22 交易完成 BuyerServer 和 SellerServer 所顯的訊息

第八章 結論

軟體代理人相較於物件導向程式中的物件是更高層階的概念，除了原本物件導向資料封裝的特性之外，軟體代理人還被賦與自主性、合作能力、行動力、與學習能力這四種特性，使得代理人更貼近人類的思維模式，因此開發人員可以很自然地利用軟體代理人來塑模 (Model) 多執行緒的大型分散式系統。

現今軟體代理人程式的使用仍不夠普遍，原因除了代理人各項特性的標準尚未確定之外 [12][14][15]，現今輔助使用者開發軟體代理人程式的開發工具相當貧脊也是主因之一。目前雖有些廠商開始投入代理人開發工具的設計，但都著重於代理人基礎類別的開發，缺乏一套能引導使用者漸進式地開發代理人系統的輔助工具。因此，本論文提出一套軟體代理人系統之開發工具，讓使用者以圖形化方式定義出軟體代理人系統架構圖，在使用者設計與實作系統中各代理人程式，則依據系統架構圖提供使用者所需之樣板程式碼，以節省軟體開發的時間；並支援使用者定義劇本的佈局工具，方便使用者執行與驗証代理人程式。

本研究之成果包含：

(1) 設計一套圖形化代理人系統架構編輯工具以支援設計師定義系統架構與目標，並對代理人的角色與互動方式做安排，讓設計師能對整個系統架構更清楚。

(2) 明確定義出代理人特性的使用模式供設計師套用，並利用工具檢查套用模式間的一致性以減少使用者發生設計上的錯誤。

(3) 設計一套程式編輯工具支援使用者常見的樣板程式碼套用，以增進開發效率。

(4) 在執行驗証階段提供佈局規畫介面，方便使用者設定各代理人啓動執行之劇本並檢視執行結果。

未來的研究方向爲

(1) 本論文所設計之工具可協助使用者定義代理人系統架構圖，但關於各代理人類別內部的設計，以及從代理人類別分析出的設計類別 (Design Class) 則尚未提供圖形化的塑模工具，未來應整合傳統 UML [14] 的塑模表示法，以提升本工具的實用性 。

(2) 除了分析與建構代理人系統之外，另一項迫切需要的工具爲視覺化除錯工具 (Visualising Agent Behavior Debug Tool) [16]，幫助使用者檢視代理人的行爲，包括程式內部的資料狀態、訊息傳送、以及遷移動作。

參考文獻

[1] Danny B.Lange,Mitsuru Oshima, Programming and Deploying Java Mobile Agents with aglets,1995

[2] M. Wooldridge and P.Ciancarini. Agent-Oriented Software Engineering: The State of the Art In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*. Springer-Verlag Lecture Notes in AI Volume 1957, January 2001.

[3] Carlos Lucena, José Alberto Sardinha, Alessandro Garcia, Jaelson Castro, Alexander Romanovsky, Paulo Alencar, Donald Cowan
, Software engineering for large-scale multi-agent system, Proceedings of the 24th international conference on Software engineering, May 2003

[4] Paolo Ciancarini, Michael Wooldridge, Agent-Oriented Software Engineering, Proceedings of the 22nd international conference on Software engineering, June 2002

[5] Philippe Massonet, Yves Deville, Cédric Nève, Agent oriented software engineering：From AOSE methodology to agent implementation.

[6] Nicholas R. Jennings, An agent-based approach for building complex software systems.

[7] Gruia-Catalin Roman, Gian Pietro Picco, Amy L. Murphy, Software engineering for mobility：a roadmap, Proceedings of the conference on The future of Software engineering, May 2000

[8] AgentBuilder:An Integrated Toolkit for Constructing Intelligent Software Agents User’ s Guide, Version 1.3 Rev.0, April 30, 2000

[9] AgentBuilder: An Integrated Toolkit for Constructing Intelligent Software Agents Reference Mannel, Version 1.3 Rev0, Aprial 11, 2000

[10] Chella, A., Cossentino, M., Infantino, I., and Pirrone, R. An agent based design process for cognitive architectures in robotics in proc. Modena, Italy, Sept. 2001

[11] Antón, A.I., McCracken, W.M., and Potts, C. Goal Decomposition and Scenario Analysis in Business Process Reengineering in proc. Of Advanced Information System Engineering: 6th International Conference, CAiSE '94 (Utrecht, The Netherlands, June 1994) 94-

[12] M. Cossentino, C. Potts - "A CASE tool supported methodology for the design of multi-agent systems" - The 2002 International Conference on Software Engineering Research and Practice, June 24-27 2002 – Las Vegas,USA

[13] P. Burrafato, M. Cossentino - "Designing a multi-agent solution for a bookstore with the PASSI methodology" - Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems,May 2002

[14] AUML http://www.auml.org

[15] Bernhard Bauer,” UML Class Diagrams: Revisited in the Context of Agent-Based Systems” ,Proc of Agent-Oriented Software Engineering (AOSE) 2001

[16] Divine T. Ndumu, Hyacinth S. Nwana, Lyndon C. Lee, Jaron C. Collis, “Visualising and debugging distributed multi-agent system”, Proceedings of the third annual conference on Autonomous Agents**,** April 1999

[17] 許嘉容, “利用行動代理人的互助式學習之網路管理系統”, 國立交通大學資訊工程研究所, 碩士論文, 2001

[18] Objectspace Voyager, http://www.recursionsw.com/voyager.htm

[19] Aglets Open Source Project, http://sourceforge.net/projects/aglets/

[20] Emerson F. A. Lima, Patrícia D. L. Machado, Fláavio R. Sampaio, Jorge C. A. Figueiredo, Article abstracts with full text online: An approach to modelling and applying mobile agent design patterns, ACM SIGSOFT Software Engineering Notes**,** May 2004

[21] Marthie Schoeman, Elsabé Cloete, Architectural components for the efficient design of mobile agent systems, **P**roceedings of the 2003 annual research conference of the

South African institute of computer scientists and information technologists on Enablement through technology,2003

[22] The Source for Developer of Java, http://java.sun.com