Indice

Introduzione

Negli ultimi anni l'evoluzione tecnologica ha portato una serie di novità. La diffusione di *Internet* è stata rapidissima ed ha offerto nuove grandi opportunità sia ai fornitori di servizi, che possono ora riferirsi ad un mercato potenzialmente mondiale, sia ai fruitori dei servizi stessi, che hanno possibilità di scelta prima inimmaginabili.

Il settore delle *telecomunicazioni* ha subito parallelamente un grande sviluppo, che ha già portato alla diffusione capillare della telefonia cellulare, ma ha mete più ambiziose. Si stanno infatti progettando sistemi di telecomunicazione con l'obiettivo di fornire servizi avanzati e personalizzati, in maniera il più possibile indipendente dal tipo di connessione utilizzata e dal luogo da cui si intende effettuare la comunicazione.

Contemporaneamente sono apparsi sul mercato *computer portatili* piccoli e potenti, capaci di utilizzare i canali di comunicazione wireless forniti dai nuovi sistemi di telecomunicazione per accedere ai servizi disponibili su Internet.

Questi presupposti aprono un nuovo scenario: quello del *Mobile Computing*, che ha lo scopo di fornire, su un computer portatile con connessione mobile, servizi simili a quelli disponibili con una connessione fissa, rendendo così l'utilizzatore libero di spostarsi. Per questo è necessario studiare e progettare nuovi sistemi software, che siano in grado di soddisfare le nuove esigenze, fronteggiando le difficoltà che si pongono.

Come prima cosa bisogna scegliere un buon *paradigma di progettazione* delle applicazioni, che fornisca strumenti adeguati per modellare la realtà, in modo da consentire la realizzazione di progetti funzionali ed efficienti. In seguito vanno ricercate le *tecnologie* necessarie per realizzare nella pratica i progetti.

In questo lavoro viene studiato il *Paradigma ad Agenti Mobili*, recentemente introdotto e molto promettente per applicazioni distribuite. Questo paradigma introduce un'astrazione di *spazio*, in cui situare le *risorse* a disposizione, e soprattutto di *agente mobile*, come entità autonoma, capace di spostarsi nello spazio per accedere alle risorse di cui necessita per svolgere il compito assegnato.

Il paradigma ha il vantaggio di essere molto *intuitivo* e di avere una *flessibilità* tale da facilitare la progettazione di applicazioni con particolari obiettivi di efficienza o con funzionalità anche molto specifiche. Queste caratteristiche si sposano molto bene con l'esigenza di utilizzo di canali di comunicazione lenti ed inaffidabili e con quella di fornire servizi personalizzati. Per questo la scelta del paradigma appare molto valida per applicazioni di mobile computing.

Nel primo capitolo affronteremo lo studio del *Paradigma ad Agenti Mobili*, presentando prima le tecnologie a codice mobile, necessarie per l'implementazione delle architetture, poi altri paradigmi di progettazione basati sulla mobilità del codice. Passeremo quindi ad una descrizione generale del paradigma ad agenti mobili, sottolineandone i pregi peculiari, i punti deboli, e le possibilità di applicazione.

Nel secondo capitolo ci occuperemo di *Mobile Computing*. Dopo aver introdotto le questioni generali relative alla mobilità, descriveremo due possibili risposte ai problemi posti: una nell'ambito di Internet e l'altra nel settore delle telecomunicazioni.

Nel terzo capitolo, dopo aver indicato i vantaggi dell'utilizzo del paradigma ad agenti mobili nel mobile computing, descriveremo l'architettura e le funzionalità di *SOMA*, un sistema ad agenti mobili realizzato al Dipartimento di Elettronica Informatica e Sistemistica dell'Università di Bologna. Proporremo, quindi, un nuovo *modello* per adattare SOMA alle esigenze del mobile computing ed accenneremo ad altri progetti riguardanti sistemi ad agenti mobili utilizzati nel settore della mobilità.

Nel quarto capitolo, dopo aver delineato brevemente alcune *caratteristiche* della versione attuale del sistema, descriveremo le *modifiche introdotte*, nell'ambito di questo lavoro, per il supporto al mobile computing.

Cap.1 Agenti Mobili

1.1 Tecnologie a Codice Mobile

1.1.1 I sistemi distribuiti classici

Prima di introdurre le tecnologie a codice mobile, è opportuno dare una descrizione schematica della struttura di un sistema distribuito classico [FugPV98], che si può rappresentare come in Figura 1.1.

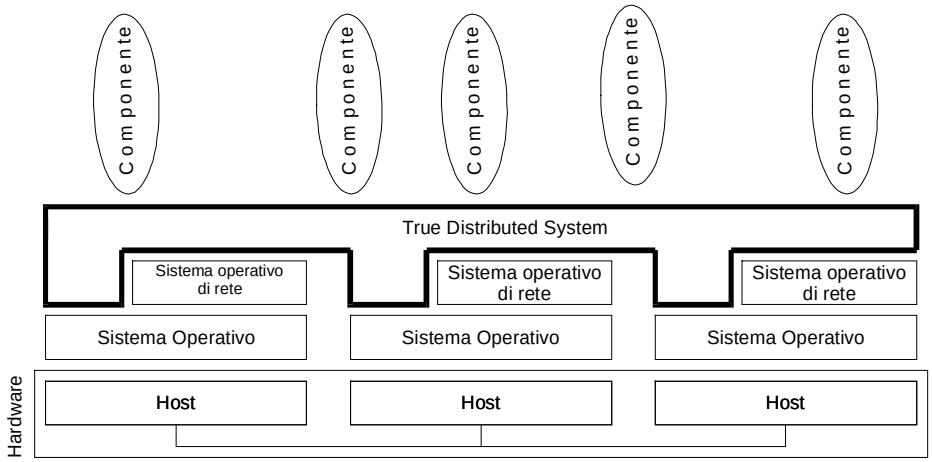

Figura 1.1 Sistema distribuito tradizionale.

Al livello direttamente superiore all'hardware si trova il *sistema operativo*, che fornisce funzionalità di base come file system, gestione della memoria e gestione dei processi. I servizi di comunicazione non trasparenti sono forniti dal *sistema operativo di rete*, per mezzo del quale le applicazioni possono comunicare con altre macchine indicando *esplicitamente* l'host destinatario della

comunicazione. Le socket, ad esempio, sono un tipico servizio fornito dal sistema operativo di rete, ed infatti una socket deve essere aperta specificando esplicitamente il nodo di rete di destinazione. Il sistema operativo di rete, almeno concettualmente, utilizza i servizi forniti dal sistema operativo, per esempio la gestione della memoria. La trasparenza del livello di rete è fornita soltanto dal *true distributed system*, che realizza una piattaforma dove i componenti, situati in differenti siti della rete, sono percepiti come se fossero locali. In questo modo, gli utenti dei servizi di un true distributed system non devono essere consci della distribuzione fisica sottostante. Quando un servizio viene invocato, non si sa in nessun modo quale sarà il nodo della rete che lo fornirà, e non si rileverà nemmeno la stessa presenza di una rete. Per fare un esempio di tutto ciò, basta pensare a CORBA[Vin97], i cui servizi possono essere considerati servizi di un true distributed system, in quanto un programmatore che utilizza CORBA non è cosciente della struttura di rete sottostante, ed interagisce sempre con lo stesso *object broker*[1]. Anche in questo caso, almeno in linea di principio, un true distributed system utilizza i servizi forniti al livello sottostante dal sistema operativo di rete.

1.1.2 I sistemi distribuiti a codice mobile

Le tecnologie che forniscono un supporto alla mobilità del codice hanno un approccio diverso: infatti la struttura di rete non è nascosta al programmatore, ma viene presa in considerazione esplicitamente. Per rappresentare questa differenza si è sostituito, al livello più alto, il true distributed system con una serie di *Ambienti di Computazione*: uno per ogni host. Ogni Ambiente di Computazione lascia trasparire l'identità dell'host su cui è situato, e fornisce alle applicazioni la capacità di spostare dinamicamente i

[1] *Object broker* - Il componente responsabile della comunicazione fra un client e gli oggetti di cui il client ha bisogno.

propri componenti su host diversi. Così utilizza i canali di comunicazione gestiti dal sistema operativo di rete ed i servizi di basso livello del sistema operativo per permettere lo spostamento del codice, ed in certi casi anche dello stato dei componenti software.

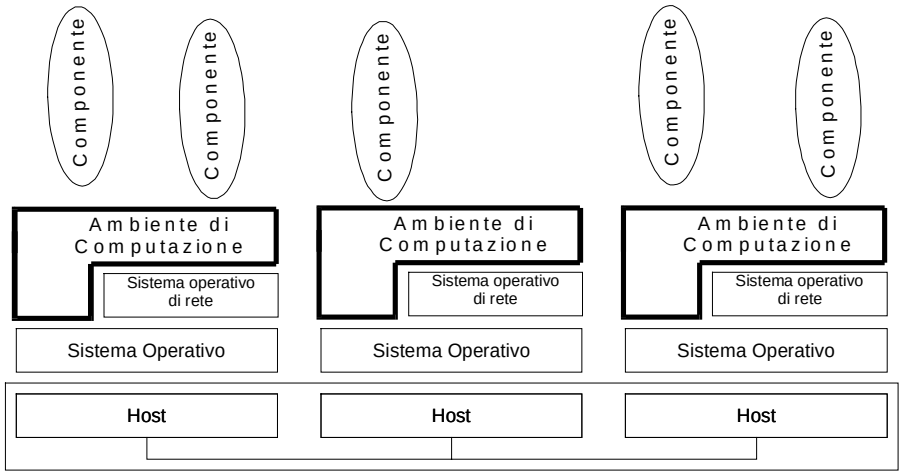

Figura 1.2 Sistema distribuito a codice mobile.

I componenti ospitati da un Ambiente di Computazione, a loro volta, si suddividono in due categorie*: Unità di Esecuzione* e *Risorse*.

- Le *Unità di Esecuzione* rappresentano flussi sequenziali di computazione, per esempio processi con un unico thread, o singoli threads appartenenti ad un processo multi-threaded.
- Le *Risorse* rappresentano entità che possono essere condivise fra più Unità di Esecuzione: sono risorse i files di un file system, gli oggetti condivisi da più threads di un linguaggio ad oggetti e multithreaded, o anche le variabili di sistema di un sistema operativo.

  Una *Unità di Esecuzione* è costituita da due elementi:
- Un *Segmento Codice*, che fornisce una descrizione statica del comportamento di una computazione.

- Uno *Stato*, costituito dallo *Spazio dei Dati* e dallo *Stato di Esecuzione* vero e proprio.
  - Lo *Spazio dei Dati* è un insieme di riferimenti alle risorse accessibili dall'Unità di Esecuzione. Come si spiegherà in seguito, le risorse non si trovano necessariamente nello stesso Ambiente di Computazione dell'Unità di Esecuzione.
  - Lo *Stato di Esecuzione* contiene dati privati che non possono essere condivisi, così come informazioni di controllo legate allo stato dell'Unità di Esecuzione, come lo stack delle chiamate e l'instruction pointer.

  Per fare un esempio:

Unità di Esecuzione: un interprete Java [Jav97], che esegue un programma, costituito da:

- *Segmento di codice*: l'insieme di classi da cui è costituito il programma.
- *Spazio dei dati*:
  - Gli oggetti che contengono i riferimenti alle risorse a cui il programma accede, come files, dispositivi di output grafico, …
- *Stato di esecuzione*:
  - Tutti gli altri oggetti, ossia le istanze delle classi create a runtime.
  - Program counter.
  - Stack delle chiamate della Java Virtual Machine.
  - …

1.1.3 I meccanismi per la mobilità

Nei sistemi convenzionali, ogni Unità di Esecuzione è legata ad un Ambiente di Computazione per tutto il suo ciclo di vita, e i collegamenti fra l'Unità di Esecuzione ed il suo segmento di codice sono in generale statici. Anche in ambienti che supportano i collegamenti dinamici (dynamic linking), il codice collegato

appartiene all'Ambiente di Esecuzione locale. Questo non è più vero nei *Sistemi a Codice Mobile*, in cui il segmento di codice, lo spazio dei dati e lo stato di esecuzione possono essere trasportati in un altro Ambiente di Computazione. In linea di principio ognuna di queste tre componenti potrebbe migrare indipendentemente dalle altre, ma noi considereremo solo le combinazioni considerate utili e adottate nei sistemi esistenti.

1.1.4 Strong Mobility e Weak Mobility

I sistemi a codice mobile esistenti offrono due forme di mobilità, caratterizzate dai componenti dell'Unità di Esecuzione che possono essere trasportati. La *Strong Mobility* è la capacità di un sistema a codice mobile di trasportare sia il codice che lo stato di esecuzione di un'Unità di Esecuzione, verso un altro Ambiente di Computazione. La *Weak Mobility* è la capacità di trasportare il codice, ma non tutto lo stato di esecuzione. Ovviamente è sempre possibile trasportare dei dati di inizializzazione.

Ad esempio Java è un linguaggio che permette la Weak Mobility, ma non la Strong Mobility. Infatti in Java è possibile caricare dinamicamente le classi anche da un sito remoto, in maniera completamente programmabile dall'utente, e questo permette di trasportare il codice. Inoltre viene fornito un meccanismo, anch'esso completamente personalizzabile, per la serializzazione e la deserializzazione, ossia per salvare lo stato di un oggetto su uno stream qualsiasi e per recuperarlo in un secondo tempo. In questo modo è possibile trasferire lo spazio dei dati e di parte dello stato di esecuzione, che però non può essere trasportato completamente, in quanto alcune sue componenti, come lo stack e l'instruction pointer, sono nascoste dalla Java Virtual Machine e non sono direttamente accessibili al programmatore. Questo, come vedremo, influenzerà pesantemente la progettazione dei sistemi ad agenti mobili basati su Java.

La **Strong Mobility** può essere ottenuta fornendo due meccanismi di natura differente: la *migrazione* e la *clonazione remota*. Il meccanismo di *migrazione* sospende una Unità di Esecuzione e la trasmette all'Ambiente di Esecuzione di destinazione dove verrà riattivata. La migrazione può essere *proattiva* o *reattiva*: nella migrazione proattiva, l'istante della migrazione e la sua destinazione vengono determinati autonomamente dall'Unità di Esecuzione stessa. Nella migrazione reattiva, lo spostamento è innescato da un'altra Unità di Esecuzione, che ha un qualche tipo di relazione con l'Unità di Esecuzione che deve migrare, per esempio una Unità di esecuzione che svolge il ruolo di gestore delle Unità di Esecuzione mobili.

Il meccanismo di *clonazione remota* crea una copia di una Unità di Esecuzione su un Ambiente di Computazione remoto. La clonazione remota differisce dalla migrazione in quanto l'Unità di Esecuzione non è distaccata dal suo Ambiente di Esecuzione attuale. Come la migrazione, la clonazione remota può essere proattiva o reattiva.

Per fare un esempio utile per chi ha già familiarità con il concetto di *agente mobile*, per costruire un sistema ad agenti mobili occorre supportare una Strong Mobility, di tipo proattivo, in quanto è l'agente che decide quando migrare e dove. Per quanto riguarda l'altro tipo di classificazione, è possibile sfruttare sia la clonazione remota che la migrazione.

I meccanismi che supportano la **Weak Mobility** forniscono la possibilità di trasferire il codice fra diversi Ambienti di Computazione, e di collegarlo dinamicamente ad un'Unità di Esecuzione già attiva, oppure di usarlo come segmento di codice per una nuova unità di esecuzione.

Questi meccanismi si classificano in base alla direzione del trasferimento di codice, alla natura del codice trasportato, alla sincronizzazione coinvolta ed al momento in cui il codice viene veramente eseguito nel sito di destinazione.

- Quanto alla direzione, il codice può essere richiesto (*fetch*) o spedito (*ship*)
- La granularità del codice trasferito varia: si può trasferire un *frammento di codice*, o un programma autonomo (*stand-alone code*). I frammenti di codice vengono caricati dinamicamente, mentre il codice stand-alone serve per avviare una Unità di esecuzione ex novo.
- Il meccanismo può essere *sincrono* o *asincrono*, a seconda che l'Unità di Esecuzione di partenza attenda o meno l'esecuzione del codice.
- Nei meccanismi *asincroni*, l'esecuzione può essere *immediata* o *differita*. Nel primo caso il codice è eseguito appena viene ricevuto, nel secondo quando si verifica una determinata condizione, per esempio l'invocazione di una parte del frammento di codice, o un evento.

Tornando al nostro esempio, il linguaggio Java prevede il caricamento dinamico delle classi, in maniera completamente programmabile, con l'utilizzo dei Class Loaders[2], che possono prelevare le classi anche da remoto. Questo è chiaramente un meccanismo di *fetch*, ma si può facilmente progettare un sistema di spedizione di classi (file .class) da eseguire all'altro capo, realizzando così la modalità *ship*. Questo, per inciso, avviene nel sistema ad agenti che descriveremo in seguito.

Inoltre, essendo la granularità del trasferimento a livello di classe, è naturale trasferire *frammenti di codice*. D'altra parte, anche una singola classe scaricata dalla rete può considerarsi concettualmente come *un'applicazione stand-alone*, come avviene nel caso di *applet* integrate in pagine Web.

---

[2] *Class Loader* - Una classe dell'API standard di Java, che viene utilizzata per caricare altre classi nell'ambiente di esecuzione da un file o per caricare classi distribuite in una connessione di rete [Jav97].

Infine, essendo la gestione delle classi programmabile, è possibile realizzare i meccanismi di sincronizzazione del tipo desiderato. In conclusione, Java permette di realizzare tutti i meccanismi di trasferimento codice sopra elencati, sempre nel caso di Weak Mobility.

1.1.5 Gestione dello Spazio dei Dati

Con la migrazione di un'Unità di Esecuzione verso un nuovo Ambiente di Computazione, il suo Spazio dei Dati, ossia l'insieme dei collegamenti alle risorse accessibili dall'Unità di Esecuzione, deve essere riorganizzato. Quest'operazione può comprendere l'eliminazione dei collegamenti alle risorse, il ristabilimento di nuovi collegamenti, o anche lo spostamento di alcune risorse verso l'Ambiente di Computazione di destinazione, insieme all'Unità di Esecuzione. La scelta dipende sia dalla natura delle risorse coinvolte, che dal tipo di collegamento a tali risorse, che anche dalle necessità imposte dall'applicazione. Possiamo modellare una risorsa come una tripla *Risorsa = <I, V, T>*, dove I è un identificatore unico, V è il valore della risorsa e T è il suo tipo, che determina la struttura delle informazioni contenute nella risorsa e la sua interfaccia. Il tipo della risorsa determina anche se la risorsa è *trasferibile* o *non trasferibile*, ossia se, in linea di principio può essere trasportata attraverso la rete o no. Per esempio, una risorsa di tipo "Informazioni sulla borsa", può essere trasferita, mentre una risorsa di tipo "Stampante", probabilmente no. Le risorse trasferibili possono essere marcate come *libere*, oppure *fisse*. Le prime, ancora una volta, possono essere trasportate, mentre le seconde no. Questa volta però la scelta non sarà di principio, ma dipenderà da criteri di opportunità: difficilmente posso ritenere opportuno il trasferimento di un file contenente un'intera base di dati.

1.1.5.1 Tipi di collegamento

Le risorse possono essere collegate con le unità di esecuzione da tre tipi di collegamento, che vincolano a loro volta diversi meccanismi di gestione che possono essere sfruttati nella migrazione.

- Il legame più forte è quello **per identificatore**. In questo caso l'Unità di Esecuzione richiede di essere collegata, in ogni momento, ad una risorsa ben precisa, identificata univocamente. Questo tipo di collegamento è utilizzato quando una risorsa non è sostituibile con un'altra equivalente.
- Un collegamento stabilito **per valore** dichiara che, in ogni momento, la risorsa deve rispettare un certo tipo, ed il suo valore non può cambiare in seguito ad una migrazione. Questo tipo di collegamento è sfruttato generalmente quando un'Unità di Esecuzione è interessata ai contenuti di una risorsa, e desidera accedere ad essi in locale. In questo caso l'identità della risorsa è irrilevante, mentre la risorsa migrata deve avere lo stesso tipo e valore di quella presente nell'Ambiente di Esecuzione di partenza.
- La forma più debole di collegamento è **per tipo**. In questo caso, l'Unità di Esecuzione richiede che in ogni momento la risorsa collegata sia di un certo tipo, indipendentemente dal suo valore attuale o dalla sua identità. Questo tipo di collegamento è sfruttato solitamente per collegare risorse che sono disponibili su ogni sistema operativo, come variabili di sistema, librerie o network devices. Per esempio, se un'Unità di Esecuzione mobile ha bisogno di accedere al display locale per interagire con l'utente attraverso un'interfaccia grafica, potrebbe sfruttare un collegamento con una risorsa di tipo "display". Il valore attuale e l'identificatore della risorsa sono irrilevanti, e la risora collegata attualmente viene determinata dall'Ambiente di Computazione corrente.

È da notare che è anche possibile avere diversi tipi di collegamenti alla *stessa* risorsa. Nell'esempio di cui sopra, supponiamo che l'Unità di esecuzione Mobile, oltre ad interagire con l'utente locale attraverso il display, abbia bisogno di presentare i risultati della computazione all'utente "proprietario" dell'Unità di Esecuzione. Questo si realizza creando, all'avvio, un legame per identificatore al display dell'utente ed un legame per tipo alla stessa risorsa. Come spiegheremo fra poco, dopo la prima migrazione i collegamenti saranno riconfigurati cosicché il collegamento per identificatore manterrà l'associazione con il display dell'utente, mentre il collegamento per tipo sarà associato al display dell'Ambiente di Esecuzione di destinazione.

1.1.5.2 Meccanismi di gestione dello spazio dei dati

La discussione precedente individua due classi di problemi che devono essere affrontati dai meccanismi di gestione dello spazio dei dati con la migrazione di una Unità di esecuzione: il trasporto delle risorse e la riconfigurazione dei collegamenti. Il modo in cui i sistemi esistenti affrontano questo problema è vincolato sia dalla natura delle risorse coinvolte che dalla forma dei collegamenti a tali risorse.

Lo scenario da considerare è costituito da un'Unità di Esecuzione U che migra, ed il cui spazio dei dati contiene un collegamento B ad una risorsa R. Un primo meccanismo generale, che è indipendente dal tipo di collegamento o di risorsa è la ***rimozione del collegamento***. In questo caso, quando U migra, B è semplicemente scartato. Si tratta di una soluzione un po' drastica, quindi, se si desidera conservare l'accesso alle risorse, si devono utilizzare altri meccanismi.

- Se U è legato ad R **per identificatore**, sono disponibili due meccanismi per conservare l'identità della risorsa. Il primo è la rilocazione ***per spostamento***, ed in questo caso R è trasferita

insieme ad U all'Ambiente di Computazione di destinazione ed il collegamento non è modificato. Chiaramente questo meccanismo può essere sfruttato solo se R è una risorsa libera trasferibile, altrimenti bisogna utilizzare un meccanismo di ***riferimento di rete***. In questo caso, R non è trasferito, ed una volta che U ha raggiunto l'Ambiente di Computazione di destinazione, il collegamento B è modificato in maniera tale da riferirsi ad R che è rimasta nell'ambiente di partenza. Di conseguenza ogni successivo tentativo dell'unità U di accedere ad R attraverso il collegamento B produrrà qualche tipo di comunicazione con l'ambiente di origine. Nella maggior parte dei casi, però, la creazione di collegamenti fra Ambienti di Computazione diversi non è desiderabile perché espone U ai problemi legati alla rete, come partizionamenti o ritardi, e rende difficoltosa la gestione della consistenza dello stato, in quanto lo spazio dei dati è effettivamente *distribuito* sulla rete. D'altra parte, prelevare una risorsa dal suo Ambiente di Computazione può provocare problemi ad altre unità di esecuzione che hanno collegamenti alla stessa risorsa, ed ancora una volta le soluzioni due: stabilire riferimenti di rete, o rimuovere semplicemente i collegamenti, scatenando eccezioni ai successivi tentativi di accesso.

- Se il collegamento B è **per valore**, il meccanismo più conveniente è la gestione dello spazio dei dati ***per copia***, perché l'identità della risorsa non è rilevante. In questo caso si crea una copia di R, chiamata R', il collegamento ad R si fa puntare su R' e quindi R' è trasferita insieme all'Unità di Esecuzione U all'ambiente di destinazione. Anche la gestione ***per spostamento*** soddisfa i requisiti imposti dal collegamento per valore ma, in alcuni casi, può essere meno conveniente in quanto la rimozione di R sarebbe un evento di cui dovrebbero tenere conto le altre Unità di Esecuzione che avevano riferimenti ad R e che si vedrebbero sottratta la risorsa.

Chiaramente, se R non è trasferibile, il riferimento di rete rimane l'unica scelta, con tutti i suoi inconvenienti.

- Infine, se l'Unità di Esecuzione U è collegata alla risorsa R **per tipo**, il meccanismo più conveniente è il ***ricollegamento***. In questo caso, il collegamento B è rimosso, e ristabilito, dopo la migrazione di U, verso un'altra risorsa R' che si trova nell'ambiente di destinazione. L'unico vincolo da soddisfare è che R' abbia lo stesso *tipo*, di R, ed il vantaggio sta nell'evitare sia il trasferimento della risorsa, sia i riferimenti di rete fra un ambiente e l'altro. Chiaramente, questo meccanismo presuppone che nell'ambiente di destinazione sia effettivamente presente una risorsa dello stesso tipo di R. Altrimenti è necessario ricorrere ad uno degli altri meccanismi a seconda del tipo e delle caratteristiche della risorsa in questione.

1.2 Paradigmi di progettazione

Le tecnologie a codice mobile sono solo uno degli ingredienti necessari per costruire un sistema software. Lo sviluppo del software è un processo complesso in cui bisogna tenere in considerazione una serie di fattori: tecnologia, organizzazione e metodologia. In particolare bisogna fare attenzione alla relazione fra tecnologia e metodologia. Infatti spesso si tende a credere che la tecnologia da sola sia sufficiente ad introdurre una metodologia. Questo risulta particolarmente evidente in una fase molto critica dello sviluppo del software: la progettazione. L'obiettivo della progettazione è la creazione di un'architettura software, che può essere definita come la scomposizione di un sistema in termini dei componenti software e delle loro interazioni. Architetture con caratteristiche simili possono essere rappresentate da *stili di architettura* o *paradigmi di progettazione*, che definiscono astrazioni architettuirali e strutture di riferimento che possono essere istanziate in architetture software reali. Un paradigma di

progettazione non è necessariamente indotto dalla tecnologia utilizzata per lo sviluppo: paradigmi e tecnologie sono due cose concettualmente separate. Così ad esempio nessuno vieta di scrivere del codice “a spaghetti” utilizzando un linguaggio ad oggetti come Java. Certamente, caratteristiche specifiche di un linguaggio possono essere particolarmente indicate per garantire alcune proprietà del programma, ma un “buon” programma non è soltanto la conseguenza diretta della selezione di un buon linguaggio.

Gli approcci tradizionali della progettazione software non sono sufficienti nella progettazione di sistemi distribuiti di larga scala che sfruttano la mobilità del codice e la riconfigurazione dinamica di componenti software. In questi casi, i concetti di locazione (dall’inglese location), distribuzione dei componenti fra le diverse locazioni e migrazione dei componenti verso locazioni diverse devono essere presi in considerazione esplicitamente nella fase di progettazione. L’interazione fra componenti che risiedono sullo stesso host è notevolmente diversa da quella fra componenti che risiedono su host diversi di una rete, in termini di tempi di latenza, accesso alla memoria, guasti parziali e concorrenza. Il tentativo di trascurare le diversità fra interazioni locali e remote può portare ad inaspettati problemi di prestazioni ed affidabilità dopo la fase di implementazione.

È quindi importante identificare paradigmi di progettazione adeguati, per i sistemi distribuiti che sfruttano la mobilità del codice, e discutere le loro relazioni con le tecnologie che possono essere utilizzate per implementarli.

1.2.1 Concetti Basilari

Prima di introdurre i paradigmi di progettazione, presentiamo alcuni concetti di base, che sono una astrazione delle entità che costituiscono un sistema software, come files, valori di variabili,

codice eseguibile o processi. In particolare, introduciamo tre concetti architetturali: *componenti*, *interazioni* e *siti*.

- I *componenti* sono i costituenti base di una architettura software. Possono essere ulteriormente suddivisi in:
  - ♦ *Componenti codice*, che incapsulano il know-how per effettuare una certa computazione.
  - ♦ *Componenti risorsa*, che rappresentano dati o dispositivi utilizzati durante la computazione.
  - ♦ *Componenti computazionali*, che sono gli esecutori attivi, capaci di portare avanti una certa computazione, utilizzando un certo know-how.
- Le *interazioni* sono eventi che coinvolgono due o più componenti, ad esempio un messaggio scambiato fra due componenti computazionali.
- I *siti* ospitano componenti e supportano l'esecuzione di componenti computazionali. Un sito rappresenta la nozione intuitiva di locazione. Le interazioni fra componenti che risiedono nello stesso sito sono considerate meno costose delle interazioni che avvengono fra componenti che si trovano in siti diversi. Inoltre, una computazione può essere effettivamente eseguita solo quando il know-how che la descrive, le risorse che usa, ed i componenti computazionali responsabili dell'esecuzione si trovano *nello stesso sito*.

I paradigmi di progettazione sono descritti in termini di schemi di interazione che definiscono la rilocazione ed il coordinamento fra i componenti richiesti per espletare un compito. Considereremo uno scenario dove un componente computazionale A, che si trova in un sito S_A, ha bisogno dei risultati di un servizio. Ipotizziamo l'esistenza di un altro sito S_B, che sarà coinvolto nel completamento del servizio.

Identifichiamo tre principali paradigmi di progettazione che sfruttano la mobilità del codice: la *remote evaluation* (valutazione remota), il *code on demand* (codice su richiesta), e il paradigma ad

agenti mobili. Questi paradigmi sono caratterizzati (vedi Tabella 1):

- dalla posizione dei componenti prima e dopo l'esecuzione del servizio,
- dal componente computazionale che è responsabile dell'esecuzione del codice,
- dalla posizione in cui il ha effettivamente luogo la computazione.

Paradigma	**Prima**		**Dopo**	
	S_A	**S_B**	**S_A**	**S_B**
Client-Server	A	Know-how Risorsa B	A	Know-how Risorsa **B**
Remote Evaluation	Know-how A	Risorsa B	A	*Know-how* Risorsa **B**
Code on Demand	Risorsa A	Know-how B	Risorsa *Know-how* **A**	B
Agenti Mobili	Know-how A	Risorsa	-	*Know-how* Risorsa ***A***

Tabella 1

Questa tabella mostra la posizione dei componenti prima e dopo l'esecuzione di un servizio. Per ogni paradigma viene evidenziato in **grassetto** il componente che esegue il codice, in *corsivo* ogni componente spostato.

1.2.2 Il Client-Server

Il paradigma client-server è molto noto, ed utilizzato universalmente. In questo paradigma, un componente computazionale B (il server), che offre un insieme di servizi, è situato nel sito S_B. Le risorse ed il know-how necessario per l'esecuzione del servizio sono ospitate anch'esse dal sito S_B. Il componente client A, che si trova in S_A, richiede l'esecuzione di un servizio con un'interazione con il componente server B. In risposta,

B fornisce il servizio richiesto eseguendo il know-how ed accedendo alle risorse coinvolte, situate anch'esse in S_B. In generale, il servizio produce un qualche tipo di risultato, che viene riconsegnato al client con un'ulteriore interazione.

1.2.3 La Remote Evaluation

Nel paradigma Remote Evaluation, un componente A ha il know-how necessario per eseguire un servizio, ma non possiede le risorse necessarie, che però si trovano nel sito remoto S_B. Di conseguenza, A spedisce il know-how del servizio ad un componente computazionale B, che si trova nel sito remoto. B, da parte sua, esegue il codice utilizzando le risorse disponibili nel suo sito. Una seconda interazione riporta i risultati ad A.

1.2.4 Il Code on Demand

Nel paradigma Code on Demand, il componente A è già in grado di accedere alle risorse di cui ha bisogno, che si trovano nel suo stesso sito S_A. Purtroppo, però, in S_A non è disponibile nessun'informazione su come manipolare tali risorse. Di conseguenza, A interagisce con un componente B in S_B, richiedendo il know-how necessario per il servizio, know-how anch'esso disponibile in S_B. Una seconda interazione avviene quando B spedisce il know-how ad A, che può quindi eseguirlo.

1.2.5 Il Paradigma ad Agenti Mobili

Nel paradigma ad Agenti Mobili, il know-how del servizio è posseduto da A, che è inizialmente situato in S_A, ma alcune delle risorse sono situate in S_B. Di conseguenza, A migra in S_B, portandosi dietro il know-how e possibilmente alcuni risultati intermedi. Dopo essersi trasferito in S_B, A porta a termine il servizio utilizzando le risorse disponibili in loco. Il paradigma ad

agenti mobili è diverso dagli altri paradigmi a codice mobile poiché le interazioni associate comportano la mobilità di un componente computazionale già *esistente*. In altre parole, mentre nella Remote Evaluation e nel Code on Demand si trasferisce solo del codice fra i componenti, nel paradigma ad agenti mobili si trasferisce un intero componente computazionale verso un sito remoto, con il suo stato, il codice di cui ha bisogno ed alcune risorse necessarie per il compito da svolgere.

1.3 Il Mondo degli Agenti

1.3.1 Introduzione

A questo punto inizieremo a descrivere quello che definirei *Mondo degli Agenti* ad un livello di astrazione più elevato possibile. Grazie al paradigma ad agenti mobili è infatti possibile costruire un *modello* della realtà ad un livello di *astrazione* molto elevato, caratterizzato da una notevole *intuitività*.

Nella storia dell'informatica c'è stata una progressiva evoluzione che è passata per la programmazione in linguaggio macchina, la programmazione strutturata, poi la programmazione ad oggetti, elevando ogni volta il livello di astrazione. L'introduzione del nuovo paradigma può essere vista come una ulteriore fase di questa evoluzione.

Immaginiamo di dover spiegare ad un neofita dell'informatica cos'è un programma: probabilmente avremo delle difficoltà. Potremmo, ad esempio, descriverlo come una lista di istruzioni scritte in un certo linguaggio, comprensibile sia all'uomo che alla macchina, e che consente all'uomo di spiegare alla macchina cosa deve fare. Questa descrizione, anche se semplice e magari riduttiva, evidenzia comunque una serie di livelli di astrazione, il livello degli umani, quello delle macchine ed un linguaggio di programmazione intermedio che funge da tramite fra i due. La presenza di diversi

livelli di astrazione non solo riduce l'intuitività di uno strumento, ma ne rende più difficile il suo utilizzo pratico.

Il paradigma ad oggetti ha già il pregio dell'astrazione, e a questa unisce semplicità ed intuitività. È *astratto* perché prescinde da tutti i dettagli tecnici e modellizza direttamente la realtà, *semplice* perché mette in campo un solo tipo di entità: gli oggetti, *intuitivo,* perché gli oggetti del modello hanno proprietà analoghe agli oggetti del mondo reale.

Con il paradigma ad agenti mobili si riesce a modellare una realtà molto complessa come quella dei sistemi distribuiti, o addirittura della rete mondiale, mantenendo le caratteristiche positive viste per il paradigma ad oggetti:

- *Astrazione* - Viene costruito un Mondo degli Agenti, che modellizza aspetti rilevanti del mondo reale.
- *Semplicità* - Le entità sono poche: *agenti* e *risorse*, situate in uno *spazio*.
- *Intuitività* - Le entità del modello hanno proprietà mutuate dal mondo reale, quindi si comportano nel modo in cui ci aspettiamo.

1.3.2 Agenti, Risorse, Place

Il Mondo degli Agenti ha caratteristiche simili e mutuate dal mondo reale. Un agente è il corrispettivo di un essere vivente: l’agente nasce, vive e muore. Quando l’agente è in vita, agisce autonomamente, interagendo con ciò che lo circonda e prendendo decisioni di conseguenza, al fine di raggiungere i propri obiettivi, essendo stato creato per svolgere una missione ben precisa. L’agente si sposta in uno spazio, che è metafora dello spazio in cui viviamo. Questo spazio è formato da luoghi, detti *place,* separati fra loro. In un *place* si possono trovare due tipi di entità: gli *agenti* e le *risorse.* Una risorsa è qualcosa di cui l’agente può aver bisogno per portare a termine la sua missione. Le risorse possono essere

logiche o fisiche [CorCS97]: un esempio di risorsa fisica può essere una stampante, un fax, un calcolatore ad elevata capacità elaborativa, o, perché no, un utente, con cui l'agente potrà interagire. Una risorsa logica può essere, ad esempio l'interfaccia ad una base di dati, o una *blackboard*, una specie di lavagna virtuale su cui un altro agente può aver scritto informazioni utili, o ancora un servizio informativo, dal quale l'agente può trarre informazioni circa la posizione di un determinato tipo di risorsa.

1.3.3 L'Astrazione di Località

Si è detto i place sono separati fra loro, è però possibile per l'agente, sotto certe condizioni, spostarsi da un place all'altro, per svolgere i propri compiti.

Innanzitutto non è necessariamente detto che tutti gli agenti debbano spostarsi, è infatti possibile che un agente sia progettato per svolgere il proprio compito in un determinato place, senza doversi mai spostare, in tal caso si dice che l'agente è *stazionario*. In questo caso si farà in modo che l'agente disponga "in loco" di tutte le risorse di cui necessita. Un esempio può essere un agente che funge da interfaccia per l'accesso ad un servizio fornito dal sistema: in tal caso saranno gli altri agenti a spostarsi per comunicare con lui, che resterà per tutta la vita nello stesso place.

Nel caso in cui un agente debba spostarsi, è evidente che dovrà rispettare le regole imposte dallo spazio in cui si sposta. Questo infatti avrà delle proprietà, come accade nello spazio comune, in cui sarebbe ad esempio difficile immaginare di andare sulla luna a piedi. Così lo spazio dei place non è necessariamente totalmente connesso, e da ogni place sarà possibile migrare solo verso alcuni degli altri place, a seconda delle proprietà dello spazio. È possibile che lo spazio dei place abbia una struttura gerarchica, come lo spazio fisico, in cui i continenti sono divisi in stati, divisi in regioni, e così via. In questo caso probabilmente lo spostamento da

un place ad un altro appartenente alla stessa unità gerarchica, diciamo alla stessa regione, sarà più “semplice”, rispetto allo spostamento verso un place che si trova in un’altra unità gerarchica: diciamo un altro continente.

1.3.4 La Comunicazione

Sarà poi necessario stabilire dei meccanismi con cui gli agenti possano comunicare fra di loro. La comunicazione fra due entità può essere, in generale, *sincrona* o *asincrona*.

- La comunicazione *asincrona*, non prevede risposta.
- La comunicazione *sincrona* prevede un risultato, ossia prevede che il destinatario della comunicazione spedisca a sua volta una risposta. Un meccanismo di comunicazione sincrona può essere:
  - ♦ *bloccante*: il mittente interrompe la sua attività nell'attesa della risposta,
  - ♦ *non bloccante*: l'attività prosegue senza attesa. In questo caso si devono prevedere meccanismi per rilevare l'arrivo della risposta. Si possono usare le interruzioni (*interrupt*), oppure prevedere che il processo in attesa interrompa periodicamente la propria attività per verificare l'arrivo della risposta (*polling*).

  Per quanto riguarda i meccanismi di interazione fra agenti, si individuano due categorie:
- *interazione stretta*: presuppone forte accoppiamento fra gli agenti interessati. Ad esempio può essere basata su meccanismi di comunicazione sincroni e bloccanti,
- *interazione lasca*: presuppone un debole accoppiamento. Ad esempio può essere basata su meccanismi asincroni.

Chiaramente sarà necessario considerare almeno due situazioni diverse: due agenti che si trovano nello stesso place, per la cognizione comune che abbiamo di spazio, comunicheranno in maniera più facile rispetto ad agenti che si trovino distanti fra loro.

Per questo, nella progettazione di meccanismi di comunicazione, ha senso privilegiare meccanismi di interazione lasca fra agenti che si trovano su place diversi, e di interazione stretta nell'ambito dello stesso place.

Per consentire la comunicazione fra agenti situati su *place diversi* è possibile fornire l'astrazione di messaggio: un oggetto che un agente può spedire ad un altro, indirizzandolo all'agente destinatario, indipendentemente dalla sua posizione attuale.

Se due agenti, invece, si trovano nello *stesso place* la comunicazione è facilitata dalla possibilità di accesso alle stesse risorse. È quindi possibile immaginare nuove metafore come lavagne su cui lasciare messaggi, o contenitori in cui depositare oggetti per altri agenti.

1.3.5 L'Accesso alle Risorse

Una caratteristica fondamentale dello spazio appena costruito è che le risorse sono accessibili solo nel place in cui sono situate, e questo non è limitativo, è anzi un punto di forza. Infatti si potrebbe obiettare ad esempio di voler garantire l'accesso ad una base di dati da tutti i nodi di una rete. Questo si mappa rendendo la risorsa "interfaccia di accesso alla base di dati", onnipresente nell'ambito dei place appartenenti alla rete in questione. Un vantaggio di questo approccio sarebbe ad esempio quello di poter aprire il sistema all'accesso da parte di agenti autorizzati provenienti da altre reti.

Visto, quindi, che non è limitante permettere l'accesso ad una risorsa soltanto nell'ambito del place in cui è situata, si deve affrontare un nuovo problema: è necessario fornire servizi informativi per gli agenti che sono alla ricerca di una determinata risorsa, ad esempio l'elenco di libri di una certa biblioteca, o di una risorsa di un certo tipo, ad esempio una stampante al laser a colori.

Una volta realizzato un sistema di informazioni di questo tipo, sarebbe possibile immaginare tutta la rete come un unico sistema

ad agenti in cui sia possibile lanciare un agente con il compito di trovare autonomamente le risorse che gli sono necessarie. In questo modo avremmo realizzato un comportamento per gli agenti che potrebbe apparire intelligente, nel senso dato al termine dall'Intelligenza Artificiale.

1.3.6 La Sicurezza

Quello della sicurezza è un tema molto attuale. In questo periodo, infatti, la diffusione esponenziale di Internet rende l'esigenza di apertura dei propri sistemi all'ordine del giorno. Però, oltre agli ovvi vantaggi derivanti dall'apertura, si devono considerare i notevoli rischi di attacchi provenienti dall'esterno, che possono provocare danni e perdite. Per questo si deve affrontare seriamente il problema della sicurezza.

Il paradigma ad agenti mobili, ancora una volta, permette di affrontare il problema in modo semplice ed intuitivo, anche se, ovviamente, non risolve di per sé nessuno dei problemi posti. Infatti rimane valido il principio per cui non esistono ricette magiche per semplificare problemi intrinsecamente complicati, e per abbattere difficoltà concettuali. A mio avviso, il problema della sicurezza informatica è abbastanza complesso da non permettere di essere affrontato con leggerezza e di essere aggredito con soluzioni "semplici".

In ogni caso, per capire quali sono i problemi di sicurezza dei sistemi ad agenti è sufficiente pensare ai problemi di sicurezza nel mondo reale. La prima cosa da fare nel mondo reale è identificare le persone, e stabilire per conto di chi agiscono, nel caso in cui agiscano per conto di qualcuno. Allo stesso modo, sicuramente un agente agirà per conto di qualcuno, per cui è necessario identificare con certezza gli agenti e le persone o le organizzazioni per conto di cui questi lavorano. Questo è fondamentalmente un problema di *paternità*. Più precisamente si tratta di costruire una funzione che

porta dall'insieme degli agenti all'insieme degli utenti, e di determinare con certezza entrambe le entità.

Nel mondo reale, inoltre, c'è un sistema di leggi per stabilire ciò che si può e ciò che non si deve fare, al fine di garantire il rispetto dei diritti dei singoli. Allo stesso modo, nei sistemi ad agenti, è necessario tutelare i reciproci interessi in gioco: bisogna quindi proteggere da un lato i sistemi dai possibili attacchi degli agenti, e dall'altro gli agenti dagli attacchi di altri agenti o dei sistemi.

L'istanza più difficile da affrontare è quella degli attacchi agli agenti perpetrati dai sistemi ostili che li ospitano, in quanto sono loro a dettare le regole del gioco ed a garantire la sopravvivenza stessa degli agenti. Infatti, un place ostile può non solo decifrare ed alterare i dati dell'agente, se non cifrati, ma ha a disposizione anche il codice, che, per essere eseguibile, dovrà essere in chiaro. È possibile quindi leggerlo ed alterarlo, imponendo all'agente comportamenti diversi da quelli per cui era stato creato. Ad esempio, se un agente trasporta con se un messaggio cifrato e l'algoritmo necessario per decifrarlo[3], il place può utilizzare l'algoritmo per decifrare il messaggio. Un place ostile, infine, può anche semplicemente distruggere un agente.

Parlando di leggi, un ulteriore parallelismo può essere fatto: nel mondo reale il sistema legislativo varia a seconda della posizione in cui ci si trova: è stabilito da regolamenti internazionali, da leggi nazionali, regionali e così via. Allo stesso modo, nei sistemi ad agenti viene naturale impostare le politiche di sicurezza su base, per così dire, geografica. Così, ma questo non è assolutamente vincolante, si può stabilire che l'organizzazione delle politiche di sicurezza rispecchi l'organizzazione gerarchica dello spazio, visto che, per coerenza, l'organizzazione geografica virtuale

[3] Per esempio, una *chiave crittografica* può essere vista come un meccanismo per selezionare la funzione di decifrazione di un messaggio cifrato. Quindi trasportare una chiave crittografica equivale a trasportare l'algoritmo di decifrazione.

corrisponderà ad un insieme di strutture reali. Cosicché una *nazione* virtuale può corrispondere al sistema informativo di un'università, o di una società; una *regione* ad un singolo dipartimento, o ad una divisione, e così via. In quest'ottica è comprensibile che ogni università vorrà avere il suo sistema di sicurezza, che verrà specializzato a livello di dipartimento, e ancora di più a livello di singola macchina, utente o gruppo di lavoro.

1.3.7 La Gestione delle Risorse

Per gestione delle risorse, intendiamo una corretta ripartizione delle risorse utilizzate in base ai vantaggi ottenuti dal loro impiego. Immaginiamo un agente che esegua un ciclo infinito a vuoto impedendo ad altri l'accesso alla CPU su cui esegue, si tratta evidentemente di uno spreco inutile e dannoso. Nel mondo reale, però, raramente accade di vedere grossi sprechi fine a se stessi, perché le risorse hanno un costo, ed essendo il danaro a disposizione limitato, usufruire di alcune risorse significa privarsi di altre.

Allo stesso modo, soprattutto in un sistema distribuito aperto, si pone l'esigenza di utilizzare nella maniera più efficiente possibile le risorse a disposizione, vincolando tutti al rispetto di certe regole. Ora, si potrebbe stabilire di introdurre un qualche tipo di valuta, utilizzata dagli agenti per pagare le risorse cui accedono, in modo tale che ogni utente, avendo a sua disposizione solo una certa somma di questo danaro elettronico, spinga i propri agenti a farne il miglior uso possibile.

1.4 Applicazioni del Paradigma ad Agenti Mobili

1.4.1 Vantaggi

In linea di principio, con il Paradigma ad Agenti Mobili è possibile realizzare qualsiasi tipo di Software distribuito. Si tratta quindi di capire in quali casi la scelta si possa rivelare conveniente.

Innanzitutto, si tratta di individuare il tipo di risorsa critica: se è importante l'efficienza in termini di capacità computazionale, il paradigma sarà probabilmente da scartare, in quanto, nella maggior parte dei casi, si fa uso di linguaggi interpretati, con una netta penalizzazione del livello di prestazioni rispetto ad applicazioni in codice nativo. Chiaramente l'ostacolo si può aggirare, ma diciamo che l'obiettivo primo del paradigma ad agenti mobili non è l'efficienza computazionale.

Cerchiamo di individuare quali possano essere i punti di forza del paradigma, ossia quali caratteristiche possano rendere il suo utilizzo più naturale o più conveniente per determinati tipi di applicazione.

- *Flessibilità* - Una volta installato il middleware di supporto, ossia un'infrastruttura software che permetta la realizzazione di applicazioni ad agenti, con i sufficienti requisiti di affidabilità, sicurezza ed efficienza, è possibile, per ogni diverso tipo di esigenza, progettare un'applicazione ad agenti diversa, che avrà accesso a risorse disponibili in siti diversi della rete.
  Questo perché l'agente trasporta il know how *di cui* ha bisogno e che può utilizzare *dove* ne avrà bisogno. Se pensiamo al Client/Server classico, ci accorgiamo che la situazione è completamente diversa: il client ha accesso al set di servizi imposto staticamente dal server. [FugPV98] Questo set di servizi, dovendo soddisfare le richieste di client diversi, non potrà necessariamente rispondere ad esigenze specifiche.

Il paradigma ad agenti prevede sostanzialmente che ad ogni esigenza diversa corrisponda un agente diverso che ha il compito di soddisfarla: si ha quindi una *personalizzazione dei servizi* con l'obiettivo di raggiungere una grande *flessibilità*.

- *Autonomia* - Un agente mobile non trasporta solo dati e know-how, ma anche il suo *stato*, che si trova sempre completamente *localizzato* nel place in cui si trova l'agente. In altre situazioni, come, ancora una volta nel Client/Server, lo stato di una computazione è *distribuito* su località diverse della rete, in particolare si troverà in parte sul client ed in parte sul server. Si avranno quindi problemi di sincronizzazione e di consistenza dello stato. Questi Problemi diventeranno tanto più critici quanto maggiori saranno le difficoltà di comunicazione.
  Nel paradigma ad agenti, è il supporto che si occupa di garantire la sopravvivenza degli agenti nei loro spostamenti. A livello di applicazione, quindi, non ci si deve preoccupare di problemi legati ad uno stato distribuito, e si potrà considerare quindi l'agente come un'entità relativamente *autonoma* e indipendente da altre.
- *Comportamento adattativo* - Questa proprietà è conseguenza delle due precedenti. Se un agente viene progettato in base ad esigenze specifiche e può essere considerato autonomo da altre entità della rete, si può fare in modo che i suoi comportamenti varino a seconda delle caratteristiche dell'ambiente che lo circonda. Lo si può progettare in modo che ad ogni passo scelga fra le azioni possibili quella che sembra più promettente ai fini del risultato finale, con un approccio mutuato dall'intelligenza artificiale.
  Per esempio, nel caso di un problema di ricerca su siti diversi, si potrebbe valutare la scelta in base sia alla probabilità di trovare le informazioni richieste, sia allo stato dei canali di comunicazione.

- *Scalabitiltà* - Il paradigma ad agenti fornisce una soluzione naturale al problema della scalabilità di un'applicazione: se il lavoro da effettuare può essere suddiviso in una serie anche molto grande di attività relativamente disaccoppiate e poco interdipendenti, ogni attività può essere affidata ad un singolo agente, dopo di che sarà sufficiente avviare un gran numero di agenti, sfruttando il parallelismo dell'esecuzione.
- *Apertura* - Se si è pronti a pagare il costo della standardizzazione del middleware di supporto agli agenti, si ottengono grandi vantaggi di apertura e di interoperabilità fra sistemi di organizzazioni diverse. Infatti, la standardizzazione e l'apertura dei sistemi permettono da un lato di accedere con facilità a servizi e risorse messi a disposizione da altre organizzazioni, dall'altro di avere un ampio "mercato" cui fornire le proprie risorse e servizi. Tutto questo esalta i vantaggi derivanti dalla flessibilità, che permette di utilizzare nella maniera più conveniente le risorse a disposizione.
- *Efficienza* - L'efficienza difficilmente può essere considerata come caratteristica peculiare di un paradigma, ma va ricercata utilizzando gli strumenti che il paradigma fornisce. La *flessibilità*, ossia la possibilità di ritagliare un'applicazione in base ad esigenze specifiche, può consentire di individuare la risorsa critica, ad esempio un certo canale di comunicazione, per ottimizzarne l'utilizzo. Nel caso in cui lo scenario, nell'ambito di una stessa applicazione, vari nel tempo, si potranno realizzare *comportamenti adattativi*, con lo scopo di massimizzare l'efficienza. Se poi la componente critica di un'applicazione è la quantità di lavoro da effettuare o l'ampiezza dello spazio su cui muoversi, si può far leva sulla facile *scalabilità* di un'applicazione ad agenti, che può portare a una maggiore efficienza, rispetto, ad esempio, ad approcci centralizzati che possono introdurre colli di bottiglia.

Elenchiamo ora una serie di applicazioni per cui può avere senso prediligere il paradigma ad agenti mobili rispetto ad altri, evidenziando i vantaggi ottenibili.

1.4.2 Network Management

Il controllo e l'amministrazione di reti di grandi dimensioni necessita lo sviluppo di strumenti che permettano di eseguire in remoto le procedure di normale manutenzione dei nodi.

Una delle necessità quotidiane, ad esempio, è la verifica del corretto funzionamento di tutti i componenti della rete. Uno strumento di monitoring di questo tipo necessita di grande flessibilità e tolleranza ai guasti, in quanto uno dei suoi compiti primari è appunto la notifica ed eventualmente il ripristino di situazioni di errore attraverso comportamenti adattativi. [Tar98]

Vediamo quali caratteristiche del paradigma si dimostrano utili in questo settore: la *scalabilità* permette di gestire facilmente anche grandi reti, su cui bisogna ripetere operazioni dello stesso tipo su tutti in nodi. La *flessibilità* può ridurre i costi di realizzazione di nuovi servizi di gestione, realizzabili con la progettazione di agenti ad hoc. *L'autonomia* ed i *comportamenti adattativi* sono utilizzabili per garantire un adeguato livello di fault tolerance: ogni agente potrebbe prendere, in caso di guasti o malfunzionamenti, decisioni opportune in maniera indipendente. *L'apertura*, infine, può facilitare la connesione di nuove macchine in rete. In [Tar98], infine, è proposto un sistema di Network Management che, in determinate condizioni, ha *prestazioni* superiori a quelle di un sistema tradizionale centralizzato.

1.4.3 Information Retrieval

L'information retireval è la ricerca di informazioni in rete. Questa può trarre grandi benefici dal paradigma ad agenti mobili.

Innanzitutto esistono basi di dati in cui le informazioni sono molto strutturate o complesse, come nel caso dei progetti realizzati con un sistema CAD/CAM[4] o di immagini derivanti da radiografie o ecografie. In questo caso un sistema client/server può essere inadeguato per fornire l'accesso alla base di dati, e può essere più conveniente affidare ad un agente il know-how necessario per effettuare la ricerca, sfruttando quella che abbiamo indicato come *flessibilità* del paradigma.

Un altro caso da considerare, ad esempio, è quello di una ricerca incrociata su basi di dati diverse che si trovano in siti diversi (ad esempio un'operazione di join). In questo caso un approccio tradizionale prevede di trasportare verso il client un sottoinsieme dei dati contenuti nei due database, per fare la ricerca incrociata in locale e produrre, magari, un numero molto esiguo di risultati utili. Questo comporta la necessità di trasportare grandi masse di dati pur ricercando un risultato poco voluminoso. Con l'approccio ad agenti sarebbe possibile ottimizzare la ricerca, considerando l'efficienza dei diversi canali di comunicazione e programmando un agente per minimizzare i tempi di trasferimento. Nel caso precedente, ad esempio, avrebbe senso trasportare uno dei due risultati della ricerca direttamente verso l'altro database, per riportare infine a destinazione solo i risultati realmente interessanti e non informazioni ancora da filtrare. In questo modo si può far leva sulla *flessibilità* del paradigma per ottenere *efficienza*. Se poi l'efficienza dei canali di comunicazione varia nel tempo, si potrebbero progettare *comportamenti adattativi*.

Anche la standardizzazione e *l'apertura* dei sistemi potrebbero rivelarsi utili, in quanto renderebbero disponibili grandi quantità di dati, anche di tipo eterogeneo, su cui fare ricerche ad hoc, con criteri ogni volta ritagliati sulle proprie esigenze. Se poi lo spazio di ricerca, ossia il numero di siti potenzialmente da esplorare, si

[4] *CAD/CAM* - Computer Aided Design, Computer Aided Manufacturing.

rivelasse troppo ampio per un solo agente, si potrebbero effettuare ricerche in parallelo (*scalabilità*) o determinare dinamicamente (*comportamentio adattativo*) i siti da esplorare utilizzando tecniche dell'intelligenza artificiale o dell'ottimizzazione combinatoria

1.4.4 Commercio Elettronico

Per Commercio Elettronico si intende la possibilità di effettuare transazioni commerciali in rete, con forme di pagamento non tradizionali. Le problematiche sono molteplici, soprattutto incentrate sulla necessità di garantire la sicurezza dei pagamenti e l'autenticazione dell'identità dei partecipanti alla transazione.

Nell'approccio ad agenti mobili, le problematiche di sicurezza dovrebbero essere affrontate il più possibile a livello di supporto. Si dovrebbe anche procedere ad un processo di standardizzazione ed *apertura*, per poi costruire negozi virtuali capaci di accogliere agenti che abbiano il compito di ricercare prodotti, di contrattare offerte particolari e di effettuare le transazioni commerciali, il tutto in maniera *personalizzabile* e *autonoma*.

1.4.5 Sistemi di Controllo a Distanza

Una soluzione naturale è quella di attivare un place sul dispositivo da controllare e di spedire un agente programmato secondo le necessità del controllo da effettuare. Si sfrutterebbe la *flessibilità*, nel senso che si potrebbe con facilità sostituire l'agente controllore a seconda delle esigenze, l'*autonomia*, nel senso che l'agente potrebbe lavorare autonomamente, senza un contatto costante con un'ipotetica centrale operativa, i *comportamenti adattativi*, in quanto l'agente potrebbe avere il know-how necessario per prendere decisioni "intelligenti" a seconda delle necessità. La *scalabilità*, in questo caso va vista come la possibilità di controllare un gran numero di dispositivi con agenti dello stesso tipo, semplicemente creandoli e spedendoli a destinazione.

1.4.6 Documenti Attivi

Si potrebbe immaginare di modellare un documento con un agente. Un documento avrebbe così un contenuto, uno stato e delle operazioni effettuabili (fra cui il trasferimento), il tutto definito in maniera formale, in modo da garantire, ad esempio, la consistenza del contenuto di versioni diverse prodotte da persone diverse in luoghi diversi. L'agente potrebbe anche gestire esplicitamente gli spostamenti del documento, la sua distribuzione, ecc.

1.4.7 Mobile Computing

Quello del Mobile Computing sembra essere un importante settore di applicazione del paradigma ad agenti mobili. Nel secondo capitolo si parlerà diffusamente di Mobile Computing e di mobilità, mentre nel terzo capitolo vedremo in che modo il paradigma ad agenti mobili possa rivelarsi adatto a rispondere alle esigenze della mobilità.

Cap.2 Mobile Computing

2.1 Introduzione

Negli ultimi anni il mondo dell'informatica da un lato e quello delle telecomunicazioni dall'altro, hanno subito enormi cambiamenti. I prodotti delle nuove tecnologie non sono rimasti oggetto di curiosità, o per specialisti e addetti ai lavori, ma hanno in poco tempo invaso i mercati e raggiunto il grande pubblico. Le curve di diffusione sul mercato di Internet o dei telefoni cellulari sono molto più ripide rispetto a quelle di prodotti innovativi del passato come la radio, la televisione o il telefono, che hanno richiesto molto più tempo per raggiungere una diffusione capillare.

Nel prossimo futuro i colossi dell'informatica si stanno preparando a fronteggiarsi su un campo relativamente nuovo: quello dei sistemi operativi e delle applicazioni per architetture hardware compatte ed economiche, tali da poter essere utilizzate non solo nei computer palmari o nei personal digital assistants, ma addirittura negli elettrodomestici, o come centraline di controllo intelligenti, che si collegano alla rete quando è necessario. Per una esemplificazione è sufficiente pensare a *Windows CE* [Mic], il sistema operativo per computer portatili con cui la Microsoft conta di ripetere il successo ottenuto nell'ambito dei Personal Computers. Un approccio alternativo è quello proposto dalla Sun Microsystems che, con la tecnologia *Java Embedded* [Jav], vorrebbe vedere il suo linguaggio utilizzato universalmente, dai grandi sistemi fino ai computer palmari o alle architetture specializzate, con processori in grado di eseguire direttamente il Bytecode della Java Virtual Machine. La SUN ha anche proposto recentemente la tecnologia *Jini* [Jin99], grazie alla quale apparecchiature di uso comune, come

televisori, telefoni cellulari, fax e, ovviamente, computer, potrebbero connettersi in rete ed interagire fornendo servizi ed accedendo a servizi forniti da altri, in maniera dinamica e senza bisogno di configurazione o installazione preventiva. Il tutto, anche in questo caso, è basato sull'utilizzo del linguaggio Java.

Nell'ambito delle telecomunicazioni la competizione è forse ancor più agguerrita, i monopoli nazionali stanno cadendo un po' ovunque e le multinazionali si fanno concorrenza a livello mondiale, tanto da essere spesso costrette a fusioni per sfruttare le sinergie fra le proprie posizioni strategiche, per ridurre gli enormi costi di gestione delle infrastrutture e creare così vantaggi competitivi.

In questo momento sono nella fase di standardizzazione i cosiddetti Sistemi Mobili della Terza Generazione, che introdurranno molte novità rispetto ai sistemi attuali come il GSM (Global System for Mobile Communication), che è considerato di seconda generazione. L'obiettivo non è solo quello di aumentare la velocità di trasmissione dei dati, ma di rendere i servizi di telecomunicazione disponibili ovunque ed in maniera personalizzata.

Viste queste premesse non è difficile immaginare come nei prossimi anni osserveremo una rapida diffusione di computer portatili che si collegheranno alla rete sfruttando connessioni senza fili (wireless), e vorranno accedere ai servizi forniti su Internet possibilmente come se fossero macchine collegate in maniera stabile alla rete. Il principale elemento di novità introdotto è proprio quello della mobilità: per questo si parla di *Mobile Computing*, termine che può indicare sia la mobilità del *terminale* di accesso alla rete, che della *persona* che accede ai servizi informatici. Ora, se il mondo delle telecomunicazioni ha già affrontato il problema, Internet è stata progettata avendo in mente comunicazioni fra macchine fisse, collegante con connessioni relativamente stabili. Così, ad esempio, un nodo della rete è

individuato dal suo indirizzo IP, che fa riferimento direttamente alla sottorete di appartenenza e non ha alcun concetto base per il supporto della mobilità. Allo stesso modo il paradigma di progettazione di applicazioni distribuite più diffuso è il Client-Server, che, nella configurazione classica, prevede un collegamento stabile fra cliente e servitore per tutta la durata della loro interazione. È evidente che questi meccanismi non rispondono, così come sono, alle esigenze del Mobile Computing, quindi serviranno strumenti e concetti nuovi.

Questo è ovviamente solo un aspetto della questione, che non interessa soltanto gli ingegneri elettronici, informatici o delle telecomunicazioni che si occupano della progettazione e del funzionamento dei sistemi. Infatti, se con un telefono cellulare è già possibile entrare in contatto con chiunque, in qualsiasi momento e da un luogo qualsiasi, nelle stesse condizioni, in un prossimo futuro, sarà possibile accedere a tutte le risorse e a tutti i servizi forniti in rete. Questo ridurrà notevolmente l'esigenza di spostarsi per accedere alle risorse e cambierà notevolmente il nostro stile di vita. Basti pensare che la risorsa di cui si parla può essere una banca, una sala conferenze in cui incontrarsi con altre persone, un'aula scolastica, un negozio, un'équipe medica, o anche un gruppo di amici con cui giocare o suonare.

2.1.1 Tipi diversi di Mobilità

Innanzitutto bisogna osservare che il termine Mobile Computing è piuttosto generico, ed è necessario chiarire meglio i termini del discorso quando si parla di mobilità. Una prima distinzione che può essere fatta è quella fra *mobilità personale (personal mobility)* e *mobilità a livello del terminale (terminal mobility).* [JunP98] Questa distinzione viene fatta principalmente nel settore delle telecomunicazioni, che si è trovato ad affrontare questo tipo di

problemi prima del settore informatico, e di conseguenza ha già fornito delle classificazioni e delle risposte.

2.1.2 La Teminal Mobility

La *terminal mobility* prevede che l'utente mobile usufruisca dei servizi, sottoscritti presso un service provider, attraverso un ben preciso terminale mobile, che si tratti di un telefono cellulare o di un fax gsm, o altro ancora.

Nell'ambito delle reti di calcolatori la configurazione tipica è quella di una rete di comunicazione wired che collega host fissi, ed un insieme di host mobili che si collegano a determinati host fissi attraverso collegamenti di varie tipologie, a seconda delle situazioni o delle esigenze. È possibile che con la stessa macchina e nello stesso giorno, l'utente colleghi il portatile direttamente alla propria LAN in ufficio, poi si ricolleghi a Internet attraverso un cellulare GSM, poi da casa via modem su rete telefonica terrestre. È ovvio che questo utente ipotetico desidererebbe utilizzare le stesse applicazioni, nello stesso modo in tutti e tre i casi, per non essere costretto ad installare ed apprendere l'utilizzo di software diversi a seconda delle esigenze. Chiaramente rimangono le pesanti differenze esistenti fra le caratteristiche dei canali di comunicazione utilizzati: infatti, se da un lato il collegamento alla LAN del proprio ufficio è affidabile, ha una grande larghezza di banda ed è a costo zero, dall'altro, il collegamento GSM è molto costoso, ha una banda ridotta e può cadere in ogni momento. Di conseguenza è auspicabile che le stesse applicazioni siano abbastanza flessibili da poter operare in condizioni così diverse adattando i propri comportamenti alle risorse disponibili e alle loro caratteristiche, tutto ciò nella maniera più trasparente possibile nei confronti dell'utente finale.

Nell'ambito della terminal mobility è possibile fare un'ulteriore distinzione: il sistema può permettere la mobilità fra una sessione

di lavoro e l'altra, ma non nell'ambito della stessa sessione di lavoro, oppure anche nell'ambito della stessa sessione. Nel primo caso, quando non è ammessa la mobilità ad applicazioni avviate, si parla più propriamente di *portabilità*, mentre se è possibile spostarsi in maniera trasparente alle applicazioni (ad esempio cambiare il proprio indirizzo IP senza dover riavviare il sistema), allora si parla di *moblità* vera e propria [Per97].

È stato proposto un protocollo per affrontare questi problemi direttamente a livello di rete, nella gerarchia OSI. Si tratta del protocollo Mobile IP, che permette di realizzare la mobilità in maniera trasparente alle applicazioni. Affronteremo quindi in maniera più approfondita queste questioni nella sezione dedicata al protocollo Mibile IP.

2.1.3 La Personal Mobility

La *personal mobility*, a differenza della terminal mobility, prevede che l'utente non sia costretto a portarsi dietro un terminale ben preciso, ma solo le sue informazioni di identificazione e autenticazione. Quando sarà necessario, potrà registrarsi presso un terminale qualsiasi ed usufruire dei servizi da lui sottoscritti a suo nome, e non relativamente ad un terminale. Le informazioni necessarie alla propria registrazione, che potrebbero semplicemente essere un nome utente ed una password, in realtà saranno tali da garantire una maggior sicurezza e praticità, e verranno quindi memorizzate tipicamente su smart card standard. In questo contesto si introdurrà anche un altro concetto: quello di *Virtual Home Environnment*. Ogni utente definirà un proprio ambiente personalizzato, in cui definirà, ad esempio, nel caso delle telecomunicazioni, la politica di gestione delle telefonate entranti, o delle sessioni di teleconferenza. Ogni volta, quindi, che accederà ai servizi, anche su terminali diversi o su reti diverse, si vedrà

presentare il proprio ambiente personalizzato. Di tutto questo parleremo più approfonditamente nella sezione 2.3.

2.2 La mobilità a livello di rete: il protocollo Mobile IP

Per affrontare il problema della mobilità a livello di rete la Internet Engineering Task Force[5] ha proposto l'introduzione di un nuovo protocollo di rete: il *Mobile IP*, estensione dell'Internet Protocol (IP) su cui si basa tutto il funzionamento della rete Internet [Per97].

Il protocollo Mobile IP presuppone la cooperazione di tre grandi sottosistemi. Innanzitutto, c'è un meccanismo tale che i computer mobili possano determinare il loro nuovo indirizzo IP quando si spostano da un posto all'altro su Internet. Secondo, una volta che il portatile conosce l'indirizzo IP del nuovo punto di connessione, si *registra* presso un agente che lo rappresenta nella rete di partenza (*home network*), nel senso che gli comunica la sua nuova posizione. Infine, il protocollo definisce semplici meccanismi per recapitare datagrammi al nodo mobile quando questo è lontano dalla sua home network.

2.2.1 Problemi di Realizzazione della Mobilità

Gli indirizzi IP vengono impiegati oggi per identificare una macchina particolare. In questo senso, gli indirizzi IP vengono spesso considerati semanticamente equivalenti ai Fully Qualified Domain Names (FQDN) dei Domain Name Servers (DNS) [CorS98]. In altre parole si può concettualmente usare sia l'indirizzo IP che il FQDN al fine di individuare un nodo

[5] *Internet Engineering Task Force (IETF)* - Una comunità internazionale grande ed aperta di progettisti, ricercatori e operatori commerciali interessati all'evoluzione dell'architettura di Internet ed a suo buon funzionamento operativo.[IETF]

particolare fra i milioni di computer che fanno parte di Internet. Sistemi di trasporto comuni come il Transmission Control Protocol (TCP) tengono traccia dello stato della sessione di comunicazione utilizzando gli indirizzi IP dei due punti fra cui avviene la comunicazione.

Gli indirizzi IP vengono anche utilizzati per trovare un percorso che unisce due nodi sulla rete, percorso che non deve essere necessariamente lo stesso in entrambi i versi di percorrenza. Modellando la sessione come un byte stream bidirezionale, l'indirizzo IP di destinazione per i datagrammi che fluiscono in un verso coinciderebbe con l'indirizzo IP di partenza dei i datagrammi che fluiscono nel verso opposto. Tipicamente il percorso selezionato per un datagramma dipende solo dall'indirizzo di destinazione e non, ad esempio, dall'indirizzo di origine, dall'ora del giorno o dalla lunghezza totale del messaggio. L'unico altro fattore che influenza la scelta è lo stato attuale di congestione della rete. In altre parole, un cammino che viene prescelto in condizioni normali da un router intermedio per una destinazione particolare, potrebbe venire scartato se il traffico in quella direzione è ritardato o interrotto completamente per problemi di congestione.

Le funzionalità di identificazione e ritrovamento assoluto dell'indirizzo IP portano ad una situazione contradditoria per il mobile computing: da un lato il computer mobile ha bisogno di avere un indirizzo IP stabile al fine di essere identificabile dagli altri computer su Internet; dall'altro, se l'indirizzo è stabile, anche il percorso verso il computer mobile lo è, negando quindi la mobilità. Il protocollo Mobile IP, di conseguenza, estende il protocollo IP dando al computer mobile la possibilità di utilizzare due indirizzi: uno per l'identificazione e l'altro per il routing.

Sono stati fatti vari tentativi per gestire lo spostamento dei computer su Internet. In prima approssimazione, è sicuramente possibile per un nodo mobile acquisire un nuovo indirizzo IP ad ogni punto di connessione. Questo funzionerà finché il nodo

mobile non si sposterà: allora il vecchio indirizzo non sarà più utilizzabile, e sarà necessario procurarsene un altro. Sfortunatamente, questo significa di solito che ogni programma Client sul nodo mobile cesserà di funzionare, cosicché il computer mobile dovrà riavviare tutti i sottosistemi di gestione della rete. Molti utenti non attueranno soluzioni così specifiche e semplicemente riavvieranno tutto il sistema. Questo approccio non è così controindicato se un consistente lasso di tempo interviene fra due successive nuove connessioni. Le esigenze di molti utenti di computer mobili sono soddisfatte da questo modo di operare, che viene indicato con il termine **portabilità**. In altre parole un sistema viene detto *portabile* quando può connettersi a vari punti della rete, cambiando ogni volta il proprio indirizzo IP.

Anche nell'utilizzo di un sistema portabile, comunque, sorgono alcune difficoltà di implementazione: la maggior parte delle applicazioni identificano inizialmente un nodo di Internet attraverso il suo nome di dominio (FQDN), ma in seguito utilizzano soltanto l'indirizzo IP del nodo. Al fine di contattate il nodo, l'applicazione consulta il server DNS appropriato per ottenere un indirizzo IP. Così, se l'indirizzo IP viene allocato dinamicamente, o il server avrà l'indirizzo sbagliato o dovrà ricevere aggiornamenti da parte del nodo portabile. Siccome il DNS è tipicamente nel cuore amministrativo nella maggior parte delle reti connesse ad Internet, ogni protocollo progettato per alterarne il contenuto informativo deve essere accuratamente progettato, implementato e gestito. Più sono frequenti gli aggiornamenti ai record del DNS, più è probabile che vi siano errori, o che tutto il sistema collassi. Come minimo sorgono sensibili problemi di sicurezza, visto che ogni amministratore di rete dovrebbe lasciare a migliaia di nodi mobili la possibilità di alterare qualche record qua e là nel proprio DNS.

Con il termine **mobilità** si indica qualcosa in più rispetto alla portabilità: i computer mobili wireless possono connettersi ad

Internet e rimanere connessi anche quando si spostano da un posto all'altro, stabilendo nuovi collegamenti e lasciando i vecchi. Si è quindi pensato che risolvere il problema a livello di rete modificando lo stesso protocollo IP, avrebbe portato grandi vantaggi, fra cui la trasparenza dello spostamento a livello di applicazione, e lo spostamento senza interruzioni (*seamless roaming*). La trasparenza a livello di applicazione è un requisito fondamentale, perché è impensabile costringere tutti gli utenti mobili all'acquisto di nuove applicazioni che gestiscano esplicitamente la mobilità. Il seamless roaming, sebbene non sia strettamente necessario, sarà sicuramente apprezzato, quando saranno ampiamente sfruttate le infrastrutture che permettono la mobilità dei sistemi wireless. Inoltre il seamless roaming è un altro passo avanti verso la trasparenza della mobilità a livello di applicazione.

2.2.2 Terminologia

Prima di entrare nel dettaglio della descrizione del protocollo, è bene inquadrare il discorso fissando un po' di terminologia specifica. Le specifiche del Mobile IP introducono le seguenti unità funzionali:

Nodo mobile - Un host o un router che cambia il suo punto di connessione da una rete o sottorete ad un'altra, senza cambiare il suo indirizzo IP. Un nodo mobile può continuare a comunicare con gli altri nodi su Internet utilizzando il suo indirizzo IP fisso.

Home agent - Un router nella home network del nodo mobile che spedisce datagrammi ai nodi mobili lontani, e mantiene le informazioni relative alla posizione corrente di ciascun nodo.

Foreign agent - Un router sulla rete visitata dal nodo mobile che coopera con l'Home agent per completare il recapito dei datagrammi al nodo mobile che si trova lontano da casa.

Un nodo mobile ha un home address, che è un indirizzo IP di lungo termine nella sua home network. Quando è lontano da casa, gli viene associato un indirizzo *care-of (c/o)*, che riflette il punto di connessione corrente del nodo. Il nodo mobile usa il suo indirizzo "di casa" come indirizzo di origine per tutti i datagrammi che spedisce, eccetto nei casi in cui si richiede un comportamento diverso, per alcuni datagrammi contenenti richieste di registrazione.

I seguenti termini sono utilizzati di frequente in relazione al Mobile IP:

Mobility Agent - Sia un Home Agent, che un Foreign Agent.

Care-of Address - Il punto terminale di un tunnel verso un nodo mobile, per datagrammi rispediti (forwarded) al nodo mobile quando è lontano da casa. Ci sono due tipi di care-of address: il *foreign agent care-of address* è l'indirizzo di un foreign agent presso cui il nodo mobile si è registrato; mentre il *collocated care-of address* è un indirizzo locale ottenuto dall'esterno, che il nodo mobile ha associato con una delle sue interfacce di rete.

Agent advertisement - I Foreign Agents rendono nota la loro presenza utilizzando un messaggio speciale, che è costruito attaccando un'estensione speciale ad un router advertisement

Home address - Un indirizzo IP assegnato per un periodo di tempo esteso ad un nodo mobile. Questo rimane invariato indipendentemente dal punto di connessione del nodo mobile alla rete.

Home network - Una rete, che può anche essere virtuale, che ha un prefisso di rete che corrisponde a quello dell'home address del nodo mobile. Notare che il meccanismo di routing dell'IP standard spedirà alla home network del nodo mobile i datagrammi destinati al suo home address.

Foreign network - Ogni altra rete diversa dalla home network del nodo mobile.

Virtual network - Una rete con nessuna istanza fisica oltre il suo router, che ha un'istanziazione fisica su un'altra rete. Il router (nel nostro caso l'home agent), generalmente comunica la raggiungibilità della rete virtuale con protocolli di routing convenzionali.

Tunnel - Il percorso seguito da un datagramma quando è incapsulato. Il modello stabilisce che il datagramma viene incapsulato e spedito ad un agente conosciuto, che estrae il datagramma originale e lo spedisce alla destinazione finale.

2.2.3 Descrizione del protocollo

Il Mobile IP fornisce un modo per svolgere tre funzioni collegate:

Agent Discovery - I Mobility Agents rendono nota la loro disponibilità su ogni collegamento (link) per cui forniscono servizio.

Registration - Quando il nodo mobile si trova lontano da casa, *registra*, ossia comunica il suo Care Of Address al suo Home Agent.

Tunnelling - Al fine di recapitare i datagrammi al nodo mobile lontano da casa, l'home agent deve spedire i datagrammi al care-of address del nodo mobile.

In seguito verranno delineate le operazioni previste dal protocollo. La Figura 2.1 può essere di aiuto nel visualizzare il ruolo giocato dalle diverse entità in gioco.

- I *mobility agents* rendono nota la loro presenza spedendo messaggi di *agent advertisement*. Un nodo mobile impaziente può opzionalmente sollecitare un messaggio di *agent adevrtisement*.
- Ricevuto un *agent advertisement*, un nodo mobile determina se si trova sulla sua *home network* o su una *foreign network*. Un

nodo mobile si comporta come ogni altro nodo, quando si trova sulla sua rete di origine.

- Quando un nodo mobile si allontana dalla sua home network, ottiene un *care-of address* sulla *foreign network*, per esempio sollecitando, oppure aspettando un *agent advertisement*, oppure contattando il Dynamic Host Configuration Protocol (DHCP) o il Point-to-Point Protocol (PPP) [CorS98].
- Quando è lontano da casa, il nodo mobile registra ogni nuovo *care-of address* presso *l'home agent*, possibilmente attraverso il *foreign agent*.
- I datagrammi spediti all'*home address* del nodo mobile sono intercettati dall'*home agent*, spediti incapsulati (tunneled) dall'*home agent* al *care-of address*, ricevuti all'altro capo del *tunnel* (che può essere il sia il foreign agent, che lo stesso nodo mobile) e finalmente recapitati al *nodo mobile*.
- Nella direzione opposta, i datagrammi spediti dal *nodo mobile* sono generalmente inviati alla destinazione utilizzando i meccanismi di routing del protocollo IP standard, non necessariamente passando attraverso l'*home agent*.

Quando l'home agent spedisce attraverso il tunnel un datagramma al suo care-of address, l'intestazione contenente l'indirizzo IP di destinazione (ossia l'home address del nodo mobile) è effettivamente nascosta ai router che ricevono il pacchetto in un punto intermedio del percorso, cosicché il pacchetto raggiunge il care-of address. Solo lì il pacchetto spedito originariamente viene estratto e recapitato al nodo mobile.

È compito di ogni home agent attirare ed intercettare i datagrammi destinati all'home address di ogni nodo mobile registrato presso di lui. Quando vengono utilizzati i foreign agents, similmente, il modo naturale di procedere suggerisce che il nodo mobile sia capace di collegarsi al suo foreign agent. Sono comunque possibili altre configurazioni non definite dal mobile IP, e quindi invisibili ad esso. Si noti come, se l'home agent è il solo

router che avverte la raggiungibilità della home network, ma non c'è nessun collegamento fisico che istanzia la home network, allora tutti i datagrammi trasmessi ai nodi mobili domiciliati in quella home network raggiungeranno naturalmente l'home agent senza bisogno di creare nessun collegamento speciale.

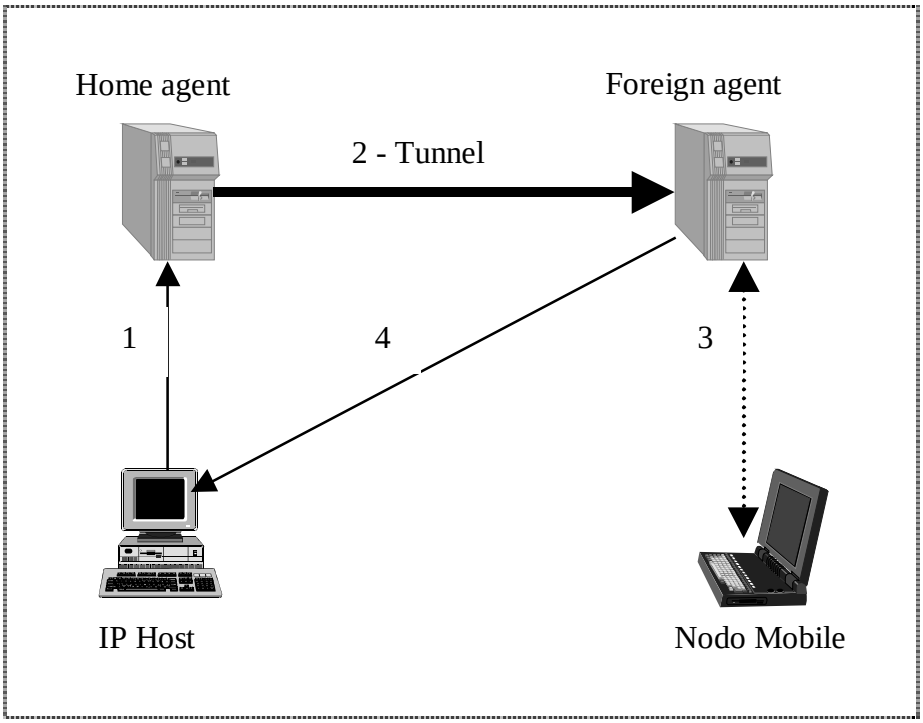

Figura 2.1 Protocollo Mobile IP, percorso dei datagrammi da e verso un nodo mobile.

La Figura 2.1 illustra il percorso di datagrammi da e verso un nodo mobile che si trova lontano da casa, una volta che si è registrato presso il suo home agent. Si presume che il nodo mobile utilizzi un care-of address fornito dal foreign agent:

- Un datagramma spedito al nodo mobile arriva alla home network, con il routing del protocollo IP standard.
- Il datagramma è intercettato dall'home agent ed è spedito attraverso il tunnel al care-of address, come indicato dalla freccia più spessa.

- Il datagramma è estratto e spedito al nodo mobile.
- Per datagrammi spediti dal nodo mobile, si applica il routing previsto dallo standard IP, ossia questi vengono inviati direttamente alla loro destinazione. Nella figura il foreign agent è il router di default del nodo mobile.

A questo punto non è necessario aggiungere ulteriori dettagli relativi al protocollo, a parte qualche accenno ai criteri di sicurezza della fase di registrazione di un nodo mobile.

2.2.4 La sicurezza in Mobile IP

La procedura di registrazione nel protocollo IP mobile deve essere resa sicura, cosicché registrazioni fraudolente possano essere riconosciute e respinte. Altrimenti ogni utente malizioso di Internet potrebbe interrompere la comunicazione fra l'home agent ed in nodo mobile con il semplice espediente di fornire una richiesta di registrazione contenente un care-of address contraffatto, magari proprio l'indirizzo IP dell'utente malizioso. Questo interromperebbe tutto il traffico destinato al nodo mobile.

Quello dell'identificazione e dell'autenticazione degli utenti è un problema comune ai sistemi che permettono a più persone di accedere a risorse comuni. L'obiettivo è quello di identificare in maniera certa l'utente, ossia di autenticarlo, in modo da potergli attribuire la responsabilità delle azioni compiute a suo nome. L'autenticazione di un utente si basa tradizionalmente [Ten98] sulla combinazione di elementi di tre tipi:

- si chiede all'utente *qualcosa che possiede,* come una smart card,
- si chiede all'utente *un segreto che conosce,* come una password,
- si verificano determinatii *attributi* dell'utente, ad esempio le sue impronte digitali o il disegno della sua returna.

Il protocollo stabilito nelle specifiche, per proteggersi da attacchi di utenti ostili, prevede l'inclusione di un valore non falsificabile nella richiesta di registrazione, valore che cambia ad

ogni nuova registrazione. Al fine di rendere ogni valore diverso, viene inserito nel campo di identificazione un *timestamp*, ossia l'orario della registrazione, o un nuovo numero casuale, un cosiddetto *nonce*[6]. L'home agent ed il nodo mobile devono accordarsi su valori ragionevoli per il timestamp o il nonce, e il protocollo permette operazioni di risincronizzazione.

Ci sono tre estensioni di autenticazione definite per il mobile IP:

- L'estensione per l'autenticazione mobile-home.
- L'estensione per l'autenticazione mobile-foreign.
- L'estensione per l'autenticazione foreign-home.

L'autenticazione mobile-home è richiesta in tutte le richieste di registrazione ed in tutte le risposte. Il Security Parameters Index (SPI)[7] dentro ognuna delle estensioni di autenticazione definisce il contesto di sicurezza utilizzato per calcolare e verificare l'autenticatore. Un nodo mobile deve essere in grado di associare a valori arbitrari dell'SPI i rispettivi algoritmi e modi di autenticazione che implementa.

---

[6] *Nonce* - Un valore estratto a caso, diverso dai valori precedenti, inserito in un messaggio per proteggersi dai *replays*, ossia per evitare che un utente malizioso intercetti il messaggio e lo spedisca di nuovo in maniera non autorizzata.

[7] *SPI* Security Parameters Index - Un indice che identifica un contesto di sicurezza fra una coppia di nodi, scelto fra i contesti disponibili nella *Mobility Security Association.* Questo è un insieme di contesti di sicurezza fra coppie di nodi, contesti che possono essere applicati nello scambio di messaggi fra i nodi stessi nel protocollo mobile IP. Ogni contesto indica un algoritmo ed un modo di autenticazione, un segreto (una chiave condivisa, oppure appropriate coppie di chiavi pubbliche/private), ed uno stile di protezione dal *replay* (vedi nota precedente).

2.2.5 Punti deboli di Mobile IP

Si è visto come il protocollo Mobile IP si propone di affrontare il problema della mobilità a livello di rete, in modo da rendere la mobilità trasparente ai livelli OSI superiori. Questo impone ai pacchetti diretti al nodo mobile di seguire un percorso piuttosto tortuoso, in quanto questi devono prima raggiungere l'home agent, poi il foreign agent ed infine il nodo mobile interessato. Se si pensa che sicuramente, in un collegamento wireless, il problema di individuare la posizione del nodo mobile per recapitargli effettivamente le informazioni deve essere affrontato di nuovo ad un livello OSI inferiore, ci si accorge che si ha una sovrapposizione di funzioni. Si potrebbero quindi considerare soluzioni alternative al Mobile IP, come quella di utilizzare direttamente le reti wireless ATM come supporto per la mobilità [Var97].

Inoltre l'overhead introdotto dal Mobile IP potrebbe diventare molto pesante, ad esempio nel caso in cui l'host che spedisce il messaggio ed il nodo mobile si trovino "vicini", mentre l'home agent sia in una posizione difficilmente accessibile a causa, ad esempio, di uno stato di congestione della rete (Figura 2.2). In questo caso potrebbe essere opportuno gestire esplicitamente la mobilità e fare in modo che la comunicazione fra i due pari segua un percorso più diretto.

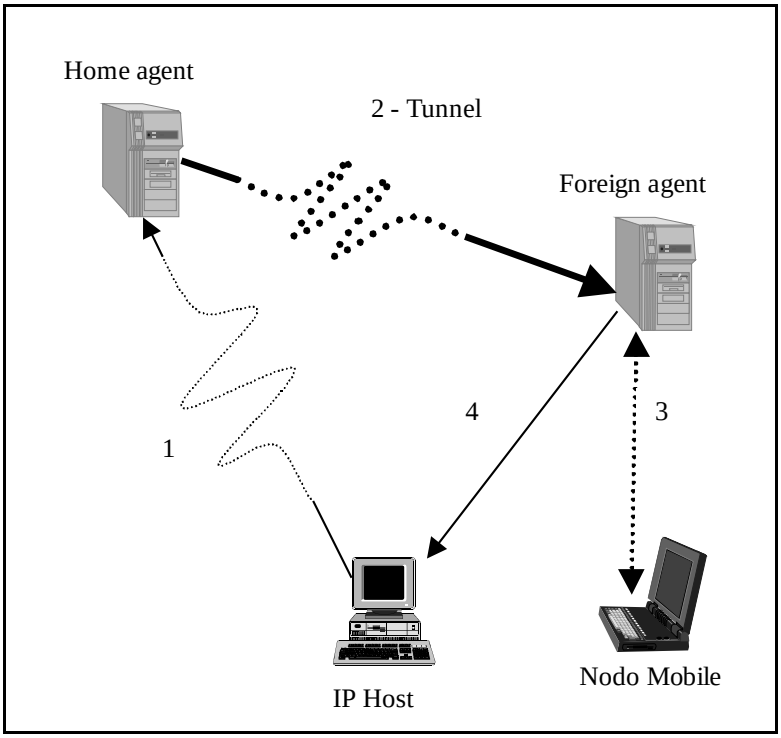

Figura 2.2 Protocollo Mobile IP: difficoltà di accesso allo Home Agent.

Occorre poi fare un'altra osservazione in merito al Transmission Control Protocol (TCP), che non è stato progettato per collegamenti senza fili, ma per collegamenti cablati, che introducono raramente errori nella trasmissione dei pacchetti IP. In un ambiente del genere, TCP presuppone che il mancato ricevimento di un pacchetto derivi da una situazione di congestione della rete: di conseguenza il protocollo riduce la velocità di trasmissione dei pacchetti, in modo da favorire il decongestionamento. Questo comportamento, se molto indicato per collegamenti wired, può essere addirittura controproducente nel caso di collegamenti wireless, in cui la perdita di un pacchetto è più probabile venga causata da errori nel canale. In questo caso TCP dovrebbe aumentare la frequenza di spedizione e non diminuirla [JamSJ98]. Per risolvere questo problema è stato proposto uno stratagemma che agisce a livello di linea: viene introdotto uno

snooping agent che osserva ed inserisce in una cache i pacchetti TCP spediti dal nodo mobile in una linea wireless. Lo snooping agent effettua un monitoraggio dei pacchetti di acknowledge spediti dal nodo mobile a conferma della ricezione dei pacchetti a lui diretti. Con questa tecnica l'agente può rilevare la mancata ricezione da parte del nodo mobile dei pacchetti, e rispedire i quelli che sono andati persi, prelevandoli dalla sua cache. In questo modo l'agente evita gli effetti dannosi della perdita dei pacchetti, che, se rilevata dal mittente, produrrebbe un rallentamento della frequenza di spedizione.

2.3 La Mobilità nel mondo delle telecomunicazioni

Il mondo delle telecomunicazioni ha affrontato il problema della mobilità già da tempo, ed anche se il settore è in rapidissima evoluzione, l'argomento può ormai essere considerato maturo. Per questo è molto utile vedere con quale approccio sono già stati affrontati gli stessi problemi che ora si presentano nell'ambito del Mobile Computing.

D'altra parte tutte le soluzioni possibili proposte per il Mobile Computing dovranno comunque appoggiarsi a sistemi di telecomunicazioni avanzati ed efficienti, che, a livello più basso, realizzeranno effettivamente le comunicazioni, nell'ottica della separazione dei diversi livelli di astrazione. Quindi è fondamentale capire in che direzione si sta muovendo il mondo delle telecomunicazioni, e che tipo di infrastrutture saranno disponibili nei prossimi anni.

Per questo presentiamo la descrizione dell'UMTS (Universal Mobile Telecommunication System) [HagBM98] standardizzato per opera dello European Telecommunications Standards Institute (ETSI), su cui si baseranno probabilmente i sistemi di comunicazione mobili del futuro. Questi sistemi, noti come sistemi

mobili della terza generazione, verranno messi in opera nel giro di qualche anno.

L'UMTS supporterà comunicazioni wireless con un'ampiezza di banda che arriva ai 2 Mb, che permetterà lo sviluppo e la diffusione di servizi oggi disponibili solo su rete fissa, come i servizi di video conferenza o la trasmissione di dati ad alta velocità. Questi nuovi servizi verranno forniti da terminali mobili attraverso interfacce radio innovative e migliori, e attualmente si intende renderli globalmente disponibili in modo indipendente dalla posizione dell'utente. La loro disponibilità sarà assicurata utilizzando diverse tecnologie di accesso, che includono reti fisse, radio e satellitari.

L'offerta di nuovi servizi e nuove possibilità renderà l'UMTS molto attraente sia per i *service providers*, che per gli utilizzatori finali. Alcuni principi chiave di questo sistema sono:

- Il supporto del QoS (Quality of Service), che vuol dire offerta garantita di servizi wireless attraverso reti di tecnologia differente.
- Il supporto del Personal Communication Support (PCS), che può essere visto come uno sviluppo ulteriore della mobilità personale, con una maggior raggiungibilità dei servizi ed una loro migliore integrazione.
- La realizzazione del concetto di Virtual Home Environnemnt, che permette agli utenti mobili un accesso unificato, ossia un comune "look and feel", ai servizi forniti anche attraverso reti diverse e providers con capacità variabile.

Tutti insieme, questi principi permetteranno l'offerta di servizi ed informazioni ovunque, in qualsiasi momento ed in qualsiasi forma. Fondamentalmente l'UMTS sarà sviluppato come un ambiente in evoluzione, e questo implica un approccio modulare nella definizione delle parti della rete, principio che è in linea con le raccomandazioni della "multimedialità globale".

2.3.1 L'architettura della rete UMTS

La rete UMTS è formata da due strutture separate: una *Access Network* (AN) che è una generica rete radio di accesso, ed una *Core Network* (CN). L'Access Network, che sarà direttamente connessa all'utente ed al terminale con un collegamento senza fili, sarà responsabile dei meccanismi di indentificazione e di autenticazione e dovrà garantire l'efficacia dei meccanismi di sicurezza dal lato utente. La Core Network invece, collegherà diversi sistemi, UMTS e non. La separazione tra Access Network e Core Network offre la possibilità di rimpiazzare ogni CN con ogni altra rete. Ad esempio, si potrà sostituire una rete GSM con una B-ISDN). Le interazioni fra Access Network e Core Network saranno realizzate attraverso Interworking Units specifiche della CN, a cui l'AN si collega. Questa distinzione fra AN e CN permetterà anche la separazione fra Service Provider e Network Provider, fornitori rispettivamente di accesso alla rete ed dei canali di comunicazione.

L'utente finale disporrà, per l'accesso alla rete, di un terminale di accesso e di un USIM: UMTS Subscriber Identity Module, che sarà una smart card contenente le informazioni di identificazione ed autenticazione, insieme ai permessi di accesso ai servizi a cui si è precedentemente iscritto.

Questo porta all'introduzione di una nuova generazione di smart card aperte, che includono un generico sistema operativo che fornisce i servizi di base (come accesso ai file, operazioni di I/O, gestione della sicurezza, …), attraverso un'API standard. Siccome Java è un linguaggio particolarmente adatto per la comunicazione di rete, la sicurezza ed il downloading delle applicazioni, potrà essere utilizzato come collegamento fra la smart card e la struttura di rete. Inoltre la card offrirà la possibilità di scaricare ed eseguire le funzionalità del terminale, ad esempio le applicazioni di gestione del profilo-utente.

2.3.2 Il Virtual Home Environnment

Il concetto di Virtual Home Environment, sviluppato per l'UMTS ed in fase di standardizzazione, permetterà il supporto della mobilità del servizio offerto, mobilità indicata in termini di *service roaming* e di *service portability*. L'idea di fondo è che un utente mobile disporrà del proprio ambiente personalizzato anche quando utilizzerà terminali diversi su reti diverse. Il Virtual Home Environnment permetterà la mobilità del servizio all'utente, il che significa che l'utente disporrà durante i suoi spostamenti di tutti i servizi che ha sottoscritto e personalizzato. Da un lato questo richiede l'adattamento delle interfacce di gestione, dall'altro bisogna garantire che tutti i terminali disponibili su tutte le reti siano in grado di fornire i servizi richiesti. Durante le procedura di registrazione, il VHE permetterà alla rete visitata di accedere alle informazioni relative al service provider dell'utente, al profilo personalizzato dei servizi richiesti, ed alle prestazioni richieste per offrire servizi specifici di quel service provider.

Il Virtual Home Environnment può quindi essere considerato come un livello intermedio (*middleware*) che nasconde all'utente le diverse caratteristiche della rete, e alla rete le specificità dell'utente e del suo service provider.

2.3.3 Il Personal Communication Support

Il Personal Communication Support permetterà all'utente UMTS di personalizzare sia la gestione delle chiamate entranti, che la conversione delle chiamate a seconda delle prestazioni del terminale utilizzato.

Nei sistemi di telecomunicazioni utilizzati attualmente un utente può essere registrato in un solo terminale, e da questo unico terminale ha possibilità molto limitate di trasferire tutte le chiamate in arrivo verso un altro terminale, o una segreteria telefonica, o una mailbox per fax. Con la realizzazione del Personal Communication

Support l’utente disporrà di un meccanismo molto versatile per definire le politiche di gestione delle chiamate. In questo modo potrà definire diversi criteri di gestione delle chiamate entranti, criteri condizionati dall’orario, dal tipo di servizio richiesto e dall’utente chiamante.

Siccome l’UMTS supporterà la mobilità personale *(personal mobility*), l’utente potrà essere registrato presso vari terminali, con prestazioni diverse, di conseguenza sarà necessario che il Personal Communication Support sia in grado di adattare una chiamata entrante alle prestazioni del terminale presso cui si è registrato l’utente. Questo aumenterà notevolmente le possibilità del sistema di raggiungere l’utente finale, in quanto sarà l’utente stesso che stabilirà il modo in cui saranno gestite le chiamate che lo interessano.

Cap.3 Il sistema SOMA e il Mobile Computing

3.1 Il Paradigma ad Agenti Mobili nel Mobile Computing

3.1.1 Vantaggi

Ora che abbiamo descritto le caratteristiche del Paradigma ad Agenti Mobili e le problematiche poste dal Mobile Computing, vediamo quali caratteristiche del paradigma possono rivelarsi vantaggiose nell'ambito della mobilità.

Quando un computer portatile si collega alla rete con un collegamento wireless, si hanno vari problemi: il collegamento è tipicamente lento, costoso e inaffidabile. Per questo una prima esigenza è quella di ridurre i tempi di collegamento, ottimizzando l'utilizzo del canale, che si rivela la vera risorsa critica da gestire in maniera *efficiente*. L'approccio più naturale, di conseguenza, sembra essere quello di effettuare due collegamenti successivi per accedere ad un servizio in rete: in un primo collegamento si specifica qual è il tipo di servizio richiesto ed in un secondo si ricevono i risultati.

Volendo utilizzare, ad esempio, un paradigma client-server, bisogna prevedere la possibilità di interazioni successive e discontinue fra client e server. Per questo il server va adattato in modo da accettare richieste in una connessione iniziale, preparare i risultati e memorizzarli in attesa di una connessione futura. Alla successiva connessione, identificato ed autenticato il client, il server dovrebbe spedirgli i risultati, se questi sono pronti.

Proviamo a rispondere alla stessa esigenza con il paradigma ad agenti mobili.

1) Alla prima connessione del portatile viene spedito in rete un agente con il compito di effettuare un certo servizio.
2) L'agente effettua il suo compito in maniera autonoma, quindi si mette in attesa.
3) Alla successiva connessione del portatile, l'agente viene avvisato e riporta i risultati desiderati.

In questa sede non ci interessa tanto fare un confronto completo ed esaustivo fra i possibili paradigmi, quando delineare i possibili vantaggi derivanti dalla scelta del paradigma ad agenti mobili. Per organizzare il discorso riprendiamo in considerazione la terminologia introdotta nel paragrafo 1.4.1, in cui si erano individuati alcuni vantaggi ottenibili con l'utilizzo del paradigma: *flessibilità, autonomia, comportamento adattativo, scalabitiltà, apertura, efficienza.*

La prima esigenza da soddisfare è quella di *flessibilità* del servizio ottenuto: per ottenere servizi diversi è sufficiente avviare agenti diversi con obiettivi diversi, e non c'è la necessità di adattare il fornitore di servizio alle proprie esigenze. Non si sarà nemmeno costretti ad utilizzare i servizi forniti così come sono, in quanto questo comporterebbe la necessità di trasportare verso il portatile risultati parziali ancora da elaborare, il che riduce l'efficienza dell'utilizzo del canale se i risultati parziali sono voluminosi. In questo caso si può far leva sull'*autonomia* di un agente fornendogli il know how necessario per completare da solo ed in rete tutto il servizio richiesto. L'agente riporterà sul portatile i risultati solo a computazione terminata, ottimizzando così l'utilizzo del canale (*efficienza*).

Anche il *comportamento adattativo* può essere utile: basta immaginare il caso in cui si desideri che il livello di dettaglio dei risultati da restituire vari in base alla larghezza di banda del canale utilizzato. Se le prestazioni del canale saranno limitate, l'agente ci restituirà solo un "riassunto" delle informazioni richieste,

ottimizzando così l'utilizzo della risorsa critica. Per un esempio più dettagliato si veda la sezione 3.3.1.

La *scalabilità*, in questo caso, si traduce nella possibilità di spedire un unico agente attraverso il canale, con l'ordine di replicarsi una volta raggiunta la rete fissa ed utilizzare il parallelismo per aggredire un problema complesso che consenta questo approccio. Nel riportare il risultato si potrebbe prevedere la presenza di un agente che attenda risultati dagli altri, li riassuma e li trasporti verso il portatile.

L'*apertura* infine può essere vista dal lato del portatile, che presumibilmente non fornirà servizi, ma fruirà dei servizi forniti da altri. Ipotizzando l'esistenza di un'ampia piattaforma per supporto agli agenti mobili standard e aperta, il portatile potrebbe connettersi a diversi sistemi senza dover far riferimento sempre alla sua rete di origine. In questo modo, affrontando il problema della mobilità del portatile a livello di supporto, si potrebbe anche aggirare l'ostacolo incontrato dal protocollo Mobile IP e descritto nella sezione 2.2.5.

3.1.2 Problemi da risolvere a livello di supporto

Si presentano, a questo punto, delle difficoltà, che andranno risolte al livello di supporto per nascondere il più possibile la mobilità agli agenti.

- Ad ogni nuova connessione, il portatile, pur cambiando indirizzo, dovrà essere identificato ed autenticato dal sistema a cui si collega e che lo dovrà accogliere. Bisogna quindi fornire al sistema di accoglienza questa funzionalità e garantire i necessari requisiti di sicurezza.
- In seguito alla connessione del portatile al sistema, si devono avvertire gli agenti che abbiano l'esigenza di migrare sul portatile.
- Bisogna fornire agli agenti un servizio di informazione sullo stato di un portatile. In particolare bisogna in ogni momento

sapere se, ed eventualmente a con quale indirizzo, un portatile è connesso. In questo modo gli agenti potranno *adattare* i propri comportamenti allo stato della connessione ed alla posizione del portatile. Sarebbe anche utile poter discriminare fra diversi tipi di collegamento: ad esempio wireless, via modem o su LAN.

In risposta a queste esigenze proviamo a proporre un modello, di cui presenteremo un'implementazione nel capitolo successivo.

3.2 La Mobilità dei Place: un modello

3.2.1 Introduzione

Abbiamo detto che nel mondo degli agenti c'è già un concetto di spazio ben definito, ed in ogni caso ogni tipo di mobilità dovrà tenere conto di questa struttura spaziale. Si è anche detto, in maniera più o meno esplicita, che alla struttura spaziale virtuale corrisponderanno ben precisi significati: ad un entità "geografica" virtuale può corrispondere:

- un'entità geografica reale, ossia una rete di calcolatori situata in un certo posto,
- un'organizzazione, che sia un'università, un'azienda, o un'organizzazione amministrativa, …
- un dominio di sicurezza,
- altro…

O ancora tutte queste cose insieme. Così i concetti introdotti per fornire mobilità non dovranno alterare la struttura spaziale già costruita, che si fonda su basi concettuali ben precise, ed anzi dovranno armonizzarsi ad essa il più possibile.

Definiamo innanzitutto cosa si intende per mobilità nel Mondo degli Agenti. Se consideriamo, ad esempio la Terminal Mobility (vedi 2.1.2), e il caso particolare in cui un utente si sposti col suo computer portatile e si colleghi al sistema utilizzando collegamenti

di vario tipo: LAN, wireless o doppino telefonico. Se, in tutte le situazioni, sul portatile verrà attivato lo stesso place, ossia un place con lo stesso nome e nella stessa posizione nello spazio dei place, in realtà non si avrà nessun tipo di mobilità, in quanto gli agenti non percepiranno alcuna differenza fra questo caso ed il caso del terminale fisso. L'unica differenza percepibile sarà l'impossibilità di saltare dal place del portatile agli altri, e viceversa, in determinate situazioni. Questa impossibilità di movimento si verificherà, ovviamente, nei periodi in cui il portatile non è connesso alla rete.

Questo tipo di gestione della mobilità, completamente nascosta agli agenti e gestita ai livelli di astrazione inferiore, non mi sembra coerente con l'approccio finora descritto, in cui lo spazio degli agenti ha proprietà che rispecchiano quelle della realtà sottostante. Che significato avrebbe, infatti, collegarsi con un portatile ad una rete diversa dalla solita, magari in un altro paese ed in un contesto completamente diverso, ed apparire agli agenti come un place locale alla rete di partenza, o alla organizzazione di partenza? È quindi opportuno ideare un meccanismo diverso.

3.2.2 L'identità di un Place

L'idea proposta in questo lavoro è quella di associare ad ogni place un'*identità*, separata dalla sua locazione nello spazio dei place, e che permetta di ritrovare un place anche nel caso di un suo spostamento. L'introduzione di un identificatore unico per ogni place deriva da un'ambiguità che è presente nel caso di place non mobili, e che in quel contesto è fonte di semplificazione.

L'ambiguità è quella fra

- l'*identità*, che deve permettere di riconoscere un place anche a seguito di un suo spostamento, e
- la *località*, che indica la posizione occupata da un place nello spazio dei place e rappresenta la rete a cui è connesso.

Questo discorso è simile a quello già fatto per il protocollo Mobile IP (vedi 2.2): se un place non può cambiare la sua *località*, allora lo si può *identificare* proprio con questa, cosicché le operazioni di identificazione e ritrovamento saranno concomitanti. Questo ha anche un notevole vantaggio per quanto riguarda la distribuzione delle informazioni circa la locazione di un place: organizzando lo spazio in maniera gerarchica, come nel caso dei nomi su Internet o di quelli dei file in un comune file system, ogni livello gerarchico dovrà conservare le informazioni solo sui suoi sottolivelli: così un "continente" dovrà conoscere i propri "stati", che dovranno conoscere le proprie "regioni", e così via.

In caso di mobilità, però, questo meccanismo non è più sufficiente, perché facendo coincidere l'*identità* con la *località*, nello spostamento si può conservare solo una delle due, perdendo necessariamente l'altra. Quindi, o si perde la nozione di identità, in quando il place spostandosi acquisterà un altro nome, o si perde quella di località, conservando lo stesso nome negli spostamenti, ma abbiamo detto che è bene che gli agenti percepiscano lo spostamento dei place, perché la struttura del loro spazio deve avere proprietà simili a quelle del mondo sottostante.

3.2.2.1 Identificatori Semiparlanti

Visto in un'altra ottica, questo problema è simile a quello che si ha nel settore delle basi di dati con l'uso dei cosiddetti *identificatori semiparlanti.*

- Un *identificatore* può essere definito come un oggetto che permette di individuare un altro oggetto all'interno di un insieme, in maniera univoca. In altre parole deve realizzarsi una corrispondenza biunivoca fra gli elementi dell'insieme degli identificatori e gli elementi dell'insieme di oggetti identificati.
  Ad esempio, considerando i cittadini italiani, la coppia (Nome, Cognome) non è un identificatore, in quanto esistono casi di

omonimia, ossia di persone diverse con nomi uguali. Paradossalmente neanche l'aggiunta della data di nascita non consente di avere la certezza assoluta dell'identificazione. Per questo è stato costruito il Codice Fiscale, che identifica biunivocamente ogni cittadino italiano.

- Un *identificatore semiparlante* è un identificatore che ha al suo interno delle informazioni circa l'oggetto identificato. Un esempio è il codice fiscale stesso, che è costruito a partire dal Nome, Cognome, Data e Comune di Nascita e Sesso della persona. Nel caso in cui qualcuna di queste informazioni dovesse cambiare, ci sarebbe un'incongruenza, perché si dovrebbe o alterare l'identificatore, o eliminare la corrispondenza fra codice fiscale e informazioni relative alla persona. Così, se il soggetto deve cambiare identità, magari per ragioni di sicurezza, si è costretti a cambiare anche il suo codice fiscale se non si vuole lasciarvi traccia dell'alterazione. In questo modo il codice fiscale perde la proprietà di identificare biunivocamente una persona, in quanto una stessa persona, nel corso della vita, si trova ad avere due codici fiscali diversi.
  Brevemente, un altro esempio di identificatore semiparlante è il numero di matricola degli studenti dell'Università di Bologna, in cui le prime due cifre indicano la Facoltà, ed altre due il Corso di Laurea, per cui, cambiando semplicemente ordinamento io mi sono visto cambiare il numero di matricola.
  Per questo, nella fase di progettazione di un identificatore, è sempre buona norma evitare che contenga informazioni sull'oggetto identificato. In altre parole è bene evitare gli identificatori semiparlanti.

3.2.3 I Place Mobili: un nuovo concetto

Tornando agli agenti mobili, utilizzare la locazione di un place per identificarlo, in questo ordine di idee, è molto di più che

utilizzare un identificatore semiparlante, e se è possibile accettare che cambino due cifre in un numero di matricola, non è accettabile che cambi l'intero identificatore di un place ad ogni suo spostamento. Per questo è stato deciso di introdurre, contemporaneamente, il concetto di *Place Mobile* e quello di *Identificatore* di un Place.

Vediamo qual è il risultato di questa operazione. Esistono due categorie distinte di place:

- I *place fissi* nell'arco di tutta la loro vita mantengono sempre la stessa posizione.
- I *place mobili* si spostano da una regione all'altra nello spazio dei place, ma, ovviamente, mantengono sempre lo stesso identificatore unico.

È da notare che sia i place fissi che i place mobili sono mappati nello stesso spazio di identificazione, ossia che, dato un identificatore, questo può sempre individuare un place, che sia fisso oppure mobile, e dato un place qualsiasi, è sempre possibile determinare il suo identificatore.

D'altra parte, nel momento in cui un place mobile si sposta da una regione all'altra, entra a tutti gli effetti a fare parte della regione di destinazione, per cui, se si conosce la sua nuova posizione geografica, è comunque possibile raggiungerlo. Anzi, nella pratica, il sistema ad agenti dovrà semplicemente fornire un meccanismo per poter determinare in maniera trasparente, dato l'identificatore di un place, la sua posizione effettiva, in modo che possa essere raggiunto.

A questo punto, però, la nostra metafora sembra scricchiolare: come si può rappresentare il concetto di un luogo che si muove? Un luogo, per definizione è fisso! Chiaramente, quella dell'identificatore di un place deve essere vista come infrastruttura costruita sulla struttura dello spazio di base. Così, l'identificare posti diversi nel tempo, ma unici in un dato istante, con lo stesso luogo, è una evidente convenzione. Questo però accade anche nella

vita reale: pensiamo ad esempio a concetti come “casa mia”, o “la città in cui risiedo”, o “l’ufficio anagrafe del comune di Bologna”. In tutti e tre i casi si tratta di luoghi diciamo *virtuali*, che in un certo istante si trovano in una posizione ben precisa, ma che possono cambiare nel tempo. Così, ad esempio, si può decidere di andare a vivere in un’altra città, cambiando casa e città di residenza, ed il comune di Bologna può decidere di spostare l’ufficio anagrafe. In questo modo abbiamo dato un’interpretazione del concetto di identificatore di place, che può essere visto come un significato che si dà ad un certo luogo, e che ha una sua identità particolare, identità che non cambia quando si decide di cambiare luogo a cui si assegna tale significato.

3.2.3.1 La Raggiungibilità di un Place

Un place mobile ha anche un’altra proprietà in più rispetto ad un place fisso, oltre alla sua posizione attuale: la sua stessa *presenza* o *raggiungibilità*. C’è infatti un’altra notevole diversità in questo senso fra place fissi e mobili: nei place fissi l’irraggiungibilità di un place è da considerarsi un caso anomalo, un’eccezione, mentre i place mobili, nella fase di spostamento da una regione all’altra, sono sempre irraggiungibili, per un intervallo di tempo che può essere trascurabile nel caso di un passaggio “al volo”, ma che può essere anche relativamente lungo.

Ma scendiamo ancor di più nell’analisi: possiamo immaginare l’irraggiungibilità di un *place fisso* come un cambiamento delle proprietà dello spazio dei place: “prima era possibile raggiungere Bologna da Modena, Reggio Emilia e Forlì, ora Bologna è irraggiungibile, perché?”. Chiaramente nella nostra metafora le città sono place, e, se Bologna non è raggiungibile, è perché è cambiato qualcosa nel corrispettivo reale: ad esempio il place è stato disattivato, la macchina che lo supportava si è guastata o sono caduti i collegamenti di rete. In ogni caso si tratta di situazioni

anomale. D’altra parte, se Bologna in qualche modo riapparirà, immancabilmente si troverà “vicino” alle tre città menzionate, in quanto, essendo un place fisso, non potrà allontanarsi dall’Emilia Romagna. Magari in seguito non sarà più possibile raggiungere Bologna direttamente da Reggio Emilia, e si dovrà passare per Modena, ma Bologna dovrà comunque rispettare il sistema di leggi dell’Emilia Romagna, sarà gestita dalle stesse autorità e sarà nota alle città “vicine”.

Per quanto riguarda invece i *place mobili*, la mobilità è vista come la scomparsa da una posizione, seguita da un intervallo di “assenza”, seguita da una ricomparsa, possibilmente in un’altra posizione. In quest’ottica il non poter raggiungere un place ricercandolo nella sua posizione precedente non è un fatto straordinario o eccezionale, ma è da considerarsi un evento proprio della stessa natura di un place mobile, quindi va affrontato con mezzi diversi. Questo anche perché alla scomparsa si può accompagnare lo spostamento, per cui non ha più senso aspettarsi che un place mobile si riattivi nello stesso posto. La soluzione proposta è quella di associare ad un place mobile anche uno stato, che avrà due possibili valori: il place è *presente* nello spazio dei place, il place è *assente*. Nella nostra metafora, se io devo trascorrere un periodo in Inghilterra, dovrò trasferire il place mobile “Casa Mia” dalla mia abitazione in Italia al luogo in cui andrò a vivere in Inghilterra. Per questo, dal momento in cui lascio la mia abitazione in Italia, al momento in cui deciderò dove risiedere in Inghilterra, il place mobile “Casa Mia” sarà nello stato *non* p*resente*, in quanto sarebbe inutile spedirmi la posta o telefonarmi a casa in Italia, perché non riceverò mai i messaggi, ed è impossibile sapere quando e dove ricomparirà casa mia. Quindi è necessario gestire esplicitamente la presenza o assenza del Place Mobile. Notare che, “Casa Mia”, è un Place Mobile che ha due risorse che si muovono con lui: una buca per le lettere ed un numero telefonico.

3.2.4 Connessione e Disconnessione di un Place Mobile

Procedendo ulteriormente, rimane un ultimo elemento da aggiungere alla nostra costruzione: nella metafora, quando decido di trasferirmi in Inghilterra, ci sono due istanti di tempo che hanno una particolare importanza per me: il momento della partenza, e quello in cui decido di fissare la mia nuova dimora. Al momento della partenza, il place "Casa Mia" cessa di essere presente, in quanto, ad esempio non posso più ricevere lettere e telefonate dal telefono di casa. Al contrario, quando fisso la nuova dimora, "Casa Mia" torna ad essere presente, ed i miei amici possono chiamarmi a casa e spedirmi di nuovo lettere. Questi due momenti possono essere visti come due *Eventi*, ovvero cose che accadono ed hanno una importanza tale da modificare il comportamento di altri. Infatti, a seguito di un *Evento di Disconnessione* di "Casa Mia", vorrei che i miei amici, trovando libero il telefono di casa, non inferiscano che sono uscito o l'ho staccato, ma sappiano che sono nel corso di un trasferimento. Allo stesso modo, vorrei che, quando il place "Casa Mia" si *connette* di nuovo allo spazio dei place, chi ha esigenza si contattarmi sia subito avvertito e possa immediatamente raggiungermi telefonicamente, o spedirmi una lettera con la certezza che io avrò accesso alla mia buca delle lettere.

Gli *eventi*[Ten98], molto utilizzati nel campo dei sistemi distribuiti, sono entità di un determinato *tipo*, cui sono associate alcune *informazioni*. Questi sono generati da *produttori*, e utilizzati da *consumatori*. Sia consumatori che i produttori dovranno in qualche modo registrarsi, i primi per rendere noti i tipi di eventi che produrranno, i secondi per evidenziare i tipi di eventi cui sono interessati. Il grosso vantaggio di questo modello, è che può essere usato come meccanismo di comunicazione in cui chi produce l'evento non conosce l'identità di chi lo consumerà.

Questo è esattamente ciò di cui abbiamo bisogno nel nostro modello per la connessione e disconnessione dei place mobili. Nel nostro caso i produttori degli eventi di Connessione e

Disconnessione saranno sicuramente i place mobili stessi, mentre i consumatori saranno da un lato gli Agenti interessati, dall'altro il sistema sottostante, che dovrà reagire in qualche modo ad ogni evento.

3.2.5 Il Modello

Riassumendo quindi un Place Mobile è dato da:

1) Un *identificatore* unico ed immutabile.
2) Uno *stato* con due possibili valori: {*presente*, *assente*}.
3) Una *posizione* nello spazio dei place, definita quando lo stato è *presente*.
4) Eventi di *Connessione*, generati nella transizione da *assente* a *presente.*
5) Eventi di *Disconnessione*, generati nella transizione da *presente* ad *assente*.

Il comportamento è il seguente:

- *Sono a casa a Bologna.* Il Place Mobile "Casa di Livio Profiri" (*identificatore*) è *presente* a Bologna (*posizione*)
- *Parto per l'Inghilterra.* Generato un evento di *Disconnessione*, "Casa di Livio Profiri" è *assente*, la *posizione* è indefinita.
- *Mi stabilisco a Londra*. Generato un evento di *Connessione*, "Casa di Livio Profiri" è *presente*, la *posizione* è Londra.

3.3 Applicazione del Modello al Mobile Computing

Questo è il punto di congiunzione delle due grandi tematiche affrontate in questo lavoro: da un lato si è parlato, a vari livelli di astrazione, di un paradigma di progettazione, quello ad Agenti Mobili. Dall'altro si sono descritte le esigenze poste, nel mondo reale, dalla mobilità delle persone e terminali di calcolo. Si è anche costruito un modello per rappresentare la mobilità nel mondo degli Agenti Mobili. Qui vogliamo provare ad utilizzare il modello

proposto per rispondere alle esigenze manifestate, per verificare l'efficacia e l'efficienza dell'implementazione che ne consegue.

Consideriamo quindi i due tipi di mobilità visti nella sezione 2.1.1: la Terminal Mobility e la Personal Mobility

3.3.1 Terminal Mobility

Si è detto che lo spazio dei place ha proprietà simili alle entità reali rappresentate, e il Place Mobile rappresenta una sovrastruttura, qualcosa che è mobile, ma ha anche un'identità. Applichiamo tutto questo costruendo l'infrastruttura di supporto ad un place mobile su un computer portatile, o su un altro dispositivo portatile che sia in grado di sostenere il carico di un place mobile, ad esempio un telefono cellulare degli anni a venire. In questo modo nel Place Mobile saranno disponibili le Risorse presenti nel computer portatile.

A questo punto ci si interroga sull'opportunità di fornire *portabilità* o *mobilità*, nel senso di (vedi sezione 2.1.2). In altre parole ci si chiede se la mobilità debba essere affrontata a basso livello, nascondendola a livello più alto, o debba essere resa manifesta ad alto livello. Nel primo caso si ha il vantaggio che le applicazioni non si accorgono che qualcosa si muove sotto di loro, quindi non devono gestire il problema. Questo, se è un vantaggio, perché porta ad una semplificazione delle applicazioni, è uno svantaggio, perché impedisce alle applicazioni di adattare i loro comportamenti alla situazione. Nel secondo caso la situazione è rovesciata.

Immaginiamo di avere un browser web e di navigare su internet. Sarebbe auspicabile, quando siamo connessi attraverso un collegamento lento e costoso, ad esempio wireless, che il browser ci presentasse solo i contenuti testuali della pagine, evitando di perdere tempo nel dowloading di grossi file di immagine o di audio. Al contrario, collegandosi alla propria LAN, con un

collegamento veloce e “gratuito”, vorremmo goderci tutto ciò che internet è in grado di fornirci, compresi i filmati ed il sonoro. In altre parole pretendiamo un comportamento *adattativo* del browser (beneficio), che deve essere cosciente del tipo di collegamento sottostante e deve avere l’intelligenza di comportarsi in maniera opportuna nei vari casi (costo). Chiaramente non è detto che tutta la mobilità debba essere gestita a livello di applicazione: ad esempio, in un collegamento con telefono cellulare, non deve essere il browser a gestire il passaggio da una cellula all’altra, ma questo compito deve necessariamente essere delegato al sistema di gestione della comunicazione, ai fini di mantenere le auspicate distinzioni di livello di astrazione.

Immaginiamo un caso un po’ più complesso ed affrontiamolo con un approccio ad agenti. Un ingegnere deve ricercare, nel database del suo CAM aziendale (Computer Aided Manufacturing), tutti i progetti in cui si usa un sistema di iniezione di carburante che abbia determinate caratteristiche. Vuole anche poter lanciare la ricerca dal portatile ed ottenere i dati in risposta personalizzati in base al tipo di collegamento utilizzato in una seconda connessione. Si tratta di un’applicazione di Information Retrieval, in un contesto di Terminal Mobility.

Un agente verrà creato sul portatile con il compito di svolgere tutte le operazioni: si sposterà dal portatile alla rete nel corso di un breve collegamento wireless. Arrivato sulla rete, sarà in grado di agire autonomamente e potrà spostarsi sul place dove si trova l’interfaccia al DBMS del CAM. Qui verrà autorizzato all’accesso solo dopo essere stato identificato ed autenticato. Effettuata la ricerca, l’agente si sposterà in un place fidato che abbia la possibilità di ospitarlo fino a che il portatile non si collegherà e, appena ciò accadrà, l’agente tornerà di nuovo al mittente con i risultati desiderati.

Concentriamoci ora sugli aspetti relativi alla mobilità: vogliamo che, se la seconda connessione del portatile è fatta via telefono

cellulare, l’agente ritorni fornendo il numero di progetti trovati e qualche informazione su ognuno, per un totale di qualche decina di kbytes. Se invece il collegamento è via rete telefonica fissa, vogliamo anche una descrizione testuale di ogni progetto, e siamo disposti ad attendere il trasferimento anche di molte centinaia di kbytes. Quando il collegamento è diretto via LAN, poi, vogliamo visualizzare con un CAD i progetti e tutte le immagini in rendering. Come realizzare tutto questo? Viene spontaneo proporre di mappare il portatile in un place mobile, in modo che all’identità del portatile corrisponda quella del place mobile. A questo punto dobbiamo esprimere le differenti caratteristiche della connessione nello spazio dei place. Per esempio, a seconda del tipo dei connessione, il portatile si collocherà nello spazio dei place in una posizione diversa. Nel più semplice dei casi si farebbe corrispondere a una certa collocazione “geografica” il collegamento LAN, ad un’altra quello telefonico ed ad una terza quello cellulare. Con questa modellizzazione, dal punto di vista dell’agente il comportamento richiesto non sarebbe complesso:

1) Saltare dal place mobile ad un place fisso.
2) Recarsi nel place in cui si trova il DBMS.
3) Effettuare la ricerca.
4) Recarsi in un determinato place fisso e rimanere in attesa dell’evento *Connessione del Place Mobile.*
5) Determinare la posizione del place mobile, ossia il tipo di collegamento.
6) Selezionare le informazioni da trasportare in base al tipo di collegamento.
7) Saltare sul place mobile.
8) Presentare i risultati.

Si può osservare che tutte le operazioni legate alla rete (1, 2, 4, 5, 7) possono essere mappate in semplici primitive per gli agenti. Così, il passaggio dall'analisi all'implementazione dell'applicazione è veramente immediato, visto che tutta la programmazione di rete

si risolve nell'utilizzo di poche primitive, che fanno uso dei servizi messi a disposizione dal middleware di supporto al sistema ad agenti mobili. Questo permetterà di distogliere l'attenzione dagli aspetti dell'applicazione legati alla rete ed alla mobilità, per lavorare solo sulle parti realmente caratterizzanti l'applicazione:
3) Una complessa ricerca in una base di dati di componenti meccanici.
6) La selezione delle informazioni da presentare in base al tipo di collegamento.
8) La presentazione dei risultati nella maniera più efficace.

Con questo esempio abbiamo cercato di chiarire un concetto: il modello proposto per i place mobili, con l'identificazione di un place mobile con un terminale portatile, può fornire un valido strumento per aggredire problemi di Terminal Mobility in maniera semplice ed efficace, con l'accorgimento di mappare le caratteristiche che ci interessa rendere esplicite a livello di applicazione, nello spazio dei place. Aggiungiamo che, se il supporto è realizzato con accortezza, si possono anche raggiungere obiettivi di efficienza programmando gli agenti in modo da fare il miglior uso possibile delle risorse critiche.

Si noti che in questa sezione trascuriamo volutamente tutti i problemi che si dovranno affrontare a livello di supporto (protocolli, sicurezza, comunicazione, fault tolerance, standardizzazione…), perché ci interessa la prospettiva percepita dall'utente finale e dal progettista di applicazioni ad agenti mobili.

3.3.2 Personal Mobility

Se per la terminal mobility avevamo assegnato ad un terminale un place mobile, nella terminal mobility, in maniera altrettanto naturale, proveremo ad assegnare un *place mobile* ad un *singolo utente*. In tal caso l'identificatore unico del place mobile si farà corrispondere all'identità dell'utente. In questo modo la presenza

del place mobile nello spazio dei place avrà per gli agenti un significato molto preciso: coinciderà con la disponibilità della *risorsa* rappresentata dall'*utente* stesso.

Potremmo immaginarci una situazione molto simile a quella descritta nell'esempio precedente, con l'unica differenza che la ricerca sarebbe avviata da una macchina, e i risultati ricevuti su un'altra. Sarebbe un caso di personal mobility, visto che a spostarsi non sarebbe un terminale, ma un utente. Dal punto di vista degli agenti e del progettista di applicazioni ad agenti, il problema sarebbe analogo. Le diverse macchine sarebbero mappate come diverse posizioni nello spazio dei place, in cui il place mobile si verrebbe a trovare seguendo gli spostamenti dell'utente nello spazio reale. Chiaramente il punto 6 (sezione 3.3.1) dovrebbe essere riprogettato, nel caso in cui restasse il problema di adattare la quantità di informazione da trasportare in base alla posizione di destinazione.

Quando si parla di Personal Mobility, però, si possono perseguire obiettivi più ambiziosi. Abbiamo visto (sez.2.3) che, nel campo delle telecomunicazioni, in futuro, si prevede una riorganizzazione di tutti i sistemi, sia a livello tecnico, che a livello di rapporti economico-finanziari fra le entità in gioco. Tutto questo lavoro di riorganizzazione ruota attorno ad un unico centro di interesse: l'utente finale. In un'ottica più utilitaristica, si può affermare che il terreno su cui ci si farà concorrenza è quello del soddisfacimento dei bisogni del cliente finale.

In ogni caso, il concetto più interessante che è stato introdotto è quello di Virtual Home Environnment, che ha l'obiettivo di fornire sempre la stessa interfaccia, le stesse metafore e lo stesso tipo di servizi all'utente, indipendentemente dal terminale da lui utilizzato.

Proponiamo quindi il paradigma ad Agenti Mobili, e più precisamente il modello del Place Mobile, per fornire l'astrazione di Virtual Home Environnment.

Immaginiamo, ad esempio, un medico che lavora in un sistema ospedaliero. Come la maggior parte dei medici, sarà molto specializzato nel suo settore, ma non avrà molta confidenza con l'informatica. Ciò nondimeno, sarà circondato da un'infinità di apparecchiature elettroniche, indispensabili per lo svolgimento della sua professione. In questo contesto, costretto ad utilizzare macchinari diversi, in luoghi diversi, desidererebbe almeno che tutti seguissero una logica comune e avessero lo stesso tipo di interfaccia. Per questo, lo si potrebbe dotare di una smart card personale, che garantisca identità e sicurezza, smart card a cui si associ un place mobile, che si attivi in ogni macchina utilizzata. In questo modo agenti mobili potrebbero, seguendo il modello proposto, saltare sul place mobile ad ogni evento di connessione, ossia ogni volta che il medico utilizza un'apparecchiatura, per personalizzarne i comportamenti secondo le sue esigenze.

In questo modo sarebbe possibile fornire l'astrazione di Virtual Home Environnment in maniera molto immediata.

Di nuovo, si nota come tutte le difficoltà di compatibilità, standardizzazione, comunicazione e sicurezza sono scomparse a livello applicativo, ma riappariranno a livello di supporto.

3.4 Il sistema SOMA

SOMA, Secure and Open Mobile Agent, è un sistema ad Agenti Mobili realizzato al Dipartimento di Elettronica Informatica e Sistemistica dell'Università di Bologna. Il sistema può essere considerato un'istanza del paradigma presentato nelle sezioni precedenti. Lo descriveremo quindi come caso particolare all'interno del quadro generale delineato.

3.4.1 La Comunicazione

Nel progettare i meccanismi di comunicazione di SOMA è stato seguito un criterio molto rigido: i meccanismi di *interazione stretta*

(vedi sezione 1.3.4) sono ammessi solo fra agenti che si trovano nello stesso place. Questo criterio è fondato su uno dei principi base del paradigma ad agenti mobili, che prevede la non trasparenza della località delle risorse e degli agenti sulla rete. In quest'ottica non ha senso prevedere forte accoppiamento fra agenti che si trovano su place diversi, quindi i meccanismi di *interazione stretta* vanno riservati alle comunicazioni nell'ambito dello stesso place.

3.4.1.1 La Comunicazione fra Agenti situati su Place diversi.

SOMA fornisce un solo meccanismo di comunicazione fra agenti che si trovano in place diversi: lo scambio di *messaggi*. Altre forme di comunicazione a distanza, però, possono essere realizzate a livello di applicazione, utilizzando tutti gli strumenti messi a disposizione dal sistema operativo che ospita il place.

Un messaggio è un oggetto composto da 3 elementi:

1) L'identificatore dell'agente mittente.
2) L'identificazione dell'agente destinatario.
3) Il contenuto del messaggio, un oggetto di qualsiasi tipo, che possa essere trasferito da un place all'altro.

Un agente può spedire un messaggio ad un altro agente indipendentemente dal place in cui si trova il destinatario. Il mittente non è neanche tenuto a conoscere la posizione attuale dell'agente destinatario, che potrebbe anche essere in una fase di spostamento. L'unica informazione necessaria per la spedizione è l'identità del destinatario.

Spedito un messaggio, l'agente non riceve alcuna conferma di ricezione da parte del destinatario, ma il sistema garantisce il recapito del messaggio, se il destinatario si trova su un place raggiungibile.

Ogni agente dispone di una Mailbox, che conserva i messaggi ricevuti e non ancora letti. È possibile vedere in maniera non

bloccante se sono presenti messaggi nella mailbox, oppure attenderli in maniera bloccante.

Non tutti gli agenti dispongono del servizio di messaggeria. Infatti, per poter recapitare un messaggio a un agente, il sistema deve sempre tenere traccia dei suoi spostamenti, e questo comporta un certo overhead, soprattutto in termini di utilizzo della rete. Si è quindi deciso di poter creare agenti *not traceable*, non rintracciabili, e quindi privi di Mailbox, per migliorare l'efficienza di applicazioni che non necessitano di questo servizio. In ogni caso, a default ogni agente è *traceable*.

3.4.1.2 La Comunicazione nell'ambito di un Place

Per quanto riguarda invece la comunicazione fra agenti che si trovano nello stesso place, i meccanismi forniti sono due:

- La *condivisione di oggetti*: oggetti di qualsiasi tipo possono essere depositati in uno speciale contenitore, ricevendo in cambio un identificatore con il quale sarà possibile accedere di nuovo all'oggetto da parte dello stesso o di un altro agente.
- La *blackboard*, una lavagna su cui lasciare messaggi del tipo *(chiave, contenuto)*, dove *chiave* è una stringa di descrizione e *messaggio* un oggetto qualsiasi.

La differenza è che la *chiave* della *blackboard* è di tipo stringa, e fornisce una descrizione "in chiaro" del contenuto, a differenza dell'identificatore del caso precedente che, invece, viene fornito dal sistema. Così, affinché un altro agente possa aver accesso ad un oggetto condiviso, deve essergli fornito l'identificatore da colui che lo ha depositato, mentre nel caso della blackboard è sufficiente accordarsi sul "nome" (*chiave*) da dare ad un certo tipo di informazione.

In sostanza la blackboard è l'unico dei tre meccanismi visti che permette di comunicare con agenti di cui non si conosca già l'identificatore (*interazione anonima*), e può essere utilizzato

proprio per mettere in contatto agenti che altrimenti non si potrebbero conoscere.

3.4.2 L'Astrazione di Località e le proprietà dello Spazio dei Place

Lo spazio dei place di SOMA ha una struttura gerarchica, limitata, per semplicità, a due livelli. I *Place*, infatti, sono semplicemente raggruppati in aree geografiche dette *Domini*, che sono considerati entità separate e non ulteriormente aggregabili (Figura 3.1). Ogni place ha un nome unico all'interno del suo dominio, e i domini hanno nomi a loro volta unici, di conseguenza, per individuare la posizione di un place, è necessario e sufficiente specificare la coppia (dominio, place).

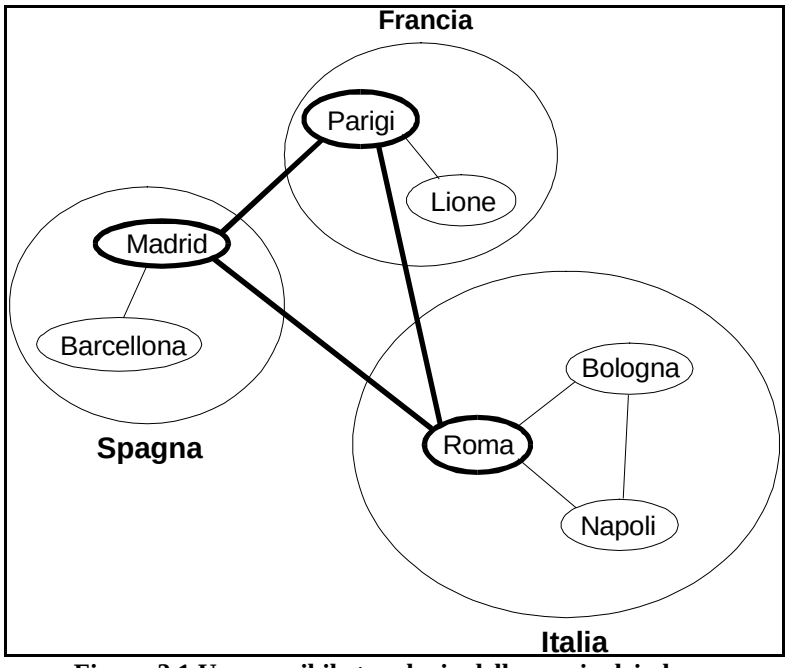

Figura 3.1 Una possibile topologia dello spazio dei place. Italia, Francia e Spagna sono Domini, le città sono Place.

3.4.2.1 Spostamento Intra-Dominio

Come si vede, l'insieme dei place di un dominio è *totalmente connesso*, nel senso che per ogni coppia di place esiste un arco che li congiunge. Con questa struttura gli agenti si possono spostare da un place ad un altro qualsiasi dello stesso dominio, utilizzando la primitiva g*o(Place)* fornita dal supporto.

Esempio:

Spostamento da Bologna a Napoli: *go(Napoli).*

È da osservare che non sempre il salto fra due place si può effettuare, non però per i vincoli imposti dalla topologia, ma per una circostanza di tipo eccezionale. Per esempio è possibile che il place di destinazione sia stato disattivato, che ci siano problemi nelle connessioni di rete, o che ci sia un malfunzionamento del sottosistema software di gestione della rete. In questo senso le proprietà dello spazio dei place rispecchiano quelle della realtà sottostante, a tempo di esecuzione.

Se l'agente non riesce a saltare, rimane nel place di origine e gli viene subito notificata la circostanza eccezionale, che il progettista di applicazioni deve gestire esplicitamente.

3.4.2.2 Spostamento Inter-Dominio

Ogni dominio ha un place particolare, il *Default Place* che ha il ruolo di collegamento con gli altri domini. L'insieme dei Default Place è totalmente connesso, ed in questo modo i Domini sono tutti collegati fra loro. Questo rende il grafo formato da tutti i place di tutti i domini, non totalmente, ma *semplicemente connesso,* nel senso che per ogni coppia di place è sempre possibile trovare un cammino che li congiunga. Se definiamo la distanza fra due place come il numero minimo di archi necessari a formare un cammino che li unisca, allora due place avranno, al più, distanza pari a 3, nel caso appartengano a domini diversi e nessuno dei due sia il default place del suo dominio.

Esempi:

Da Napoli a Barcellona: (Napoli,Roma), (Roma,Madrid), (Madrid,Barcelona), distanza=3.

Da Parigi a Barcellona: (Parigi,Madrid), (Madrid,Barcellona), distanza=2.

Le primitive di spostamento Inter-Dominio sono due: *go(Dominio)* e *go(IdentificatoreDelPlace).* La prima era già presente nelle precedenti versioni del sistema, mentre la seconda è stata introdotta nell'ambito di questo lavoro, per consentire la migrazione di un agente verso un Place Mobile, indipendentemente dal suo stato e posizione attuale.

- La primitiva *go(Dominio),* trasferisce l'agente nel default place del Dominio di destinazione.

  Esempi:

  Da Lione: *go(Italia)* Cammino (Lione,Parigi), (Parigi,Roma).

  Da Parigi: *go(Italia)* Cammino (Parigi,Roma).

Notare che l'agente, nel primo caso, in *condizioni normali*, non percepisce il passaggio per Parigi, ma crede di saltare direttamente da Lione a Roma.

- La primitiva *go(IdentificatoreDelPlace)* trasferisce l'agente al place di destinazione identificato da *IdentificatoreDelPlace*

  Esempio:

  Da Napoli: go(Identificatore di [Spagna, Barcellona]),

  Cammino (Napoli, Roma),(Roma, Madrid),(Madrid, Barcellona)

Anche in questo caso l'agente, in *condizioni normali*, non percepisce il passaggio per Roma e Madrid, ma salta direttamente da Napoli a Barcellona, nel senso che viene riattivato solo a destinazione raggiunta.

Nel caso di un cammino di lunghezza unitaria, si è detto che o l'agente raggiunge la destinazione prefissata, o viene subito segnalata l'anomalia e l'agente rimane al punto di partenza. Nel caso di cammini, diciamo, multipli, ossia composti da più salti, non

è possibile conoscere l'esito dei salti futuri, quindi occorre fornire un comportamento diverso da quello del salto singolo. SOMA prevede che, in caso di fallimento del primo salto, l'anomalia viene segnalata, altrimenti non viene segnalato nessun errore, e l'agente viene riattivato nel place intermedio in cui si è verificata l'anomalia. Questo comportamento è strettamente legato a caratteristiche dell'implementazione in Java (vedi sezione 4.2.3). È l'agente che deve verificare se il place in cui viene riattivato è quello di destinazione o è un place intermedio, e agire di conseguenza.

Vediamo che cosa succede se il place di destinazione è un Place Mobile. Vi sono vari casi da considerare. Innanzitutto bisogna chiarire che le primitive *go(Place)* e *go(Dominio)* non fanno alcun riferimento all'*identità* del place di destinazione, quindi il comportamento sarà lo stesso a tutti gli effetti. In particolare, se il place mobile sarà *assente*, non sarà possibile raggiungerlo, quindi verrà segnalata l'anomalia o si interromperà la migrazione al passo precedente.

La primitiva *go(IdentificatoreDelPlace)*, invece, è stata progettata con l'obiettivo di gestire la mobilità del place e renderla il più possibile trasparente agli agenti, quindi, nel caso in cui il place di destinazione sia mobile:

- Se al momento della chiamata il Place Mobile è *presente*, il comportamento è lo stesso che per un place fisso.
- Se al momento della chiamata il Place Mobile è *assente*, l'agente rimane inattivo finché il place non torna ad essere *presente*. In seguito a questo evento, l'agente è trasportato sul place mobile, dove vene riattivato. In questo modo l'agente non percepisce in nessun modo la differenza fra place mobili e fissi, in quanto durante l'attesa rimane disattivo, come durante ogni altro trasferimento. In sostanza, dal punto di vista dell'agente, un salto ad un place mobile, con l'attesa della connessione, è solo un trasferimento che richiede molto tempo.

Questo approccio nel trattamento dei salti verso place mobili, se decisamente pratico, può in alcuni casi essere fortemente limitante, in quanto non è detto che l'agente voglia rimanere ad aspettare per tempi molto lunghi l'attivazione di un place mobile. Per questo sono stati previsti meccanismi di risveglio dall'attesa, ed in questo caso l'agente viene riattivato in un place ben determinato: la Home Base del Place Mobile. Affronteremo l'argomento nella sezione 3.4.3.

È anche possibile gestire lo spostamento verso place mobili direttamente a livello di applicazione, in quanto sono disponibili primitive per:

- Determinare lo stato (*presente* o *assente*) di un place mobile, conoscendo il suo identificatore,
- Determinare la posizione di un place mobile, dato il suo identificatore,
- Registrarsi in attesa di eventi di connessione.

Grazie a queste, un agente può disporre della posizione di un place mobile pochi istanti dopo la sua connessione e recarsi verso il place fisso utilizzando *go(Dominio)* e *go(Place)*, che sono orientate alla posizione e non all'identificatore, e con le quali non si corre il rischio di rimanere bloccati in attesa.

3.4.3 Il Place Mobile

I Place Mobili in SOMA sono stati progettati seguendo il modello presentato nella sezione 3.2.5.

Il modello presentato non fa alcun riferimento al modo in cui il sistema gestisce le informazioni sullo *stato* la *posizione* dei Place Mobili, in quanto si tratta di scelte architetturali. In SOMA si è stabilito di associare ad ogni Place Mobile un place fisso, che gestisce le informazioni relative alla mobilità e gli agenti che desiderano migrare verso un place mobile. Questo place è denominato *Home Base* del Place Mobile. Nelle sezioni successive

faremo spesso riferimento alla Home Base, nella descrizione dei rapporti fra Place Mobili e sistema.

Un'altra precisazione va fatta a proposito degli eventi di *Connessione* e *Disconnessione* di un Place Mobile (vedi sezione 3.2.4). In realtà in SOMA non è previsto un sistema dei gestione degli eventi per gli agenti. Questo, infatti, sarebbe un meccanismo di interazione stretta da fornire indipendentemente dalla posizione dell'agente e, adottandolo, si andrebbe contro la logica del sistema che prevede solo meccanismi di interazione lasca per le comunicazioni in remoto (vedi sezione 3.4.1). Di conseguenza gli eventi di *Connessione* e *Disconnessione* sono notificati agli agenti sotto forma di semplici messaggi spediti dalla Home Base. L'oggetto del messaggio contiene tutte le informazioni sullo stato del Place Mobile. Si è deciso di riservare al sistema un sottoinsieme dei valori possibili per l'identificatore di agente: uno di questi valori verrà utilizzato come identificatore del mittente per il messaggio con cui si comunica l'evento, questo per non moltiplicare il numero di entità del sistema. In seguito sarà possibile adottare approcci differenti.

3.4.4 Il Ciclo di Vita degli Agenti

Il ciclo di vita di un agente si compone di 5 fasi (Figura 3.2):

- ***Nascita***: l'agente viene creato, tutte le sue strutture dati sono inizializzate, quindi viene messo in esecuzione. La creazione di un agente può essere opera o dell'utilizzatore o di un altro agente.
- ***Esecuzione***: è l'unico stato in cui l'agente è attivo. È l'agente che decide quando interrompere l'esecuzione e portarsi in uno dei tre possibili stati: Idle, Spostamento o Morte.
- ***Idle***: l'agente è inattivo, congelato in uno stato consistente. È gestito dal sistema, che riattiva l'agente o quando questo riceve

un messaggio o, se è il caso, alla connessione di un place mobile.

- ***Spostamento***: l'agente è inattivo, congelato in uno stato consistente, nel corso di uno spostamento. Uscirà da questo stato al termine dello spostamento, andando in esecuzione o in Idle.
- ***Morte***: l'agente ha portato a termine la sua missione. Tutte le risorse occupate vengono liberate.

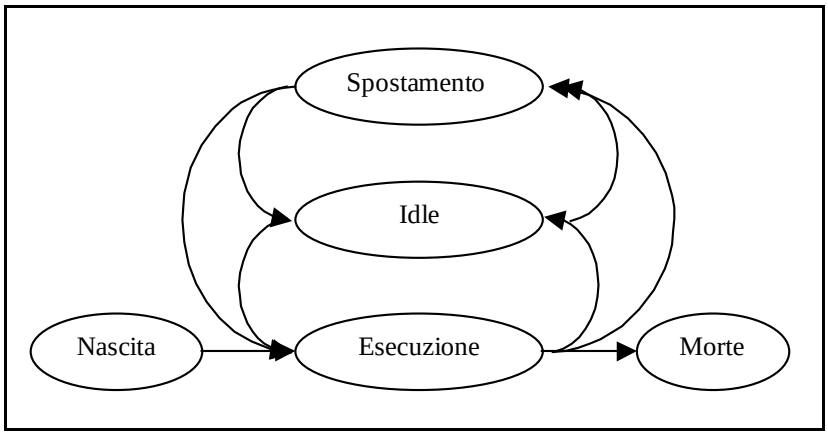

Figura 3.2 Ciclo di vita di un Agente.

3.4.4.1 Lo stato Idle

Lo stato *Idle,* introdotto nell'ambito di questo lavoro, pur essendo uno stato di attesa di messaggi, differisce molto dallo stato in cui si trova un agente quando si mette in attesa di messaggi nella sua Mailbox, con la relativa primitiva. Infatti, nel secondo caso l'agente è attivo ma bloccato, mentre, quando è Idle, è disattivo. A riprova di questo, nel secondo caso è possibile che l'agente si sblocchi autonomamente, ad esempio dopo un *timeout* fissato, pur non avendo ricevuto messaggi. È invece concettualmente impossibile, per un agente, uscire dallo stato Idle in maniera autonoma, in quanto appunto è congelato ed affidato al sistema,

che si incarica di risvegliarlo al momento opportuno. La situazione si capovolge, invece, se vista dagli occhi del sistema: finché un agente è attivo, il sistema non può disporne in nessun modo: non può, ad esempio spedirlo a sua insaputa verso un altro place, o salvarne lo stato su disco in maniera consistente. È solo possibile attendere che l'agente, autonomamente, si porti in uno dei tre stati: Idle, Spostamento o, al limite Morte.

È chiaro che esistono meccanismi di basso livello per terminare forzosamente l'esecuzione di un agente e liberare tutte le risorse da lui occupate, ma si tratta di operazioni distruttive che non forniscono alcuna garanzia di conservazione di stati consistenti per tutti gli agenti o risorse con cui l'agente terminato stava interagendo.

Concettualmente, quando un agente è in Idle, il suo consumo di risorse è ridotto rispetto a quando è in Esecuzione, sia perché non gli è associato alcun *Componente computazionale*, (vedi sezione 1.2.1), sia perché il sistema può disporre dello stato dell'agente a piacimento, potendo decidere ad esempio se conservarlo in memoria o salvarlo temporaneamente su hard disk.

Ancora, nel caso in cui un guasto o una qualche circostanza eccezionale dovesse imporre la terminazione del sistema di supporto ad un place, il sistema potrebbe salvare lo stato degli agenti in Idle per recuperarlo in una successiva attivazione, mentre non potrebbe in alcun modo garantire continuità agli agenti rimasti in Esecuzione, che verrebbero persi nella disattivazione del place.

Questo è quanto accade tipicamente nei computer portatili alimentati a batterie, in cui quando la batteria si sta per esaurire, si susseguono ripetuti avvertimenti e consigli di salvataggio ed uscita da tutte le applicazioni attive. Il supporto al place mobile realizzato, prevede il salvataggio di tutti gli agenti in Idle alla disattivazione di un place, ed il loro recupero alla successiva attivazione. Questo avviene in maniera completamente trasparente. È anche previsto un meccanismo di *fault tolerance*, che garantisce

il recupero degli agenti in Idle anche in caso di disattivazione improvvisa del place. Si tratta però di un meccanismo piuttosto rudimentale che viene tenuto, di default, disattivo, per ragioni di efficienza.

Nella versione attuale del sistema, sono due i casi in cui l'agente esce dallo stato Idle, ma in seguito potrebbero esserne previsti altri. I due casi sono:

1) L'agente *riceve un messaggio* e passa allo stato di *Esecuzione*. Questo è un comodo meccanismo di coordinamento fra agenti: se un agente deve aspettare un messaggio, invece di rimanere in Esecuzione bloccato, ha la possibilità di andare in Idle, con tutti i vantaggi citati, per essere riattivato solo all'arrivo del messaggio. Chiaramente un'operazione di passaggio da uno stato all'altro dell'agente è molto più pesante rispetto all'operazione di bloccaggio e sbloccaggio di un thread o di un processo, in quanto le prime sono gestite dal supporto agli agenti, le seconde a livello più basso: di sistema o di linguaggio. Quindi, in un puro calcolo di efficienza, si tratta di vedere se i costi di attivazione e disattivazione di un agente ripagano i benefici derivanti dalla liberazione di risorse caratteristica dello stato Idle. In definitiva, se si conta di ricevere spesso messaggi, conviene rimanere in Esecuzione, bloccati; altrimenti è preferibile andare in Idle

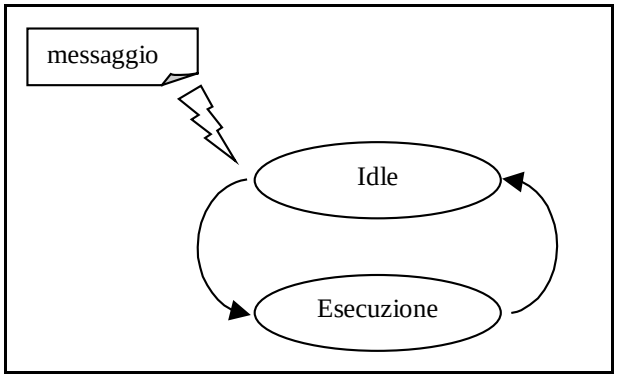

Figura 3.3 Il ciclo Esecuzione-Idle

2) Alla *connessione di un Place Mobile*, se l'agente era in attesa dell'evento, passa allo stato *Spostamento***.** (Vedi 3.4.4.2)

3.4.4.1.1 Agenti che gestiscono altri agenti

Lo stato *Idle* potrebbe essere utilizzato in futuro per fornire agli agenti la possibilità di gestire altri agenti in Idle. Per esempio si potrebbero fornire, ad agenti autorizzati, primitive di inserimento, estrazione ed eliminazione di agenti in Idle, dall'insieme degli agenti in Idle di un place, e magari primitive per mandare in Esecuzione o spostare agenti in Idle da un place all altro. In questo modo sarebbe possibile fornire nuovi servizi direttamente a livello di applicazione, senza dover modificare il supporto. Inoltre questi agenti fornitori di servizio potrebbero, alla disattivazione del place su cui risiedono, andare in Idle e venire essi stessi salvati, per essere recuperati nello stesso stato alla successiva attivazione del place. In questo modo si realizzerebbe un ulteriore strato di supporto, completamente realizzato ad agenti, e programmabile a livello di applicazione. Chiaramente si tratta di meccanismi da progettare con cura, per ragioni di sicurezza, per garantire che la manipolazione di agenti in Idle sia possibile solo ad agenti espressamente autorizzati a farlo. I meccanismi per ottenere un tale grado di sicurezza, in ogni caso, sono già disponibili in SOMA.

3.4.4.2 La migrazione verso un Place Mobile: i passaggi di stato

Esaminiamo più in dettaglio i passaggi di stato che hanno luogo quando un agente invoca la primitiva *go(IdentificatoreDelPlace)*, e l'identificatore si riferisce ad un Place Mobile. Innanzitutto, il sistema deve determinare in che stato si trova il Place Mobile, e, nel caso in cui sia *presente*, qual è la sua posizione effettiva. Descriveremo questo meccanismo nella sezione 4.5.7. A questo punto vi sono due possibilità:

1) Il Place Mobile è *presente*, ne conosciamo quindi la posizione. L'agente passa dallo stato di Esecuzione allo stato di Spostamento, per essere rimesso in Esecuzione a trasferimento ultimato, o fallito, rispettivamente nel place di destinazione, oppure in quello di origine o in uno intermedio. In questo caso il comportamento è quello illustrato in Figura 3.4. ed è analogo a quello di uno spostamento verso un Place Fisso.

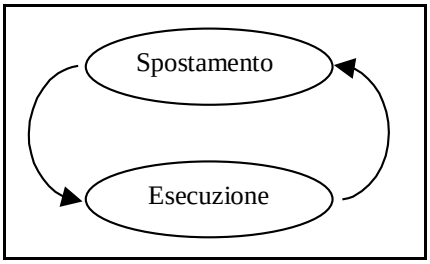

Figura 3.4 Passaggi di stato: migrazione di un agente.

2) Il Place Mobile è *assente*, in questo caso l'agente deve rimanere in attesa della sua connessione, quindi viene messo in Idle, non però nel place in cui si trovava quando ha invocato la primitiva *go(IdentificatoreDelPlace)*, ma in un determinato place fisso: la Home Base del Place Mobile (vedi sezione 3.4.3), che si incarica di gestire gli agenti in attesa. Ci sono quindi due ulteriori possibilità:
   a) L'agente si trova già nella Home Base, quindi viene subito messo in Idle.

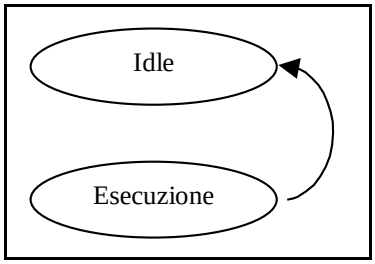

Figura 3.5 Passaggi di stato: agente messo in Idle.

b) L’agente deve recarsi nella HomeBase, quindi passa allo stato Spostamento. In caso di trasferimento fallito, viene rimesso in Esecuzione con la solita semantica, in caso di trasferimento portato a termine viene messo in Idle in attesa.

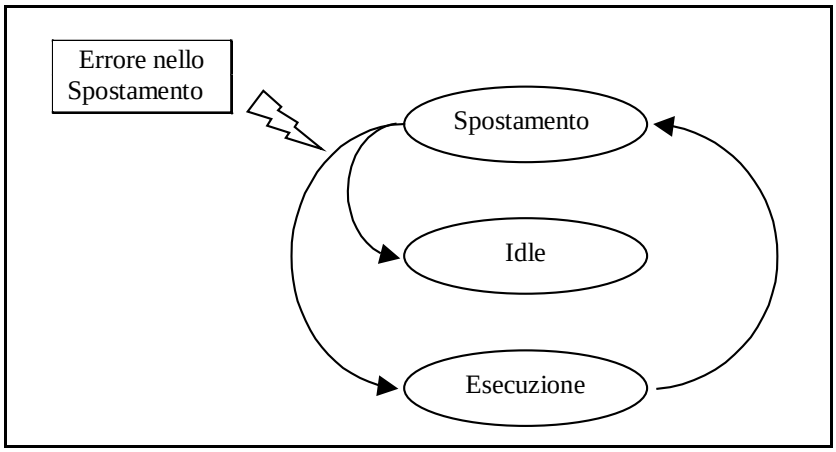

Figura 3.6 Passaggi di stato: trasferimento in attesa della *Connessione* di un Place Mobile.

A questo punto, se tutto si è svolto normalmente, l’agente si trova nella HomeBase, nello stato Idle. Anche in questo caso possono avvenire due cose:

3) L’agente riceve un messaggio, quindi viene subito rimesso in Esecuzione ed eliminato dall’elenco degli agenti in attesa della connessione. Per rimettersi in attesa, dovrà utilizzare di nuovo la primitiva *go(IdentificatoreDelPlace)*, saltando al caso 1) o 2a). Con questo meccanismo si possono evitare attese troppo lunghe o indesiderate, semplicemente incaricando altri agenti di spedire opportuni messaggi.

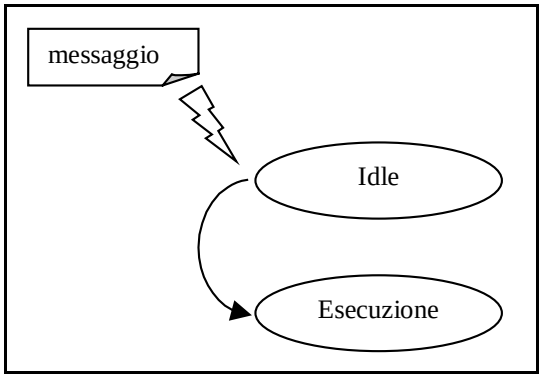

Figura 3.7 Passaggi di stato: risveglio di un agente in Idle.

4) Si verifica un evento di Connessione del Place Mobile verso cui l'agente desiderava migrare. In questo caso l'agente passa dallo stato Idle allo stato di Spostamento, infine allo stato di Esecuzione, a trasferimento effettuato o fallito, secondo la solita semantica.

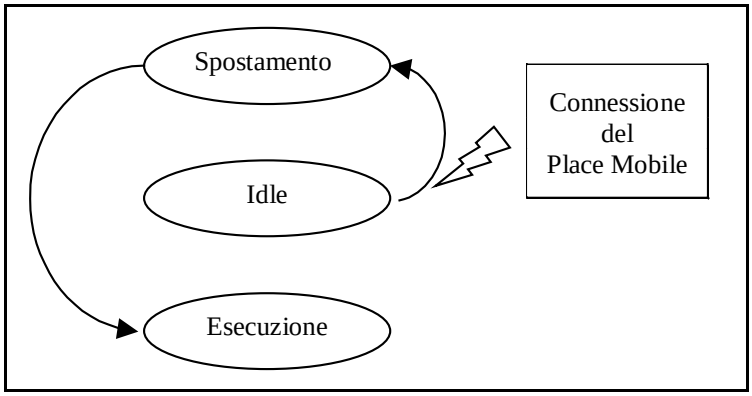

Figura 3.8 Passaggi di stato: *Connessione* di un Place Mobile.

3.4.5 La Sicurezza

La versione attuale di SOMA ha sistemi di sicurezza orientati in due direzioni: la protezione del sistema, delle risorse e degli agenti da possibili attacchi di agenti ostili, e la protezione dei canali di

comunicazione, da tutti i possibili attacchi che possono venire portati da sistemi ostili. In questa sezione siamo più interessati al primo meccanismo di protezione, che riguarda direttamente gli agenti e le applicazioni ad agenti, mentre, relativamente al secondo, accenniamo semplicemente che il supporto utilizza un sistema di cifratura a chiave pubblica-chiave privata per garantire la sicurezza nelle comunicazioni fra place.

Per quanto riguarda la protezione delle entità del sistema dagli attacchi di agenti ostili, o, al limite, mal progettati, SOMA prevede un meccanismo per stabilire, azione per azione, quali agenti hanno il permesso di compierla e quali no. Si tratta di un meccanismo assolutamente necessario in un sistema ad agenti aperto, che, in sua assenza, sarebbe un po' come una piattaforma di accoglienza per virus mobili capaci di arrecare danno a piacimento a qualsiasi risorsa disponibile.

Il modello utilizzato è quello delle politiche di sicurezza, che stabiliscono, in maniera flessibile e modificabile dinamicamente, per ogni coppia (agente, operazione), se l'agente può compiere l'operazione.

L'operazione può essere molto generica, ad esempio "Accesso al disco rigido", o molto specifica: "Accesso al file xyz.doc in scrittura". Sono anche prese in considerazione operazioni specifiche dell'ambiente ad agente, come l'accesso ad un place o l'accesso agli oggetti depositati in un place.

Per quanto riguarda gli agenti, questi acquisiscono diritti in tre modi diversi:

- Ogni agente agisce per conto di un *principal*, che è l'utente che lo ha cerato. All'ingresso dell'agente in ogni place, viene determinata l'identità del suo principal ed, in base a questa, ed eventualmente alla classe dell'agente, gli viene associato l'insieme di operazioni consentite, o più precisamente, gli viene associato un *dominio di protezione*.

- Il principal può anche stabilire un insieme di *preferenze*, che sono ulteriori limitazioni alle operazioni permesse ad un singolo agente. Questo meccanismo permette di stabilire regole che variano addirittura da agente ad agente, anche nell'ambito dello stesso principal e classe di agenti.
- Infine, un agente, se autorizzato, può chiedere ad un place una *credenziale*, che è una prova che l'agente ha avuto accesso a quel determinato place. In questo modo l'agente "spera" che i place in cui migrerà successivamente lo considerino più "credibile", accordandogli permessi più ampi.

Per una descrizione più accurata del modello si sicurezza di SOMA si veda [Ten98].

Anche le politiche di sicurezza possono essere viste come proprietà dello spazio dei place, ed in effetti, anche nello stabilire una politica, si rispetta la gerarchia organizzativa di questo spazio. Infatti, per ogni dominio, viene stabilita una *politica di dominio*, che viene distribuita a tutti i suoi place. Ogni place, dal canto suo, può stabilire le proprie regole, alterando la politica di dominio con la propria *politica di place*. Gli approcci possibili sono diversi: o una severa politica di dominio, con i place che stabiliscono regole più permissive; o al contrario regole piuttosto permissive a livello di dominio, e più severe a livello di place; o ancora politiche ibride.

Ancora una volta, se le politiche di sicurezza sono proprietà dello spazio dei place, queste rispecchiano fedelmente proprietà del mondo reale. Infatti, immaginiamo che un dominio rappresenti un'organizzazione, la politica di dominio, di conseguenza, rappresenterà il meccanismo per difendere gli interessi dell'organizzazione stessa dai possibili attacchi di agenti che operano per conto di altri soggetti. La politica di place, allo stesso modo, difenderà gli interessi dell'entità rappresentata dal place, nei confronti delle altre entità, interne o esterne all'organizzazione.

Ora ribaltiamo il discorso: come garantire gli interessi del princpal di un agente, garantendo l'incolumità dell'agente stesso?

Prima di provare a rispondere, facciamo un esempio che chiarisca il problema: se un agente che si occupa di commercio elettronico trasporta, in forma più o meno protetta, il numero della carta di credito della persona che lo ha creato, questa avrà un forte interesse alla segretezza dell'informazione. In che modo può garantirsi affinché non venga violata la privacy dell'agente? Chiaramente esistono meccanismi nel supporto che possono garantire la sicurezza degli agenti, ma il supporto è fornito da altri, che hanno appunto interessi diversi, quindi non è possibile farvi affidamento. Si tratta quindi, da un lato di evitare place e domini non fidati, dall'altro di gestire esplicitamente la sicurezza a livello di applicazione, basandosi magari sui place che si ha la certezza siano fidati.

Per esempio, un agente non dovrebbe mai trasportare una chiave crittografica con sé, perché questa potrebbe essere facilmente scoperta ed utilizzata per violare i sistemi che avrebbe dovuto proteggere. Così, dovendo effettuare una computazione su place non fidati, l'agente potrebbe, dopo ogni place visitato, recarsi in un place fidato e farsi crittografare i risultati parziali, in modo che questi non vengano alterati da altri. Questo è un esempio di gestione esplicita della sicurezza a livello di applicazione.

3.4.5.1 La Sicurezza di un Place Mobile

In questo lavoro non è stato affrontato direttamente il problema della sicurezza. L'unica considerazione fatta è che, per un Place Mobile, ha poco senso parlare di *politica di dominio*, in quanto nei suoi spostamenti, un Place Mobile potrà cambiare frequentemente dominio di appartenenza. Per questo, il Place Mobile non ha una politica di dominio da modificare, ma deve impostarsi autonomamente la propria. Questo rispecchia il principio che la politica deve essere ritagliata in base alle entità con cui il Place

Mobile dovrà interagire effettivamente, intesi come domini di accoglienza ed agenti che desidereranno accedere al place stesso.

3.5 Altri Sistemi ad Agenti Mobili per il Mobile Computing

3.5.1 Supporto alla Mobilità del Terminale nei Sistemi ad Agenti

Alcuni dei sistemi ad agenti mobili realizzati sono stati adattati allo sviluppo di applicazioni per il mobile computing. In particolare, nella maggior parte dei casi, l'obiettivo è stato quello di realizzare servizi di *terminal mobility*, che consentano il lancio di agenti da computer portatili in collegamenti via cavo o wireless ed il successivo recupero degli stessi agenti con i risultati desiderati. A tal proposito citiamo brevemente alcuni esempi:

- In [KovRR97] si descrive il sistema MASE, in cui è previsto che un nodo della rete fissa, il *Mobility Gateway*, funga da interfaccia fra rete fissa e rete wirelless e fornisca tutti i servizi necessari alla mobilità. Le comunicazioni sono basate sullo standard UMTS (vedi sezione 2.3.1). Il sistema, per permettere agli agenti di effettuare il ritorno sul nodo mobile fornisce una lista, aggiornata dinamicamente, contenente i place connessi al sistema.
- In [LazS98] si descrive il sistema Discovery, in cui il nodo mobile, ad ogni connessione alla rete e ad ogni disconnessione, spedisce a tutti gli agenti che aveva lanciato sulla rete fissa un messaggio di notifica dell'evento. Gli agenti interessati, quindi, riceveranno messaggi di tipo CONNECT e DISCONNECT, grazie ai quali potranno coordinarsi con il nodo mobile.
- In [KotGN97]e [GraKN97] si descrive come il sistema Agent TCL fornisce mobilità. Ad ogni nodo mobile si associa un nodo

fisso che funge da *Docking Station* per il nodo mobile. Il sistema di nomi utilizzato prevede, indicando con *D* il nome di un nodo mobile, che la sua Docking Station sia chiamata *D_dock*. La migrazione di un agente da e verso un nodo mobile avviene in maniera *trasparente*. È il sistema, infatti, che organizza, sia su *D* che su D_*dock*, una coda di agenti che desiderano migrare all'altro capo del canale. Gli agenti sono tenuti inattivi e la coda è memorizzata su disco. Ad ogni connessione, tutti gli agenti che desiderano migrare verso la rete vengono spediti a *D_dock*, tutti quelli che vogliono tornare sul portatile vengono spediti da *D_dock* a *D*. Gli agenti vengono riattivati solo a destinazione raggiunta.

- In [SahM97] e [SahMB97] si descrive il sistema MAGENTA, che stabilisce nomi unici per i place, che sono chiamati *lieu*[8]. I place hanno nomi unici, e, se sono mobili, possono comparire e scomparire, in maniera prevista o imprevista. Il sistema ha il compito di gestire questi eventi. Agli agenti si fornisce la possibilità di rimanere in un place qualsiasi, in attesa della "comparsa" di un place "scomparso" su cui si desidera migrare.

Si può osservare come in tutti i casi si affrontano le stesse problematiche, per le quali si propongono soluzioni diverse. In particolare alcuni dei problemi sono:

- Fornire al nodo mobile uno o più nodi fissi di riferimento.
- Gestire le connessioni del nodo mobile alla rete e le sue disconnessioni, desiderate o meno.
- Stabilire meccanismi per permettere il ritorno di un agente su un nodo mobile.
- Permettere la persistenza di agenti mobili in ambienti di esecuzione che possono essere disattivati e riattivati.

[8] *Lieu* - in Francese, luogo.

3.5.2 Supporto alla Mobilità Personale su Internet

Il supporto alla mobilità personale, già ampiamente affermato nelle telecomunicazioni, è un concetto ancora abbastanza nuovo per Internet. Infatti il routing dei pacchetti, basato sul protocollo IP, è orientato agli host, e non agli utenti. Così, se è possibile individuare univocamente una macchina fornendo il suo indirizzo IP, non esiste alcun tipo di identificatore unico per gli utenti che permetta di indirviduarne la posizione. Un indirizzo di posta elettronica, ad esempio, consente di spedire un messaggio ad un utente e non ad una macchina, ma non permette di sapere dove l'utente si trova o quando si collega, cosicché non può essere utilizzato per fornire servizi più avanzati.

In [Jung98], si descrive un progetto sviluppato all'Università di Seoul, in cui, con un sistema ad agenti mobili, si forniscono servizi di mobilità personale agli unteti di Internet. Questo è interessante, perché quello della mobilità personale è un settore abbastanza ignorato da parte dei progettisti di sistemi ad agenti mobili. La proposta è quella di realizzare una rete di agenti, che costituisca un livello intermedio fra il livello di Internet e quello delle comunicazioni interpersonali. Si ha così una rappresentazione a tre livelli di astrazione:

- Rete interpersonale - schematizza le comunicazioni fra le persone.
- Rete di agenti - comunicazione fra agenti mobili che rappresentano le persone.
- Rete fisica - comunicazione fra i computer su cui eseguono gli agenti.

Il concetto più importante di questa rappresentazione è quello che la mobilità delle persone può essere mappata efficacemente nella mobilità degli agenti, col risultato di fornire un modello semplice, espressivo ed efficace della mobilità personale.

Il progetto prevede che ogni utente della rete interpersonale possieda un agente, detto Agente Personale, nella rete di agenti.

Chiunque voglia comunicare con un altro utente fa una richiesta al proprio Agente Personale, che ricerca l'Agente Personale dell'altro utente e prova a contattarlo. Questo vuol dire che le comunicazioni interpersonali fra gli utenti sono realizzate effettivamente dagli Agenti Personali in maniera indipendente dalla posizione fisica degli utenti stessi. Quindi l'Agente Personale, per permettere la comunicazione fra utenti, deve avere la possibilità di determinare la posizione del proprio utente e di poter migrare in quella posizione.

Un place capace di accogliere agenti può essere costruito su vari tipi di piattaforme e reti, se le Unità di Esecuzione rispettano gli stessi protocolli. Di conseguenza le differenze fra reti e sistemi diversi possono essere mascherate dallo strato intermedio che fornisce il supporto di esecuzione agli agenti. La rete ad agenti può essere costruita su ogni tipo di rete, cosicché sia le reti di telecomunicazioni che la rete Internet possono fornire la struttura di base senza conflitti.

A livello di implementazione, in realtà, ogni utente ha non uno, ma due agenti personali: uno si trova nell'home site dell'utente, mentre l'altro segue l'utente nei suoi spostamenti e fornisce i servizi da lui richiesti. Al momento della connessione l'utente inserisce username e password, vengono creati i due agenti personali, e sono quindi accessibili i servizi forniti dal sistema: scambio di messaggi, E-mail e servizio di talk fra utenti.

È da notare che il modello proposto, con due agenti, uno fisso e l'altro mobile, assomiglia per funzionalità al modello del Mobile IP (vedi sezione 2.2), con la differenza che il Mobile IP permette la terminal mobility e non la personal mobility. Riguardo poi al termine *agente*, nei due ambiti è usato in accezioni completamente diverse: nel Mobile IP vedremo che l'Home Agent ed il Foreign Agent sono due router, mentre nel sistema appena descritto si tratta di agenti mobili.

Cap.4 Architettura di SOMA e Dettagli di Implementazione

4.1 L'Evoluzione di SOMA

SOMA è stato sviluppato materialmente da studenti della Facolta di Ingegneria dell'Università di Bologna nell'ambito delle loro Tesi di Laurea. Al sistema è anche dedicato un sito web [SOMA], costantemente aggiornato da parte dei responsabili del progetto, in cui è possibile reperire tutte le informazioni più recenti, insieme al codice, che è distribuito liberamente.

Le fonti di informazione più dettagliate, quindi, sono proprio le Tesi di Laurea, che descrivono accuratamente l'evoluzione del progetto nelle sue fasi successive, che elenchiamo bervemente.

1) L'Ing. Fabio Tarantino ha sviluppato la prima versione del sistema, in cui erano presenti tutte le funzionalità di base, escluse quelle relative alla sicurezza [Tar98]. Il sistema, d'altra parte, era realizzato sul JDK 1.1 [Jav], che non fornisce una gestione della sicurezza abbastanza flessibile per essere facilmente utilizzata in un sistema ad agenti mobili.
2) L'Ing. Luigi Tenti ha sviluppato la seconda versione, basata sul JDK1.2 beta 2, curando principalmente la sicurezza del sistema contro gli attacchi di agenti non fidati, e del trasferimento di agenti e comandi fra place, prevedendo un sistema di cifratura, con un meccanismo a chiave pubblica-chiave privata [Ten98].
3) L'Ing. Claudia Chiusoli e l'Ing. Fabrizio Coccia hanno lavorato insieme alla terza versione del sistema, nella quale è sono stati realizzati i meccanismi dinamici di gestione del sistema [Chi98][Coc98]. In particolare sono stati aggiunti tool visuali per la gestione di Place ed Utenti e per il lancio di Agenti, ed è stato realizzato il meccanismo di distribuzione dinamica ai

place, delle informazioni di configurazione, e di tutte le chiavi crittografiche necessarie ai sistemi di sicurezza. È stata inoltre fornita un'interfaccia grafica per il Place e si è sostituito il protocollo TCP con un protocollo basato su pacchetti UDP, nella comunicazione fra le entità del sistema.

4) Contemporaneamente alla presente tesi, Cristiano Cavallari ha realizzato una versione di SOMA compatibile con lo standard MASIF, proposto dalla OMG [OMG], e basato sullo standard CORBA. La sua versione è stata realizzata a partire dalla [2].
5) Infine in questo lavoro si adatta SOMA al Mobile Computing, a partire dalla versione [3].
6) È previsto il porting del sistema alla versione definitiva del JDK 1.2 [Jav], uscita in dicembre 1998, e la convergenza delle versioni [4] e [5], sviluppate separatamente. Allo stato attuale, tutte le versioni si basano sul JDK1.2 beta 2.

In tutte le versioni sono presenti applicazioni ad agenti di esempio per verificare le funzionalità fornite.

4.2 Descrizione del Sistema

Descriveremo l'architettura del sistema nella maniera più sintetica possibile, fornendo dettagli di implementazione solo quando li riterremo necessari alla comprensione dei meccanismi generali, o quando potranno essere di aiuto a chi volesse intraprendere uno studio dettagliato del codice sorgente.

Il sistema è realizzato completamente in Java [Jav], quindi, riferendoci alla classificazione data nella sezione 1.2.1:

- I *componenti codice* sono classi Java.
- I *componenti risorsa*, anche quando rappresentano dispositivi, sono oggetti Java.
- I *componenti computazionali* sono thread Java, che condividono un comune spazio di indirizzamento.

- Le *interazioni* sono chiamate di metodi o alterazioni dirette di attributi di oggetti, nell'ambito della stessa Java Virtual Machine. Per realizzare interazioni fra componenti ospitati su macchine virtuali diverse, si utilizza lo scambio di comandi, tecnica descritta nella sezione 4.2.1.
- I *siti*, chiamati *place* in SOMA, corrispondono a macchine virtuali Java.

4.2.1 Le Interazioni tramite Comandi

In Java, macchine virtuali diverse hanno spazi di indirizzamento diversi, mentre tutti i thread avviati sulla stessa macchina virtuale condividono lo stesso spazio di indirizzamento. Di conseguenza tutte le interazioni che avvengono nell'ambito della stessa macchina virtuale, ossia dello stesso place, anche fra thread diversi, possono far uso di riferimenti al comune spazio di indirizzamento, quindi sono molto più efficienti delle interazioni che avvengono fra place diversi.

Sia interazioni fra place diversi, che quelle fra applicazioni di supporto (vedi sezione 4.2.6) e place, avvengono tutte con lo stesso meccanismo: lo scambio di *comandi*.

Un comando è un oggetto derivato dalla classe astratta Command:

```
…
public abstract class Command implements Serializable
{
  …
  public abstract void exe();
  …
}
```

Quando il comando viene ricevuto dal place "server", viene eseguito il suo metodo exe(). Esistono anche comandi di risposta, ma non ci interessa scendere nel dettaglio. Per un loro utilizzo si veda la classe:

`AgentSystem.MobileHomenameRequestCommand.`

In effetti si utilizza un meccanismo di Weak Mobility di tipo *ship* di un *frammento di codice*, *asincrono*, con esecuzione *immediata* (vedi sezione 1.1.4). Non sempre è richiesta risposta da parte del destinatario del comando.

Per fare un esempio, l'Agent Launcher, applicazione di supporto, per mettere in esecuzione un agente in un place, spedisce un comando di classe "Ccreate" ad un demone del place stesso. Il comando Ccreate, che viene subito eseguito, chiama semplicemente un metodo del modulo di gestione degli agenti di quel place. Il modulo è visibile al comando in quanto, appartenendo allo stesso place, si trova nello stesso spazio di indirizzamento. Quindi un'interazione remota si risolve sempre nella spedizione di un comando, che effettua la stessa interazione in locale. In questo caso, l'Agent Laucher non prevede risposta dal place.

Cerchiamo di capire che paradigma di progettazione viene utilizzato. Se il comando spedito ha il know-how per effettuare l'operazione può essere visto come una Remote Evaluation, altrimenti, concettualmente, anche se si ha trasferimento di codice, è piuttosto un Client-Server, in cui il meccanismo per scegliere l'operazione da effettuare è rappresentato dalla chiamata di un metodo di un oggetto già presente nel server. In ogni caso non sempre è prevista risposta ad un comando, e, dove è prevista, si risolve nella spedizione di un comando di risposta.

4.2.2 I Collegamenti fra i Place e fra i Domini

Il place sono collegati fra loro utilizzando [Chi98] un protocollo basato su socket UDP [Jav97]. I comandi vengono prima serializzati su stream di bytes, quindi manipolati e spediti. Quando i pacchetti UDP sono ricevuti, viene ricostruito lo stream di bytes, quindi l'oggetto è deserializzato. Per questo, tutti gli oggetti che si

intende spedire da un place all'altro devono implementare l'interfaccia `Serializable`, e rispettare le regole della serializzazione degli oggetti di Java.

La topologia dei collegamenti, già descritta nella sezione 3.4.2, presenta due tipi di collegamenti: quelli fra domini e quelli fra place.

- I collegamenti fra domini avvengono tramite il ***Gateway***, un modulo di comunicazione installato sul *Default Place* di ogni dominio. Ogni Gateway spedisce comandi agli altri Gateway, e a tutti i Place del suo Dominio.
- Ogni place può spedire comandi agli altri place e al Gateway del suo dominio.

Per informazioni più dettagliate, vedere [Tar98] e [Chi98], in cui si descrivono accuratamente sia il protocollo di comunicazione che i demoni che ne sono responsabili.

4.2.3 Gli Agenti Mobili

Un Agente Mobile in SOMA è una classe derivata dalla classe astratta di base Agent, di cui presentiamo gli attributi e i metodi pubblici:

```
package AgentSystem;
public abstract class Agent implements Serializable
{
  private AgentID My_ID = null;
  public String Start;
  public boolean Traceable;
  public Mailbox Mail;
  public ArrayList preferenze;
  public ArrayList credenziali;

  public Agent()
  {…}

  public AgentID getID()
  {…}

  public abstract void putArg(Object obj);
  public abstract void run();
```

```
  public final void go(PlaceName p,String metodo)
    throws CantGoException
  {…}

  public final void go(DomainName d,String metodo)
    throws CantGoException
  {…}

  public final void go(HomeName h,String metodo)
    throws CantGoException
  {…}

  public final void addPref(GrantEntry ge)
  {…}

  public final void giveMeCredential()
  {…}

  …
}
```

Osserviamo subito che un agente ha, come ci aspettavamo, un identificatore, una mailbox, un set di preferenze ed un set di credenziali, insieme al campo traceable che specifica se l'agente è rintracciabile o meno, ossia se è possibile spedirgli dei messaggi. Ha anche metodi per ottenere l'identificatore, per aggiungere preferenze e per richiedere credenziali ad un place.

Per fare un esempio, con un grande sforzo di fantasia, presentiamo l'agente HelloWorld:

```
public class HelloWorld extends AgentSystem.Agent
{
  public void putArg(Object o){}

  public void run()
  {
    System.out.println( "Hello world!" );
  }
}
```

La classe HelloWorld estende Agent, che si trova nel package AgnetSystem. Non c'è costruttore, ed, in ogni caso, l'unico

costruttore ammissibile sarebbe quello senza parametri. I due metodi da ridefinire obbligatoriamente sono run e putArg.

- Il metodo `run`, che viene sempre chiamato per avviare l'agente, in questo caso manda in uscita la famosa frase.
- Il metodo `putArg`, invece, viene chiamato ancora prima di `run` e servirebbe, nelle intenzioni iniziali del progetto, per fissare lo stato iniziale dell'agente in base al parametro passato in argomento. Nella versione attuale, in ogni caso, gli agenti vengono avviati con l'Agent Launcher, che non permette di passare nessun argomento, quindi putArg non è utilizzabile.

Rimangono da descrivere i tre metodi `go` ed il significato del campo `start`. I metodi `go` consentono all'agente di spostarsi in un altro place, come è stato descritto nella sezione 3.4.2:

- `go(PlaceName p,String metodo)` specifica il place di destinazione.
- `go(DomainName d,String metodo)` specifica il dominio di destinazione.
- `go(HomeName h,String metodo)` specifica l'identificatore del place di destinazione e corrisponde alla primitiva *go(IdentificatoreDelPlace)* introdotta per i place mobili. Evidentemente la classe `HomeName` rappresenta l'identificatore del place.

Osserviamo che, insieme alla destinazione del salto, si specifica un *metodo* sotto forma di stringa. Si tratta del *metodo che verrà attivato all'arrivo* a destinazione, o in una tappa intermedia di uno spostamento multiplo in cui non si riesca a raggiungere la destinazione finale (vedi sezione 3.4.2.2).

Come si era visto nella sezione 3.4.2.2, se invece lo spostamento è singolo, la condizione anomala viene notificata subito: vediamo qui che il meccanismo utilizzato è il lancio dell'eccezione `CantGoException`. Questo perché, se il primo

salto non riesce, il *Network Manager* è subito in grado di notificarlo.

In definitiva la semantica della `go` è la seguente: si effettua il tentativo di trasferimento:

- se il trasferimento riesce l'istruzione `go` ritorna senza errori,
- se non riesce, viene lanciata una `CantGoException`, con un messaggio di descrizione dell'errore.

A questo punto è anche possibile capire la funzione del campo `start`: a seguito di una `go`, il campo `start` conterrà il nuovo metodo di partenza. Quando l'agente verrà riattivato nel place di destinazione, il sistema leggerà il contenuto di `start` e lancerà il metodo corrispondente.

Proviamo a chiarire il tutto con un esempio. Programmiamo un agente per recarsi in un place di cui conosciamo il nome:

```
public class UnSalto extends AgentSystem.Agent
{
  public void putArg(Object o){}
  public void run()
  {
    System.out.println("UnSalto: sono nato");
    try
    {
      go( new AgentSystem.PlaceName( "PlaceVicino"), "run2" );
    }
    catch(Exception e)
    {
     System.out.println(
       "UnSalto: impossibile raggiungere PlaceVicino");
    }

    System.out.println("UnSalto: ultima istruzione");
  }

  public void run2()
  {
    System.out.println(
      "UnSalto: sono arrivato nel PlaceVicino!");
  }
}
```

Osserviamo che:

- se il trasferimento fallisce, viene eseguito il corpo della catch (),
- altrimenti, all'arrivo dell'agente a destinazione, viene avviato run2().
- In ogni caso, viene eseguita l'istruzione che stampa la stringa "UnSalto: ultima istruzione". Bisogna osservare infatti che, nel caso in cui la migrazione ha successo, il flusso di esecuzione non viene interrotto, ma continua con l'istruzione successiva alla go, sta quindi al programmatore fare in modo l'agente termini il più presto possibile la sua esecuzione. In ogni caso, le strutture dati dell'agente, a seguito di una go con esito positivo, vengono cancellate dal place di origine, quindi ulteriori operazioni effettuate da parte dell'agente (come ulteriori tentativi di migrazione), porterebbero a situazioni di errore.

Chiariamo infine che il trasferimento di un agente avviene grazie ad un comando: il `TransCommand`, che si occupa di trasferire agenti o comandi, da un place di un dominio, ad un qualsiasi altro place, di un qualsiasi dominio. In pratica l'agente o il comando viene trasportato come array di bytes, che, se è settatta una determinata opzione, viene anche cifrato prima della spedizione e decifrato una volta a destinazione. In questo modo si cerca di garantire la sicurezza del canale. In ogni caso per ottenere l'array di bytes da trasportare si utilizza il meccanismo di serializzazione fornito in Java.

4.2.4 La Sicurezza

I sistemi di sicurezza di SOMA sono basati sul modello di sicurezza di Java1.2beta 2, ed è in fase di sviluppo una versione basata sul modello della versione definitiva 1.2 di Java. Attualmente sono disponibili sistemi di protezione del sistema contro attacchi di agenti ostili, e di protezione dei canali di comunicazione.

Per informazioni più dettagliate sull'implementazione si vedano [Ten98] e [Coc98].

4.2.5 La Gestione Dinamica

SOMA permette la gestione dinamica di place ed utenti. Sul default place di un particolare dominio viene avviato, oltre al Gateway, il ***Supervisor***, un modulo che ha il compito di distribuire le informazioni di configurazione e di sicurezza di tutto il sistema ai Gateway degli altri domini, che, a loro volta, le distribuiranno ai place.

Per ulteriori informazioni vedere [Chi98][Coc98]. In particolare in [Chi98], si descrive il *meccanismo di avviamento* del place che ospita il *Supervisor*, di un default place, che ospita un *gateway* e di un *place* normale.

4.2.6 Le Applicazioni di Supporto

Sono attualmente disponibili diverse applicazioni di supporto:

1) Il *Setup Tool* [Chi98][Coc98], che permette di creare il file contenente la configurazione iniziale di tutto il sistema, prima dell'avvio del Supervisor.
2) Il *System Manager* [Chi98][Coc98], che, interagendo col Supervisor, permette di gestire utenti e place. In particolare è possibile, da un qualsiasi nodo della rete ed a sistema già avviato:
   a) creare e cancellare utenti,
   b) creare nuovi domini,
   c) creare e cancellare place di un dominio.
   d) avviare nuovi place, anche da remoto, se nella stessa macchina è già presente un place attivo.
3) L'*Agent Launcher* [Chi98] [Coc98], che permette di lanciare, da un qualsiasi nodo della rete, un agente in un place, a nome di un determinato utente, che viene prima identificato ed

autenticato via username e password. Le informazioni sugli utenti ed sulla topologia del sistema vengono richieste, con un collegamento iniziale, al *Supervisor*. A partire dalla presente versione l'Agent Launcher può anche richiedere le informazioni di cui necessita ad un Place Mobile (vedi sezione 4.5.1)

4) L'*Agent Policy Tool* [Ten98], che permette di impostare le politiche di sicurezza.

4.3 La Modularità

Il sistema SOMA è organizzato in diversi moduli, ognuno responsabile di alcune attività:

Nome Modulo	Oggetto o Classe	Funzione Svolta
Main	`Main`	Avvio del sistema (è una classe statica)
Agent Manager	`Main.agentManager`	Gestione agenti
Network Manager	`Main.netManager`	Gestione connessioni di rete
Security Manager	`Main.securityMan`	Gestione sicurezza dei canali di comunicazione
Agent Policy Manager	`Main.agentPolicy`	Gestione politiche di sicurezza
Gateway Manager	`GateMain.gateManager`	Gestione Gateway
Supervisor Manager	`SupervisorManager`	Gestione del Supervisor (è una classe statica)

Per una descrizione più dettagliata dei moduli è possibile far riferimento a [Tar98], ed a [Ten98] per i moduli realtivi alla sicurezza.

Con l'implementazione del Place Mobile si è presentata un'esigenza nuova nell'ambito dell'evoluzione di SOMA: quella di adattare il tipo di supporto installato in base alle proprie esigenze. Così ora si desidera avere dei place fissi e dei place mobili, dei gateway "normali" e dei gateway che supportano la mobilità dei place. Si è anche deciso di rendere opzionale il supporto agli agenti nello stato Idle.

Chiaramente, per fornire i nuovi servizi, si sono dovute modificare alcune funzionalità già offerte dal sistema, e se ne sono dovute aggiungere delle nuove. Per questo si è deciso di utilizzare il più possibile l'ereditarietà del codice, creando nuove classi che ridefiniscono i metodi da sostituire e ne aggiungono nuovi, se necessario.

Nella struttura precedente, però, tutti i gestori del sistema (Agent Manager, Network Manager, …) erano realizzati tramite classi statiche, ossia classi i cui attributi e metodi sono tutti statici, mentre non ci sono attributi e metodi di istanza. Questo rispecchiava fedelmente l'idea che i gestori fossero unici, ossia che non ci fossero, ad esempio, più sistemi di gestione degli agenti nell'ambito della stessa macchina virtuale, ossia dello stesso place.

Questa struttura rende difficoltosa la sostituzione dei metodi vecchi con i nuovi, in quanto, per chiamare un metodo di una classe statica si fa esplicitamente riferimento al nome della classe, cosicché per invocare un metodo della classe derivata, si dovranno sostituire tutte le chiamate di tipo `ClasseMadre.Metodo();` con chiamate di tipo `ClasseFiglia.Metodo();`. Per questo si è deciso di trasformare tutti i metodi e gli attributi dei gestori da statici a dinamici, in modo da non dover cambiare le chiamate ai metodi, ma solo il *tipo* di gestore istanziato, ossia il gestore "base", o un gestore "derivato". Le istanze dei gestori vengono create tutte nel modulo Main, in cui, in base ai parametri di avvio, si decide che tipo di place verrà lanciato.

Facciamo un esempio:
L'oggetto gestore degli agenti è:

```
Main.agentManager
```

Nel modulo Main si crea:

- un Agent Manager di base con l'istruzione:
  `agentManager = new AgentManager();`

- un Agent Manager con supporto agli agenti in Idle con l'istruzione:
  `agentManager = new IdleAgentManager();`

Ora, indipendentemente dal tipo di gestore installato, questo verrà inizializzato con la stessa chiamata:

```
Main.agentManager.init( … );
```

I moduli da istanziare per avviare un place mobile hanno nomi ottenuti preponendo il prefisso "Mobile", al nome originario del gestore: `MobileNetManager`, `MobileSecurityMan`, `MobileAgentPolicy` e `MobileIdleAgentManager`, che deriva dall'`IdleAgentManager`, derivato a sua volta dall'`AgentManager`.

Un place mobile, come ogni altro place, per potersi connettere al sistema deve far riferimento ad un gateway, ed in particolare al gateway del dominio a cui vuole appartenere temporaneamente. Non tutti i gateway offrono, però, questa possibilità, ma solo quelli il cui gestore è un'istanza di `MobileGateManager`.

Per quanto riguarda il Supervisor, si noti che `SupervisorManager` è rimasta una classe statica, in quanto, nell'implementazione attuale, è presente un solo Supervisor, quindi ha scarso senso costruirne delle sottoclassi.

4.4 Lo stato Idle

Come abbiamo visto nella sezione 1.1.4, Java non fornisce Strong Mobility, in quanto il meccanismo di serializzazione non è in grado di salvare gli stati interni della Java Virtual Machine. In particolare, con la serializzazione viene perso lo stack delle chiamate ai metodi, con tutte le variabili locali. Per questo, nel simulare Stong Mobility col meccanismo descritto nella sezione 4.2.3, si fa implicitamente l'assunzione che, nel passaggio dallo stato di *Esecuzione* a quello di *Spostamento*, si perdono tutte quelle

componenti dello stato di *Esecuzione* dell'agente che non possono essere serializzzate, in particolare tutte le *variabili locali* di tutti i metodi chiamati e non ancora usciti. Quindi, nel progettare applicazioni ad agenti, è importante tenere a mente che le informazioni che si desidera conservare nello spostamento devono essere memorizzate nelle variabili di istanza dell'agente, ossia nei suoi attributi. Parallelamente, bisogna eliminate dagli attributi tutti i riferimenti ad oggetti non serializzabili.

Nel passaggio dallo stato *Esecuzione* allo stato *Idle* si fa un'assunzione analoga, visto che quando un agente è in *Idle* deve poter essere serializzato e salvato su disco, oppure trasportato.

Non tutti gli agenti possiedono questo stato, ma solo quelli derivati dalla classe `IdleAgent`, che è si presenta così:

```
package AgentSystem;
…
public abstract class IdleAgent extends Agent
{
  public IdleAgent()
  {
    super();
  }

  // Manda l'agente in Idle.
  // Verrà riattivato col metodo Metodo.
  public final void Idle( String Metodo )
  {
    ((IdleAgentManager)Main.agentManager).Idle( this, Metodo );
  }
}
```

La classe è derivata da Agent, astratta, perché non ridefinisce `run` e `putArg`, ed ha un unico costruttore, che si limita ad invocare quello della classe Agent. L'unico metodo che viene aggiunto è `Idle`, che, ovviamente, manda l'agente in Idle, specificando, come nel caso della `go`, il metodo con cui verrà riavviato l'agente quando verrà rimesso in *Esecuzione*.

Il metodo `Idle` dell'`IdleAgentManager`, da parte sua, effettua due semplici operazioni: estrae l'agente dalle strutture dati dell'`AgentManager` e lo salva nell'archivio degli agenti in Idle.

Si può osservare la sintassi della chiamata, che necessita un casting ad `IdleAgentManager`, visto che il metodo `Idle` non è presente in `AgentManager`, classe di `Main.agentManager`.

4.4.1 L'Idle Agent Manager

L'Idle Agent Manager è il modulo che gestisce gli agenti in Idle. Le sue funzionalità sono implementate nella classe `IdleAgentManager`, derivata da `AgentManager`. Le principali operazioni previste sono:

- Mettere in Idle di un agente.
- Mandare in Esecuzione di un agente in Idle, recapitandogli, o meno, un messaggio.
- Determinare, dato l'identificatore di un agente, se è in Idle.
- Salvare gli agenti in Idle su disco. Questo avviene alla chiusura di un place.
- Caricare gli agenti in Idle precedentemente salvati. Questo avviene all'avvio di un place.
- Stampare su uno stream l'elenco degli agenti in Idle.

Infine, per integrare la classe con il resto del sistema, si sono ridefiniti i metodi di *inizializzazione* e di *recapito messaggi agli agenti*, dell'`AgentManager`.

4.5 Il Place Mobile

L'interfaccia grafica di un place è stata ridisegnata nell'ambito di questo lavoro. Nella versione attuale del sistema, un place si presenta come una finestra con:

- un'area di output, a cui sono associati gli stream di sistema `System.out` e `System.err`,
- una linea di testo per l'input, a cui è associato lo stream `System.in`,

- un menù personalizzabile, a cui si possono facilmente delle funzionalità.

Un Place Mobile (Figura 4.1) si differenzia da un place fisso solo per la presenza di due voci in più nel menù principale: le voci di *connessione ad un gateway* e *disconnessione*, queste operazioni, infatti, devono essere completamente controllabili dall'utente.

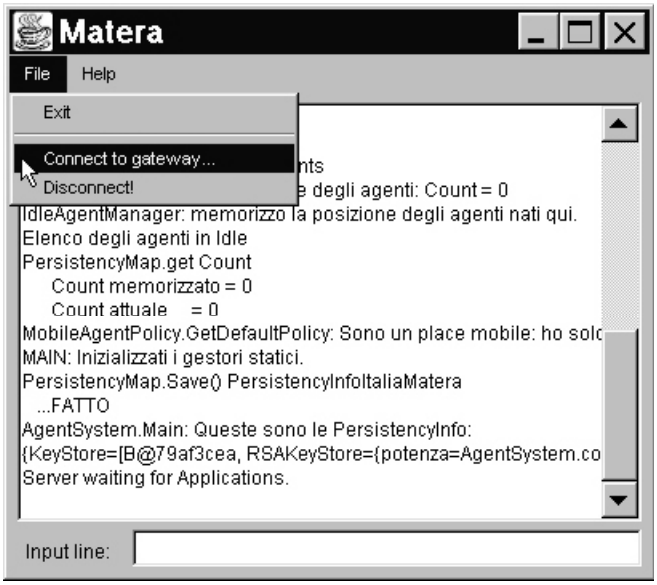

Figura 4.1 Place Mobile: finestra e menu principale.

4.5.1 L'Indipendenza di un Place Mobile dal Resto del Sistema

La prima importante differenza fra un place fisso e uno mobile è proprio la connessione al sistema, che è effettuata automaticamente all'avvio di un place fisso, mentre deve essere avviata dall'utente in un place mobile.

Nella procedura di avvio di un *place fisso* è prevista una fase iniziale in cui si effettua un collegamento con il Gateway del

proprio dominio, al fine di notificare la propria presenza e scaricare le informazioni circa:

1) la topologia del dominio di appartenenza,
2) le informazioni di sicurezza dei canali di comunicazione verso gli altri place,
3) la politica di dominio,
4) le informazioni di sicurezza sugli utenti.

Se un place fisso non dispone di queste informazioni, non può essere avviato. Questo ha senso nell'ottica di un place costantemente collegato alla rete.

Il caso di un *place mobile* è completamente diverso: un place mobile deve potersi avviare anche in modalità off-line, quindi deve essere del tutto indipendente. Nel suo caso le informazioni indicate ai punti 1, 2 e 3 non sono necessarie, mentre lo saranno quelle sugli utenti registrati (4), che vengono scaricate e salvate ad ogni connessione ad un gateway, in modo da poter essere utilizzate all'avvio di un place mobile quando non è possibile accedere alla rete. Viene imposto il vincolo che al *primo avvio* del place mobile questo si debba necessariamente connettere al suo gateway per scaricare le informazioni di sicurezza (4), che, una volta salvate, saranno sempre disponibili.

Le informazioni di sicurezza sugli utenti sono necessarie per il lancio di agenti. Infatti, per ogni agente, deve essere sempre possibile stabilire l'identità del *principal* che lo ha creato e per conto di cui agisce.

Per effettuare il lancio di agenti su un place mobile si ricorre, come nel caso di un place fisso, all'*Agent Launcher*, descritto brevemente nella sezione 4.2.6. In questo caso, però, la connessione non sarà indirizzata al *Supervisor*, ma direttamente al *Place Mobile*, a cui verranno richieste le informazioni di sicurezza sugli utenti per autenticare l'utente prima di autorizzarlo al lancio di agenti. In questo modo il place mobile è completamente indipendente dal resto del sistema.

4.5.2 Identificatore di un Place Mobile

Un place mobile è identificato in base al suo dominio di origine e al nome del *place*. Queste informazioni, però, non ne indicheranno la posizione nello spazio dei place, ma solo l'identità, come spiegato nella sezione 3.2.2. L'identità di un Place Mobile viene specificata nel comando di avvio, che ha il seguente formato:

```
java AgentSystem.Main 5200 Italia Matera 5001 bino Mobile
```

In questo caso:

- `Italia` è il dominio di origine,
- `Matera` il nome del Place Mobile,
- `5200` la porta su cui il place attende connessioni,
- (`bino`:`5001`) indica la coppia (host:porta) di riferimento per il gateway del dominio Italia
- `Mobile` indica che il place è mobile.

In questo caso il place mobile è individuato dalla coppia (Italia, Matera) anche quando è collegato altrove o è disconnesso.

Per informazioni dettagliate sull'avvio di place fissi, gateway e del Supervisor si veda [Chi98].

Selezionando la voce "Connect to Gateway…" del menu principale, compare la finestra con tutte le informazioni di connessione di un place mobile rappresentata in Figura 4.2.

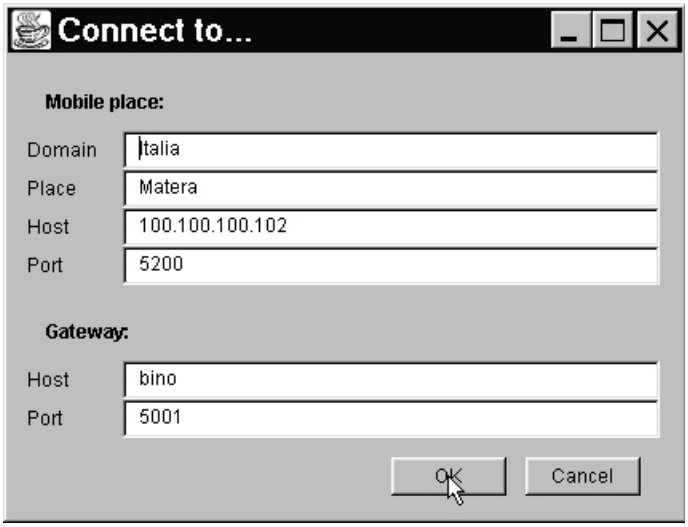

Figura 4.2 Finestra con le informazioni di connessione di un Place Mobile.

I primi quattro campi indicano la posizione attuale (Dominio, Place), nello spazio dei place, del place mobile, il suo indirizzo e la sua porta. Gli altri due campi indicano l'indirizzo e la porta del Gateway a cui si desidera connettersi. Questi devono essere specificati manualmente dall'utente, mentre gli altri campi sono stabiliti dal sistema.

4.5.3 Il Cambio di Indirizzo IP

Si noti come l'indirizzo IP del place mobile viene specificato nella forma numerica e non con il suo nome, come avviene, ad esempio per il gateway. In effetti il terzo campo indicato nella figura precedente viene aggiornato ogni volta che la finestra appare a video, in modo che venga presentato sempre l'indirizzo attuale della macchina su cui esegue il palce mobile. Questo è molto importante, perché ad ogni nuova connessione ad Internet l'indirizzo IP di una macchina può cambiare, bisogna tenere quindi in considerazione questo cambiamento, in quanto altrimenti gli altri place ed il gateway non riuscirebbero a connettersi correttamente al place mobile.

L'operazione di rilevamento dell'indirizzo attuale non è immediata, in quanto il JDK oppure il sistema operativo (Windows 95 nella fattispecie), attuano forme di caching nella conversione fra i nomi delle macchine e gli indirizzi IP in formato numerico. Si è dovuto ricorrere quindi ad uno stratagemma per poter rilevare l'indirizzo più recente: viene attivato un nuovo processo di sistema a cui viene chiesto l'indirizzo attuale.

Questi meccanismi permettono di gestire tutte le variazioni di indirizzo IP che avvengono in successive connessioni a Internet, in maniera completamente trasparente agli agenti.

4.5.4 La Registrazione di un Place Mobile

Alla pressione del tasto *OK* (Figura 4.2), si avvia la procedura di registrazione del place mobile presso il gateway specificato dall'utente. La procedura di registrazione prevede la spedizione al Gateway di tutti i dati raccolti nella finestra precedente, insieme ad alcune informazioni di sicurezza, necessarie per l'autenticazione del place mobile e per la sua autorizzazione all'accesso. Allo stato attuale il meccanismo di sicurezza è previsto ma non implementato. Fra le informazioni di registrazione viene incluso un generico oggetto "*informazioni di sicurezza*". Nella procedura di registrazione eseguita sul Gateway del dominio di accoglienza, viene chiamato un metodo che verifica il contenuto delle "*informazioni di sicurezza*" e decide se autorizzare, o meno, la registrazione. Questo è stato fatto nell'ottica della *modularità* del sistema: è infatti sufficiente sottoclassare il gestore del gateway e ridefinire il metodo che controlla l'accesso, per definire la politica di sicurezza di un dominio nei confronti dei place mobili.

Durante la procedura di registrazione, compare una finestra (Figura 4.3) che fornisce informazioni sulle operazioni che sono in fase di svolgimento, e con cui è possibile interrompere la procedura, nel caso in cui si verifichino problemi. Questo è

necessario per evitare il blocco delle attività di un place mobile, in caso di fallimento della procedura di registrazione.

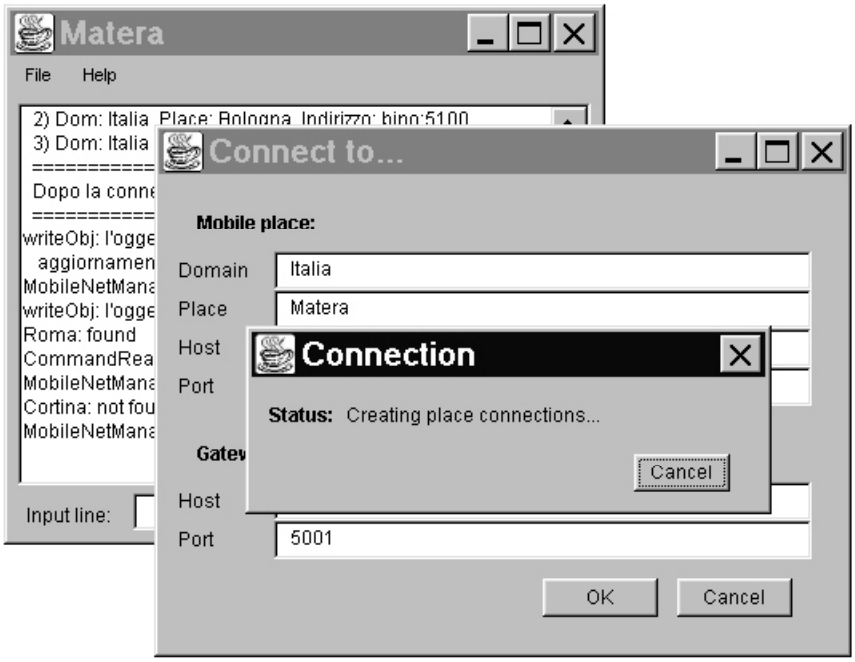

Figura 4.3 Place Mobile: connessione in corso.

A seguito della registrazione di un Place Mobile, vengono effettuate le seguenti operazioni:

- Il Gateway notifica a tutti gli altri place del dominio la presenza del Place Mobile, esattamente come avviene con l'introduzione di un place fisso [Chi98].
- Il Place Mobile riceve dal Gateway le informazioni sulla topologia del dominio e quelle per la sicurezza dei canali di comunicazione. Non viene presa in considerazione la politica di dominio, in quanto il Place Mobile ha una politica propria.
- Il Place Mobile stabilisce le connessioni con tutti gli altri place del dominio.
- Il Place Mobile notifica alla sua *Home Base* (vedi sezione 3.4.3) la sua *posizione* attuale. Questo equivale all'evento di

Connessione del Place Mobile, che quindi passa allo stato *presente*, secondo il modello presentato nella sezione 3.2.5.

A questo punto la procedura di registrazione è terminata e le due finestre si chiudono, per lasciare spazio al normale funzionamento del Place Mobile.

4.5.5 Il Recupero degli Agenti in Attesa

Quando la Home Base del Place Mobile riceve la notifica dell'evento di *Connessione* avvengono tre cose:

- Viene aggiornato lo *stato* del Place Mobile.
- Vengono spediti verso il Place Mobile tutti gli agenti che si trovavano in attesa della sua connessione nello stato Idle.
- Vengono spediti messaggi di notifica della connessione agli agenti che ne avevano fatto richiesta.

4.5.6 La Disconnessione di un Place Mobile

La disconnessione di un place mobile avviene con la selezione dell'opportuna voce del menu principale (Figura 4.1) e si compone delle seguenti fasi:

- Il Place Mobile notifica alla sua *Home Base* l'evento di *Disconnessione*.
- Il Place Mobile invia al Gateway una richiesta di disconnessione, con informazioni di sicurezza dello stesso tipo.
- Il Place Mobile disattiva tutte le connessioni di rete che erano state stabilite, con la terminazione dei demoni che ne sono responsabili. È stata prevista per tutti i demoni un'apposita procedura di terminazione, in modo da non generare gli errori a tempo di esecuzione che si possono verificare nel caso di brusca interruzione di un thread che effettua operazioni di input-output.
- Il Gateway comunica a tutti i place del dominio la disconnessione del Place Mobile.

Si noti come l'implementazione realizzata risponde alla logica indicata nella sezione 2.2.1, relativamente alla *portabilità* delle applicazioni, in quanto viene gestita esplicitamente la mobilità, con la disattivazione di tutti i sottosistemi di gestione della rete, in vista della riattivazione per un collegamento successivo.

4.5.7 La Ricerca di un Place Mobile

Vi sono almeno due situazioni in cui il sistema deve determinare la posizione di un place mobile:

1) quando un agente richiede lo stato e la posizione di un place mobile,
2) quando un agente chiama la primitiva *go(IdentificatoreDelPlace)* (vedi sezione 3.4.2.2) e l'identificatore rappresenta un place mobile. In questo caso il place su cui si trova l'agente, per decidere se spedire l'agente verso il Place Mobile o verso l'Home Base del Place Mobile, deve conoscere posizione e stato del place mobile.

In entrambi i casi avviene la stessa cosa: il place interessato spedisce all'Home Base del place mobile un comando di richiesta di informazioni e si mette in attesa di risposta. È anche previsto un meccanismo di timeout per interrompere l'attesa in caso si protragga troppo a lungo, ad esempio per un malfunzionamento della rete. In questo caso l'impossibilità di ottenere informazioni sul place mobile viene notificata all'agente, con un risultato particolare (punto 1) o con un'eccezione di tipo `CantGoException` (punto 2) (vedi sezione 4.2.3).

4.5.8 Gli Agenti nati da un Place Mobile

Con l'implementazione di un place mobile è stato necessario apportare un'importante modifica al meccanismo di ritrovamento degli agenti *traceable* (vedi sezione 4.2.3). Nella precedente versione di SOMA, ad ogni spostamento di un agente, viene inviato

un comando di aggiornamento della sua posizione al place su cui l'agente è nato. Questo meccanismo è evidentemente inadatto ad agenti nati da place mobili, in quanto un place mobile non è sempre presente, quindi l'informazione sulla posizione dell'agente si perde, ed il meccanismo di recapito dei messaggi cessa di funzionare.

Anche in questo caso, nel progetto iniziale si è fatto uso di un *identificatore semiparlante* (vedi sezione 3.2.2.1). Infatti, l'indentificatore di un agente è formato da 4 elementi:

- il *dominio* di origine
- il *place* di origine
- il nome della *classe* che definisce l'agente
- un *numero* di serie

Per un esempio si veda la Figura 4.4, in cui, nella barra del titolo della finestra, compare l'identificatore dell'agente: *Italia* è il dominio, *Roma* il place, *Shell* la classe e *4* il numero di serie.

Il meccanismo implementato nelle precedenti versioni prevede di estrarre dall'identificatore la coppia (dominio, place) di origine e di spedire a quella destinazione il comando di aggiornamento della posizione dell'agente. Anche in questo caso, quindi, l'identificatore dell'agente ha una seconda funzione: quella di stabilire il place a cui recapitare le informazioni sulla posizione.

Nella presente versione il meccanismo è stato riprogettato separando la nozione di "identità di un agente" da quella di "place a cui inviare il comando di aggiornamento della posizione di un agente".

A livello di implementazione viene fatta distinzione fra agenti nati su place fissi ed agenti nati su place mobili. Per i primi, si mantiene la vecchia semantica, per i secondi, le informazioni sulla posizione dell'agente sono memorizzate nella *Home Base* del Place Mobile.

4.6 Un’applicazione di Esempio: Shell

Con l'introduzione della modularità, dello stato Idle e del concetto di Place Mobile nel sistema SOMA, è sorta l'esigenza di testare metodicamente una serie di meccanismi nuovi, che in alcune situazioni possono rivelarsi anche di una certa complessità. Invece di scrivere una serie di applicazioni di test specifiche, si è deciso di progettare un unico agente, dotato di una interfaccia utente *user friendly*, grazie al quale l'utente sia in grado di testare tutte le funzionalità disponibili agli agenti. (Figura 4.4)

Figura 4.4 Shell: agente per la verifica di tutte le funzionalità del sistema.

4.6.1 Funzionalità Fornite

Le funzionalità fornite dall'agente Shell sono state raggruppate in quattro categorie: operazioni sull'agente, gestione messaggi, informazioni sulla topologia dello spazio dei place e traccia dei place già visitati.

Il menu *Agent* fornisce le funzionalità relative all'agente, in particolare permette di determinare il suo identificatore e di invocare le primitive di spostamento, di messa in Idle e di terminazione.

Il menu *Mail* consente di spedire messaggi ad altri agenti e di leggere quelli ricevuti.

Il menu *Place* permette di chiamare tutte le primitive che forniscono all'agente informazioni sullo spazio in cui si muove, come, ad esempio, la lista dei place del dominio o lo stato di un particolare place mobile. Sono anche fornite funzionalità per il debugging, come la possibilità di visualizzare l'elenco degli agenti in Idle o di quelli in attesa di connessione.

Il menu *History* permette di visualizzare *l'elenco* dei place visitati in ordine cronologico e *l'insieme* dei place visitati, in cui non sono presenti ripetizioni. Questo è utile per verificare le funzionalità di identificazione di un place mobile, indipendentemente dalla sua posizione. Infatti, ipotizziamo che un agente nei suoi spostamenti visiti due volte lo stesso place mobile *PM1*. Ipotizziamo che nella prima visita il place si trovi nella posizione *p1*, mentre nella seconda si trovi in *p2*: l'*elenco* dei place indicherà due voci relative a *PM1*, una prima indicante la posizione *p1*, una seconda indicante *p2*. L'*insieme* dei place, invece, avrà una sola voce relativa a *PM1*. Il menu *History* permette anche di migrare direttamente verso uno dei place precedentemente visitati, o di avere informazioni su un place mobile, selezionando il place interessato dall'*insieme* dei place visitati.

4.6.2 La Classe OutputFrame

Per l'interfaccia è stata utilizzata la classe `OutputFrame`, progettata appositamente ed utilizzata anche per i Place fissi e mobili (vedi sezione 4.5). Si tratta di un componente molto versatile, con le seguenti caratteristiche:

- Mette a disposizione un `InputStream` ed un `BufferedReader`[Jav97], che ricevono input dalla riga di input.
- Mette a disposizione un `OutputStream` che manda l'output all'area di output.
- Mette a disposizione meccanismi per gestire l'evento di uscita, prodotto dalla selezione della voce Exit del menu o dalla pressione dell'apposito tasto sulla barra del titolo.
- Ha un menu principale personalizzabile.
- È serializzabile, in quanto sono stati programmati i meccanismi di serializzazione e deserializzazione degli *stream* utilizzati.

Queste caratteristiche la rendono molto utile per la programmazione di agenti di esempio. Chiaramente, se si ricercano obiettivi di efficienza, è controproducente permettere ad un agente di trasportare un'intera interfaccia grafica: questa dovrebbe considerarsi una risorsa *trasferibile*, ma *fissa* (vedi sezione 1.1.5). D'altra parte, il trasporto di un'interfaccia grafica ha un notevole impatto visivo e rappresenta in maniera molto evidente lo spostamento dello stato di un agente. La soluzione è anche pratica in quanto la finestra di output permette di ripercorrere tutto il tracciato della vita dell'agente.

4.7 Proposta di Estensione: Come fornire Mobilità Personale con SOMA

Osserviamo infine che, con il concetto di place mobile, è possibile associare ad ogni utente non solo agenti, ma anche place che lo seguano nei suoi spostamenti, fornendo in questo modo mobilità personale e permettendo, ad esempio, di realizzare l'astrazione di *Virtual Home Environnment*. Questa proposta può essere un'alternativa a quella vista nella sezione 3.5.2.

C'è una differenza fondamentale fra la mobilità del terminale e la mobilità personale. Nel primo caso, un utente trasporta, con il

suo terminale, tutto il software di supporto al sistema ad agenti ed in particolare al Place Mobile. Il computer portatile può anche ospitare agenti e risorse relative all'utente, utilizzabili nel corso della computazione. Nel caso di mobilità personale, invece, l'utente trasporta solo le sue informazioni di identificazione ed autenticazione e desidererebbe che tutti i servizi venissero forniti dal terminale cui accede, in maniera il più possibile indipendente dalla posizione. Si presentano, quindi, due problemi distinti: quello di fornire i *servizi* all'utente e quello di fornire il *supporto* agli agenti mobili.

I *servizi* potrebbero essere forniti da agenti che, ad ogni connessione del place mobile dell'utente, vi trasportassero tutto il know how e le informazioni necessarie. L'astrazione di *Virtual Home Environnment* verrebbe realizzata in maniera del tutto naturale, visto che i servizi sarebbero offerti dagli stessi agenti, indipendentemente dalla posizione dell'utente.

Per quanto riguarda il *supporto*, si potrebbe progettare in modo da permettere l'avvio di un place generico, capace di assumere l'identità del place mobile di un determinato utente, una volta introdotte le informazioni di identificazione ed autenticazione dell'utente. Questo fornirebbe *mobilità personale* nell'ambito di tutti terminali in cui è istallato il supporto di base. Chiaramente si presenterebbero problemi di *sicurezza*, in particolare riguardo alla procedura di registrazione del Place Mobile, realizzata a partire da un supporto potenzialmente non fidato.

In una successiva evoluzione, si potrebbe anche utilizzare il paradigma *Code On Demand* per rendere disponibile il supporto anche su terminali in cui non sia già istallato.

Tutto questo, però, esula dagli obiettivi specifici del mio lavoro di tesi e fornisce un possibile percorso per una sua prosecuzione.

Conclusioni

Il criterio fondamentale seguito in tutte le fasi di questo lavoro è stato quello della distinzione dei livelli di astrazione. Sono state individuate le *esigenze* da soddisfare, poi è stato scelto un *paradigma di progettazione* per definire *un'architettura software* di cui è stata infine realizzata *un'implementazione*. È stata costruita una semplice *applicazione dimostrativa* capace di verificare le funzionalità dei servizi forniti in risposta alle esigenze che si erano poste.

Le *esigenze* da soddisfare sono state individuate nell'ambito della mobilità. I tipi di mobilità considerati sono due: la *mobilità dei terminali* e la *mobilità personale*. Nel primo caso, l'obiettivo è stato di consentire ad un terminale mobile l'accesso agli stessi servizi fruibili da un terminale fisso. Nel secondo caso, si è avvertita la necessità di fornire uniformità, nei servizi e nelle interfacce cui un utente mobile ha accesso, anche a partire da terminali diversi, in modo da consentire di accedere alle stesse funzionalità sempre nella stessa maniera, non dovendo adattare i propri comportamenti al terminale utilizzato. In altre parole, l'obiettivo è quello di realizzare il concetto di *Virtual Home Environnment*, tipico del mondo dei servizi di telecomunicazione.

Il *paradigma di progettazione* analizzato è quello ad *Agenti Mobili*. Dopo un'analisi del paradigma e delle tecnologie di supporto, si è evidenziata l'espressività del modello nel contesto dell'elaborazione distribuita. Si sono anche sottolineate le caratteristiche su cui è possibile far leva per progettare applicazioni efficaci ed efficienti, e che possono quindi motivare la scelta del paradigma, sia per i sistemi distribuiti in generale, che, in particolare, in un contesto di mobilità.

Per rappresentare la mobilità nel contesto del paradigma ad agenti mobili si è proposto un *modello*, quello del *Place Mobile*. In un Place Mobile i concetti di identità e di posizione sono stati mantenuti rigidamente separati, permettendo la riallocazione di un place e la conservazione della sua identità. E' stato considerato necessario fornire un certo grado di trasparenza al progettista di applicazioni per il mobile computing, risolvendo a livello di supporto una serie di problemi riguardanti la mobilità dei place. Ma si è altresì considerato importante che lo spostamento del place non fosse trasparente agli agenti, per mantenere visibilità di informazioni come il tipo di collegamento fra place e resto del sistema, il grado di sicurezza della connessione o la fiducia che si può accordare al sistema che accoglie il place mobile.

Si è proposto di associare un *Place Mobile* a un *terminale di computazione*, per soddisfare le esigenze di mobilità del terminale, e di associare un *Place Mobile* ad ogni *utente*, per fornirgli mobilità personale.

Nell'ambito del sistema SOMA è stata definita *un'architettura* in grado di fornire *mobilità del terminale* ai computer portatili connessi al sistema. A tal fine sono stati eseguiti cambiamenti nell'architettura del sistema per garantire una maggiore *modularità* e per permettere l'introduzione *dei place mobili* in maniera coerente con il resto del sistema. L'operazione è stata facilitata dalla valida strutturazione dell'architettura di base di SOMA, che si presta molto bene ad operazioni di estensione e di aggiunta di nuovi componenti. La scelta di un linguaggio come Java ha inoltre facilitato l'implementazione grazie alle sue caratteristiche di linguaggio object-oriented, portabile e adatto per la realizzazione di computazione distribuita.

Il lavoro di *implementazione* ha portato alla realizzazione del progetto e di un'applicazione dimostrativa con interfaccia visuale, con cui è possibile verificare tutte le funzionalità offerte agli agenti dal sistema di supporto. Il sistema SOMA costituisce ora un

framework per la realizzazione di applicazioni ad agenti mobili in un contesto di *mobilità del terminale*. Il supporto fornisce tutti gli strumenti necessari per trattare con facilità i problemi legati alla programmazione di rete e alla mobilità. Il progetto di applicazioni per il mobile computing potrà dunque trascurare questi aspetti e dedicare maggiore attenzione alle funzionalità specifiche del servizio che si intende realizzare.

Sono possibili ulteriori sviluppi in varie direzioni. Grazie all'implementazione del concetto di place mobile, si potrà facilmente fornire mobilità personale in SOMA, consentendo ad un utente di avviare il proprio place mobile da terminali diversi, indipendentemente dalla posizione fisica dell'utente e attraverso un'interfaccia diffusa come un browser web. Restano ancora da approfondire alcune nuove problematiche di sicurezza, introdotte dalla nuova capacità di muovere entità come i place: un place mobile va protetto dai possibili attacchi del sistema che dinamicamente lo accoglie e, viceversa, il sistema ospite va difeso dal comportamento possibilmente malintenzionato dei place mobili accolti. Infine, un approccio integrato al supporto della mobilità di terminali, su reti wireless e altamente dinamiche, richiede necessariamente di affrontare anche una serie di problematiche di più basso livello relative alla comunicazione, che possono essere risolte mediante un'accurata e specifica riprogettazione dei protocolli più bassi nello stack OSI.

Bibliografia

[Che98] D. Chess, "Security Issues in Mobile Code Systems", High Integrity Computing Lab, IBM T. J. Watson Research Center, Hawtorne, NY, USA, 1998

[Chi98] C. Chiusoli, Tesi di Laurea presso l'Università di Bologna, Facoltà di Ingegneria, 1998

[Coc98] F. Coccia, Tesi di Laurea presso l'Università di Bologna, Facoltà di Ingegneria, 1998

[CorCS97] A. Corradi M. Cremonini, C. Stefanelli "Melding Abstractions with Mobile Agents", Dipartimento di Elettronica, Informatica e Sistemistica, Università Di Bologna, 1997

[CorCS98] A. Corradi M. Cremonini, C. Stefanelli "Locality Abstractions and Security Models in a Mobile Agent Environnment ", Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, 1998

[CorS98] D. E. Corner, D. L. Stevens, "Internetworking with TCP/IP. Design, Implementation and Internals", Prentice Hall, 1998

[CorST98] A. Corradi, C. Stefanelli, F. Tarantino, "How to employ Mobile Agents in System Management", Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, 1998

[CugGP98] G. Cugola, C. Ghezzi, G. P. Picco, G. Vigna, "Analyzing Mobile Code Languages", Dip. Elettronica e Infomrazione, Politecnico di Milano e Dip. Automatica e Informatica, Politecnico di Torino, 1998

[FugPV98] A. Fuggetta, G.P. Picco, G. Vigna, “Understanding Code Mobility”, Politecnico di Milano, Politecnico di Torino, IEEE Transactions on Software Engineering, Vol. 24, 1998

[GraKN97] R.Gray, D. Kotz, S. Nog, D. Rus, G. Cybenko "Mobile Agents: The Next Generation in Distributed Computing", Department of Computer Science, Dartmouth College, Hannover, Germany, 1997

[HagBM98] L.Hagen, M. Breugst, T. Magedanz "Impacts of Mobile Agent Techonogu on Mobile Communication System Evolution", IKV++ GmbH/Technical University of Berlin, Germany, IEEE Personal Communications, August 1998

[IETF] Internet Engineering Task Force, Sito web: http://www.ietf.org/

[JamSJ98] A. Jameel, M. Stuempfle, D. Jiang, A. Fuchs, “Web on Weels: Toward Internet Enabled Cars”, IEEE Computer, Gennaio 1998

[Jav] Java™ Technology, Sito Web http://www.javasoft.com/

[Jav97] A.A. V.V. , "Java 1.1 Tutto & Oltre", Apogeo, 1997

[Jin99] "Jini™ Technology Executive Overview", Sun Microsystems, Sito web: http://www.sun.com/jini/

[JueG97] J. Jue, D. Ghosal, "Design and Analysis of Replicated Servers to Support IP-Host Mobility in Enterprise Networks", Department of Electrical and Computer Engineering, University of California, Davis, CA, USA, IEEE 1997

[JunP98] E. Jung, Y. Park, “Mobile Agent Network for Supporting Personal Mobility”, Dept. Of Electronics Hanyang University, Seoul, Korea, 1998

[KarT98] N. Karnik, A. Tripathi, "Design Issues in Mobile-Agent Programming Systems", Department of

Computer Science, University of Minnesota, USA, IEEE Concurrency 1998

[KotGN97] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, G. Cybenco "AGENT TCL: Targeting the Needs of Mobile Computers", Dartmouth College, Hannover, Germany, 1997, IEEE Internet Computing, 1997

[KovRR97] E. Kovacs, K. Röhrle, M. Reich, "Integrating Mobile Agents into the Mobile Middleware", Sony International (Europe) GmbH, Telecommunication Research, Fellbach, Germany, 1997

[LazS98] S. Lazar, D. Sidhu, "Laptop Docking Support for Mobile Agent Based Applications", Maryland Center for Telecommunications Research, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, USA, 1998

[Lew98] S. Lewandowsky, "Frameworks for Component-Based Client/Server Computing", Department of Computer Science, Brown University, Providence, USA, ACM Computing Surveys, March 1998

[Mic] Microsoft Corporation, Sito web: http://www.microsoft.com/

[OMG] Object Management Group, Sito web: http://www.omg.org/

[Per97] C. Perkins, "Mobile IP", IEEE Communications Magazine, May 1997

[RacD96] G. Racherla, A. Das, "Mobile Computing. A promising future that still requires much work", IEEE Ottobre/Novembre 1996

[SahM97] A. Sahai, C. Morin, "Mobile Agents for Enabling Mobile User Aware Applications", INRIA-IRISA, Campus de Beaulieu, Rennes CEDEX, France, 1997

[SahMB97] A. Sahai, C. Morin, S. Billiart, "Intelligent Agents for a Mobile Network Manager (MNM)", IRISA, Campus de Beaulieu, Rennes CEDEX, France, 1997

[SOMA] Secure And Open Mobile Agent, Sito web: http://www-lia.deis.unibo.it/Software/SOMA/

[Tar98] F. Tarantino, Tesi di Laurea presso l'Università di Bologna, Facoltà di Ingegneria, 1998.

[Ten98] L. Tenti, Tesi di Laurea presso l'Università di Bologna, Facoltà di Ingegneria, 1998.

[Var97] U.Varshney, "Supporting Mobility with Wireless ATM", Washburn University, IEEE Computer, Gennaio 1997

[Vin97] Steve Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", IEEE Communications Magazine, Febbraio 1997