# 行政院國家科學委員會專題研究計畫　期中進度報告

## 用行動代理人設計與實作具可存活性之感測器網路(1/3)

計畫類別： 個別型計畫
計畫編號： NSC94-2213-E-009-118-
執行期間： 94 年 08 月 01 日至 95 年 07 月 31 日
執行單位： 國立交通大學資訊工程學系(所)

計畫主持人： 葉義雄

報告類型： 精簡報告

處理方式： 本計畫可公開查詢

中　華　民　國 95 年 6 月 14 日

行政院國家科學委員會補助專題研究計畫 □ 成 果 報 告
☑期中進度報告

# 用行動代理人設計與實作具可存活性之感測器網路(1/3)

計畫類別：☑ 個別型計畫　　□ 整合型計畫
計畫編號：NSC－94－2213 －E －009－118
執行期間：　94 年 08 月 01 日至　95 年 07 月 31 日

計畫主持人：葉義雄
共同主持人：林祝興
計畫參與人員： 李鎮宇、李雅婷、張長蓉、朱伯昕

成果報告類型(依經費核定清單規定繳交)：☑精簡報告　□完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
□出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立交通大學資訊工程學系
東海大學資訊工程與科學系
中　華　民　國　95 年　06 月　1 日

# 目錄

# 圖目錄

# 一、 摘要

## 1. 中文摘要

隨著科技的突飛猛進，現今在感應器、網路和計算領域的發展使我們可部署大量既便宜且小的感應器，來達成許多複雜應用所需。尤其以 Distributed Sensor Network (DSNs)技術不斷的被改進再改進、價錢不斷的降低而使得在軍事、消防、醫療、生態研究等項目上被廣泛的利用。

然而，傳統的架構中，所有的 DSNs 感應器資料是集中送到一個資訊中心。但是在軍事部署的 DSNs 中，傳送非必要的資料會增加被偵測到的風險，而且也會消耗電池的動力與頻寬。供電問題，DSNs 配置的靈活度、DSNs Nodes 間資訊傳遞的安全性、DSNs System 的存活性(Survivability)…等問題也待解決。

計劃將行動代理人的技術應用在 Sensor Network 環境中，可逐步改善 DSNs 的問題，例如：提高 DSNs System 的存活率，以及增加 DSNs Nodes 間資訊傳遞的安全性…等。行動代理人(Mobile Agent)會選擇性的拜訪感應器，並且逐漸的整理合適的資料。它可以自來源端派遣到遠端執行。執行相關動作收集所需的資訊後，行動代理人會帶著結果離開。因此將可以透過行動代理人解決 DSNs 的安全性問題並提升 Sensor Nodes 的存活率。

本期的研究的首先是研究學習 Mobile Agent System，了解 MA 的運作原理與方法，再分析已開發之 MA 系統的各處優缺點，選擇適合的 MA 系統進行研究與探討。本期研究之結果將可以為下一階段之研究進行連接---Distributed Sensor Network。預期在下一階段中，團隊

將會以本期研究的 MA 系統搭配 DSN 並結合兩方之優勢將可以透過行動代理人解決 DSNs 的安全性問題並提升 Sensor Nodes 的存活率。

## 2. 英文摘要

With the progress of science and technology, the progress in sensor, network and computer enables us to dispose a large amount of cheap and small sensor node to approach a lot of complicatedness applications. Especially improvement of Distributed Sensor Network (DSNs) and price reducing make applications in military, fire control, medical treatment, ecological research, etc. useful.

Unfortunately, the traditional sensor network, all sensor nodes' data must be sent to an information center. This way will increase the detected risk of data, and will consume the battery and bandwidth. In addition, power supply, flexibility dispose, security, and survivability of sensor nodes are remained to be solved, too.

The project is to apply the mobile agent technology on Sensor Network environment. It will solve some problems of DSNs, for example, improving the survivability of DSNs system, enhancing the security of comiunicatoin between DSNs nodes, etc.. The mobile agent visits sensor node selectivelym and fuses the information. It is issued form source and executed remotely. After executing the collecting useful information, the mobile agent leaves with results to the next node or back to the center. Thus the security problem of DSNs can be solved by using mobile agent and improves the survivability concurrently.

# 二、 前言與研究目的

自 1990 年代起，Multiple Sensor Systems 成為許多研究的目標，尤其是在 Distributed Sensor Network (DSNs)。這當中技術不斷的被改進再改進、價錢不斷的降低而使得在軍事、消防、醫療、生態研究等項目上被廣泛的利用。由於 DSNs 能夠提供的應用層面廣而彌補以往傳統的網路不足的地方，因此 DSNs 能夠迅速而廣泛的應用。

DSNs 是被許多以電池為動力的微處理器所組成，而此微處理器上面附著了許多極小的感應器。因而形成了所謂的 Wireless Ad Hoc Network。由於感應器的體型很小且數量很多，因此相較於以往的系統上提供了較好的解決方法，可以解決問題的規模也較為廣大。例如: 我們居住地的環境監控、局部性的氣候研究、醫療護理和架構上的監控、動植物生態研究等。

隨著科技的突飛猛進，現今在感應器、網路和計算領域的發展使得我們可以部署大量既便宜且小的感應器，來達成許多複雜應用所需的品質需求。在一個部署遠端操作的 DSNs 子課題中，透過低頻寬的無線網路往往是感應器間通訊的唯一手段。這些感應器通常在處理能力、電池能力和通訊頻寬上具有限制。在這些感應器間作通訊會消耗有限的電力，為了保證他們所能負擔的操作，在電力上的消耗必須保持最小。顯然的，DSNs 必定成為將來的主流，而他們勢必朝向更廉價、更容易操控且更靈活的方向邁進。

然而，現在的 DSNs 依然存在許多問題仍需解決，諸如：供電問題，DSNs 配置的靈活度、DSNs Nodes 間資訊傳遞的安全性、DSNs System 的存活性(Survivability)…等。現有的 DSNs 軟體仍然稀少，而使得在很多方面仍然無法滿足在現今生活上的許多應用。絕大多數的

DSNs 系統都是依據特殊目的而設計的，而 DSNs 必須在應用之前必須被部署和安裝，但是被部署了以後 DSNs 只能稍微適應環境的變化情形，所以一但外在的狀況改變，則系統必須大幅度地修正。舉例來說，假設 Sensor Network 事先就已經被部署用來偵測是否有森林火災，若森林的火災被偵測出來之後且消防對趕來時，消防隊又必須重新設計網路來使得網路可以搜尋並且對外提供救援的申請。當然，我們可以將偵測和提供救援的申請這兩種方法整合在一起，但是對於一般的 DSNs，仍然受限於他的記憶體大小及儲存元件方面。

另一方面，在傳統的架構中，所有的 DSNs 感應器資料是集中送到一個資訊中心。但是在軍事部署的 DSNs 中，傳送非必要的資料會增加被偵測到的風險，而且也會消耗電池的動力與頻寬。為了迎接這些新的挑戰，Mobile Agent-Based on Distributed Sensor Networks (MADSNs) 已經被 Qi， Iyengar， and Chakrabarty 所提出。一個行動代理人(Mobile Agent)會選擇性的拜訪感應器，並且逐漸的整理合適的資料。行動代理人是一個特別的程式，它可以自來源端派遣到遠端執行。當抵達一個遠端節點時，行動代理人會發出他的憑證，以獲得存取當地資料與服務的權限，執行相關動作收集所需的資訊後，行動代理人會帶著結果離開。因此將可以透過行動代理人解決 DSNs 的安全性問題。

本計畫將規劃與設計一個具有可存活特性之 Sensor Network 環境。在研究背景中有描述關於 DSNs 目前存在的問題，諸如：供電問題，DSNs 配置的靈活度、DSNs Nodes 間資訊傳遞的安全性、DSNs System 的存活性(Survivability)…等。將行動代理人的技術應用在 Sensor Network 環境中，將可逐步改善 DSNs 的問題，例如：提高 DSNs System 的存活率，以及增加 DSNs Nodes 間資訊傳遞的安全性…。等。

本計劃將研究透過定期發送行動代理人對 Sensor Nodes 進行監控，一但有任何 Sensor Nodes 因為任何原因不能運作順暢，例如：電池電力不足、網路中斷…等，DSNs Center 就可以及時知道問題所在，並且準確地針對有問題的 Sensor Node 或是其相關路徑上的障礙排除，這樣就可以提高 DSNs Nodes 的存活率。

本計劃亦希望研究透過行動代理人本身可攜帶資料以及可運算的特性，將密碼學與資訊安全的技術運用於行動代理人。設計使所有 DSNs Nodes 間的資訊傳遞都由行動代理人負責，以增加 DSNs Nodes 間資訊傳遞的安全性。

# 三、 文獻探討

## 1. Mobile Agent 概念

行動代理者是一種具有獨特能力的代理者程式，可以把自己經由網路，從一個主機系統中移動(或被移動)到另一個主機系統，以存取當地的資源，完成作業的執行。這種可以移動的特性，允許行動代理者到網路中一些有提供服務的系統去存取他所需要的服務。 行動代理者具有自主性與行動能力，並可以降低網路的流量，在網路頻寬有限與網路品質不穩定的網際網路中，它提供了系統具有強健(Robust)與高度的容錯能力(Fault-Ttolerant)的能力。

依照行動代理者執行的特性，行動代理者有兩種主要的方式：

(1) 弱機動性(Weak Mobility)：

弱機動性的行動代理者，程式碼可以傳送到網路上其他的節點來執行，以達成程式碼機動(Code mobility)的功能。

弱機動性的行動代理者系統，其實現的方式是：行動代理者在移動時只帶著起始的資料，並不會傳送行動代理者在執行作業時所產生的一些的動態狀況。這一類行動代理者的生命週期模式，通常是以任務作為基礎(task-based)的型式。在傳送行動代理者前，必需明確的賦予行動代理者，到下一個網路節點時所要開始的任務為何。大多數以 JAVA 為基底所完成的行動代理者系統(Java-based mobile agent systems)，大多是利用弱機動性來設計的，如：IBM Aglet，Odyssey，MOLE，Concordia，Voyager 等。

(2) 強機動性(Strong Mobility)：

強機動性行動代理者系統，當這一類的行動代理者被傳送時，它會暫停它正在執行的作業，並將它的程式碼及執行狀態(包含靜態資料及動態狀態)傳送至下一個節點，並在下一個節點處繼續它所要執行的任務。 強機動性行動代理者系統，其實現的方式可分為遷移(Migration )與遠端複製(Remote Cloning)兩方面：

- 遷移：行動代理者行動時，先暫停作業的執行，並將程式碼及執行狀態傳送至下一個節點，等行動代理者到達下一個節點時再恢復作業的執行
- 遠端複製：行動代理者行動的方式，是在下一個節點處，另外複製一個行動代理者來執行任務。這兩種方式之差異在於，在下一次任務執行時，行動代理者本身與它的複製行動代理者是否同時存在。

具有這種強機動特性的行動代理者系統，有 Telescript，TACOMA，及 Agent Tcl 等。行動代理者具有遷移與自主的特性，可將工作依照實際需要，攜帶至適當的地方去執行，以便利用當地的資源，來完成工作，再將執行結果送回原來主機。一般來說，行動代理者技術的優點有：

- 降低網路負荷：系統將作業交由行動代理者帶到資料所在的機器上執行，等作業執行完成後，將結果傳回至原來的機器上，可減少資料透過網路來傳遞與交換的數量與次數，節省網路頻寬的浪費並增加系統的彈性。
- 克服網路延遲問題：在一些緊要的即時系統(Critical Real-time

System)中，例如在製造流程中的機器人(Robots)，需要即時回應在他們環境中的改變。若經由工廠的網路來控制這類系統，會因為網路有許多可觀的延遲問題，而無法適用於即時系統。使用行動代理者系統，可以由中央控制端派遣行動代理者到各局端，依照控制端的指令在局端直接作業，如此可以即時回應環境的改變，不會有網路延遲性的問題。

- 封裝通訊協定：在分散式系統中，當資料需要交換時，必須考慮各種系統主機的不同通訊協定，必須在資料外送時適當的編碼，並解譯接收進來的資料。採用行動代理者技術，可以將通訊協定封裝起來，將行動代理者遷移至遠端主機時，可以依照適當的通訊協定建立連線通道(Channel)，系統發展者不需再考慮不同通訊協定間之溝通問題。
- 非同步執行與自主行動：行動式上網設備與網路作連接時必須考慮兩各方面的議題：連線費率與網路的穩定度。若須要與網路持續不斷的連線才能執行的作業，在經濟及技術的考量上都是不可行的。行動代理者技術，可將作業任務嵌進行動代理者中，當行動代理者被派遣出去後，便可具非同步執行與自主行動的特性，即使發出行動代理者的機器暫時離線或關機，行動代理者也可以自行將任務完成後，等原來的機器連上線後，再將資料傳回。

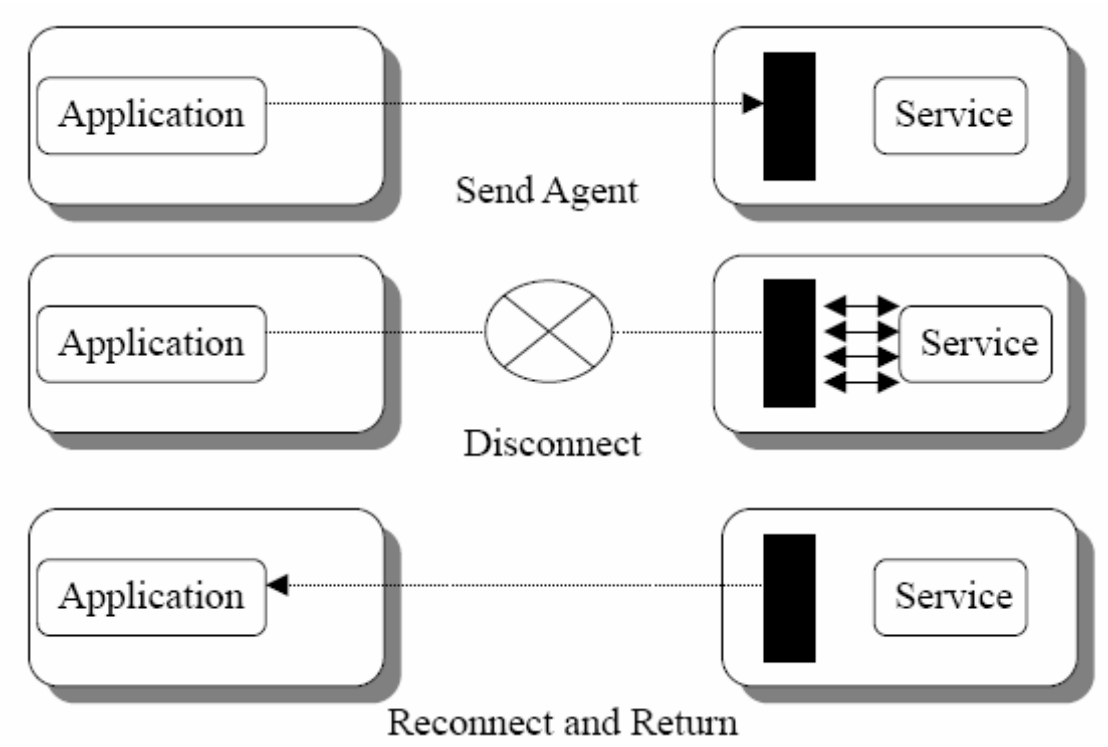


圖 1-行動代理者和離線操作

- 適應變動(Adapt Dynamically)：行動代理者有能力去偵測他們所執行的環境，並可以對環境的變化自主反應。
- 異質(Naturally Heterogeneous)：從硬體或軟體的角度來看，網路計算基本上是異質的。而行動代理者一般都是與電腦或網路傳輸層(Transport-layer)獨立無關，只與他們所執行的環境有關。所以行動代理者可以為系統無縫(Seamless)的整合提供一個最佳的條件。
- 強健與容錯(Robust and Fault-Tolerant)：當環境及條件不理想時，行動代理者可以動態反應的能力，使得行動代理者技術易於建立強健與容錯的分散式系統。例如：當某一主機要關機時，在這台主機上執行的所有行動代理者，便會收到警告，並會被派遣至網路上其他台主機，以繼續他們的作業執行。

- 因為行動代理者的架構具有彈性、易於擴充，可以改善主從式架構的缺點，提供了系統高度的延伸性(Scalability)。

## 2. Mobile Agent 要素

一個完整的行動代理者系統有下列幾個重要的元素：代理者(Agent)、場所(Place)、機動性(Mobility)、溝通(Communication)、授權(Authority)、與允許(Permit)。

- 代理者：定義為個人的行為代表，可為一個動態的物件(Mobile Oobject) 或是一個靜態物件。一個代理者可以在場所誕生(Create)、消失(Dispose)、行動代理者則可以被派遣(Dispatch)到另一個場所執行，同一個時間，一個代理者只能在一個場所中執行。
- 場所：可以把網路上一些相互連結的電腦視為一些場所的集合。以 Aglet 系統為例，場所稱為 Context，為一個靜態物件(Stationary Object)，提供代理者生存和執行的環境。例如將一個購物商場視為一個場所，一個購物代理人可以在此執行授權者賦予給它的購物的任務。
- 機動性：在行動代理者系統中，行動代理者可以在系統間到處遊歷，從一個場所移動到另一個場所，去存取局端的資源來完成它被賦予的任務。例如可以賦予購物代理人購物與比價的任務，提供所需的物品與可接受的價格，交由購物代理人至各個電子商場去查詢與比價，以期買到最適合自己的物品。在 Aglet 系統中，可以利用 dispatch 這個指令，將行動代理人派遣至另一個場所。

- 溝通：行動代理者之間，可以在相同的場所內或不同的場所之間互換訊息，經由互換訊息，行動代理者可以要求另一個行動代理者協助，行動代理者系統也可以將一個任務交由多個行動代理者共同完成，行動代理者之間也必須藉由溝通來合作完成。當溝通失敗時，行動代理者也可以自己決定下一步該如何做。例如購物代理者與銷售代理者進行交易，可以經由代理者之間的溝通來進行詢問、比價與殺價。
- 授權：在代理者系統中，場所和代理者必須能夠知道一個外來的行動代理者是誰、從哪裡來，並確定它說的是否正確。一個場所可以拒絕一個未授權的行動代理者進入。
- 允許：經過一些分配的允許授權，來限制行動代理者可以做哪些事。允許可以做的事，包括可執行的動作，及可使用的資源兩種。場所必須防範惡意的使用者，並保護資料的安全性，一個未經允許的代理者不能隨意的執行動作或存取資料。

# 3. Mobile Agent 應用領域

在分散式系統的研究中，目前所用之架構多為 Client/Server 模式，然而為了減少網路之流量，分散工作處理之負荷及增加系統的彈性，以 Agent 為基礎的模式，如 Agent-Based System(ABS)或 Multi-agent system(MAS)，已經有許多的應用領域被提出來研究，如：智慧型製造、分散式居家照顧、智慧型網路、人工智慧、網路管理、電子商務等。

一般來說，行動代理者系統可以應用的領域有：

- 電子商務(Electronic Commerce)

- 個人助理(Personal Assistance)
- 安全仲介(Secure Brokering)
- 分散式資訊擷取(Distributed Information Retrieval)
- 電信網路服務(Telecommunication Networks Services)
- 工作流程與群組軟體(Workflow Applications And Groupware)
- 監控與通報(Monitoring And Notification)
- 資訊散播(Information Dissemination)
- 平行作業(Parallel Processing)

# 4. Mobile Agent 運算模式

行動代理者系統提供網路運算模式一個功能強大、具有一致性的模式(Paradigm)，使得分散式系統的設計及發展有了徹底的改革。在這個章節中，針對三種分散式運算的設計模式，作一些簡單的介紹與比較。這三種分散式運算的設計模式是：主從(Client-Server)架構、隨喚程式碼(code-on-demand)、及行動代理者。

- 主從式運算架構

主從式架構分為伺服器(Server)端與用戶端(Client)，彼此透過網路直接連接工作，伺服器公佈一組用來存取某些資源(Resources, 如資料庫)的服務(Services)，而這些提供服務的程式碼是寄宿在伺服器端。 當用戶端需要存取伺服器端的某些資源時，就必須使用伺服器端所提供的某些服務，而這些服務皆在伺服器端執行，換句話說，伺服器具有處理程序的能力。 傳統上，要執行遠端的程式，可能需要使用遠端程序呼叫(Remote Procedure Call，RPC)這類技術來達成目的。當用戶端需要存取某些資源時，會向掌握這些資源的伺服器發出

程序呼叫(Procedure Call)，伺服器收到後便會執行此程序，執行完畢後將結果傳回給用戶端。

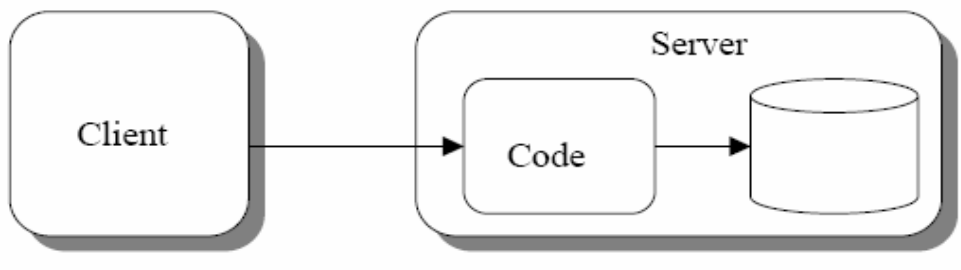


圖 2-主從運算模式

RPC 技術是主從模式常採用的方式，它的缺點是它的作業過程：用戶端發出要求服務，伺服器回應結果，會經由網路不斷的重複，造成網路流量的負荷，如圖 2-5。 利用 RPC 技術來進行系統發展，也不是一件容易的事，在不同的平台上，實作 RPC 的方式與機制也不盡相同。對行動式計算而言，以 RPC 技術為基礎的主從式架構，在執行的時候，必須維持和網路的持續連結狀態，在經濟與技術上的考量上，都不易實施。 目前大多數的分散式系統以此種模式為基礎，主機掌控全面：程式碼、資源、及程序處理。除了 RPC 外，物件要求仲介(Object Request Brokers, CORBA)，以及 Java 的遠端方法引用(Remote Method Invocation, RMI)都是支援主從架構的技術。

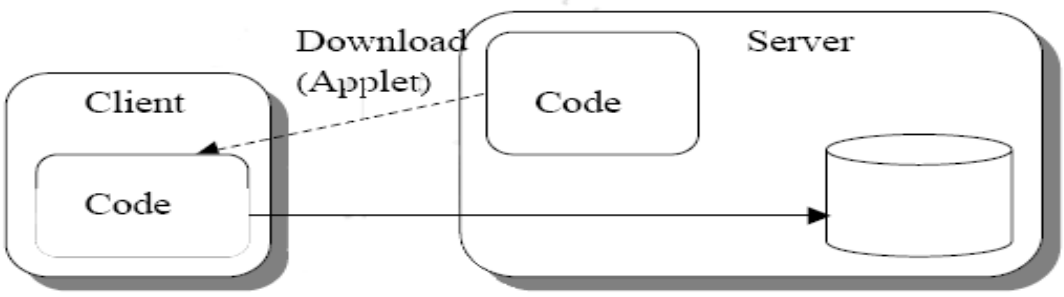


圖 3 -Code-on-Demand(Applet)模式

● Code-on-Demand

依照 Code-on-Demand 的模式，如圖 2-3，當用戶端需要程式碼時，必須先由伺服器端下載來執行。伺服器端提供程式碼，當用戶端下載程式碼執行時，計算能力是交由用戶端所決定。 用戶端掌握處理程序執行能力、局端的資源，相較於傳統的主從式架構，用戶端無須事先安裝程式碼，因為所有需要的程式碼都可以由伺服器端下載來執行。

Java applets 以及 servlets 是此種模式的特例，WWW 伺服器端提供 Applet 的程式碼，當 WWW 瀏覽器瀏覽包含 Applet 程式碼的網頁時，便會將 Applet 下載在用戶端執行。

● Mobile Agent 模式

行動代理者模式的主要特性，如圖 2-4，在網路上任一個行動代理者系統主機都具有高度的彈性，可以掌握程式碼、資源、程序處理三者的部分混合。他的程序處理能力可以與局端的資源連結在一起，程式碼並不是被一台單一主機所束縛，而是在整個網路裡都可以利用到的。

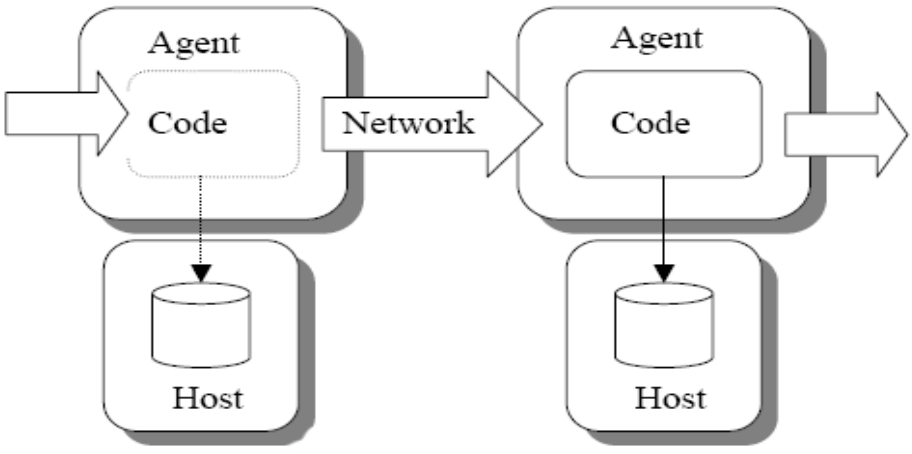


圖 4-行動代理者模式

# 5. Mobile Agent 運算模式比較

● 行動代理者與傳統主從式運算的比較

傳統主從式運算的技術，有一些缺點，例如在網路的使用率上(Network Utilization)，RPC 技術將會比行動代理者技術帶給網路更大的負荷，而且 RPC 技術必須持續的與網路連線才能完成工作，對於行動式運算而言並不實用，在使用上會受到限制。

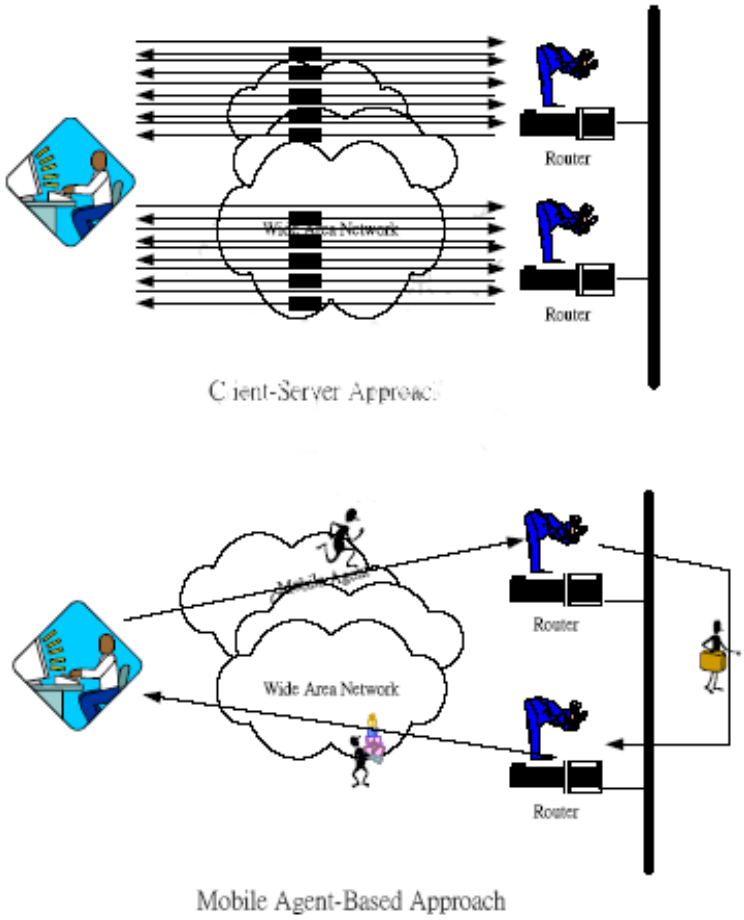


圖 5-行動代理者與傳統主從式運算比較圖

此外，在不同平台實作 RPC 的方式與機制不盡相同，對程式設計者而言會增加程式寫作的困難。而且當應用程式需要更新或擴充時，用戶端與伺服器端的程式都必須作相對應的修改，不但耗費成

本，系統更不易維護，使得系統不易擴充，具有延伸性(Scalability)的問題。

使用行動代理者的好處是可以減少無謂的通訊，而且行動代理者在執行任務時，局端用戶與伺服器端也不必一直保持連結的狀態，有益於分散式環境與行動計算上的應用。另一方面，若行動代理者使用的行動程式碼是一些和平台無關(Platform Independent)的中間碼(Intermediate Code)，更具有跨平台的好處。

另一方面，行動代理者的使用是由應用程式來提供相容性，而不用建立任何額外的應用階層的通訊協定(Application-level Protocol)，因此，應用程式的更新或擴充都不需更動客戶端的程式，而且伺服器端程式的更新或服務的修改，都不需中斷網路或修改客戶端程式的版本。

● 行動代理者與 Java applet 的比較

當我們想做一些具有互動性的網頁時，可以使用 Java applet 來完成，Java applet 採用的運算模式是 Code-on-demand，它的程式碼具有可移動的特性，但這種 Mobility-code 並不就是行動代理者。

Java applet 的使用，需先經過編譯成位元碼的形式，放置在 WWW 伺服器端，當使用者瀏覽該網頁時，瀏覽器便會從伺服器端下載該 Java applet 至用戶端，並藉由用戶端 Java-enabled 的 WWW 瀏覽器來執行。Java applet 程式碼可以被用戶端下載來執行，但不具有自主遷移的特性，而且為了安全的考量，除了伺服器端與客戶端之外，Java applet 不允許與其他的主機建立連結，故 Java applet 只能被視為具有 Mobile-code 的特性，而不能視為行動代理者。

# 6. Mobile Agent 的標準 MASIF

行動代理者是一項尚在發展中的新技術，它具有的獨立自主性與行動能力，適用於建立分散式的系統架購，並可降低網路的流量。在網路頻寬有限與網路品質不穩定的網際網路中，它提供了系統具有強健(robust)且高度容錯的能力(fault-tolerant)，因此吸引了各種應用領域的研究。在各方面的研究中，有許多的行動代理者的系統被設計出來，如 Aglets、Odyssey、Concordia、Voyager、Agent Tcl、TACOMA 等等。

這些應用在各種領域的系統，設計與架構上都有很大的不同，為了提昇系統間的共通性與系統的多樣性，某些領域的行動代理者技術需要標準化。有五家公司 Crystaliz、General Magic Inc.、GMD Fokus、IBM Corporation、及 Open Group 一起參與了多品牌行動代理者系統共通設施(Mobile Agents System Interoperability Facility, MASIF) 提議案規格的制定，並將之提出至 Object Management Group(OMG)。

MASIF 所定義的介面是介於代理者系統之間，而不是在代理者系統與代理者應用軟體之間，也就是說，MASIF 的功能不是讓不同語言所制定的行動代理者互相共通，不同語言的共通是十分困難的，MASIF 的功能是限制在讓用同一種語言(或許是不同廠商)的行動代理者系統間共通。

MASIF 並不打算將局端代理者(Local Agent)的作業標準化，這些作業，包括行動的詮釋(Interpretation)、連續(Serialization)、或執行(Execution)。MASIF 的標準包含下列四個範圍：

- Agent management：關於代理者相關的管理操作，如代理者的產生(Creation)、暫停(Suspend)、恢復(Resume)、以及終止(Terminate)。
- Agent transfer：目的在於使代理者軟體可以自由的在不同型態的代理者系統間移動，導致一個通用基礎的產生，使得代理者可以拜訪許多可利用的系統。 17
- Agent and agent system names：代理者及代理者系統的名稱所使用的標準語法(Syntax)及語意(Semantic)，允許代理者及代理者系統來證明彼此身分，同樣使用者也可以用來驗證代理者及代理者系統。
- Agent system type and location syntax：除非代理者系統的型態可以支援代理者，否則代理者不可以被傳送。位置的語法是標準的，代理者系統才可以用來定位彼此的位置。

# 四、 選擇 Mobile Agent System

本期的研究的首先是研究學習 Mobile Agent System，了解 MA 的運作原理與方法，再分析已開發之 MA 系統的各處優缺點，選擇適合的 MA 系統進行研究與探討。本期研究之結果將可以為下一階段之研究進行連接---Distributed Sensor Network。

## 1. 選擇 Mobile Agent System 之準則

本章會說明選擇適合的 MA 系統之相關準則與考量以及介紹各個列入考慮的 MA 系統。我們列舉了選擇 MA 系統開發平台的數點考量：

- 延續性：本研究計畫屬多年型計畫，因此該 MA 系統開發平台是否持續發展、更新將是很重要的參考指標。多數的 MA 系統開發平台都只有初期的簡單版本，提供了 MA 系統在學術定義上最基本的功能，也幾乎無安全性設計。故有延續性的 MA 系統開發平台會針對其弱點、缺失不斷地改良，連帶地使本計劃也會獲得改善。
- 延展性：當我們選定了此 MA 系統開發平台之後，會加以利用、修改使 MA 系統可以契合我們的需求。若延展性差，會限制接續計畫中連接 DSN 的開發，因此該 MA 系統開發平台的延展性便是很重要的指標。
- 程式語言：MA 系統開發平台所使用的程式語言是很重要的。不同的程式語言間必定存在著相容性的問題，雖然大多數都是可以解決的問題，有些不會有所影響，但是有不少問題是必須耗費效能與穩定性。因此若 MA 系統開發平台與 DSN 皆使用

相同的程式語言開發，將可以避免因不同語言所產生之各種問題。

- 技術支援：當選定了 MA 系統開發平台後，除了了解其功能之外，必定會針對其設計稍加修改企圖與 DSN 相搭配。因此 MA 系統開發平台所提供的技術支援將扮演了很重要的角色。越充足的技術支援可協助我們越快速、正確、深層地了解 MA 系統開發平台。然而大多數的 MA 系統開發平台並沒有提供很充足的技術支援，僅有讓使用者建置基本的 MA 系統，使其無法進一步地修改其較核心、深層的設計，也就無法與其他系統相連接。

# 2. Mobile Agent System 調查

本節調查了許多 MA 系統開發平台，諸如：Aglet、Zeus、Concordia、MAP(Mobile Agent Platform)、Hive、ARCA、Fargo、Grasshopper、FIPA-OS，等…。在此節僅列出部分特點加以討論。

- Aglet：
  - ✧ Originally developed by IBM Tokyo Research Lab.
  - ✧ Aglet is from Agent and Applet.
  - ✧ Became an open source project at sourceforge.net.
  - ✧ Use Java.
  - ✧ Added library on ver. 2.X.
  - ✧ Latest Version：2.0.2 published at Feb 19, 2002.
  - ✧ Document：User Manual v.2 Oct 6, 2004.

- Grasshopper：

- ✧ Developed by IKV++, which is a platform supports the development and execution of Mobile Agent.
- ✧ Implemented by Java.
- ✧ Evaluation for free.
- ✧ Optional add-on pack for FIPA-standard.

● Jumping Bean：

- ✧ A framework for quickly and easily building jumping applications with security executing.
- ✧ A computer can receive a jumping application even if the jumping application has never been install on it.
- ✧ Latest Release Version 2.8.2 at May 15, 2005.
- ✧ Hard to ask for.

● FIPA-OS：

- ✧ Foundation for Intelligent Physical Agent Open Source.
- ✧ FIPA-compatible platform.

● Voyager：

- ✧ Developed by Recursion. Inc.
- ✧ Support libraries for C++、C#(.NET)、Java.
- ✧ Documents are fully supported.
- ✧ Not free.
- ✧ Latest version 4.8 released at Jan, 2005.
- ✧ http://www.recursionsw.com

# 3. Mobile Agent System 選擇結果

在最後的考量中，由於 aglet 使用 java，其程式語言具有跨平台的特性，並且擁有較齊全的資料、以及其優良的延展性，因此我們最後選擇 aglet 作為平台。

# 五、 Introduction of Aglet

## 1. Aglet 系統簡介

Aglet 是由 IBM 公司所研發的行動代理者系統，它採用了 Java 技術的優點來設計系統，aglet 是一個可以移動的 Java 代理者程式，並支援自主執行(autonomous execution)與動態旅程(itinerary)的路徑(routing)。 Aglet 系統可以想像是 Java applets 與 servlets 的綜合與延伸。Aglets 寄宿在 Aglet server 的方式就像 Java applet 寄宿在 WWW 瀏覽器上一樣。Aglet server 提供 aglets 可以執行的環境、Java 虛擬機器(Java virtual machine, JVM)及 Aglet 的安全管理機制，因此 Aglet server 可以安全地接收 aglets 與當它的宿主。

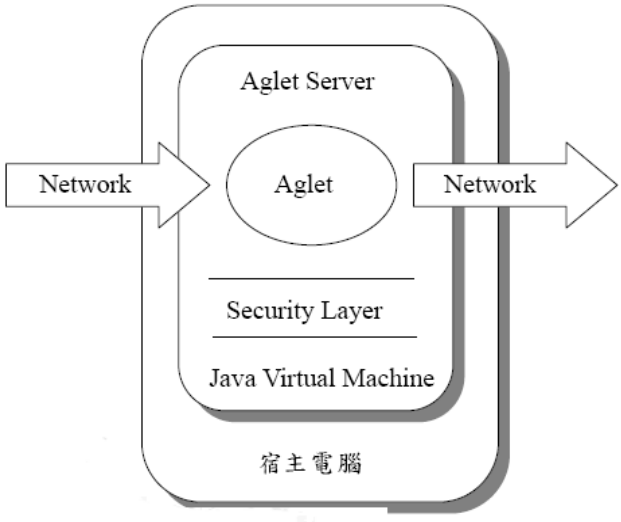


圖 6-Aglet 的架構圖

## 2. Aglet 的基本要素

在 Aglet 物件的模型中，行動代理者是一個可移動的物件，它擁有自己執行緒的控制能力，具有事件觸發的能力，利用訊息傳遞的方

式來達成代理者間的溝通。 在 Aglet 的系統模型中，它的主要元件是 Aglet、Proxy、Context 以及 Identifier：

- Aglet：一個 aglet 是一個可移動的 Java 物件，它可以經由電腦網路去拜訪提供 aglet 執行環境的主機。它具有獨立自主的特性，在到達主機時依然使用自己的執行緒在執行。
- Proxy：Proxy 是一個 aglet 的代表，它當作是 aglet 的屏障，可以避免外界直接存取 aglet 的公用方法(Public Method)。它並提供 aglet 位置的透通性(Location Transparency)，透過 Proxy 即可與 aglet 溝通，如此可以隱藏 aglet 的實際位置。
- Context：一個 Context 即是 Aglet 的工作場所，這是一個靜態的物件，它提供工具以維護及管理執行中的 aglet ，並保護主機系統的安全，不受惡意 aglets 的侵害，有良好的執行環境。在電腦網路的一個節點中，可以執行多個伺服器程序(Server Processes)，每個伺服器可以管理多個 Context。
- Identifier：每個 aglet 都有一個獨一無二的 identifier，作為 aglet 的識別作用，在 aglet 的生命周期中，此識別碼永遠是唯一且不會改變的。

# 3. Aglet 的生命週期

在 aglet 的系統中有兩個方法可以產生 Aglet：由程式碼來創造(Creation)一個 Aglet，或是由一個已經存在的 aglet 複製(Clone)而產生。為了控制 aglet 的數量，可以利用 Disposal 的方式來移除 Aglet。Aglet 的行動能力可以分為主動(Actively)與被動(Passively)兩種：主動的方式是由 Aglet 自行派遣(Dispatch)到另一個 Context 去執行，而

被動的方式則是 Context 將 Aglet 由另一個 Context 中取回(Retract)。當 Aglet 在執行時會佔用一些資源，為了減少資源的消耗，在電腦效率很低時 aglet 可以暫時的睡眠以釋出它所佔用的資源(Deactivation)，等稍後再回到執行的狀態(Activation)。

- 建立(Creation)：aglet 需要在 Context 中才能建立產生。新產生出來的 Aglet 會被賦予一個唯一的識別碼，然後安插進入 Context 中進行初始化的工作，當 Aglet 成功的初始化之後便可以馬上開始執行。
- 複製(Cloning)：aglet 在同一個 Context 中進行複製，複製時產生的 aglet 幾乎與原先之 aglet 完全相同。不同之處在於識別碼不同，執行緒也並未複製。
- 派遣(Dispatching)：派遣一個 Aglet 至其他的 Context，即是將它從目前的 Context 中移除後，並安插到目的地的 Context 中重新開始執行，也可視為是 aglet 被推(Push)至新的 Context 的行為。
- 收回(Retraction)：將 Aglet 由它目前所在的 Context 中拉(Pull)回至發出收回要求的 Context 中。
- 活化(Activation)與反活化(Deactivation)：暫時將 aglet 終止它的執行，並將它的狀態(State)儲存在輔助記憶體(Secondary Storage)中，稱為反活化，而將已經反活化的 aglet ，從輔助記憶體中還原至原來的 Context 中，恢復執行，稱為活化。
- 除去(Disposal)：終止 Aglet 目前的執行工作，並將它由目前的 Context 中移除，稱為除去。

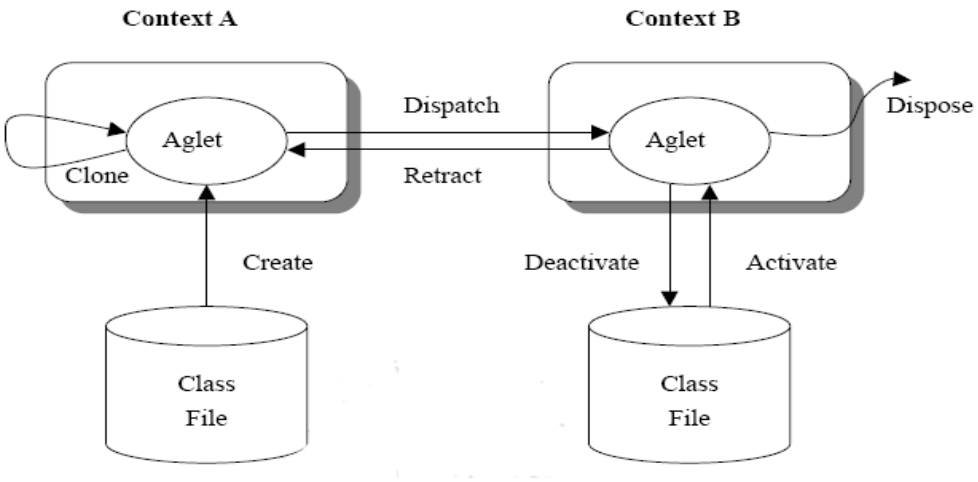


圖 7-Aglet 的生命週期

# 4. Aglet 的事件模式(Event Model)

Aglet 的程式模式是以事件為基礎的。它的事件模式允許程式設計師嵌入(plug-in)特製的 listeners 到 aglet 之中。Listener 可以在 aglet 的生命週期中，捕捉特殊的事件，並允許程式設計師根據事件，來撰寫程式，以做出相對應的動作。在系統中存在三種 listeners：

- Clone listener：捕捉關於複製的事件。程式設計師可以針對 Aglet 的複製過程，如 aglet 將要複製之時、複製真正被建立時、以及複製事件發生完畢後，在這些事件中，為此 listener 定義一些相對應的動作。
- Mobility listener：捕捉關於移動的事件。程式設計師可以針對 aglet 的移動過程，如 aglet 將要被派遣(Dispatch)至另一個 Context 之時、將要從另一個 Context 中被取回(Retract)時、以及它真正到達一個新的 Context 之時，在這些事件中，為此 Listener 定義一些相對應的動作。
- Persistence listener：捕捉關於持續性的事件。程式設計師可以

針對 aglet 的活化或者反活化過程，以這些事件，為此 listener 定義一些相對應的動作。

# 5. Aglet 的溝通模式(Communication Model)

Aglet 的溝通模式由訊息傳遞(Message Passing)來達成。訊息機制(Messaging Facility)允許 aglets 用彈性的方式來產生及交換訊息，如圖 2-9。而 aglet 的訊息機制並不假設會有同時(Concurrent)處理的狀況，所有的訊息都是一個接一個的依序處理。

除了 Aglet 和 Aglet Proxy 之外，在 Aglet 的溝通模式中還有三個重要元件：

- 訊息(Message)：訊息是一種可以在 Aglets 之間交換的物件。在 Aglets 之間交換訊息的方式，可以是同步交換也可
- 以是非同步交換。Aglets 之間利用訊息傳遞，來協同合作以及交換資訊。
- 未來回覆(Future Reply)：使用於非同步訊息傳送，允許訊息傳送者可以在非同步狀況下，接收訊息的回覆。
- 回覆集合(Reply Set)：一個回覆集合(Reply Set)可以包含多個未來回覆(Ffuture Reply)，當他們可應用時，用來取得結果。發送者(Sender)可以只選擇取得第一個回覆的結果，並忽略其餘的回覆。

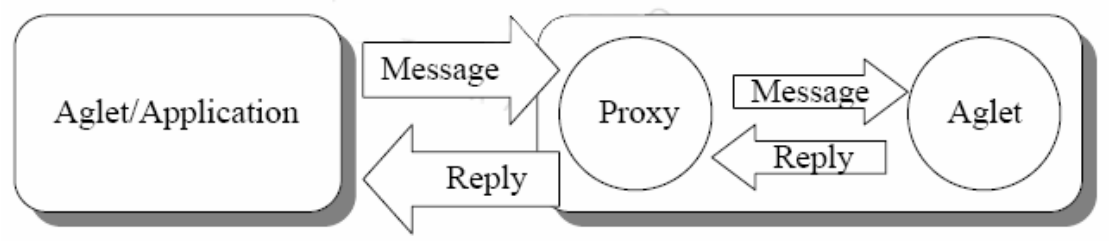


圖 8-Aglet 與 Aglet 之間的訊息溝通

# 6. Aglets 平台

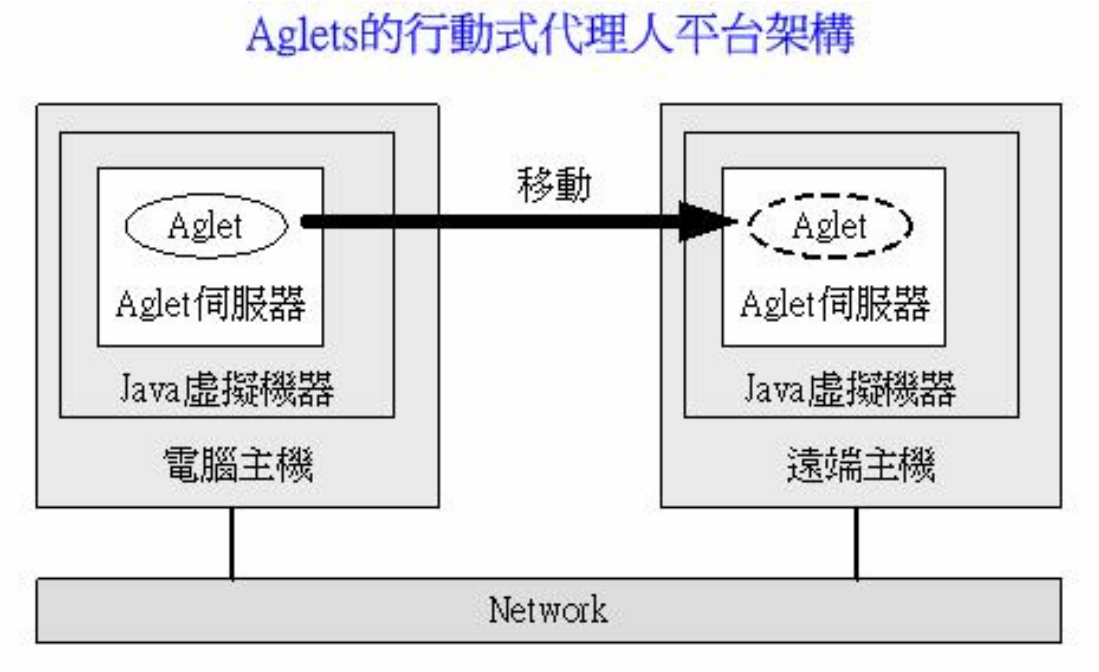


圖 9-Aglet 的行動式代理人平台架構

ASDK(Aglets Software Development Kit)為 IBM 所提供行動代理人軟體發展套件，其中 Aglet 即意輕量化的代理人 ( Light-weight Agent )，此發展套件的目的為撰寫可移動至遠端主機執行的代理人程式和提供相關執行平台。

Aglets 是有 IBM 所開發出來的一個 Mobile Agent 的系統。如圖所示，在 Aglets 的架構中，Aglet 為一個 Mobile Agent 的 Object，並執行於 Aglet Server 之上，其中 Aglet Server 提供平台中多個 Mobile Agents 相互通訊的機制，並管理整個平台的資源和 Mobile Agents。

Aglets 系統是建立於 JAVA 的平台之上，Aglets 是由一群 JAVA 的 Object 所組成的，因此必須執行於 Java 虛擬機器之上，然後藉由網路與 Aglet Server 的協助，執行中的 Aglet 程式可以從某一個主機移動到網路上的其他的主機之上的 Aglet Server 上繼續執行；詳細來

說就是 Aglets 的 JAVA Object 可以在網路上的某一個主機執行到一半，然後 dispatch 到網路上的另一台主機上繼續執行。當 Aglets 在移動的時候，他會帶著他的程式碼和整個 Object 的狀態一起移動到目的地。

由於 Aglets 由 JAVA 實作而成，因此具有以下優點以利於 Mobile Agent 的執行：

- 平台獨立性( Platform Independence )：因為 Mobile agent 用於網路架構下，可在多個主機間移動，因此可能需要再不同的作業系統上執行；然而 JAVA 提供 Virtual Machine，只要主機上有 Virtual Machine，皆可在不同的作業系同上執行，所以具有平台獨立性。
- 安全性( Security )：由於 JAVA 設計主要是往網路方面的應用設計，因此對於安全性則特別注重，在 JAVA Virtual Machine 上執行的程式皆有嚴格的安全管制。
- 動態類別載入( Dynamic Class Loading )：JAVA 所提供的動態類別載入機制，使得 JAVA Virtual Machine 可在 run-time 載入類別執行。此機制使得 aglet 藉由網路移動到遠端主機後，可以在遠端主機繼續執行。
- 多執行序( Multi-threading )：因為 JAVA 擁有多執行緒的機制，所以使得一個 Aglet 執行的平台上，可以同時具有多個 Aglets 執行，且可獨立自主的執行以及提供同步的處理，使 Aglets 可以互相的溝通。
- 序列化( Serialization )：JAVA 提供的序列化機制，提供物件的序列化和反序列化的操作，使得 JAVA 類別可以在執行中將其含有的各項資料變數保留，並在往後可重新起始該類別並將資

料變數回復。因此藉由 JAVA 序列化的機制，可在 Aglet 移動前，將 Aglet 所具有的狀態儲存為位元串列，將保留的狀態傳送至遠端平台，並以反序列的功能將重新回復 Aglet 所擁有的狀態。

將以上的 JAVA 機制實做在 Aglet API 上，使得 Aglets 與執行平台具有以上的優點和支援，以符合行動代理人的環境要求。

接下來會介紹 IBM 所提供的套件，相關套件可在 http://www.trl.ibm.com/aglets/ 下載，目前最新的版本為 2.0.2。

## 7. Aglet 2.0.2 的目錄架構

Tahiti 為 Aglet 2.0.2 套件中內附的預設 Aglets 伺服器類別，他是利用套件中所含有的執行平台相關類別實作出來，用以作為行動代理人的執行平台。可以到 IBM Aglets 網站上下載 aglets-2.0.2.jar，裡面含有相關的類別與平台設定檔。如下圖所示，aglet 2.0.2 的目錄結構。

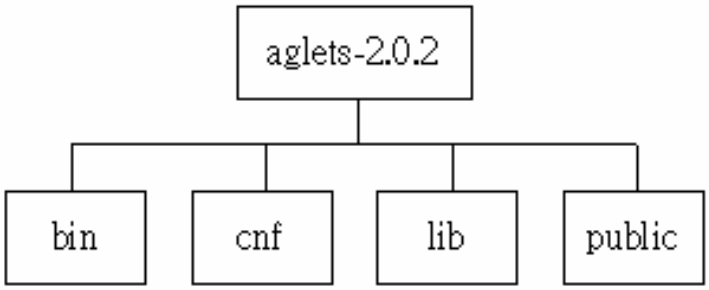


圖 10-Aglet 目錄結構

把 aglets-2.0.2.jar 解壓縮後，則會多個目錄：

- bin 的資料夾：含有啟動 Tahiti 的相關設定檔與安裝檔，所含有的 agletsd.bat.in 即為在 Windows 下的啟動批次檔範本，agletsd.in 即為在 UNIX 下的啟動 shell script 檔範本。

- cnf 資料夾：是用來存放啟動 Tahiti 時的相關設定檔，所含有的 aglets.props 即為初步設定了初始 Tahiti 時，相關資源和執行策略的規劃。
- lib 資料夾：存放相關的 Aglet 類別和介面，所含有的套件封裝黨 algets-2.0.2.jar 即為撰寫 Aglets 程式時 Java 類別和介面引用的主要來源。
- public 資料夾：因為 Aglets 中允許使用者將設計的 Aglet 放入，所以 public 資料夾為使用者放置 Aglet 程式的預設資料夾。

## 8. Aglets 2.0.2 的安裝

由於 Aglets 是利用 Java 程式語言實作而成，所以 Aglets 程式必須具備 Java 的環境。首先去 http://java.sun.com/j2se/1.4.2/ download.html 下載 Java2 標準版(J2SE)，並把 Java 環境安裝完畢，再把將 J2SE 安裝目錄的 bin 加入到系統環境變數 PATH 中，以便於編輯 Java 程式和使用相關的 Java 工具。

再到 http://sourceforge.net/project/showfiles.php?group_id=7905 下載 aglets-2.0.2.jar，解壓縮並執行 bin 目錄下的 ant.bat，若安裝成功會顯示 BUILD SUCCESSFUL 的訊息，失敗會顯示 BUILD FAILED。

執行完 ant.bat 後，啟動 Aglets 所需要的相關啟動檔即建立完成，且會自動產生.keystore 檔案，把此檔案放置作業系統的使用者目錄中(.keystore 是儲存使用者帳號和密碼的地方)。以下面的圖示，.keystore 必須 copy 到 C:\Documents and Settings\Qting\桌面。

若需要自行編輯 aglet 與 compile aglet 時，則必須將新增系統環境變數 CLASSPATH，並將 aglets-2.0.2\lib\aglets-2.0.2.jar 與 aglets-2.0.2\public 新增至 CLASSPATH 變數中。這樣在 compile 使用

者自行編輯的 aglet 程式則不會有 packet 找不到的 error 產生，以及執行失敗的問題。

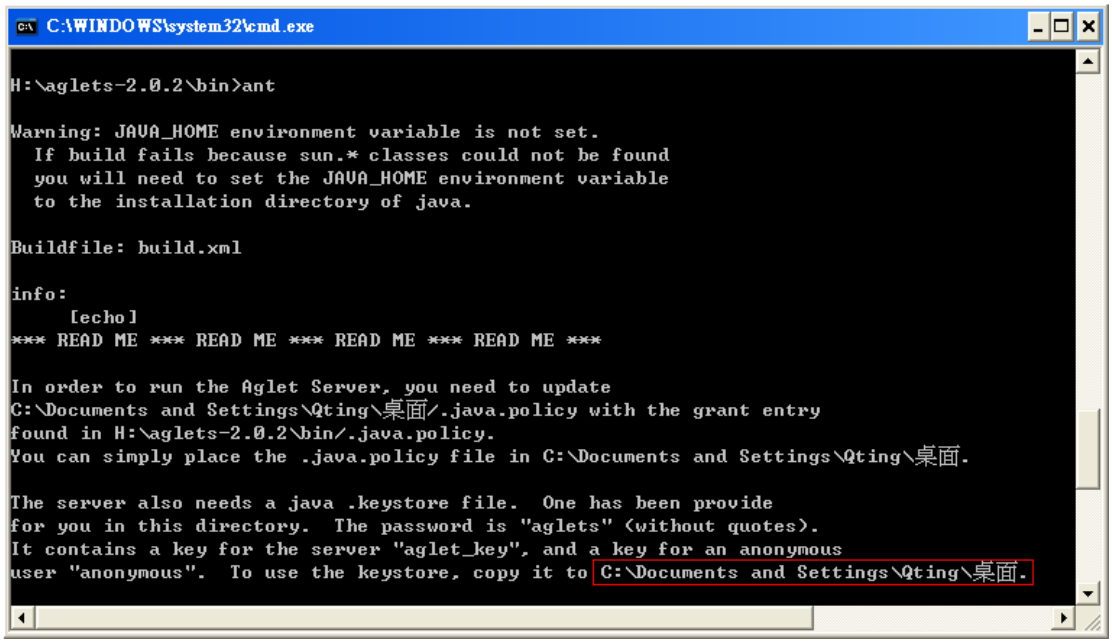


圖 11-安裝 Aglet 2.0.2

## 9. 啟動 Tahiti：

使用者可以在 MS-DOS 命令列模式下鍵入”agletsd”,

```
H:\aglets-2.0.2\bin>agletsd
```

則會進入

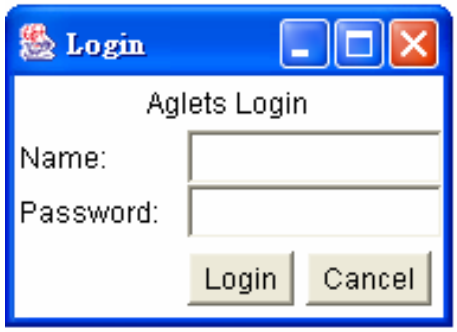


圖 12-Tahiti 登入畫面

使用者要輸入 .keystore 檔案所設定的帳號與密碼，預設情況下

為 aglet_key 和 aglets。若不想每次都要輸入帳號密碼，則可以在 MS-DOS 命令列模式下鍵入”agletsd －f ../cnf/aglets.props”。

```
H:\aglets-2.0.2\bin>agletsd -f ..\cnf\aglets.props
```

在執行 Tahiti 時會自動以 aglets.props 為設定的參考檔，並取用相關設定值，所以 Tahiti 會抓取 aglets.props 裡的帳號密碼來取用，所以使用者不須手動來輸入帳號與密碼，則可以直接開啟 Tahiti。

而 Tahiti 預設的 port 為 4434，若想改變 port 時，可以在一開始開啟 Tahiti 時所鍵入的命令裡加”-port 預設的 port”。

```
H:\aglets-2.0.2\bin>agletsd -f ..\cnf\aglets.props -port 40003
```

則會立即開啟 Tahiti。

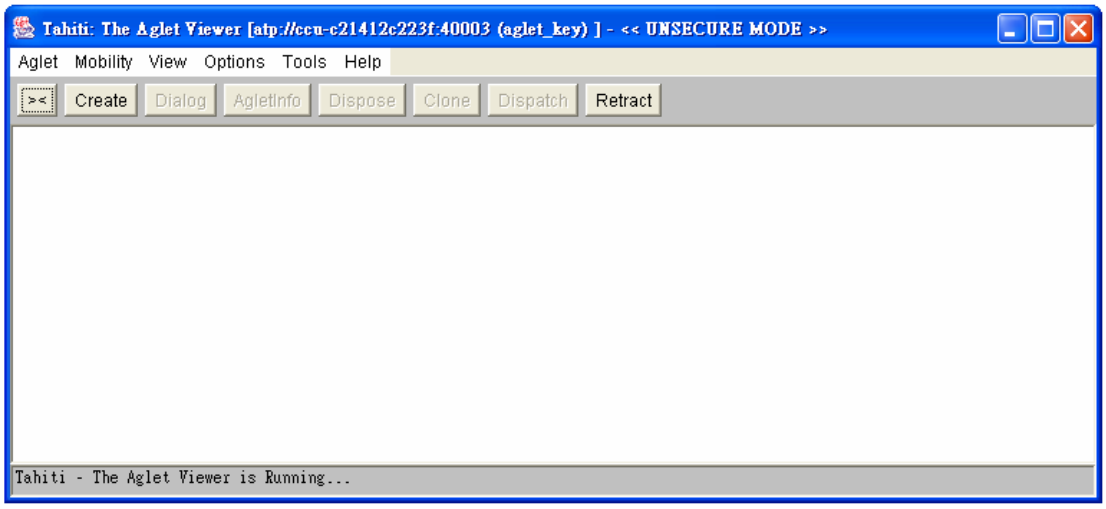


圖 13-Tahiti 開啟畫面

# 10. Tahiti 部分功能介紹

- Creation：Aglet 被 create 出來。此過程中新的 aglet 會被給予一個 identifier 然後開始初始化。在 aglet 初始化成功之後馬上開始執行。

  ✧ 程式：

```
import com.ibm.aglet.*;
public class MyAglet extends Aglet {
   public void onCreation(Object init) {
         //一個 Aglet 程式剛初始時所執行的事情，
         //並以 init 為初始化參數
   }
   public void run( ) {
         //一個 Aglet 程式的主體
   }
}
```

- Dispose: Dispose 的動作會停止 aglet 的執行並且將他從目前的 Tahiti Server 中移除。

  ✧ 程式：

```
public class MyAglet extends Aglet {
      public void run() {
         dispose();
      }
      public void onDisposing() {
         //呼叫 dispose()之後將執行此段程式
      }
   }
```

- Dispatching : Dispatching 的動作是將 aglet 從原先所在主機派送到另一台主機，在這過程中此 aglet 會從原先的主機被移出，然後被安插在目的裡，移動完成之後他會從他原先的狀態底下開始重新執行。

✧ 程式：

```
import com.ibm.aglet.event.*;
public class MyAglet extends Aglet {
  public void onCreation(Object init) {
    addMobilityListener( new
MobilityAdapter() {
          public void
onDispatching(MobilityEvent e) {
          //移動之前所做的事  }
          public void
onArrival(MobilityEvent e) {
          //到達目的地後所做的事  }
    });
 }
  public void run() {
    try{
      dispatch(targetURL);   //移動到特定
URL 的目的地
    }catch(Exception e){}
  }
 }
```

- Retraction : Retraction 的動作是將 aglet 從他目前執行的主機中移出，然後送往發出 retraction 需求的遠端主機處再開始執行。

✧ 程式：

```
import com.ibm.aglet.event.*;
public class TargetAglet extends Aglet {
  public void onCreation(Object init) {
```

```
        addMobilityListener( new
MobilityAdapter() {
            public void
onReverting(MobilityEvent e) {
                //在取回 Aglet 程式前所做的事
            }
        });
    }
 }
```

- Cloning : Aglet cloning 的動作是複製出一個和原本的 aglet 幾乎一模一樣的 aglet 出來，複製出的 aglet 和原本的差異只有 identifier 不同而已。

  ✧ 程式：

```
 import com.ibm.aglet.event.*;
   public class MyAglet extends Aglet {
      public void onCreation(Object init) {
         addCloneListener( new CloneAdapter()
{
               public void
onCloning(CloneEvent e) {
               //clone 前所做的事 }
               public void onClone(CloneEvent
e) {
               //clone 後, 另一個 Aglet 做的事
 }
               public void
onCloned(CloneEvent e) {
```

```
                //clone 後, 原始的 Aglet 做的事
    }
        });
    }
        public void run() {
            try{
                clone();
            }catch(Exception e){}
            }
    }
```

# 六、 Implementation

## 1. 實驗一

- 實驗目的：在三台不同電腦上可以利用 Aglet 平台傳送檔案，且檔案 size 並沒有大小的限制。3 台電腦(A,B,C)，3 台電腦上皆有 Aglets Server。電腦 A 準備傳送文件檔給電腦 B 與電腦 C。
- 實作流程：
  (1) 電腦 A 在 Aglet 平台上執行 MySendFileAglet Aglet。此 Aglet 主要會做到上述的動作，接下來會說明此 Aglet 執行結果。
  (2) MySendFileAglet Aglet 會自動跳到電腦 B 和儲存文件檔至電腦 B，接下來 MySendFileAglet Aglet 會再自動跳到電腦 C 以及儲存文件檔至電腦 C。
  (3) MySendFileAglet Aglet 執行結束。
- 實作結果：當 MySendFileAglet Aglet 執行結束，則電腦 B,C 皆有擁有電腦 A 所欲傳送的文件檔。
- 程式碼：

```
/* 電腦 A : atp://computer_A:40001 , 準備傳送的資料名稱 : secret_file.txt */
/* 電腦 B : atp://computer_B:40001 */
/* 電腦 C : atp://computer_C:40001 */
/* public void onCreation(Object init): */
/* 當觸發移動事件時，會執行此 function 的內容,有以下兩種情況 */
/* 1. public void onDispatching(MobilityEvent
```

```
e): */
 /*在 aglet 移動之前會執行的動作。會先把欲要傳送的檔案存到 buffer 裡 */
 /* 2. public void onArrival(MobilityEvent e): */
 /*在 alget 到達目的地後所會做的動作。新增一個檔案再把 buffer 裡的資料 */
 /* 存到檔案裡 */
 /* public void run(): */
 /* 一共會執行兩次 dispatch，第一次是 dispatch 到電腦 B ， 第二次是 */
 /* dispatch 到電腦 C */

 package Mytest;
 import com.ibm.aglet.*;
 import com.ibm.aglet.event.*;
 import com.ibm.aglet.util.*;
 import java.net.*;

 import java.awt.*;
 import java.awt.event.*;
 import java.io.*;
 import javax.swing.*;

 public class MySendFileAglet extends Aglet{
    int dispatch_num = 0 ;
    StringBuffer buf[] = new
StringBuffer[2100000];
```

```
 String
URL_addr[]={"atp://computer_A:40001",
 "atp://computer_B:40001","atp://computer_C:
40001"};
    String fdname = "secret_file.txt";
   public void onCreation(Object init){
   /* 處理 Aglet 移動時觸發的相關事件 */
    addMobilityListener(
    new MobilityAdapter() {
     public void onDispatching(MobilityEvent
e){
       /* Aglet 移動之前所做的事情 */
       RandomAccessFile output;
       File filename=new File(fdname);
       String s1;
       setText("Send :" + fdname + "  to  " +
URL_addr[dispatch_num+1]);
       /* 讀取文字檔 並存入 Stringbuffer array 裡
*/
       try{
         output =  new
RandomAccessFile(filename,"r");
        int j=0,i=0;
        buf[i] = new StringBuffer();
        dispatch_num ++ ;
        while((s1=output.readLine())!=null){
          buf[i].append(s1+"\n");
          j++;
          if(j>=1000){
```

```
            buf[++i] = new StringBuffer();
            j=0;
           }
        }
        buf[++i] = new StringBuffer();
       }
       /*  讀取文字檔存入 Stringbuffer array 結束
*/
       catch(IOException ioex) {
         System.out.println("read file
error");
       }
       }


    public void onArrival(MobilityEvent e){
       /* Aglet 到達目的地後所做的事  */
          RandomAccessFile input;
       File filename=new File(fdname);
       setText("Recive File : " + fdname + "
from :" +URL_addr[dispatch_num-1]);
         /* create 一個新檔案，再把 Stringbuffer
array 寫入此檔案 */
         File filename=new File(fdname);
         {
           input =  new
RandomAccessFile(filename,"rw");
           int k=0;
```

```
while((buf[k].toString()).length()!=0) {

 //System.out.println(buf[k].toString());
        k++;

input.writeBytes(buf[k].toString());
         }
       }
     /* 把 Stringbuffer array 寫入檔案結束 */
       catch (IOException ioex){
        System.out.println("create file
error");
      }
    }
   }
  };
 }

 public void run()
 {
   /* dispatch_num 代表 dispatch 的次數 */
    if(dispatch_num < 2 )
    {
    try{
       /* Aglet 由電腦 A 傳至電腦 B */
       if(dispatch_num==0) {
          URL desURL = new
URL(URL_addr[dispatch_num+1]);
          dispatch(desURL);
```

```
            }
            /* Aglet 由電腦 B 傳至電腦 C */
            else{
                URL desURL = new
    URL(URL_addr[dispatch_num+1]);
                dispatch(desURL);
            }
        }
        catch(Exception e)  {
            System.out.println("dispatch error");
        }
       }
      }
     }
```

● 執行結果圖；

✧ 電腦 A create MySendFileAglet，準備把 secret_file.txt 傳送到電腦 B。

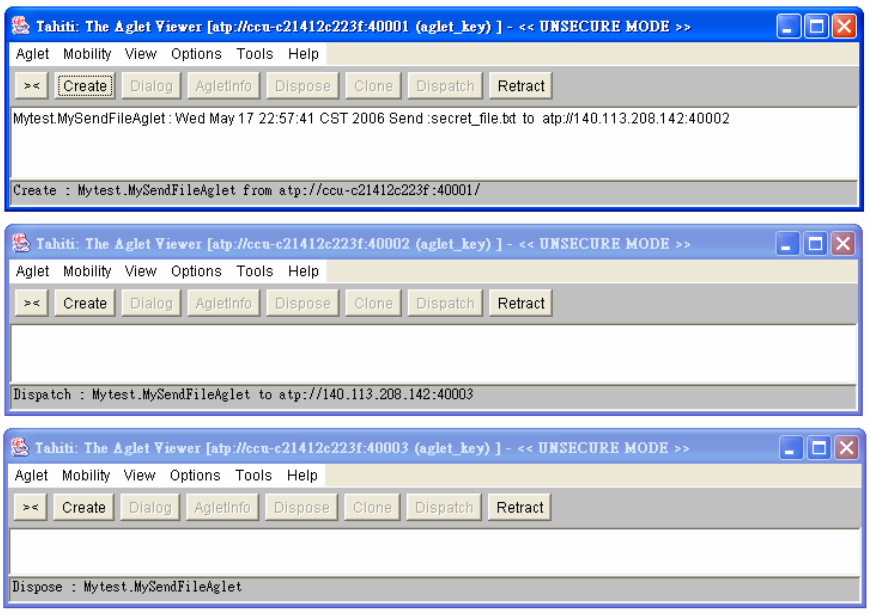


圖 14-實驗一 (1)

✧ 電腦 B 收到 MySendFileAglet，然後儲存檔案以及把 MySendFileAglet 傳送給電腦 C


圖 15-實驗一 (2)

✧ 電腦 C 收到 MySendFileAglet，然後儲存檔案


圖 16-實驗一 (3)

✧ 執行成功後，三台電腦皆有 secret_file.txt。

● 遇到的困難與解決方式：

一開始我們先在 Java 環境下，撰寫一個可以開啟檔案的程式，然後再把程式移至 aglet 伺服器上執行時，卻發生 Security Problem。


圖 17-實驗一 (4)

而因為在 Java 下可以開啟檔案，所以一開始我們推論應該是 Tahiti 伺服器的預設 security 為不可以存取檔案或圖檔，只能接受 show 一些簡單的訊息。於是更改 .\aglets-2.0.2\cnf\aglets.props 設定檔裡面的 aglets.secure 參數，原本預設為 true，更改為 false。

```
55 # (optional) If false, every activities of aglets in the server
56 #       will be allowed.
57 # default: true
58 aglets.secure=false
```

接下來開啟 Tahiti server 時，必須在 MS-DOS 命令列模式下鍵入”agletsd －f ../cnf/aglets.props “。


代表開啟的 Tahiti server 會依照 aglets.props 裡的參數設定相關的環境，於是就可以存取檔案或圖檔。

## 2. 實驗二

- 實驗目的：設計一個 Aglet 程式，可以自動移動到其他節點，最後回到 home 節點
- 實作流程：
  (1) 開啟三個 Aglet 預設的 server，分別使用 port 40000 為 home 節點，port 40001 為 hop1 節點，port 40002 為 hop2 節點。
  (2) 於 home 節點建立我們的 Aglet 程式之後，程式會從 home 節點傳送到 hop1 節點，三秒後傳送到 hop2 節點，接下來又傳回到 home 節點。在傳送過程中不需要我們手動 dispatch 程式
- 實作結果：程式能夠自動地從 home 移動到 hop1、hop2，最後回到 home，而不需要我們手動 dispatch。
- 程式碼：
  - ✧ 當我們在 Aglet server 上按下 create 之後，Aglet 便會執行所建立的 class 內的 onCreation()方法來做一些初始化的動作。
  - ✧ 利用 SimpleItinerary 類別可以讓我們簡單的將程式傳送到其他節點去，並且可以根據我們的需求自行定義所需要發送的訊息，以讓接收端根據所接收到的訊息作出相對的回應。而 Aglet 是以字串的方式來定義每個節點的 IP、port。
  - ✧ 在 Aglet 裡面預設了一個訊息處理程序 handleMessage (Message msg)，每當接收訊息的時候，Aglet server 便會執行這個程序，並依照使用者定義的方式來處理訊息。

```
public class AutoTraverse extends Aglet
{
   /* 定義各節點的位址.端口 */
   String home_addr="atp://localhost:40000";
   String hop1_addr="atp://localhost:40001";
   String hop2_addr="atp://localhost:40002";

   /* 定義一個可以將程式和訊息傳送到其他節點的類別
物件 */
   SimpleItinerary itinerary;

   public void onCreation(Object init) {
     setText("home...go to hop1 after 3 sec");
//秀出 home 訊息
      waitMessage(3000); // 等待 3 秒鐘

      itinerary = new SimpleItinerary(this);
      try {
         itinerary.go(hop1_addr, "hop1"); //
送到 hop1 去
      } catch (Exception ex) {
         ex.printStackTrace();
      }
   }
   /* 訊息處理函式，根據訊息內容來處理相對應的事 */
   public boolean handleMessage(Message msg)
{
```

```
        if (msg.sameKind("hop1")) {
            // job at the first place
            setText("hop1...go to hop2 after 3
sec"); //秀出 hop1 訊息
            waitMessage(3000); // 等待 3 秒鐘
            try {
               itinerary.go(hop2_addr, "hop2");
//送到 hop2 去
            } catch (Exception ex) {
               ex.printStackTrace();
            }
        } else if (msg.sameKind("hop2")) {
            // job at the second place
            setText("hop2...go back home after 3
sec"); //秀出 hop2 訊息
            waitMessage(3000); // 等待 3 秒鐘
            try {
               itinerary.go(home_addr, "home");
//送到 home 去
            } catch (Exception ex) {
               ex.printStackTrace();
            }
        } else if (msg.sameKind("home")) {
            // job at the third place
            setText("at home...dispose after 3
sec"); //秀出 home 訊息
            waitMessage(3000); // 等待 3 秒鐘
            dispose(); //刪除程式
```

```
        } else return false;
        return true;
    }
}
```

- 執行結果圖；
  - ✧ 按下 Aglet server 上的 create 按鈕，並建立我們寫好的程式


圖 18-實驗二 (1)

  - ✧ 程式建立後會顯示一些訊息，並於三秒後移動到 hop1 節點

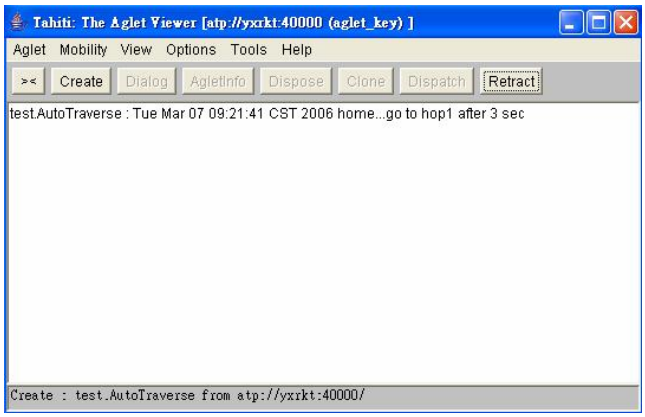


圖 19-實驗二 (2)

✧ 移動到 hop1 節點後，會顯示訊息並於三秒後移動到 hop2 節點

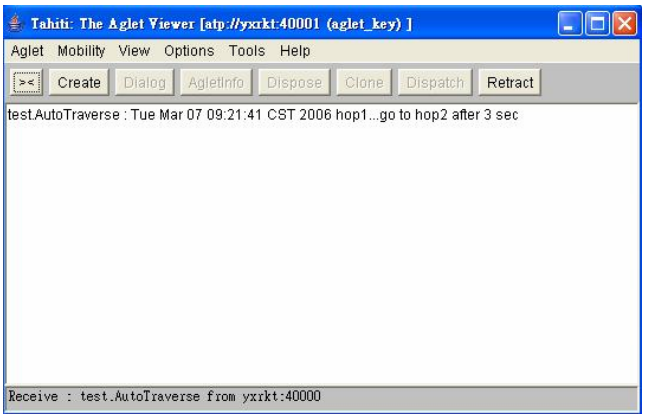


圖 20-實驗二 (3)

✧ 移動到 hop2 節點後，會顯示訊息並於三秒後回到 home 節點

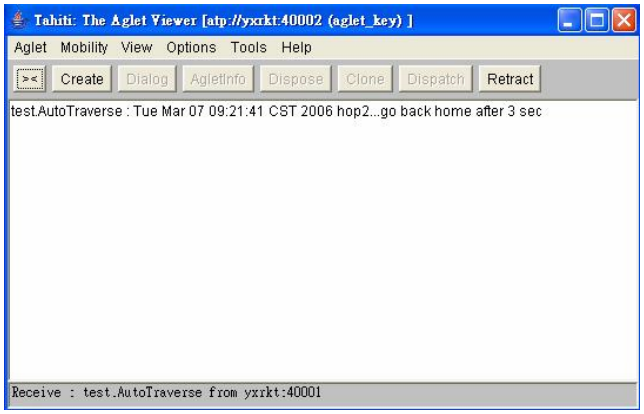


圖 21-實驗二 (4)

✧ 程式傳回到 home 節點後會於三秒後自動中斷

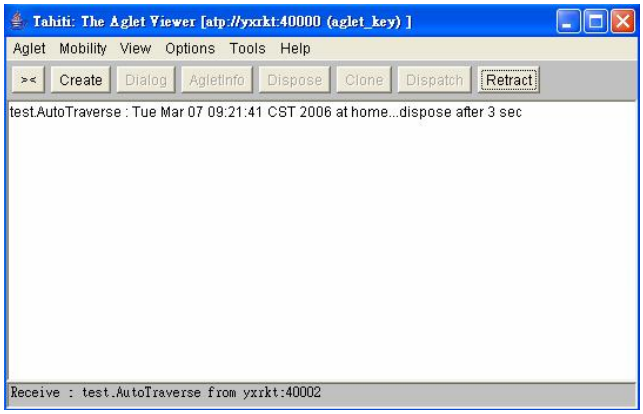


圖 22-實驗二 (5)

● 遇到的困難與解決方式：

程式在編譯上會引用到某些類別(com.ibm.agletx.*)，但是這些類別在預設的函式庫(aglets-2.0.2\lib\ aglets-2.0.2.jar)內並沒有加進去，因此我們自己把那些類別加進去並重新整合為一個新的函式庫 lib.jar。

# 3. 實驗三

- 實驗目的：測試 Aglet 所能傳送與顯示的最大訊息長度。
- 實作流程：
  (1) 開啟兩個 Aglet 預設的 server，分別使用 port 40000 為 home 節點，port 40001 為 hop1 節點。
  (2) 於 home 節點建立我們的 Aglet 程式之後，程式會從 home 節點傳送到 hop1 節點，並在 hop1 依序顯示字串 A、字串 B、字串 A+字串 B。
- 實作結果：
  - ✧ 如果我們定義一個字串長度超過 65536 bytes 的字串的話，在編譯程式的時候便會產生錯誤，因此無法定義一個太長的字串，而必須將他以 65536 bytes 為單位來切割。
  - ✧ 兩個字串 A 與 B，A 為 60k bytes，B 為 40k bytes，單獨顯示 A 或 B 並不會有錯誤。但 A+B 超過 setText()所能接受的最大長度，因此即使接收端將 A 與 B 重新組合，還是無法顯示出來。所以此最大長度為 API 設定所能印出的最大訊息長度，而不是 Aglet 所能傳送的最大訊息長度。
- 程式碼：

```
public class DisplayAglet2 extends Aglet {
    /* 定義各節點的位址.端口 */
    String hop1_addr="atp://localhost:40001";
    /* 定義一個可以將程式和訊息傳送到其他節點的類別
物件 */
    SimpleItinerary itinerary;
```

```
 /* 定義字串 A 60k bytes */
    String strA =
    "01234567890123456789012345678901234567890123456789" +  // 50 bytes
    "01234567890123456789012345678901234567890123456789" +
  (中間省略)
 "01234567890123456789012345678901234567890_strA_end" ;
    /* 定義字串 B 40k bytes */
 String strB =
    "01234567890123456789012345678901234567890123456789" +  // 50 bytes
    "01234567890123456789012345678901234567890123456789" +
 (中間省略)
 "01234567890123456789012345678901234567890_strB_end" ;
    /* 程式建立後的初始化動作 */
    public void onCreation(Object init) {
       setText("go to hop1 after 2 seconds...");
       waitMessage(2000); // 等待 2 秒鐘

       itinerary = new SimpleItinerary(this);
       try {
          itinerary.go(hop1_addr, "hop1"); //
```

送到 hop1 去

```
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    /* 訊息處理函式，根據訊息內容來處理相對應的事 */
    public boolean handleMessage(Message msg)
{
        if (msg.sameKind("hop1")) {
            setText("show strA after 2
seconds...");  //秀出 server 訊息
            waitMessage(2000); // 等待 2 秒鐘
            setText(strA);  //秀出 server 訊息(字串
A)
            waitMessage(2000); // 等待 2 秒鐘

            setText("show strB after 2
seconds...");
            waitMessage(2000); // 等待 2 秒鐘
            setText(strB);  //秀出 server 訊息(字串
B)
            waitMessage(2000); // 等待 2 秒鐘

            setText("show strA + strB after 2
seconds...");
            waitMessage(2000); // 等待 2 秒鐘
            setText(strA+strB);  //秀出 server 訊
```

```
息(字串 A+字串 B)
            }
            else return false;
            return true;
        }
    }
```

● 執行結果圖；

✧ 程式建立後，兩秒後回傳送到 hop1 去

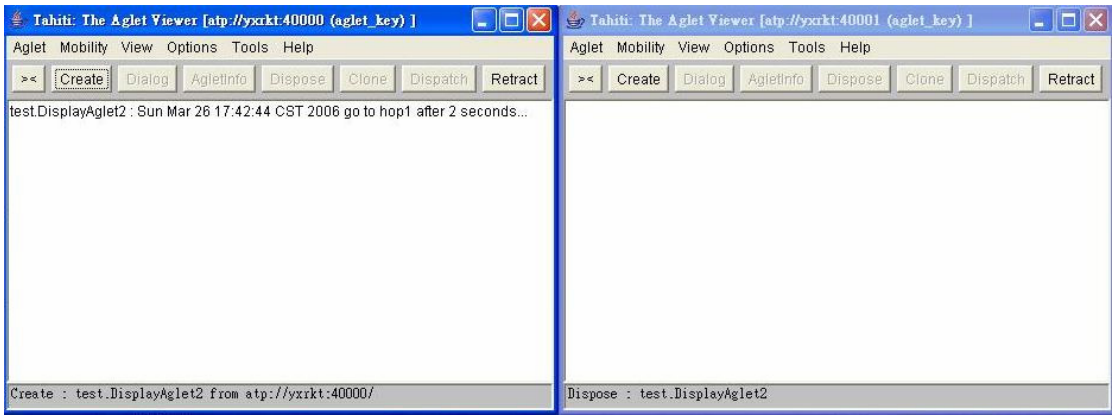


圖 23-實驗三 (1)

✧ 傳送到 hop1 後，顯示訊息說明兩秒鐘後印出字串 A


圖 24-實驗三 (2)

✧ 印出字串 A，長度 60k bytes


圖 25-實驗三 (3)

✧ 顯示訊息說明兩秒後印出字串 B


圖 26-實驗三 (4)

✧ 印出字串 B，長度 40k bytes


圖 27-實驗三 (5)

✧ 顯示訊息說明兩秒後印出字串 A 和字串 B(B 附加於 A 之後)

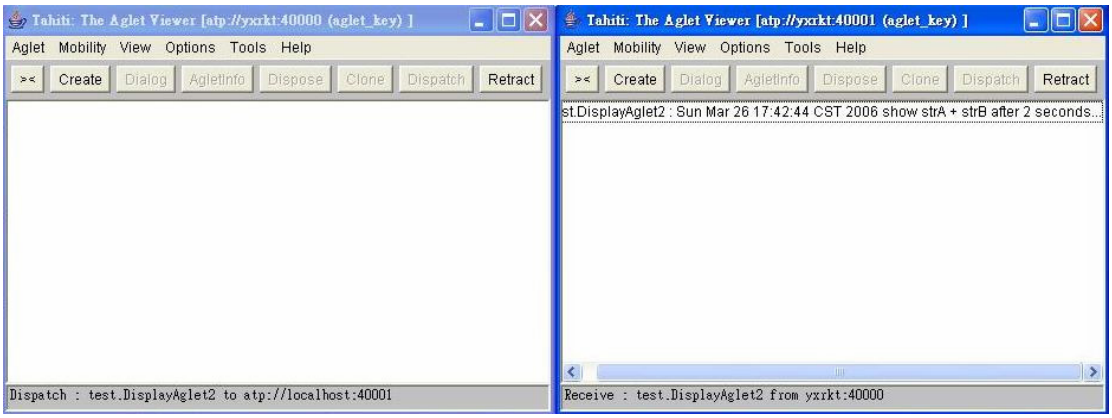


圖 28-實驗三 (6)

✧ 印出字串 A 和字串 B，總字串長度 100k bytes，但是卻無法顯示出來


圖 29-實驗三 (7)

● 遇到的困難與解決方式：

一開始在定義字串長度時無法定義太長的字串，只要所定義的字串長度超過 65536 bytes 的話就會在編譯程式時產生錯誤，如下圖所示，因此只能將長字串做切割，並於接收端合併起來顯示。

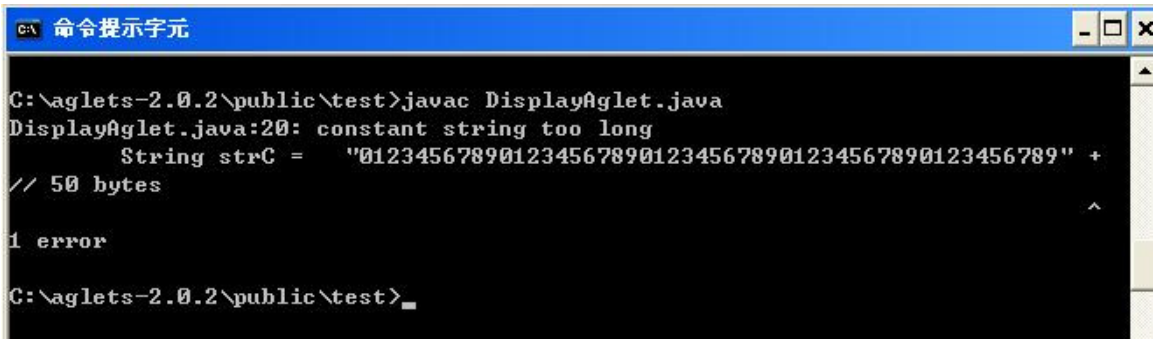


圖 30-實驗三 (8)

但是將字串合併之後，卻發現 setText()函式無法正常顯示訊息，因此下一步的解決方式便要想辦法讓程式能夠正常顯示長字串，我們想到的辦法是另外開一個新的對話視窗來顯示訊息。

# 4. 實驗四

- 實驗目的：在 Aglets server 上產生新的對話視窗顯示過長的訊息，使傳送文字長度不受限制
- 實作流程：
  (1) 開啟兩個 Aglet 預設的 server，分別使用 port 50000 為 home 節點，port 50001 為 hop1 節點。
  (2) 於 home 建立我們的 Aglets 程式後，程式會從 home 節點移動至 hop1 節點，並產生一個對話視窗，將所傳送的所有訊息全部顯示出來
- 實作結果：可以正常的顯示所有傳送的長字串而不會有錯誤。
- 程式碼：
  - ✧ Aglet 以字串的方式來定義每個節點的 IP、port：

```
String hop1_addr="atp://localhost:50001";
// 定義各節點的位址.端口
```

  - ✧ 我們自行定義的對話視窗類別物件的宣告：

```
MyDialog my_dialog = null;
```

  - ✧ 當我們在 Aglet server 上按下 create 之後，Aglet 便會執行所建立的 class 內的 onCreation()方法來做一些初始化的動作。在我們的程式碼內，當 create 之後 Aglet 程式會於兩秒鐘之後傳送到 hop1 去，並發出”hop1”的訊息。程式碼如下：

```
public void onCreation(Object init) {
        setText("go to hop1 after 2
seconds...");
        waitMessage(2000); // 等待 2 秒鐘
```

```
        itinerary = new
    SimpleItinerary(this);
        try {
            itinerary.go(hop1_addr, "hop1");
    //送到 hop1 去
        } catch (Exception ex) {
            ex.printStackTrace();
        }
      }
```

✧ 在 Aglet 裡面預設了一個訊息處理程序,每當接收訊息的時候，Aglet server 便會執行這個程序，並依照使用者定義的方式來處理訊息。下面是我們定義的訊息處理方法：

```
 public boolean handleMessage(Message msg)
{
        // 只有在 hop1 的時候才秀出對話視窗
        if (msg.sameKind("hop1")) {
            createGUI();
  // Create GUI to control this Aglet
            printString();
  // 印出所有字串
        }
       // 按下 server 上的"dialog"按鈕時秀出對話
視窗
        if (msg.sameKind("dialog")) {
            dialog(msg);
        }
        else  return false;
```

```
        return true;
    }
```

✧ 當 Aglet 程式移動到 hop1 後，server 便會執行這個訊息處理程序，然後判斷接收的訊息，如果接收的訊息符合字串 “hop1”，那麼 hop1 便會執行相對的副函式 createGUI()和 printString()。不只有在 Aglet 程式移動時才會發出訊息，當使用者按下 server 定義的按鈕時也會發出相對應的訊息，因此當我們按下”dialog”按鈕時，hop1 便會秀出對話視窗，這樣做的好處是假若我們不小心把對話視窗關掉了,這樣就不用再於 home 重新建立一次 Aglet 程式了。

✧ 以下是 createGUI()副函式的程式碼：

```
protected void createGUI() {
        my_dialog = new MyDialog();
// 配置對話視窗記憶體
        my_dialog.pack();

my_dialog.setSize(my_dialog.getPreferred
Size()); //設定視窗為預設大小
        my_dialog.setVisible(true);
// 顯示視窗
    }
```

✧ 以下是 printString()副函式的程式碼：

```
public void printString() {
        setText("Print string A...");
// 在 hop1 server 印出訊息
        my_dialog.textArea.append(strA);
```

```
  // 在對話視窗上印出字串 A
        my_dialog.textArea.append("End of
  strA. 60k bytes printed.\n");
        my_dialog.textArea.append("\n");

        waitMessage(2000);  // 等待兩秒鐘

        setText("Print string B...");
    // 在 hop1 server 印出訊息
        my_dialog.textArea.append(strB);
  // 在對話視窗上印出字串 B，附加在字串 A 的後面
        my_dialog.textArea.append("End of
  strB. 40k bytes printed.\n");
     }
```

✧ dialog()副函式的程式碼：

```
public void dialog(Message msg) {
     // check and create a dialog box
     if (my_dialog == null) {
        my_dialog = new MyDialog();
        my_dialog.pack();

  my_dialog.setSize(my_dialog.getPreferred
Size());
     }

     // show the dialog box
     my_dialog.setVisible(true);
  }
```

✧ 定義的視窗類別物件 MyDialog：

```
public class MyDialog extends Frame implements
WindowListener, ActionListener {
   /* 宣告使用者介面物件 */
   private Button hide = new Button("HIDE");
// 隱藏視窗的按鈕
   public TextArea textArea = new TextArea(15,
50);  // 顯示字串的區域

   /* Constructor，當視窗物件被建立時就開始產生對
話視窗的元件 */
   MyDialog() {
      layoutComponents();
      addWindowListener(this);
      hide.addActionListener(this);
   }

   /* 當我們按下對話視窗上的 HIDE 按鈕後，便將視窗隱
藏 */
  public void actionPerformed(ActionEvent ae)
{
      if
("HIDE".equals(ae.getActionCommand())) {
         setVisible(false);
      }
   }
```

```
  /* 編排所有的視窗元件 */
  private void layoutComponents() {
     // Layouts components
     GridBagLayout grid = new
GridBagLayout();
     GridBagConstraints cns = new
GridBagConstraints();
     setLayout(grid);

     cns.weightx = 0.0;
     cns.weighty = 0.0;
     cns.fill =
GridBagConstraints.HORIZONTAL;
     cns.insets = new Insets(2, 2, 2, 2);

     // 用來顯示字串的文字區域
     cns.gridwidth =
GridBagConstraints.REMAINDER;
     cns.fill = GridBagConstraints.BOTH;
     grid.setConstraints(textArea, cns);
     add(textArea);

     // HIDE 按鈕
     Panel p = new Panel();
     cns.fill = GridBagConstraints.NONE;
     cns.gridheight = 1;
     grid.setConstraints(p, cns);
     add(p);
     p.setLayout(new FlowLayout());
```

```
        p.add(hide);
    }

    /* 處理視窗事件 */
  public void windowActivated(WindowEvent we)
{}
    public void windowClosed(WindowEvent we) {}
    public void windowClosing(WindowEvent we)
// 若我們按下對話視窗上的
      {   dispose();     }                    //
關閉按鈕就刪除視窗物件
    public void windowDeactivated(WindowEvent
we) {}
    public void windowDeiconified(WindowEvent
we) {}
  public void windowIconified(WindowEvent we)
{}
    public void windowOpened(WindowEvent we) {}
}
```

- 執行結果圖；

✧ create 之後會顯示訊息，並於兩秒後移動到下一個節點


圖 31-實驗四 (1)

✧ 移動到下一個節點後馬上產生對話視窗，並顯示出字串 A

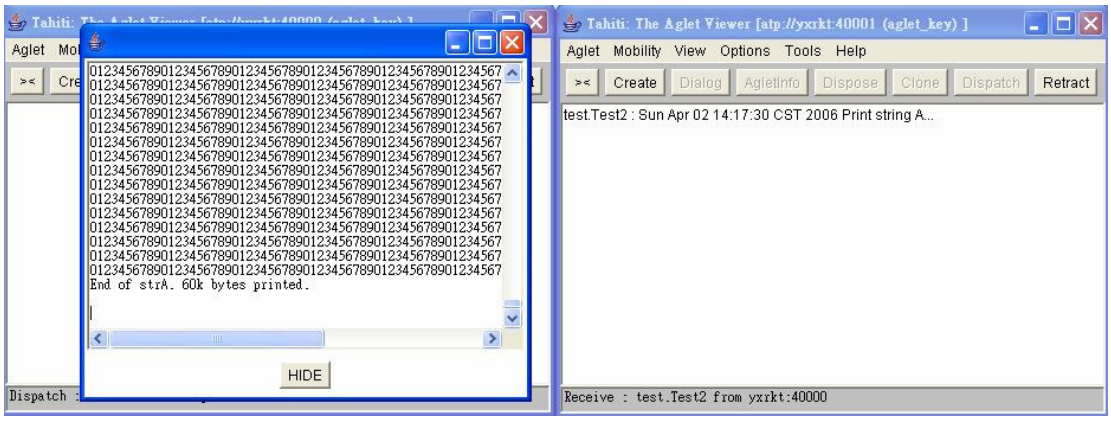


圖 32-實驗四 (2)

✧ 經過兩秒鐘之後，再顯示出字串 B，附加於字串 A 之後

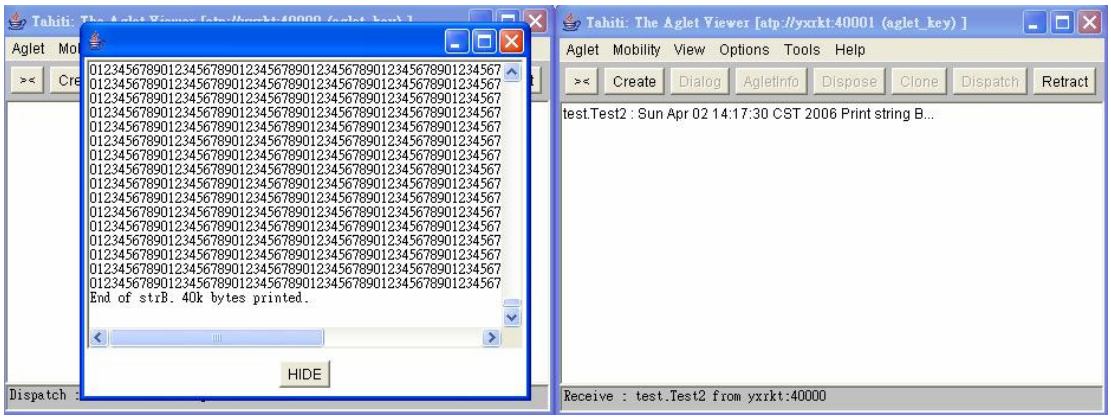


圖 33-實驗四 (3)

● 遇到的困難與解決方式：

一開始在實驗的過程中，原本以為已經順利完成了，但是將程式碼移動到另一台電腦測試時，卻發現 Aglet 程式無法順利移動到其他節點去，後來經過了多次的實驗，推測問題是起因於防毒軟體的防火牆將 Aglet 程式所使用的 port 阻擋掉，而造成 Aglet server 無法順利

送出與接收訊息。

我們的解決方式是在防毒軟體的設定上新增一些例外的 port，讓防毒軟體不檢測經過這些 port 的封包，如此 Aglet server 便能順利的收發訊息封包。

# 七、 結果與討論

本期的研究首要目的在於學習 Mobile Agent System，了解 MA 的運作原理與方法、培養 MA 人才。進一步，我們設定了許多評估 MA 系統的重要指標，再針對各項指標分析目前已開發之各種 MA 系統的優缺點。在這樣的評估過程中，我們可以了解開發 MA 系統時所面對的各種要求。當選定了適合的 Aglet Mobile Agent System 後，對 Aglet 做更深層的研究與探討，不只是單純地使用其使用者介面，透過解析原始碼的過程，Aglet 的運作架構、原理，以及 Aglet 與 JVM 之間部份互動的設計…等，都有詳實的了解。

本研究計劃下一階段之研究在於將連接 Mobile Agent System 與 Distributed Sensor Network。團隊將會以本期研究的 MA 系統搭配 DSN 並結合兩方之優勢將可以透過行動代理人解決 DSNs 的安全性問題並提升 Sensor Nodes 的存活率。預計我們將選定一合適的 DSNs 情境，透過實作的過程可以確實地評估 MA+DSNs 的相互增益。

# 八、 參考文獻


[01]http://www.trl.ibm.com/aglets/ (Aglet )

[02]http://www.recursionsw.com (Voyager)

[03]http://sourceforge.net/projects/zeusagent/ (ZEUS)

[04] 竇其仁, 林志敏, 林正敏著, 網路代理人(Network Agents), 知城, ISBN 9867231392, 2006.

[05]http://aglets.sourceforge.net/doc/overview-summary.html (aglets Development Kit)

[06]http://www.mclab.iecs.fcu.edu.tw/agenttech/TextBook1.asp

[07]Agent TCL. http://agent.cs.dartmouth.edu/

[08]Aglets. http://www.trl.ibm.co.jp/aglets.

[09]Baldi, M., Gai, S., Picco, G. P., 1997 , "Exploiting code Mobility in decentralized and Flexible Network Management," *Proceedings of the First International Workshop on Mobile Agent (MA'97)*, pp.13-26.

[10]C. G. Harrison, D. M. Chess, and A. Kershenbaum, "Mobile agents: are they a good idea?," *Technical Report RC 19887*, IBM Thomas J. Watson Research Center, March 1995. http://www.research.ibm.com/massive/mobag.ps.

[11]Camarinha-Matos, L.M., Vieira, W., "Intelligent mobile agents in elderly care," Robotics and Autonomous Systems 27, 1999, pp. 59-75.

[12]Chavez, A., Maes, P., Kasbah, "An Agent MaketPlace for buying and selling Goods," *Proceedings of the First International conf. Practical Application of Intelligent Agents and Multi-Agent Technology*, 1996, pp.75-90.

[13]Chess, D., Harrison, C., Kershenbaum, A., Mobile Agents: Are They

a Good Idea, 1995.

[14] Concordia, http://www.meitca.com/HSL/Projects/Concordia

[15] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Communications of the ACM*, vol. 42, no. 3, pp. 88–89, March 1999.

[16] D. Hearn and M.P. Baker, Computer Graphics, C Version, Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 1997.

[17] D. Milojicic, "Mobile agent applications," *IEEE Concurrency*, pp. 80–90, 1999.

[18] D. N. Jayasimha, S. S. Iyengar, and R. L. Kashyap, "Information integration and synchronization in distributed sensor networks," *IEEE Trans. Syst., Man, Cybern.*, vol. 21,pp. 1032–1043, 1991.

[19] Douglas W. Stevenson, Network Management White Paper, 1996, http://netman.cit.buffalo.edu/Doc/Dstevenson.html.

[20] E. Cho, S. S. Iyengar, K. Chakrabarty, and H. Qi, "A new fault tolerant sensor integration function satisfying local condition," Submitted for publication, 2000.

[21] Elmasri, Ramez, Navathe, Shamkant B., Fundamentals of Database Systems, Addison Wesley, 1994.

[22] FIPA, FIPA rationale, URL: http://drogo.cselt.it/fipa/fipa_rationale.htm, 1996.

[23] Fuggeta, A. Picco, G. P., Vigna, G, "Understanding code mobility," *IEEE Transactions on software Engineering*, 24(5), pp. 346-361, 1998.

[24] G.Mergen, Q. Zhao, and L.Tong, "Sensor networks with mobile agents: Energy and capacity considerations," *IEEE Journal on Selected Areas in Communications*, July 2003.

[25] Gavalas, D., Greenwood, D., Ghanbari, M., O'Mahony, M., "An Infrastructure for Distributed and Dynamic Network Management

based on Mobile Agent technology," *Proceedings of the IEEE International Conference on communications (ICC'99)*, pp. 1362-1366, 1999.

[26] Gavalas, D., Greenwood, D., Ghanbari, M., O'Mahony, M., "Advanced network monitoring applications based on mobile/intelligent agent technology," *Computer Communications*, vol. 23, pp. 720-730, 2000.

[27] Goldszmidt, G., "On distributed systems management," *Proceedings of the Third OBM/CAS Conference*, 1993, http://www.cs.columbia.edu/~german/papers.html.

[28] H. Qi, S. S. Iyengar, and K. Chakrabarty, "Multiresolution data integration using mobile agents in distributed sensor networks," *IEEE Transactions on SMC: C*, 2000.

[29] Hairong Qi, Xiaoling Wang, S. Sitharama Iyengar, Krishnendu Chakrabarty, "Multisensor Data Fusion in Distributed Sensor Networks Using Mobile Agents," *In Proc. Intl. Conf. Information Fusion*, pages 11-16, August 2001.

[30] Hamid Gharavi, Sikantap. Kumar, "Special Issue on Sensor Networks and Applications," *Proceedings of the IEEE*, Vol. 91, No. 8, pp.1151-1153, August 2003.

[31] IBM Company, Aglet, http://www.trl.ibm.com/aglets

[32] J. Kay, J. Etzl, G. Rao, and J. Thies, "Atl postmaster: a system for agent collaboration and information dissemination," *In Proceedings of the 2nd International Conference on Autonomous Agents*, pp. 338–342, Minneapolis, MN, 1998.

[33] J. S. Wong and A. R. Mikler, "Intelligent mobile agents in large distributed autonomous cooperative systems," *Journal of Systems and Software*, 47(2):75–87, 1999.

[34] Joanne L. S. H., Jairo A.G., 1998, "The Application of Web-Based

Technologies on Integrated Network Management," *IEEE*, 1094-1098.

[35]K. N. Ross, R. D. Chaney, G. V. Cybenko, D. J. Burroughs, and A. S. Willsky, "Mobile agents in adaptive hierarchical bayesian networks for global awareness," *In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 2207–2212. IEEE, 1998.

[36]Knoll and J. Meinkoehn, "Data fusion using large multi-agent networks: an analysis of network structure and performance," *In Proceedings of the International Conference on Multisensor Fusion and Integration for Intelligent Systems* (*MFI*), pages 113– 120, Las Vegas, NV, Oct. 2-5 1994.

[37]L. Lamport, "Synchronizing time servers," *Technical Report Technical Report 18*, Digital System Research Center, 1987.

[38]L. Prasad, S. S. Iyengar, and R. L. Rao, "Faulttolerant sensor integration using multiresolution decomposition," *Physical Review E*, 49(4):3452–3461, April 1994.

[39]L. Prasad, S. S. Iyengar, R. L. Kashyap, and R. N. Madan, "Functional characterization of sensor integration in distributed sensor networks," *IEEE Trans. Syst., Man, Cybern., SMC*, vol. 21, 1991.

[40]L. Tong, Q. Zhao, and S. Adireddy, "Sensor networks with mobile agents," *in Proc. 2003 Military Communications. Intl. Symp.*, Boston, MA, Oct. 2003.

[41]Lange, Danny, B., Oshima, Mitsuru, Programming and Deploying JAVA Mobile Agents with Aglets, Addison Wesley, 1998.

[42]Luiz A.G. Oliveira, Paul C. Oliveira and Eleri Cardozo, "An Agent-Based Approach for Quality of Service Negotiation and Management in Distributed Multimedia Systems," *Proceedings of*

*the First International Workshop on Mobile Agent (MA'97)*, pp.1-12, 1997.

[43]M. Hattori, N. Kase, A. Ohsuga, and S. Honiden, "Agent-based driver's information assistance system," *New Generation Computing*, 17(4):359–367, 1999.

[44]M. Lutz and D. Ascher. Learning Python. O'Reilly, April 1999.

[45]Magedanz, T., Rothermeln, K., Krause, S., "Intelligent agents：an emerging technology for next generation telecommunications?," *Proc. IEEE INFOCOM' 96, Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation*, pp.464-472, 1996.

[46]Martin-Flatin J. P., Znaty S., Hubaux J. P., A survey of Distributed Network and System Management Paradigms, v2, 1998.

[47]Milojicic, D., Breugst, M., Busse, I., Campbell, J., Covaci, S. Friedman, B., Kosaka, K., Lange, D. Ono, K., Oshima, M., Tham, C., Virdhagriswaran, S., White, J., "MASIF: The OMG Mobile Agent System Interoperability Facility," *Second International Workshop, (MA'98)*, *Lecture Notes in Computer Science, 1,477*, (Stuttgart, Germany: Springer), pp. 50-67, 1998.

[48]Mole, http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html

[49]P. Dasgupta, N. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. Magnet, "Mobile agents for networked electronic trading," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 4, pp.509–525, 1999.

[50]Papaioannou, T., Edwards, J., "Using mobile agents to improve the alignment between manufacturing and its IT support systems," *Robotics and Autonomous system*, vol. 27, pp.45-57, 1999.

[51] Rabelo, R., Luis M. Crmarinha-Matos, Hamideh Afsarmanesh, "Multiagent perspectives to agile scheduling," *L.M. Intelligent Systems for Manufacturing: Multi-agent Systems and Virtual Enterprises*, (Kluwer Academic Publishers, Dordrecht), pp.51-66, 1998.

[52] S. Shamai(Shitz) and A. D. Wyner, "Information-theoretic considerations for symmetric cellular, multipleaccess fading channels–Parts I and II," *IEEE Trans. Inform. Theory*, vol. 43, no. 6, pp. 1877–1911, Nov. 1997.

[53] Stallings William, "SNMP, SNMPv2 and RMON," *Practical Network Management*, (Addison Wesley) Stallings William, 1993, *Network Management*, (IEEE Computer Society Press) , 1996.

[54] T. Oates, M. V. N. Prasad, and V. R. Lesser, "Cooperative information-gathering: a distributed problem solving approach," *IEE Proceedings - Software Engineering*, vol. 144, no. 1, pp.72–88, 1997.

[55] Telescript. http://www.generalmagic.com.

[56] W. Caripe, G. Cybenko, K. Moizumi, and R. Gray, "Network awareness and mobile agent systems," *IEEE Communications Magazine*, pages 44–49, 1998.

[57] Wong, David, Paciorek, Noemi, and Moore, Dana, "Java-based Mobil Agents," *Communications of the ACM*, vol. 42, no. 3, pp.92-102, 1999.

[58] Wooldridge, M., Jennings, N. R., "Agent theories, architectures and languages: A survey, Intelligent Agents," *Lecture Notes in Artificial Intelligence, vol.890*, (Springer, Berlin), pp.1-39, 1995.

[59] Zapf, M., Herrmann, K., Geihs, K., "Decentralized SNMP Management with Mobile Agents," *IEEE, Intelligent and Mobile Agents*, vol. 6, pp. 623-635, 1999.