網路精靈伺服器之設計

Design of a Server for Mobile Agents

連耀南（Yao-Nan Lien），尹衍林（Yenlin Yin），詹國群（Tony Chan）
國立政治大學資訊科學系
（Department of Computer Science，National Chengchi University）

摘要

通訊品質不佳，頻寬太低，以及價格太高是行動資訊服務網路中所必須解決的問題之一。而網路精靈代表使用者遊走於網路中執行使用者事先委託的任務，正是解決此困難問題的一項技術。政大資科系的一個研究群正嘗試研究與製作一個能支援網路精靈的行動資訊服務系統—名為觔斗雲，而本文旨在介紹觔斗雲系統中的網路精靈伺服器的設計。

1. 行動資訊服務系統

網際網路上豐富的資訊資源提供了未來資訊化社會中使用者隨手可得的資訊寶藏。除了高速網路與資料庫的建立以外，如何使這種資訊資源也能很方便的提供給行動中的使用者，使得資訊資源眞正的無遠弗界，讓使用者隨時隨地很方便的使用並擷取所需的資訊，已成為資訊界最重要的課題。無線通訊網路提供方便的通訊媒介給行動中的資訊使用者；但目前的無線通訊網路頻寬太低，價格太高，通訊品質不佳，大為影響行動資訊服務之品質。最近學術界所提倡的網路精靈技術或行動代理者（mobile agent），即是用來克服此種困難的主要技術。網路精靈技術和行動代理者乃是針對下面幾點目前網路架構的下主要瓶頸：

- 頻寬太低，價格太高，通訊品質不良。
- 行動計算在目前網路架構上成本過高。
- 傳輸的資料量經常龐大但含有太多無效資訊。
- 行動中使用者隨時有中斷連線（disconnection）之潛在危機。

簡單而言，上述的網路精靈是一個客戶端送出的電子訊息，其中攜帶了可執行的電腦程式，此段程式會被精靈目前所行動至的精靈伺服器執行，由伺服器代替原客戶的角色。此外，在精靈所攜帶的訊息中，也必須有命令伺服器轉送精靈至下一站的能力。我們也通稱網路精靈及行動代理者為「行動精靈」或簡稱為「精靈」。我們的研究方向，就是依照上述幾個問題加以研究以及提供解決的辦法，使得 mobile agent 在我們所設計的行動資訊服務環境中，能夠有效率的運作。

2. 網路精靈的概念

在上一節中已經簡單敘述過所謂的網路精靈。其重要的特性如下—

- 精靈有行動能力（mobility），
- 伺服器可以執行精靈內容（如 hopto，即精靈在伺服器間的轉移），
- 管理系統可以控制精靈（如 suspend，kill），
- 精靈間可互相通訊（communication），
- 精靈必須自給自足（self-contained）。

所謂的精靈自給自足是指所有關於網路精靈行動的資訊，都應該包含在網路精靈本體之中。當網路精靈行動到一站伺服器時，伺服器會負責去執行精靈中的內容（script）一直到都執行完畢或是精靈必須前往下一站而結束。當精靈預計行動到下一站時，目前的伺服器負責包裹精靈所需的行動資訊至精靈本體中，並將精靈轉送至下一站，之後該伺服器和此精靈間的關係便結束。當然，伺服器必須記錄一些和精靈行動有關的外部資料（external status），如抵達時間，離開時間，終止狀態，預計行程（下一站）等等，而不應記錄精靈的內文。

精靈自給自足的特性和我們所熟知的 RPC（Remote Procedure Call）有很大的差異。RPC 是當某一伺服器對另一伺服器送出一個需求（request）時，在需求執行未結束前，發出需求的伺服器不可以切斷和需求間的關係，必須要等需求執行完畢回來後才可以結束整段程式碼。如此一來，如果某一 RPC 程式是要連續在好幾站伺服器執行時，其所佔用的資源將極為可觀。

除此之外，精靈還必須維持其單元性（atomicity）。當一個精靈生成後，對任一個資料庫做的交易都要一次完成，對任一站只能光臨一次，而每一個精靈的生命週期只有一次。對於精靈的單元性，在 6.3.3 會有較詳細的解釋，而精靈的執行狀態將於 4.2 節中詳細敘述。

3. 網路精靈程式語言

要設計一個健全的語言，絕非一件簡單的工作。尤其當針對一個陌生的領域而言，更是一項艱難的任務。在最近的電腦語言中，有一種名叫 JAVA 的新程式語言異軍突起，另有一種名叫 Tcl/Tk 的語言，也是來勢洶洶。然而這兩種語言

尚在發展階段，對於行動計算而言不是很成熟。因此在研究中，我們選擇設計一種較高階，並易於使用的程式語言。

在我們所設計的原型機觔斗雲（FlyingCloud）的語言中，提供單一的基本型態string和一個衍生型態list，一些list的操作及基本的流程控制(如if、while及for loop)結構。

精靈程式語言提供使用者，伺服器，管理系統一個溝通的橋樑。利用所設計的語言，精靈有行動能力（hopto），伺服器可以執行精靈內容，管理系統可以控制精靈（suspend, kill），甚至精靈和精靈之間可以溝通（communication）。下面是一個簡單的例子：

```
 1 name: danny
 2 passwd: XXXXXX
 3 LIST={apple, kiwi, lemon, banana};
 4 Jump(6);
 5 hopto STATION;
 6 STATUS = `exist Agent123`;
 7 if (!STATUS) {
 8   NEW_LIST = cuttail(2, LIST);
 9   STATION = first (NEW_LIST);
10   JUMP(5);
11 }
12 kill Agent123;
```

上面的程式是一個控制精靈，負責搜尋並終結另一個精靈。第一和第二行提供身份認證的資訊，第三行是指定的搜尋路徑，其餘的部分是搜尋演算法。在這個精靈行動的同時，一些環境變數都必須附在精靈本體的後端跟著行動，而不會留在伺服器上，而這些因為不屬於精靈本體，所以並不顯示在程式內。

我們所設計的精靈語言，遵循以下的原則：

- 語言本身為高階語言。編譯系統之製作，亦維持階層式架構，以維持可為其他語言代替之彈性。
- 精靈自給自足，所有執行該精靈所需資訊，包括內部資料及環境變數均包含在精靈內部。
- 資料型態簡單，具有string和list。
- 具流程控制能力（sequential，if and while loop）。

這個語言目前並未支援定義函數（function）之能力，因此其擴展性相當有限，這個問題將留待後續的研究中繼續探討。

在精靈程式語言的設計與製作中比較困難的問題是內部資料及環境變數的處理，以維持精靈的自給自足。

當環境變數及內部資料包裝於精靈中轉移到下一站後，必須能自精靈本身抽取出來，在新的伺服器中重建執行環境。在重建執行環境時，比較困難的是迴圈的處理，因迴圈的控制變數必須精確的附在精靈本身且遊走於網路之中。此外，精靈通訊問題的處理也非易事，我們將在網路精靈伺服器部份詳細說明。

4. 網路精靈的管理及搜尋

4.1 管理系統

由於精靈遊走於網路中，一個行動資訊服務系統必須建置一個管理系統以執行精靈之追蹤，管理及控制。

管理系統在整個基礎結構的大環境而言，可視為一個超級使用者（super user）。管理系統之權限，比一般使用者多了管理及控制其他網路精靈的能力。在往後的幾節中，我將討論精靈的狀態，控制及搜尋。

4.2 精靈狀態

精靈的執行狀態如下：

- 執行中（running）：精靈正在伺服器上執行；
- 跳躍中（hopping）：精靈正準備跳躍到下一站；
- 冷凍中（frozen）：精靈能在本站執行，但暫時停止網路漫遊的能力；
- 等待中（spinning）：精靈停止活動，正在等待某些資源；
- 終止活動（terminated）：精靈已被終止一切行動；
- 暫停活動（suspended）：精靈暫時停止一切活動。

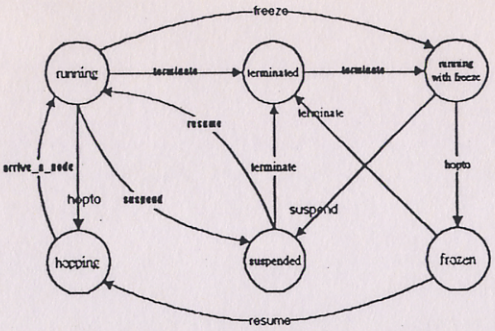
「等待中」和「暫停活動」兩個狀態之主要區別在於「等待中」的精靈可以自己決定是否繼續等待或是繼續後續的行動（如跳躍至下一站），而不需要伺服器或是管理者的許可。但是「暫停活動」的精靈，必須等待下一部的外部指示，才能繼續行動。

4.3 精靈控制

當管理系統需要控制某一台伺服器上的精靈時（不論是客戶要求或是自發性主控），管理者可以利用以下幾個功能來控制精靈：

- 終止（terminate）：停止精靈一切行動；
- 冷凍（freeze）：限制精靈不能跳出目前伺服器；
- 暫停（suspend）：暫時停止精靈一切行動；
- 繼續（resume）：恢復精靈為執行狀態。

當一個精靈開始活動後，不管是自發狀態或是有管理系統控制，都有一個相對應的狀態轉換歷程（State Transition Diagram, Finite State Machine），其狀態表示及轉換如下圖一：



圖一， 精靈狀態轉換歷程圖

相較於上述的精靈控制，我們也希望管理系統能夠在精靈將要行經的路上，預留一些控制訊息給精靈，這就是所謂的「前置控制」（ look ahead control ）。如此，我們也許可以省去搜尋精靈的時間，而直接在某一站等它並控制它。

5. 基礎系統架構

前面已大致介紹了研究內容。以下將說明系統架構的實作，本文著重於伺服器的設計。

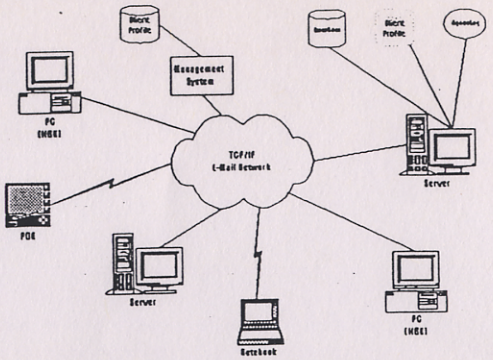
目前在政治大學所設計的原型機，定名為觔斗雲系統（ FlyingCloud System ）。整個系統所追求的目標，乃是提供一個彈性的架構，希望在不更動目前既有大環境的基礎結構下讓行動資訊服務能在既有的網路上方便的建置。因此，我們利用電子郵件（ Electronic Mail ）做為精靈的傳送機構。我們將精靈包裝在電子郵件之中，由客戶或管理者發出，精靈在精靈伺服器之間的轉移，也是以電子郵件的模式傳遞，如此便無須修正目前之電腦網路架構。

觔斗雲的系統架構，如下圖二所示。客戶不論在何處，都可以利用個人數位助理（ Personal Digital Assistant ， PDA ），手提式電腦（ Notebook ），或是基本的個人電腦（ PC ）等可以有網路連接及收發電子郵件功能的電子設備，發出精靈並執行精靈所被託付的任務。整個觔斗雲系統，都是建構在目前的 TCP/IP 網路上，因此不會對下層網路實體結構有所更動，而僅在上層邏輯結構上控制及管理系統。

觔斗雲系統大致可以分為三個部份：使用者端，伺服器端，及管理系統。使用者端包括了使用者必備的網路連接功能，電子郵件系統，和觔斗雲系統的使用者介面。伺服器端包括所有關於精靈活動必備的裝備，資料庫，及精靈伺服器。管理系統包括了精靈管理及控制的機制，以及管理者的監控畫面。

FlyingCloud System Architecture

觔斗雲系統架構



圖二， 觔斗雲系統架構圖

6. 網路精靈伺服器

以客戶端的角度而言，當使用者在通過身份確認，送出一個精靈後，這個精靈會自動在一台或多台伺服器間執行，正常執行完畢會將結果送還給使用者。結果傳回的方式有兩種，第一種是將結果以電子郵件的方式寄回給使用者，或是使用者另外指定的收信人。第二種方式是將結果轉存回使用者的註冊站（ Registered Station, Home Base Node ），再利用使用者端的觔斗雲使用者介面察看結果。

為了能使整個系統順利運作，有許多的內部資料是需要隨著精靈傳遞的。如使用者之註冊資料，精靈之執行歷程，精靈行動環境之轉移。在這些資料中，精靈行動環境的資料比較難處理。我們將在本節中詳細說明伺服器以及組成元件的設計。

6.1 精靈伺服器設計考量

設計伺服器之前，我們有很多的考量要評估。首先，我們要考慮使用高階或是低階的程式語言，來發展精靈伺服器；其次考慮伺服器的可攜性，我們希望設計出來的伺服器，在轉移到不同的作業系統時，不需修正。最後還得考慮工作環境的選擇。在詳加評估之後，我們決定利用 Perl 程式語言來設計精靈伺服器，目前是在工作站上發展（包括可以安裝 Linux 及執行 Perl 的個人電腦）。在目前的程式語言中，能夠擁有移植性的不多；又作業系統在工作站（ Sun Solaris ， HP UX，...)和一般個人電腦（ Windows 95 ， Windows NT ）上的使用環境大不相同。為了要求較多的相容性，我們便決定使用 Perl 及工作站。

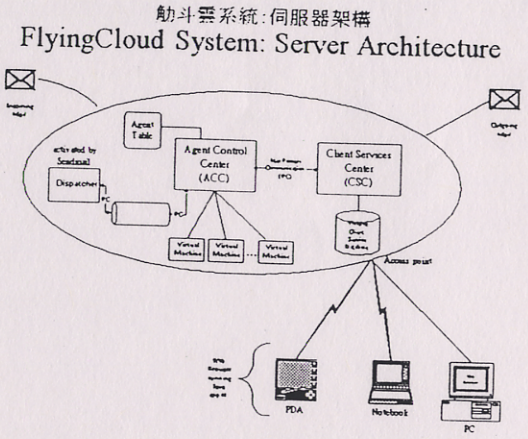
6.2 精靈伺服器設計依據

精靈伺服器的主要任務是提供一個計算環境（ Virtual Machine ），供一個精靈執行其指定的任務。然而精靈的程式是藉由電子郵件傳送的，伺服器必須在收到一封電子郵件時，判斷它是一個精靈，並且執行它內部所攜帶的指令；在精靈必須移動到下一站時，必須為其準備所需攜

帶的環境變數，連同內部狀態一起包裝成一個獨立的精靈，並藉由電子郵件轉移到下一個伺服器。此外，由於觔斗雲系統所針對的使用群眾是可能利用無線通訊或是數據機接撥上來的行動使用者，他們的連線時間通常短暫，而且品質不好。因此如何在有限的時間內，包裝並送出行動者所需的行動代理精靈，並且執行完畢後將執行結果回報給原使用户，也是設計精靈伺服器的主要考慮。

6.3 精靈伺服器組成元件

以伺服器的角度而言，為了讓使用者所發出之精靈能順利的在伺服器端執行、正常結束並得到結果，伺服器內部必須詳細分工，各盡其責，共同完成伺服器應該執行的工作。觔斗雲系統伺服器架構及流程如圖三，大體上伺服器元件可分為四個部份（1）分派者，dispatcher；（2）精靈控制中心，Agent Control Center，ACC；（3）虛擬機器，Virtual Machine，VM；（4）客户服務中心，Client Service Center，CSC，將在下面逐一討論。



圖三， 觔斗雲系統伺服器架構圖

6.3.1 分派者，Dispatcher

當使用者送出一個精靈時，其實只是發出一封電子郵件，其中包含了使用者的身份確認資訊（第一及第二行），和精靈內文（第三行之後）等字串。當精靈伺服器所在工作站的信件管理者收到精靈伺服器的新信件時，它會自動呼叫精靈伺服器中負責管理來信的程式—即「分派者」，來處理這一封新進的郵件。當分派者收到新的郵件之後，首要任務就是分辨它是不是一個行動精靈。如果它是一個我們所謂的行動精靈，則在信的內容中，第一和第二行應該是送出精靈者的身份認證資訊（ username and password ）。如果依照信件中前兩行的資訊，分派者無法在所有註冊的客户中找到對應的客户資訊或密碼不符，則這封信件將會被退回原來寄信的人，代表身份確認無法通過，精靈也必然不會執行。若第一步的身份確認通過，表示該使用者是已通過註冊的客户。此時，分派者將會把這封信重新包裹成精靈眞正要被執行的格式（第一行是精靈資訊，第二和第三行是使用者資訊，第四行起是精靈本文），分配一個唯一的 ID ，儲存在伺服器中適當的位置，並登記其ID和抵達時間，以備未來搜尋所需。另一種情況是，當分派者收到的新信件時，信件內容中第一行不是使用者資訊，而是一行精靈資訊（ agent_id: XXXXX ），下兩行才是使用者資訊。這時，分派者會很清楚的瞭解到這是一個從別站轉送過來的精靈，而把這種已經擁有 ID 的精靈同樣儲存在適當的位置。分派者在安置好精靈之後，便完成了所被託付的第一件事。接下來分派者會透過 IPC （目前我們實作是利用 IPC 中的 message queue ），要求精靈控制中心開始執行這個新的精靈。分派者的任務到此結束，等待新的精靈進來。

另外有一種情況比較特殊，分派者可能會遇到一個已經執行完畢的精靈。這情況是因為這個精靈的執行結果要送回來伺服器，等使用者來取回。分派者只須將這個已經執行完畢的精靈，儲存在使用者的目錄下。其中細節會在 VM 及 CSC 中討論。

6.3.2 精靈控制中心，Agent Control Center，ACC

顧名思義，精靈控制中心（ ACC ）就是控制精靈的總部。在觔斗雲原型機中， ACC 是一個永遠在背景執行的程式。啓動觔斗雲系統後，就持續在背景執行 ACC 。

由於我們精靈伺服器的記憶體管理是設計利用 message queue ，因此，必須有一個程式不斷的去維護這個 message queue 的完整，並且禁止他人隨意存取，又另外必須有一個跑腿者來負責傳遞訊息。因此，整個 ACC 可以分為兩個部份。先前所指必須在背景執行的程式，是指管理記憶體的那一部份，而跑腿者是視需要才呼叫的小程式。在上一節中所提到的，分派者會透過 IPC 告知 ACC 一個新的精靈已經抵達，並且準備執行。其告知方式就是利用跑腿者把一個訊息（ message ）送給 ACC ， ACC 在接到訊息之後，複製（ fork ）一個子程式並且去執行（ exec ）系統中的虛擬機器（ VM ），把訊息中所指定的精靈交由此一虛擬機器負責監督。而到此時，才眞正開始執行精靈所要執行的任務。在精靈通訊小節中，曾提及關於精靈通訊的問題。在此作一個簡單的說明。由於所有訊息的傳遞，都必須靠 ACC 來完成；因此不論精靈在和其他精靈通訊的時候或是和伺服器之間傳遞訊息，都是交由 ACC 統籌管理。精靈必須把所要傳遞的訊息，由 VM 交給 ACC ，ACC 會判斷是交給其他精靈或是給 ACC 本身的訊息。若訊息是給 ACC 的， ACC 會在接到後作適當的反應。如果訊息是要交給其他精靈，則 ACC 會負責重新包裹訊息，並交給收件的精靈，完成一方的精靈通訊。而另一方的通訊也是用相同的

方式完成，實現精靈間的通訊。除此之外 ACC 也要負責管理精靈。意指當有管理者或是用户要求控制精靈時， ACC 必須把訊息包裝後交給監督精靈的 VM ，由 VM 來執行訊息的內容，從而控制精靈。若沒有 ACC 的居中傳遞，則訊息無法交到 VM 手中，精靈也就無法控制了。在精靈搜尋小節中，曾經提到過的前置控制（ Look Ahead Control ），也需要透過 ACC 來處理。當我們已經預先知道精靈前進的路徑，而想在精靈未來前進的路程上，留下一些控制訊息給它，就必須在精靈將會前往的伺服器上，透過 ACC 預留一個訊息在 message queue 中，而當精靈來到這一站時，負責監督的 VM 會把訊息交給精靈，完成前置控制的動作。由於精靈在網路上的行徑可能不如我們所預料的順利，因此如何有效運用前置控制來掌握精靈將來的動向及控制精靈，都是下一個階段的研究重點。

6.3.3 虛擬機器，Virtual Machine，VM

當 ACC 複製（ fork ）出子程式並且去執行（ exec ）VM 之後，VM 才是眞正去執行精靈的最小單位。 VM 以順序（ sequential ）的方式去逐行執行精靈的本文（第四行起）。但重要的是，為了能夠讓 ACC 管理精靈或是要做精靈間的通訊， VM 在執行每一行精靈本文之前，必須先透過跑腿者去檢查 message queue 中是否有 ACC 或是其它精靈留給它所監督的精靈的訊息。若有， VM 必須先去執行訊息中所託付的任務（如 freeze ，resume ），然後才去執行一行精靈本文。精靈的本文有很多種類，其中包括有內部指令、迴圈、外部指令等等。基本上，除了迴圈之外的指令，我們必須依照設計的要求維持其單元性（ atomicity ）；而在迴圈之內的指令，也是依照單行指令的方式去執行，唯一的不同是，這些指令被包裹在迴圈中，處理時需要格外的注意。 VM 如此執行精靈本文，若是遇到一些關鍵字時，如 freeze ， suspend 或 hopto ， VM 必須有所應變。在先前的章節中，解釋過 freeze 和 suspend 等指令的功用，這是用於控制精靈的關鍵字，如：

suspend Agent_123;
freeze Agent_456;

上述兩行是 VM 在執行精靈本文時可能遇到的指令。假設目前我們的精靈叫 Agent_001 ，而 Agent_001 的任務就是負責暫停 Agent_123 及冷凍 Agent_456 另外兩個精靈。這時候， VM 必須把這兩個訊息，順序的透過 ACC 跑腿者，交給 ACC ，再由 ACC 負責把訊息轉交給負責監督 Agent_123 及 Agent_456 的 VM ，由他們來眞正負責暫停及終止所監督的精靈。第三個較為特殊的指令是 hopto ，這個指令顧名思義是要精靈跳躍至下一站。其語法如下：

hopto NEXT_STATION;

其中 NEXT_STATION 可以是一個臨時改變的站名，也可以是一個預先設定好的地點。當 VM 遇到這個指令的時候， VM 必須先察看目前所監督的精靈有無接獲冷凍的指令，禁止其離開本站。若有，則 VM 必須停止一切精靈本文的執行動作，而開始等待繼續（ resume ）的指令，等到繼續的指令接到之後，才能準備把精靈轉送至下一站。若沒有這道禁令， VM 必須把所有關於這個精靈的內部資料重新包裝起來並且附於精靈本文之後，結束精靈在本站的所有工作，將精靈郵寄至下一站（ NEXT_STATION ），並且作記錄，才算是完成了 VM 的基本任務。

此外，當 VM 遇到一行“!script end!”的時候， VM 會瞭解這一個精靈的本文已全部執行完畢，並且準備將結果寄回給使用者，即發信人。但如同我們在先前所述，使用者可以更改收件地址或是將結果存在 HBN 中，因此，精靈本文中還可能有著如下的指令：

“Reply: HBN;” 或是
“Reply: another person's email address;”

如此一來， VM 便會清楚的瞭解結果要交回何處。如果回覆的地址是一般的使用者，精靈便會直接將結果寄回；如果更改後的地址是 HBN ， VM 會把結果寄還給使用者所屬的 HBN ，而該站所屬的分派者在接到之後，會將這個已經執行完畢的精靈儲存在使用者的適當目錄下，等待使用者利用客户服務中心（ CSC ）的連線來察看結果。

6.3.4 客户服務中心， Client Service Center，CSC

以目前 Windows 的使用普及度而言，發展一套系統卻不支援 Windows 是一個非常不智的決定；因此我們的研究假設使用者是一個習慣坐在電腦前面玩 Windows NT 、 95 等視窗環境下的一般使用者。而 CSC 在此中便扮演一個提供友善使用者介面的角色。 CSC 提供了精靈擁有者一個可以即時監控精靈的工具，並且可以在線上取回精靈完成任務後得到的結果。為了做到所謂的即時服務（ Real Time Service ）， CSC 必須和 ACC 一樣是一個永不停止活動的程式，在客户端和伺服器端不斷的更新資料。我們所設計的 CSC ，主要是以現在最熱門的 Web 瀏覽器（如 Netscape 或是 IE)搭配 Java 程式語言所構成的使用者介面。利用瀏覽器的網路連接功能，搭配由伺服器端下載至客户端的 Java 小程式（ CSC applet ）就可以輕易的達到即時性監控精靈的要求。

先前我們曾提及，由於目前的使用者很多是習慣坐在電腦前使用 Windows 的一般使用者。對於電腦的其他基本操作，可能並不很熟。因此，為了避免使用者因為對於電子郵件不熟悉而無法使用觔斗雲系統， CSC 的使用者介面提供了這樣的使用者一種極為方便發送精靈的工具。也就是說，使用者只要利用滑鼠在 CSC applet 上點幾下，就可以透過 CSC 發送精靈。為了確保系統的安全，在 CSC 這裡也要進行使用者身份確認。

當初我們在設計 CSC 時有很多的考量，主要

是考慮到 CSC 必須對精靈的動態能隨時的掌握。關於這一點，以現今所有的程式語言中，只有利用 Java 才能在客戶端動態的掌握使用者在伺服器端所有精靈的行動。此外，現在 Internet 及 WWW 非常熱門，利用瀏覽器搭配 Java 來發展觔斗雲系統的使用者介面應是正確的選擇。第三點，也是最重要的一點，我們同樣為了考慮系統的可攜性，必須選擇一種不用更改原始程式就可以到處轉移的語言，來發展系統。而 Java 正符合了我們的需要。

CSC 的功能，主要是在客戶端提供友善的使用者介面，設計上則分為兩個部份，CSC Server 和 CSC Client （原始程式碼見最後一章節）。SCS Server 和 ACC 一樣是一個必須在精靈伺服器上不眠不休執行的程式。它主要的任務是在精靈伺服器端的某一個 Socket 不斷聆聽是否有來自客戶端的連線。而客戶端則是利用 CSC Client（一個 Java applet ）透過 Socket 和伺服器端連接。利用 Socket 是非常基本的設計網路程式的方法，而且也是最能爭取時效性的一種網路連接方式。觔斗雲系統中的精靈是以電子郵件的方式行動，因此若要掌握精靈的最新動態，利用 Socket 做網路連接會比只用電子郵件傳遞精靈最新動態更為快速。使用者可以透過 CSC 的使用者介面，線上控制自己的精靈，如 freeze ， suspend 等功能，也可以察看已經完成的精靈的最後結果。管理者也可透過 CSC 控制所有精靈。

7. 網路精靈管理系統

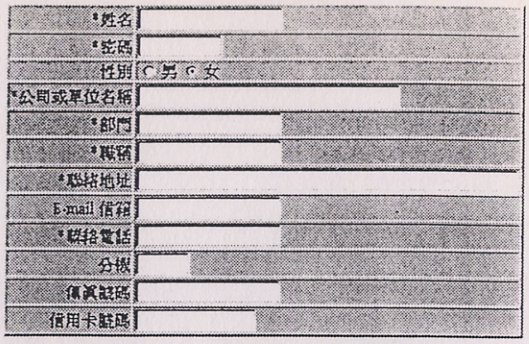
管理者其實就像是一個超級使用者（ Super User ）。同樣利用電子郵件的方式，管理者可以有較高的權限去管理及控制伺服器上的精靈。此外，管理系統有負責統籌管理使用者的註冊資訊。

由於管理系統具有監視網路狀態的功能，我們設計讓一般的使用者在和伺服器連線時也可以和管理系統連線，監看網路狀態。使用者因此有能力選擇利用較不擁擠的路徑來發送行動精靈，以提高精靈網路之執行效能。

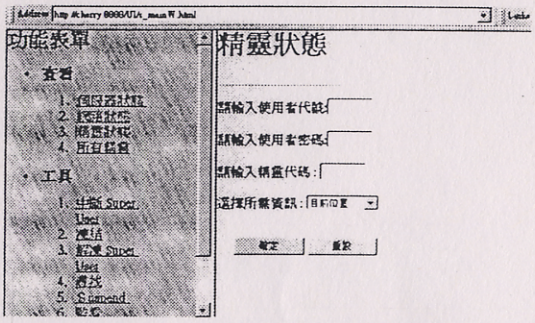
觔斗雲系統下的精靈管理系統，是一套利用 CGI 寫成的精靈及使用者管理程式。我們利用 Perl 撰成 CGI 程式，搭配使用 Netscape3.0 以上版本的瀏覽器，就可以透過 CGI 和伺服器端作連接。至於設計的詳細部分因不在本文之範圍內，故不予詳述。

8. 使用者介面

考慮到使用者可攜性和系統維護的難易程度，我們所有的介面都是利用 WWW 瀏覽器來實作。使用者可以利用目前隨手可得的 Windows 作業系統，搭配任何一種瀏覽器（ Netscape 或是 IE ），透過網路連接到觔斗雲系統，就會出現如下的圖示。使用者就可以利用使用者介面來發出行動精靈，察看目前網路狀態及精靈狀態，控制精靈，以及察看精靈完成後的結果。



圖四， 使用者註冊選單



圖五， 精靈狀態查詢選單

9. 結論

網際網路上豐富的資訊資源提供了未來資訊化社會中隨手可得的資訊寶藏。但隨著工商的發達，行動使用者的數量必定大幅增加。如何使行動中的使用者可以順利的取得網路上的資訊，是一個非常重要的關鍵。行動資訊服務環境主要就是提供行動中的使用者一個方便的橋樑去取用網路資源，但可以不必在意基礎網路架構間的不同之處。由於行動通訊品質不穩、費用太高，行動資訊服務環境必須提供使用者及使用者的行動精靈一個便宜及穩定的作業環境。相對於使用者而言，有了行動資訊服務系統，就可以把需要親自連在網路上一步一步的事情，交給行動精靈處理。如此使用者不但可以節省時間，更可以節省金錢。對於分秒必爭的現代工商社會，是一個非常實用而且有吸引力的新系統。

在未來的計畫中，我們除了繼續維護觔斗雲系統之外，研究發展更有效控制精靈的管理系統也是一項重要課題。其次不論精靈的行動路徑是否已知，隨時掌握所有精靈的最新動態並控制它也是我們在下一個階段所要研究的問題。此外，目前的實作皆以有固定網路連線的個人電腦來模擬行動中的使用者，因此下一個階段的主要研究方向即在於利用無線通訊網路來實際操作觔斗雲系統並研究相關問題。

Reference

1. TIA/EIA IS-41, "Cellular Radio

Telecommunications Intersystem Operations", Telecommunications Industry Association, Dec. 1991.
2. Mohammed Zaid, "Personal Mobility in PCS", IEEE Personal Communications, vol. 1, no. 4, 4th Qtr. 1994, pp. 12-16.
3. Charles Perkins and Pravin Bhagwat, "A Mobile Networking System based on Internet Protocol", IEEE Personal Communications, vol. 1, no. 1, 1st Qtr. 1994, pp. 32-41.
4. Jay Padgett, Christoph Gunther and Takeshi Hattori, "Overview of Wireless Personal Communications", IEEE Communications, Jan. 1995, pp. 28-41.
5. David Chess, Bejamin Grosof, Colin Harrison, David Levine, Colin Parris and Gene Tsudik, "Itinerant Agents for Mobile Computing", IEEE Communications, Oct. 1995, pp. 34-49.
6. Charles Perkins and Kevin Luo, "Using DHCP with computers that move", ACM Wireless Networks, vol. 1, 1995, pp. 341-353.
7. Yi-Bin Lin, "Determining the User Locations for Personal Communications Services Networks", IEEE Transactions on Vehicular Technology, vol. 43, no. 3, Aug. 1994, pp. 466-473.
8. Kaveh Pahlavan and Allen Levesque, "Wireless Data Communications", IEEE Proceedings, Sep. 1994, pp. 1398-1430.
9. Yao-Nan Lien, "An Open Intelligent Messaging Network Infrastructure for Ubiquitous Information Service", Proc. of First Workshop on Mobile Computing, April. 1995, pp. 2-9.
10. Yao-Nan Lien, "Perspective of Service Networks on National Information Infrastructure", CCL Technical Journal, no. 35, Dec 1, 1994, pp. 28-36.
11. Yao-Nan Lien, "An Open Architecture for Ubiquitous Information Services", Accepted to appear in Journal of Computers.
12. Yao-Nan Lien, "Client and Agent Mobility Management", Proc. of the 2nd International Mobile Computing Workshop, Mar. 1996, pp. 141-151.
13. Wen-Shyen Chen and Yao-Nan Lien, "Intelligent Messaging for Mobile Computing over the World-Wide-Web", Proc. of the 2nd International Mobile Computing Workshop, Mar. 1996, pp. 42-51.
14. Yao-Nan Lien, Yenlin Yin, Fuhan Liu and Yuli Hwang, "A Mobile Agent Service Network Prototype", Proc. of 1996 Workshop on Distributed System Technology and Applications, May. 1996, pp. 278-286.
15. Wen-Shyen Chen, Yao-Nan Lien and Wen-Yee Hsien, "An Intrastructure for Mobile Computing with Intelligent Messaging: Implementation Issues", Proc. of 1996 Workshop on Distributed System Technology and Applications, May. 1996, pp. 270-277.
16. Yao-Nan Lien and Roger Leng, "On the Search of Mobile Agents", The Seventh IEEE International Symposium on Personal, Indoor, and Mobile Radio communications (PIMRC'96), Oct. 1996, pp. 703-707.
17. Yao-Nan Lien, Yenlin Yin, Fuhan Liu, Yuli Hwang, Tony Chan, Chi-Yung Chen, Chun-Shyan Hwang, Chun-Ing Lee, Shin-Yi Leu, Chin-Hung Chen and Yang-Farn Liu, "FlyingCloud: A Mobile Agent Service Network", 1996 International Conference on Distributed Systems, Software, Engineering, and Database Systems, Dec. 1996, pp. 177-183.