INDICE

Cap.1 - SOMA

1.1 SECURE AND OPEN MOBILE AGENT

Questo capitolo è dedicato all'approfondimento del sistema ad agenti mobili chiamato "**SOMA**" (Secure and Open Mobile Agent).
Il sistema è stato creato presso il Dipartimento di Elettronica, Informatica e Sistemistica (DEIS) della facoltà di Ingegneria Informatica dell'Università di Bologna.
Esso si inserisce nell'ambito di un progetto ideato del Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) che si chiama "Project Design Methodologies and Tools of High Performance Systems for Distributed Applications".
Il sistema è scaricabile gratuitamente dal sito web del DEIS.
La realizzazione di SOMA è stata portata avanti dai laureandi di Ingegneria Informatica, perciò tutta la documentazione riguardo la sua realizzazione e evoluzione è costituita da varie Tesi di Laurea.

1.2 PROPRIETÀ DI SOMA

Il sistema è scritto interamente in **Java**.
Poiché si è deciso di **non** modificare la Macchina Virtuale Java, ma di usarne le proprietà (soprattutto quelle di serializzazione e di caricamento delle classi) per mantenere la compatibilità.
In questo modo si sfrutta appieno la "portabilità" di Java però, come vedremo meglio nel paragrafo …., ci impedisce di realizzare una mobilità di codice di tipo "forte" (poiché non è possibile catturare e ripristinare lo stato di esecuzione di un thread), perciò ci dovremmo accontentare di una "debole".
Elenchiamo ora quali sono le proprietà principali.

1.2.1 Portabilità

L'aver usato Java come linguaggio di programmazione (senza esserne andati a modificare la Macchina Virtuale) ci garantisce la completa **portabilità** del sistema.
Infatti Java è un linguaggio "semi-interpretato". L'esecuzione del codice compilato può essere fatta su un qualsiasi computer, indipendentemente dal suo hardware e software, a patto di disporre di una Macchina Virtuale Java conforme alle specifiche.
Quindi SOMA può essere eseguito in un ambiente "**eterogeneo**" di computer.

1.2.2 Scalabilità

Tramite opportune "**ipotesi di località**" (che vedremo in dettaglio nel paragrafo 1.3) si riesce a garantire la scalabilità del sistema.
Questo vuol dire che le sue prestazioni non subiscono grosse variazioni se si aumenta (anche enormemente) il numero di nodi che fanno parte di quello che chiameremo "mondo degli agenti".
Come vedremo, tali ipotesi riflettono la struttura interna di Internet, perciò il sistema è adatto a essere "scalato" su reti enormi, fino al livello di Internet.

1.2.3 Sicurezza

La sicurezza viene realizzata sfruttando gli strumenti messi a disposizione dalla versione 1.2 del linguaggio Java.
Al momento è in corso (tramite altre Tesi di Laurea parallele a questa) un completo ridisegno del modello di sicurezza di SOMA in modo da arrivare a una completa protezione sia degli ambienti di esecuzione sia degli agenti. Per questi ultimi sembra che si userà:

- il meccanismo di firma digitale per verificare l'integrità del codice;
- algoritmi crittografici per il trasferimento dello stato degli agenti.
-

1.2.4 Apertura

Il sistema può essere considerato "aperto" poiché aderisce alla standardizzazione basata su CORBA che è stata proposta da OMG.
Aderendo a tale standard, gli agenti di SOMA possono interagire:

- con qualsiasi applicazione CORBA-compliant;
- con gli agenti di altri sistemi, a patto che questi adottino questo stesso standard.

1.2.5 Dinamicità

Gli ambienti di esecuzione sono gestiti dinamicamente: in qualsiasi momento è possibile aggiungere un nuovo CE.
Le modifiche dello stato del sistema verranno "propagate" ai soli nodi che ne sono "interessati" (e non a tutti i nodi).

1.3 LE IPOTESI DI LOCALITÀ

Affinché il sistema sia scalabile occorre introdurre delle "ipotesi di località".
Poiché si voleva che il sistema fosse "scalabile" fino a Internet, le ipotesi che sono state fatte vanno a ricalcarne la struttura interna.
Internet è costituita da un insieme di **località distinte** organizzate in **gerarchia**.
Ogni località impone delle **sue decisioni** per quanto riguarda l'amministrazione e la sicurezza.
La **topologia** con cui sono connesse tra loro le località è gestita in modo **dinamico** e può quindi variare nel tempo.
Ecco allora che SOMA si rifà alla medesima struttura introducendo alcune "**astrazioni di località**" (che possono essere mappate direttamente nelle località di Internet).

1.3.1 Astrazione di Place

Il **Place** rappresenta **l'ambiente di esecuzione degli agenti**. In sostanza, è **l'astrazione del concetto di nodo**. Infatti esso può contenere agenti e risorse. Ogni Place ha un proprio nome.
Si noti che un nodo può contenere più di un Place (sebbene in genere è preferibile crearne uno solo) ma non viceversa (un Place può appartenere a uno e un solo nodo).
In Figura 1.1 ho schematizzato l'astrazione di Place.

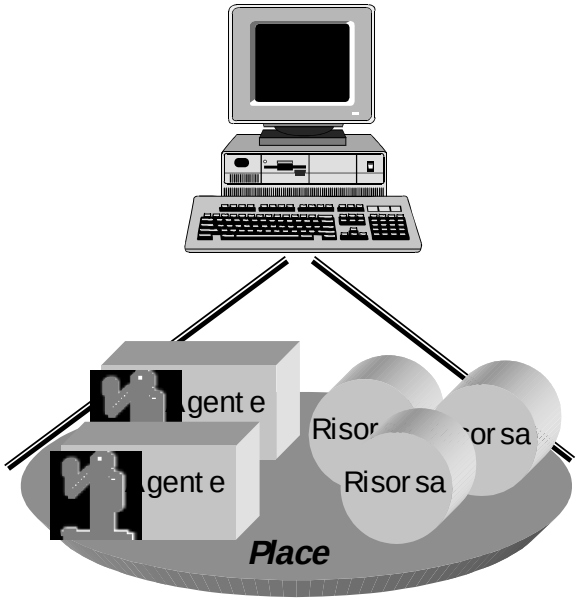


Figura 1.1 - l’astrazione di Place.

1.3.2 Astrazione di Dominio

Un **Dominio** è costituito da un insieme di Place che hanno uguali caratteristiche sotto l’aspetto fisico (per es., i Place corrispondono a nodi che appartengono a una stessa rete locale) oppure logico (per es., hanno politiche di gestione identiche).
Esso rappresenta quindi **l’astrazione del concetto di rete locale**.
A ogni nodo corrisponde uno o più Place del Dominio; l’insieme di tutti questi Place costituisce il Dominio.
Come per i Place, anche il Dominio ha un proprio nome.
In Figura 1.2 ho schematizzato l’astrazione di Dominio.

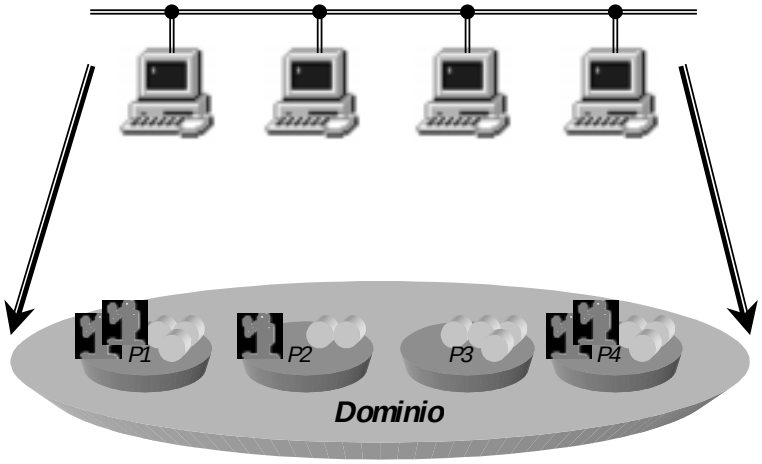


Figura 1.2 - l'astrazione di Dominio.

1.3.3 Astrazione del Mondo degli Agenti

L'idea di Dominio serve per "raggruppare" tra loro un certo numero di Place che hanno certe "caratteristiche" comuni.
Ogni Dominio è isolato dal "mondo esterno": i suoi Place possono comunicare direttamente tra loro ma non con altri Place all'esterno del Dominio.
Ecco allora che serve un'entità che faccia da "tramite" nelle comunicazioni tra un Place del Dominio e uno esterno a esso.
Tra tutti i Place del Dominio ne esiste uno molto particolare che viene chiamato "**Place di Default**" e ha il compito di amministrare il Dominio.
Con la **nuova versione** di SOMA si è abbandonata l'idea di un'unica entità coordinatrice (chiamata "Supervisor") di tutti i dominii e si è preferito strutturare la **coordinazione tra i dominii** inserendoli in un **albero gerarchico**.

1.3.3.1 Il Place di Default

All'interno di ogni dominio esiste un Place particolare che si chiama "**Place di Default**".
Esso rappresenta **l'astrazione di un "gateway" che separa la rete locale dal mondo esterno**, cioè è un "centro di servizi".
È lui che si occupa della gestione del dominio e di tutte le comunicazioni che coinvolgono i Place interni con quelli esterni; è l'intermediario tra gli ambienti interni e quelli esterni al Dominio.
Ha una **conoscenza completa** dei Place appartenenti al dominio mentre ha solo una visione parziale della topologia dell'intero sistema (vedremo tra poco come questa "visione" sia realizzata).
Il Place di Default ha lo stesso nome del Dominio.
In realtà, quando si vuole **creare un nuovo Dominio**, si andrà a creare un Place di Default su un certo nodo. Poi (e solo "poi") potranno essere creati i Place che appartengono a tale dominio.
In Figura 1.3 ho schematizzato l'astrazione di un Dominio gestito dal suo Place di Default (che corrisponde al "gateway" della rete locale).

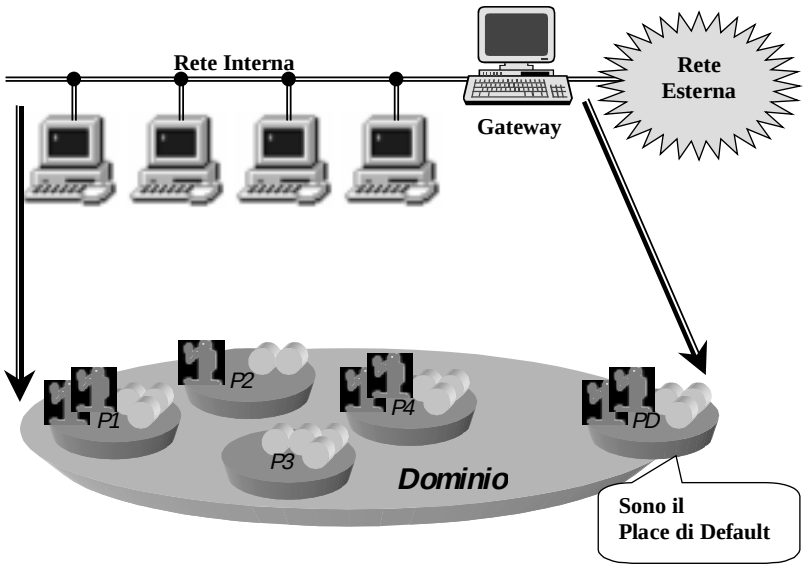


Figura 1.3 - un dominio e il suo Place di Default.

1.3.3.2 Struttura di un Dominio (il PNS)

Tutti i Place appartenenti allo stesso dominio si conoscono l'un l'altro.
Affinché ciò sia possibile, ogni Place ha una tabella, chiamata **PNS** (**Place Name Service** – Servizio di Nomi dei Place), in cui sono memorizzati, uno a uno, tutti i "nomi" (cioè gli identificatori) dei Place appartenenti al Dominio.
Il "proprietario" di tale tabella è il Place di Default. È lui che la "copia" ai nuovi Place al momento della loro creazione (per questo motivo i Place devono essere creati **sempre dopo il Place di Default**).
Sia il Place di Default sia i Place hanno la stessa tabella.
Le eventuali modifiche su una tabella vengono "propagate" al Place di Default che provvederà a informare tutti gli altri Place.

1.3.3.3 Gerarchia di Dominii (il DNS)

Sappiamo già che i dominii sono inseriti in un **albero gerarchico**, in modo tale da permettere un'elevata scalabilità del sistema.
Quindi ogni Place di Default dovrà conoscere il proprio "**genitore**" e i propri "**figli**".
Al momento della creazione di un Place di Default occorrerà specificare qual è il Place di Default genitore (a meno che non si stia creando quello che diventerà la "radice" dell'albero).
Inoltre un Place di Default può essere interessato a "conoscere" altri Place di Default, nel caso servisse instaurare con uno di loro una qualche comunicazione (su richiesta degli agenti).
Ecco allora che ogni Place di Default ha una tabella aggiuntiva, chiamata **DNS** (**Domain Name Service** – Servizio di Nomi di Dominio), in cui sono memorizzati dei "nomi" dei Place di Default.
Si noti che **non** è richiesto che siano memorizzati i nomi di tutti i Place di Default. .
Ognuno avrà una propria visione "limitata" del sistema, in modo da garantire la scalabilità.

La tabella conterrà necessariamente il nome del “genitore” e quelli dei “figli”. Inoltre può contenere altri nomi che possono essere utili.
NB: la tabella DNS è “fisicamente” identica a quella di PNS.
In Figura 1.4 ho schematizzato una gerarchia di dominii.

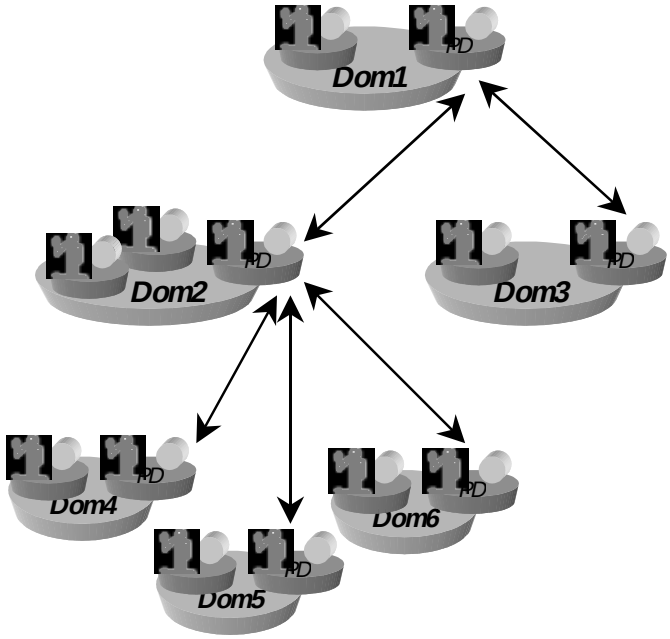


Figura 1.4 - una gerarchia di dominii.

1.3.3.4 Implementazione: il demone

Come si capirà meglio più avanti, le tabelle di DNS e di PNS servono per associare i nomi dei **Place** ai **nodi** di residenza o, per essere esatti, al nodo e al processo “**demone**”[1] corrispondente al Place che è contenuto nel nodo. Infatti la tabella di PNS memorizza le triple:

(**NomePlace**, **IndirizzoIP**, **Porta**)

Infatti ogni Place crea un proprio processo “demone” tramite il quale avviene lo scambio (asincrono) di messaggi con gli altri Place.
Questo concetto verrà chiarito nel paragrafo 1.4.4.

1.4 IDENTIFICAZIONE

Per poter essere identificati, i Place di Default, i Place e gli Agenti devono avere un “nome”, cioè un “identificatore”.

[1] Il termine “demone” è preso in prestito dalla letteratura greca; esso “non vuol dire demonio, ma anima del defunto, o meglio ancora fantasma” [Luciano De Crescenzo, “I Miti degli Eroi”]. Il termine si adatta benissimo perché in informatica un “demone” è un processo che termina l’esecuzione ma non muore del tutto poiché resta permanentemente in attesa di un qualche evento (che gestirà). Per esempio può restare in attesa di messaggi provenienti dalla rete.

1.4.1 Identificazione di un Place di Default (Dominio)

I Place di Default hanno lo stesso nome del Dominio che devono gestire; esso è costituito da una **sequenza di caratteri alfanumerici e numerici ma senza spazi**.
Il nome di un dominio dev'essere **unico** in tutto il sistema.
Esempi **validi** sono: "Pippo", "Unico", "Gigi2" e "6mio".
Non vanno bene: "Un Posto", "Or ora", "per 6".

1.4.2 Identificazione di un Place

I Place **non** di Default hanno un nome composto da **due parti** che sono separate da uno o più spazi ("bianchi").
Poiché un Place è contenuto in un dominio, la prima parte sarà il **nome del Place di Default** a cui fare riferimento; l'altra parte invece è un **nome proprio del Place**, che sarà ancora una volta costituito da una sequenza di caratteri alfanumerici e numerici ma senza spazi.
Possono esserci due Place con lo stesso "nome proprio" all'interno del sistema, ma non possono essere contenuti all'interno dello stesso Dominio.
Occorre che il "**nome completo**" risulti **unico** all'interno del sistema.
Esempi **validi** sono: "Pippo Pluto", "Unico Sub1", "Gigi2 Gigi2".
Non vanno bene: "Un Bel Place", "Ora", "Ora et labora".

1.4.3 Identificazione di un Agente

Anche per gli agenti si usa un **sistema di nomi gerarchico**, perciò sarà a sua volta composto da **due parti**.
Poiché un agente nasce in un Place, la prima parte sarà il **nome del Place di creazione** (che può essere un Place normale ma anche un Place di Default); l'altra parte sarà un **numero intero progressivo** specifico del singolo Place (per l'esattezza, un intero positivo).
Ogni Place ha internamente un "contatore" che viene usato per costruire il nome dei nuovi agenti e viene incrementato di una unità dopo ogni creazione.
In tal modo si distinguono gli agenti all'interno di un Place; aggiungendo l'indicazione del nome del Place si ottiene la garanzia di un nome **unico** all'interno del sistema.
Quindi un agente può essere sempre riferito da qualsiasi altro agente all'interno del sistema, da qualsiasi Place.
Esempi **validi** sono: "Pippo Pluto 1", "Unico Sub1 3", "Gigi 55".
Non vanno bene: "Pippo Pluto", "Or Ro 8 5", "Gigi -55".

1.4.4 Indirizzo di Trasporto & Numero di Porta

Sappiamo che un Place è creato su di un nodo, perciò a ogni Place corrisponderà un "**indirizzo di trasporto**" secondo lo standard **TCP/IP**. Ogni Place ha un processo "**demone**" associato che ha il compito di gestire tutte le comunicazioni con gli altri Place.
Questo "demone" è contenuto all'interno del nodo di residenza del Place e per identificarlo non basta l'indirizzo IP. Si ricorre perciò all'uso di un "**numero di porta**" che viene associato al "demone" al momento della sua creazione. Potendo specificare dei numeri di porta differenti, è possibile creare più Place su uno stesso nodo.

Riassumendo, ogni Place ha un suo nome logico (per es., “Uno” o “Uno Sub”) che viene “traslato” nella coppia “indirizzoIP:NumPorta” (per es. “192.168.0.1 : 1234”) dalla tabella del Servizio di Nomi (che può essere sia il PNS sia il DNS).

1.5 MIGRAZIONE DEBOLE DI CODICE

Come già preannunciato, SOMA non realizza una mobilità “forte” di codice bensì una “debole”.
Non si potrebbe fare altrimenti, perché il linguaggio Java non permette che si acceda allo stato di esecuzione dei thread, in particolar modo allo stack. Non si può quindi “catturare” lo stato di un agente (né, tantomeno, ripristinarlo).
Però Java ci permette di realizzare una sorta di migrazione “debole” di un oggetto. Ora lo vediamo in dettaglio.

1.5.1 Serializzazione & Caricatore di Classi

Per far migrare un “agente”, SOMA sfrutta la proprietà di “**serializzabilità**” che può essere imposta su un oggetto.
Gli agenti non sono dei thread! Sono invece degli “**oggetti passivi**”!
Possono quindi essere serializzati con facilità e quindi inviati su un qualsiasi canale di comunicazione.
Se l’oggetto contiene dei **riferimenti a altri oggetti “serializzabili”**, questi ultimi saranno a loro volta “serializzati” nel momento in cui avviene la serializzazione dell’oggetto che li usa.
Però tutti gli oggetti riferiti dall’agente devono essere serializzabili (fatta eccezione per quelli definiti come “transient” o “static”) altrimenti l’agente non può più migrare (si può dire che si tratta di un “agente immobile”).
Per realizzare la mobilità degli “agenti” servirà un opportuno **Caricatore-di-Classi** (Class-Loader) che riesca a spedire e ricevere questi oggetti tramite la rete.
Poiché un agente può tornare su un nodo già visitato, è conveniente che ogni nodo mantenga in una memoria temporanea il codice degli agenti che ha ricevuto. In gergo si dice che ogni nodo mantiene gli agenti ricevuti in una “**cache**”.
In questo modo, non è sempre necessario trasferire il codice dell’agente da un nodo all’altro; si parla perciò di “**codice su necessità**” (“by need”).
In questa maniera si **aumenta l’efficienza** del sistema, **ma a scapito della sua “consistenza”**.
Infatti nella cache può esserci una “vecchia versione” dell’agente in arrivo. Se si fa pieno affidamento alla “cache”, si possono avere errori o incongruenza nell’esecuzione degli agenti.
NB: nella versione attuale di SOMA **non** è previsto alcun controllo di consistenza della “cache”. Se si sta scrivendo un agente e lo si manda a un Place, questo lo memorizzerà giustamente nella sua cache.
Ma se poi si va a modificare il codice dell’agente e lo si rinvia al Place, questo andrà ciecamente a caricare la “vecchia” versione dalla sua cache. Per fortuna sotto c’è Java che controlla e nel momento in cui si fa la “deserializzazione” dei dati in arrivo, verrà generata un’eccezione e l’agente **non** sarà messo in esecuzione.
Questo è un errore tipico di chi scrive per la prima volta un agente e deve continuamente compilarlo, lanciarlo e farlo migrare qua e là.

L’unica soluzione è di svuotare la cache di ogni Place prima di ogni nuovo lancio.

1.5.2 Come, quando e dove migrare

È l’applicazione che decide “dove” e “quando” migrare, mentre è il sistema che decide “come” migrare.
Il sistema metterà a disposizione un **comando** che l’agente invocherà quando vuole migrare; in esso viene specificato “**come**” migrare.
Il **momento della sua invocazione** all’interno del codice dell’agente indicherà il “**quando**”, mentre il “**dove**” sarà indicato tramite un **parametro** passato al comando (che conterrà il **nome del Place** su cui si vuole andare).
L’agente che vuole migrare deve invocare il metodo di SOMA:

```
public void go (PlaceID destination, String method)
```

Questo metodo è contenuto nella classe “**Agent**”, da cui tutti gli agenti discendono.
Per ora concentriamoci sull’oggetto **PlaceID** passato al metodo; esso rappresenta il Place verso cui si vuole migrare e può contenere sia il nome di un **Place normale** sia quello di un **Place di Default**.
L’oggetto PlaceID serve solo per mantenere il nome di un Place e il costruttore ha la seguente sintassi:

```
public PlaceID (String domain, String place)
```

Per es., per costruire un Place di Default si può scrivere:

```
PlaceID mioPlace = new PlaceID (“Unico”, “”);
```

e per creare un Place che fa riferimento a tale Place di Default:

```
PlaceID mioPlace = new PlaceID (“Unico”, “Sub1”);
```

Si noti che nel primo caso il secondo parametro è una stringa vuota (attenzione: “vuota”, e non “nulla”).

```
DomainID mioPlace = new PlaceID (“Unico”, “”);
```

Sarà il sistema che, usando l’indicazione contenuta nel PlaceID, ricaverà attraverso il Place Name Service l’indirizzo IP del nodo.
Ma resta ancora da esaminare il secondo parametro.

1.5.3 Da dove riprendere?

Sappiamo che SOMA non realizza una mobilità forte, perciò è lecito aspettarsi che un agente, dopo che è stato trasferito sul Place di destinazione, **non** possa riprendere l’esecuzione **dall’istruzione successiva** al comando “go()”.

Ma non si vuole neppure che si riparta sempre dallo stesso punto (altrimenti che mobilità sarebbe).
In SOMA si è scelto di permettere all'agente di **specificare da che metodo riprendere l'esecuzione**, cioè che metodo deve essere invocato dopo che l'agente è stato trasferito.
Stiamo parlando del secondo parametro del metodo "go()":

```
public void go (PlaceID destination, String method)
```

Viene passata una stinga contenente il nome del metodo che dovrà essere invocato a migrazione avvenuta.
Il valore predefinito è "**run**", questo vuol dire che la prima volta che si esegue un agente (cioè al momento della sua **creazione**) il metodo da cui si parte a eseguire è "run()".
Attenzione! L'agente invoca il metodo "go()" e viene trasferito su un altro Place; su di esso il sistema provvederà a invocare un metodo specifico dell'agente. Ma l'agente in questione è una "copia" (un "clone") dell'agente originario.
Nessuno ci garantisce che l'agente originario sia stato interrotto.
Difatti stiamo realizzando una mobilità debole. Non potendo alterare la Macchina Virtuale, **dopo l'esecuzione del metodo "go()" bisogna far in modo che l'agente** "**termini in modo naturale**", andando a chiudere tutta la catena di invocazioni presente sullo stack.
Quindi, dopo l'invocazione di "go()" **l'agente non è stato fermato** ma continua nella sua esecuzione!
Ma può non essere un bene l'avere nel sistema due agenti in esecuzione con lo stesso nome!
Perciò è buona norma far sì che dopo l'invocazione di "go()" non venga fatto nulla, in modo da far terminare l'esecuzione.
Ecco perché la si trova sempre come ultima invocazione di un metodo.

1.5.4 Errori di migrazione

Poiché è possibile che la migrazione non vada a buon fine (per es., se il Place di destinazione è caduto oppure non è più raggiungibile), questo metodo può generare un'eccezione.
Essa è riportata come un oggetto di classe "**CantGoException**".
Ecco perché l'invocazione del metodo "go()" si trova sempre all'interno di un blocco "try-catch":

```
try {
  go (destPlace, metodoRun);
}
catch (CantGoException ecc) {
  .. ..  // reazione all'errore
}
```

È però possibile che il Place di destinazione non sia "conosciuto" dal Place corrente (cioè il suo nome non è contenuto nel **Place Name Service**); in tal caso l'agente viene fatto **migrare sul Place di Default** in modo che lì possa risolvere l'associazione col nome logico cercando nella tabella di **Domain Name Service**.

Se neppure in questa viene trovata l'associazione, **non** si potrà generare la "CantGoException".
Infatti l'agente ha già eseguito una migrazione, quindi lo "stato di esecuzione" caratteristico di Java (cioè lo stack) è stato perso.
Ecco allora che l'agente verrà "risvegliato" (cominciandone l'esecuzione dal metodo precedentemente specificato) però non si troverà sul Place di destinazione ma sul Place di Default relativo al dominio a cui apparteneva il Place da cui aveva chiesto la migrazione.
L'agente deve perciò **controllare** se, a seguito di una richiesta di migrazione, ci si trova effettivamente sul Place che si voleva raggiungere.

Ecco allora che ha senso scrivere il seguente codice:

```
protected PlaceID placeSuCuiMigrare;

public void run() {
  .. ..
  try {
    // Mi memorizzo il Place di destinazione
    placeSuCuiMigrare = destPlace;
    go (destPlace, "startMethod");
  }
  catch (CantGoException ecc) {
    .. ..  // reazione all'errore
  }
}

public void startMethod() {
  // Preleva il Place corrente
  PlaceID placeAtt = agentSystem.getPlaceID();

  if (placeAtt.equals(placeSuCuiMigrare)) {
    .. ..  // BENE, SONO A DESTINAZIONE!
  }
  else {
    .. ..  // NON CE L'HO FATTA!
    }
}
```

1.5.5 Come avviene una migrazione

La migrazione di un agente può essere differente a seconda del tipo dei Place di origine e di destinazione (che può essere: Place normale o Place di Default).

1.5.5.1 Migrazione intra-dominio

L'agente chiede di **migrare su un Place che appartiene allo stesso dominio** del Place su cui si trova: è una migrazione **intra-dominio**.
Poiché ogni Place ha la tabella di **Place Name Service**, il Place di origine conosce quello di destinazione e quindi può comunicare con lui (vedremo più avanti come avvenga ciò).
Non fa differenza se uno dei due Place interessati è quello di Default.
L'agente verrà trasferito direttamente dal primo al secondo Place.
In Figura 1.5 ho schematizzato questo tipo di trasferimento.

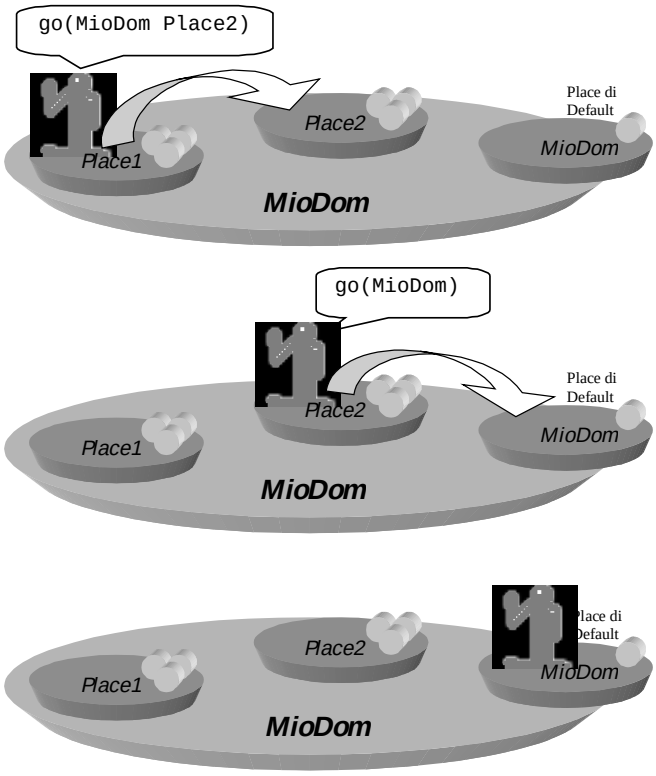


Figura 1.5 - migrazione intra-dominio.

1.5.5.2 Migrazione inter-dominio

Si ha una migrazione **inter-dominio** quando l'agente chiede di **migrare su un Place che si trova all'esterno del dominio** del Place su cui si trova.
Se il Place di origine non è quello di Default, esso non potrà trasferire l'agente sul Place di destinazione poiché non lo conosce. Infatti solo il Place di Default possiede la tabella del **Domain Name Service**.
Però neppure lui conoscerà i singoli Place all'interno del dominio di destinazione.
Ecco allora che si possono presentare casi differenti di trasferimento.
Il primo caso è la migrazione tra due Place di Default: il trasferimento è diretto poiché basta accedere alla tabella di DNS del Place di origine per ottenere quello di destinazione.
In Figura 1.6 un agente migra dal Place di Default chiamato "Dom1" a quello chiamato "Dom2".

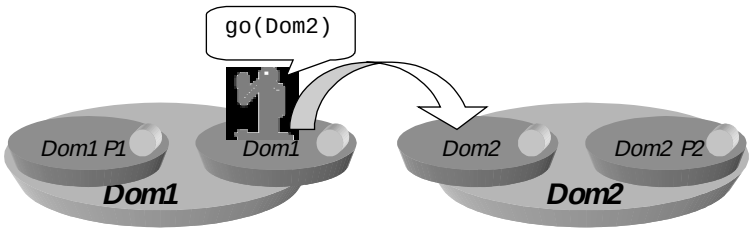


Figura 1.6 - migrazione tra Place di Default.

Se invece l'agente si fosse trovato sul Place interno al dominio (che nel disegno di sopra è "Dom1 P1"), lo spostamento avrebbe avuto luogo in due parti: dal Place al Place di Default del proprio dominio e da lì al Place di Default di "Dom2".
In Figura 1.7 ho schematizzato quest'altro caso.

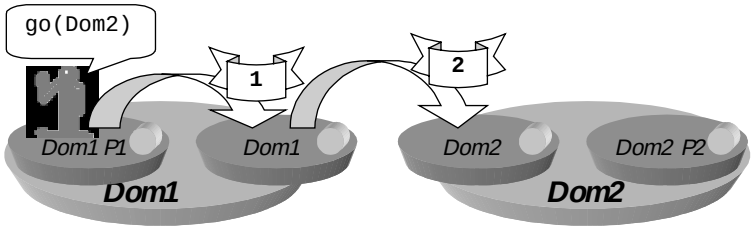


Figura 1.7 - migrazione da Place a Place di Default esterno.

E infine il caso più complicato è quando sia il Place di origine sia quello di destinazione non sono Place di Default.
Infatti occorrono tre passaggi: dal Place al Place di Default locale (si usa il PNS del Place); da lì al Place di Default del dominio a cui appartiene quello di destinazione (si usa il DNS del Place di Default); e infine da lì al Place di destinazione (si usa il PNS del Place di Default del dominio di destinazione).
In Figura 1.8 ho riportato quest'ultimo schema.

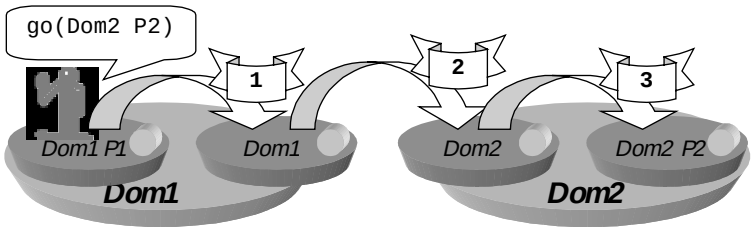


Figura 1.8 - trasferimento tra Place non di Default.

1.6 AMBIENTE DI ESECUZIONE: ENVIRONMENT

La nuova versione di SOMA definisce una classe che si chiama "**Environment**" e che rappresenta l'**ambiente di esecuzione**.
Questo oggetto contiene al suo interno tutti gli oggetti che compongono un singolo Place.

In Figura 1.9 ho schematizzato un oggetto Environment (il significato dei singoli oggetti verrà spiegato nei seguenti paragrafi, e in particolare in 1.12 vedremo come creare l'Environment di un Place).

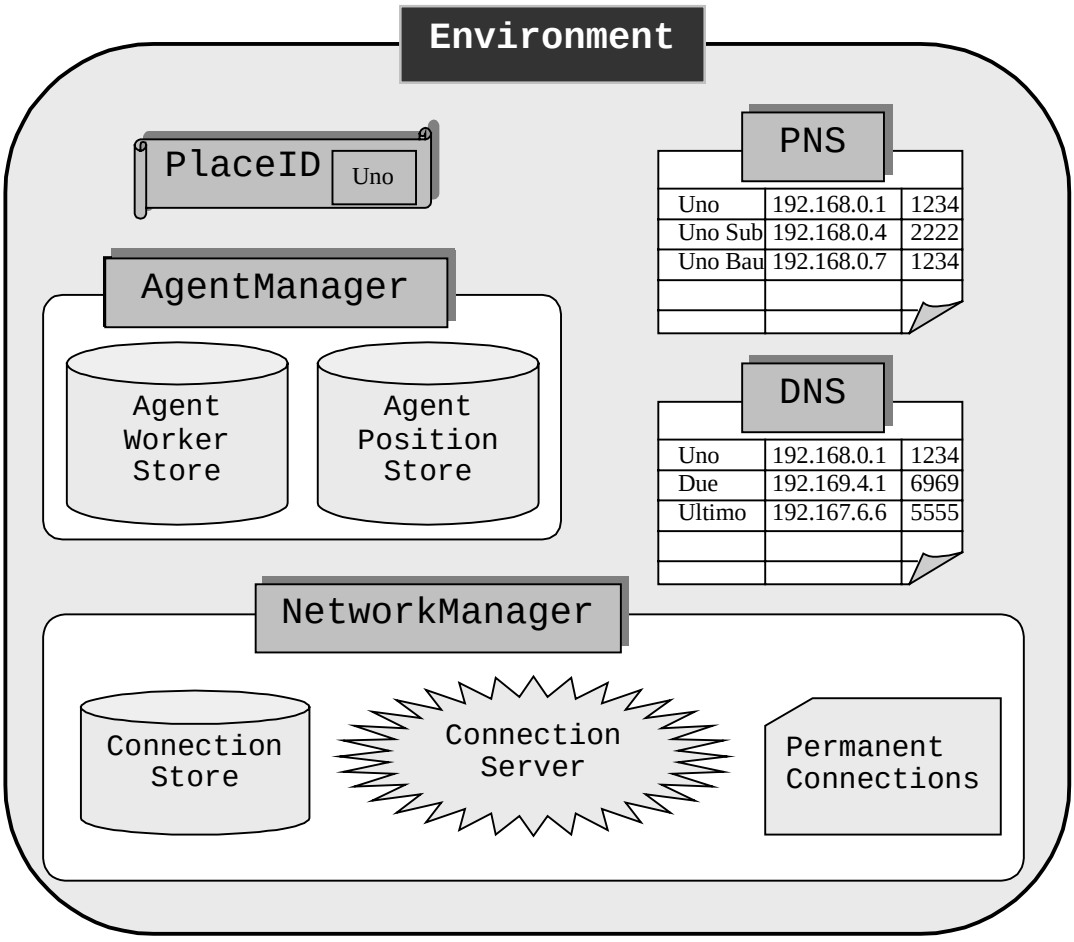


Figura 1.9 - struttura di un oggetto "Environment".

Dalla **versione 1.5** di SOMA esso definisce le seguenti variabili:

`public final PlaceID` **`placeID`**`;`
è il nome (identificatore) del Place;

`public NetworkManager` **`networkManager`**`;`
è il gestore delle comunicazioni fra Place;

`public DomainNameService` **`domainNameService`**`;`
è la tabella di DNS, cioè il Servizio di Nomi di Dominio;
ovviamente è presente solo in un Place di Default;

`public PlaceNameService` **`placeNameService`**`;`

è la tabella di PNS, cioè il Servizio di Nomi di Place;
è posseduta da qualsiasi Place (di Default o normale);

```
public AgentManager agentManager;
```
è il gestore degli agenti.

```
public DirExplorerItem dir;
```
è il riferimento al "menu a linea di comando";
lo si usa in combinazione con un "ExplorerThread";

```
public ThreadGroup threadGroup;
```
è il "gruppo di thread" a cui apparterranno tutti i thread che hanno a che fare col Place. Non viene più eseguito un Place in una Macchina Virtuale Java separata, ma si separano in gruppi tutti i thread relativi a un Place.

```
public InputStream in;
public PrintStream out;
public PrintStream err;
```
Sono i flussi (stream) di input/output/error relativi al Place.
In genere si riferiscono ai flussi standard, ma possono essere "dirottati" (e ciò viene fatto nelle finestre grafiche).

La **versione 1.6** di SOMA aggiunge i nuovi riferimenti:

```
public MobilePlaceManager mobilePlaceManager;
```
è il gestore dei Place Mobili.

La **versione "1.6 con GUI"** di SOMA (cioè la versione precedente a cui è stata aggiunta l'interfaccia grafica) aggiunge invece:

```
public ActionPlace actionPlace;
```
è il gestore della finestra del Place (nel package "gui");

```
public SportelloRichieste sportelloRichieste;
```
è il thread di attesa delle richieste da parte delle applet;

1.7 AGENTI E WORKER

Tutti gli agenti di SOMA ereditano dalla classe astratta "**Agent**", in cui vengono implementate tutte le funzionalità di un generico agente.
In essa vengono definite le seguenti variabili:

```
private AgentID myID;
```
è il nome (l'identificatore) dell'agente; una volta assegnatone uno, non dev'essere più modificabile.

`public String **start** = "run";`

è la stringa contenente il metodo da cui far partire l'esecuzione; per default contiene "run" e quindi si parte da "run()".

`public boolean **traceable** = false;`

specifica se l'agente deve o meno essere "rintracciabile", cioè si può sempre sapere dove si trova (per default vale "false" e quindi significa che l'agente è rintracciabile).

`public Mailbox **mailbox**;`

rappresenta la casella di posta in cui sono depositati i messaggi ricevuti (in modo asincrono) da altri agenti;

`public transient AgentSystem **agentSystem**;`

è colui che fa da tramite tra l'agente e il sistema SOMA; questo è il solo riferimento a un oggetto del sistema che un agente ha (e che dovrà usare); lo vedremo nel paragrafo 1.8.

`transient AgentWorker **worker**;`

il "**worker**" (di classe "**AgentWorker**") è il **thread** che gestisce l'agente. Viene assegnato di volta in volta dal Place.

Ora concentriamoci sul "worker".
Sappiamo già che un agente è un **oggetto passivo** e non un thread (poiché Java non permette la serializzazione di thread); bene, colui che si occupa materialmente del flusso di esecuzione di un agente è il "worker" a cui esso è associato.
Ci sarà quindi **un worker per ogni agente**.
Nel momento in cui un agente nasce (viene creato o arriva attraverso la rete), il sistema creerà un "worker" a cui affiderà l'agente.
Sarà il "worker" che invocherà il metodo di lancio dall'agente (quello da cui iniziare l'esecuzione); dopodiché egli attenderà che il suo agente finisca l'esecuzione.
Si noti che il "worker" è definito "transient", cioè **non** migra insieme all'agente (non potrebbe, è un thread) ma ne viene creato uno nuovo in ogni Place su cui l'agente arriva.

1.8 INTERAZIONE AGENTE-PLACE: AGENTSYSTEM

Ogni agente ha memorizzato in una propria variabile un riferimento all'oggetto di classe "**AgentSystem**".
Questa variabile viene aggiornata dal sistema dopo ogni migrazione dell'agente (infatti è dichiarata "transient"); in essa viene copiato il riferimento all'oggetto "AgentSystem" di cui è proprietario l'oggetto di classe "**AgentManager**" (che vedremo nel paragrafo 1.11) del Place corrente.
Questo è l'unico riferimento che l'agente ha con il Place su cui si trova, e quindi è l'unico modo per interagire col Place e perciò col "mondo esterno".

I metodi di “AgentSystem” che l’agente può invocare sono:

```
public PlaceID getPlaceID()
public Environment getEnvironment()
```
per ottenere rispettivamente il PlaceID e l’Environment del Place su cui ci si trova;

```
protected void go(Agent agent, PlaceID destination)
```
Esegue i controlli di sicurezza (che vedremo nel paragrafo 1.14) e poi invoca il metodo “go()” del “worker” dell’agente;

```
protected void idle(Agent agent)
```
mette in “idle” (ozio) l’agente.

```
public InputStream getIn()
public PrintStream getOut()
public PrintStream getErr()
```
per ottenere i flussi di I/O del Place

```
public PlaceID[] getPlaces()
public PlaceID[] getDomains()
```
per ottenere le informazioni contenute rispettivamente nel PNS e nel DNS del Place corrente (ossia la lista di tutti i PlaceID contenuti rispettivamente nel PNS e nel DNS)

```
public void sendMessage(final Message message)
```
invia un messaggio a un altro agente; nell’oggetto “message” è contenuto, oltre all’oggetto che costituisce il messaggio, sia l’indicazione dell’agente mittente, sia quella del destinatario.

```
public IndexHashtable sharedObjects;
```
è un contenitore di oggetti condivisi; si noti che un Place ha un solo oggetto “AgentSystem” che viene passato all’agente per “**riferimento**”, quindi gli oggetti inseriti in “sharedObjects” (che non è altro che una tabella di hash) possono essere “visti” da tutti gli agenti presenti nel Place.

1.9 LANCIO DI AGENTI

Un agente, per poter essere messo in esecuzione, ha bisogno innanzitutto che gli venga associato un “worker”.
Per crearlo si usa il seguente metodo di “AgentManager”:

```
public AgentWorker createAgent(String agentName,
                               Object argument,
                               boolean isSystemAgent,
                               boolean traceable)
```

Questo metodo va a cercare la classe il cui nome è contenuto nella stringa "**agentName**" passata come parametro e, una volta trovata, la carica in memoria (tramite il Caricatore-di-Classi) forzandola a essere una classe di tipo "Agent" (o sua discendente).
Viene quindi creato un nuovo thread "**worker**" a cui viene dato il riferimento all'agente appena creato. Però il "worker" **non** viene fatto partire ma viene reso a chi ha chiamato il metodo.
Il parametro "argument" è l'argomento (o gli argomenti) dati all'agente (che leggerà al momento dell'esecuzione) mentre "isSystemAgent" serve (se posto a "true") a forzare l'uso del caricatore di classi del sistema.
Invece "traceable" indica se si vuole che l'agente sia "rintracciabile" (cosa significhi lo vedremo meglio nel prossimo paragrafo).
Non è detto che sia sempre reso un "worker"; se il metodo rende "null" significa che non è stato possibile creare l'agente.
Altrimenti, il "worker" è stato creato e per lanciarlo (e quindi mettere in esecuzione l'agente) basta invocare il metodo "**start()**".
Un esempio di codice può essere il seguente:

```
AgentWorker worker = env.agentManager.createAgent
                              ("theAgent", null, false, true);
if (agentWorker != null)
  worker.start();
```

1.9.1 Agenti Rintracciabili e Non Rintracciabili

Sappiamo che è possibile che un agente venga creato "non rintracciabile" impostando a "false" il parametro "traceable".
Questo vuol dire che si alleggerisce il sistema dall'obbligo di conoscere in ogni momento la posizione dell'agente.
In sostanza, l'agente viene creato **senza "mailbox**" (quindi non potrà ricevere e inviare messaggi agli altri agenti) e si permette una gestione più "leggera" dell'agente.
In genere è raro che un agente venga definito non rintracciabile.
La possibilità di "rintracciare" un agente è realizzata dal sistema.
Poiché un agente nasce su un Place, sarà lui il "responsabile" dell'agente perciò sarà compito suo tenere traccia in ogni momento della posizione corrente dell'agente (cioè del Place su cui si trova).
Per come si è strutturato il sistema di nomi, il nome dell'agente contiene l'indicazione del Place di creazione, perciò ogni Place (che riceve l'agente) può andare a "dialogare" con tale Place (e, per es., informare della nuova posizione del suo agente).
Ogni Place possiede una tabella in cui tiene traccia della posizione di ognuno dei suoi agenti (quelli creati da lui).

1.10 COMUNICAZIONE

Bisogna che sia possibile che i Place comunichino tra loro (in modo da coordinarsi nello scambio di agenti e di informazioni); allo stesso modo è opportuno che anche gli Agenti comunichino tra loro.

1.10.1 Comunicazione tra Place

Nel paragrafo 1.3 abbiamo fatto delle ipotesi di località che ci hanno condotto alla scomposizione del sistema in "località".
Abbiamo detto che si privilegiano le azioni "locali" mentre quelle che "escono" dalla località sono penalizzate (ma tanto, poco importa perché vengono fatte di rado).
Quest'idea è stata mantenuta anche nella definizione delle modalità di comunicazione tra Place.

1.10.1.1 Comunicazione intra-dominio

Tutti i Place **all'interno di uno stesso dominio** sono **connessi in modo permanente** tra di loro; ognuno ha tanti canali aperti quanti sono gli altri Place del dominio.
In Figura 1.10 ho schematizzato questi canali di comunicazione.

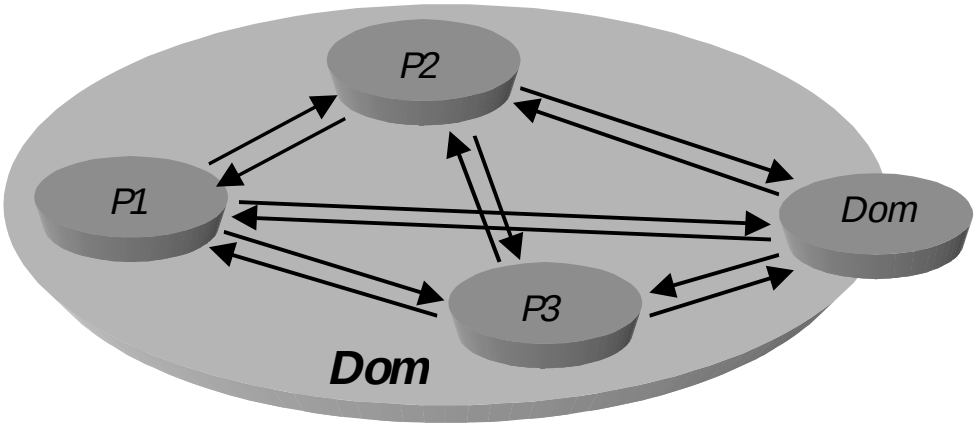


Figura 1.10 - connessioni fra i Place di un Dominio

In questa maniera non occorre ogni volta creare un canale di comunicazione su cui eseguire lo scambio di messaggi; infatti basterà andare a usare quell'unico canale che collega col Place destinatario.
Così facendo si ottimizzano le prestazioni nelle comunicazioni tra Place che sono "locali".
Per memorizzare tutte le "connessioni" verso gli altri Place del dominio si usa una tabella che fa corrispondere i vari PlaceID con la specifica connessione aperta verso di loro.
Tale tabella è realizzata da un oggetto di classe "**ConnectionStore**" il quale mette a disposizione i due metodi:

```
public Connection putConnection(PlaceID placeID,
                                Connection connection)
public Connection getConnection(PlaceID placeID)
```

Quindi la tabella è composta da coppie "(**PlaceID**, **Connection**)".
La classe "**Connection**" rappresenta il "**demone**" che è responsabile di tutte le comunicazioni che avvengono via "socket" con un certo Place.

Difatti il suo costruttore è:

```
public Connection(Socket mySocket, Environment env)
```

Quando viene invocato il metodo “start()”, il demone genera un thread separato il quale va a leggere un “comando” dalla sua socket.
Se nessun comando arriva, il thread si bloccherà in attesa che ne arrivi uno. Altrimenti leggerà il comando e ne invocherà il metodo “start()”.
Sì perché il comando è un oggetto di classe “**Command**”; essa è una classe **astratta** che è **serializzabile** (ovviamente, dev’essere trasferita su socket) e implementa l’interfaccia “**Runnable**” (deve perciò definire il metodo “run()”).
Inoltre mette a disposizione il metodo “start()” con la sintassi:

```
public void start(Connection ReturnConnection,
                  Environment env)
```

Questo metodo mette in esecuzione il codice contenuto nell’oggetto “comando” (in pratica, va a invocare il metodo “**run**()” di se stesso).
La “ReturnConnection” rappresenta la connessione (locale al Place di destinazione) con cui comunicare col Place mittente del comando; invece “**env**” è l’oggetto “Environment” relativo al Place locale, cioè quello su cui eseguire il comando.
Tutti i “comandi” ereditano dalla classe “**Command**” e non fanno altro che andare a ridefinire il metodo “**run**()” (dichiarato “astratto” nella classe genitore).
Ogni “comando” viene serializzato e trasferito su un canale di comunicazione che è “visto” come un flusso (“stream”) ordinato di oggetti (“**Object-I/O-Stream**”).
Il flusso viene realizzato usando un collegamento su “**socket-stream**”.
Questo vuol dire che si va a usare una trasmissione “**affidabile**” dei dati poiché si sfrutta il protocollo **TCP/IP**.

1.10.1.2 Comunicazione inter-dominio

Abbiamo detto che le comunicazioni tra Place appartenenti a dominii diversi sono “rare”, perciò è bene non tenere aperta una connessione permanente con ogni altro Place esistente (non sarebbe gestibile).
Quindi si è scelto che tra **Place non appartenenti allo stesso dominio** il **canale di comunicazione venga creato su necessità** (“by need”).
Quando un Place vuole comunicare con un altro “esterno” al suo dominio, apre un canale di comunicazione verso quel Place e gli invia il “comando” che deve.
Poiché questa comunicazione è di tipo “**asincrono**”, l’unica soluzione è di creare in ogni Place un “**demone**” che attende la richiesta di connessioni. Esso si chiama “**ConnectionServer**”, e non fa altro che creare una “ServerSocket” (quindi si usa ancora una volta il protocollo **TCP/IP**) su una certa **porta** locale, il cui numero corrisponde proprio a quello specificato alla creazione del Place (vedi paragrafo 1.4.4).
Su queste connessioni possono essere inviati gli stessi oggetti discendenti di “**Command**” che abbiamo visto poc’anzi.

Il “ConnectionServer” crea un **thread** “**Connection”** (la stessa classe vista nel caso delle connessioni locali) a cui delega la gestione del comando. Il thread sarà “legato” a una porta differente da quella del Place; essa viene resa in automatico dal metodo “**accept**()” (che permette di attendere l’arrivo di una richiesta di comunicazione).
Il delegare la comunicazione a un thread separato permette al demone di mettersi subito in ascolto di una nuova richiesta.
Si noti che dal lato del mittente la porta che si usa non viene cambiata.
Nella seguente Figura 1.11 ho riportato in tre schemi la sequenza di passi affinché si stabilisca la comunicazione tra due Place.

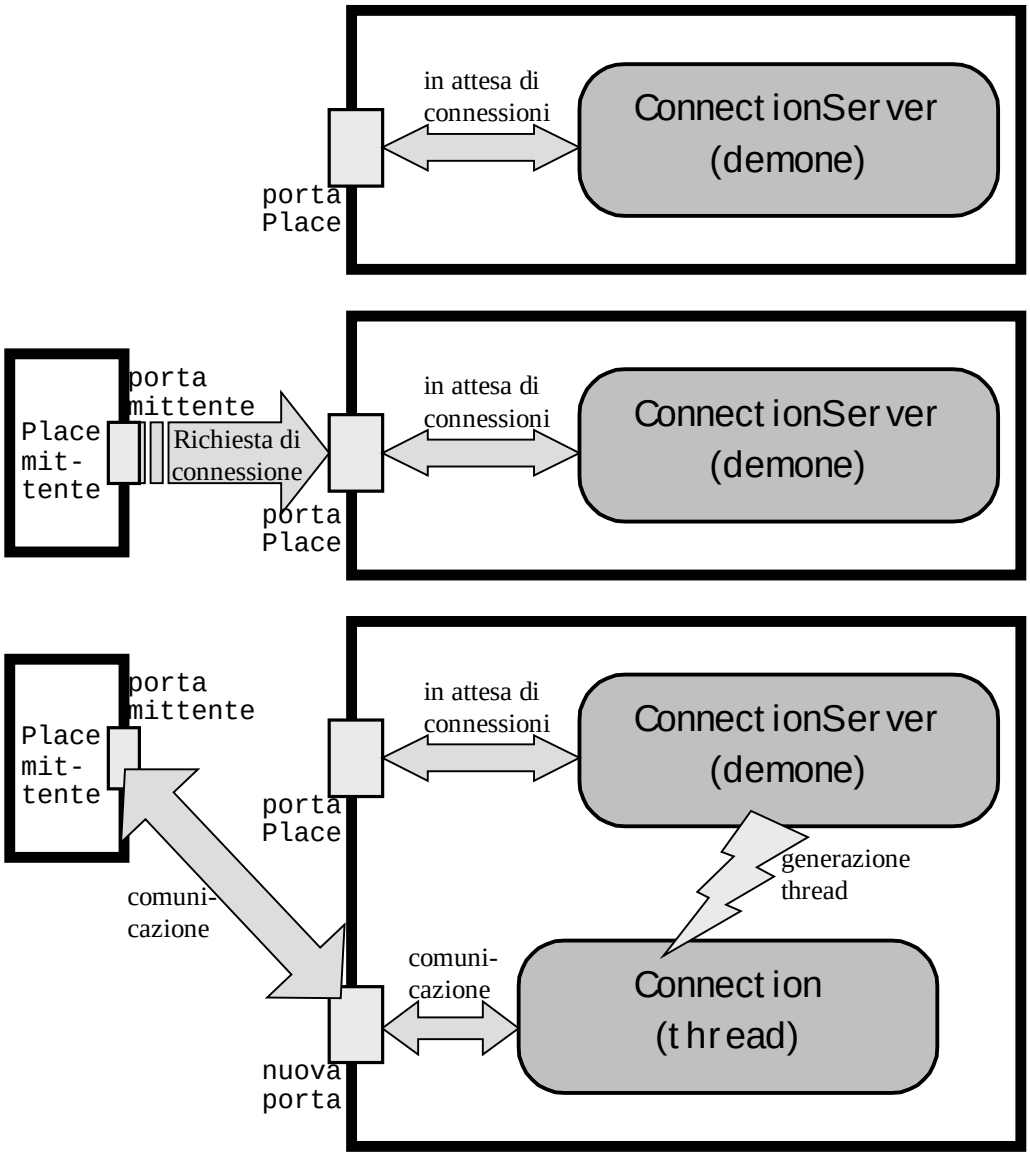


Figura 1.11 - gestione della comunicazione tra Place.

In aggiunta, ogni Place permette la creazione di "**connessioni permanenti**" verso Place "esterni". Ossia si possono creare delle connessioni fisse verso un Place esterno, come se questo fosse interno.

1.10.1.3 Il NetworkManager

La classe "**NetworkManager**" serve da "gestore unico" delle comunicazioni tra due Place qualsiasi.
Infatti essa usa i seguenti oggetti (dichiarati al suo interno):

```
public ConnectionStore connectionStore;
ConnectionServer connectionServer;
private Hashtable permanentConnections;
```

Ogni Place ha un suo gestore di questo tipo, infatti nella classe "**Environment**" viene usato un oggetto di classe "NetworkManager".

1.10.1.4 I discendenti di "Command"

Ecco una breve descrizione di tutti i comandi che due Place possono scambiarsi:

- **Movimento di codice e agenti:**
  - ✓ ClassRequestCommand: per richiedere l'invio di una classe;
  - ✓ SendClassCommand: per inviare una classe;
  - ✓ AgentTransportCommand: per trasportare un agente;
- **Gestione delle tabelle dei Dominii (DNS):**
  - ✓ DomainRefreshCommand: aggiornamento della tabella;
  - ✓ DomainRegisterCommand: nuova registrazione;
  - ✓ PutDomainCommand: inserimento di una nuova entrata;
  - ✓ RemoveDomainCommand: cancellazione di un'entrata;
- **Gestione delle tabelle dei Place (PNS):**
  - ✓ PlaceRefreshCommand: aggiornamento della tabella;
  - ✓ PlaceRegisterCommand: nuova registrazione;
  - ✓ PutPlaceCommand: inserimento di una nuova entrata;
  - ✓ RemovePlaceCommand: cancellazione di un'entrata;
- **Gestione delle connessioni tra Place:**
  - ✓ ConnectionCommand: per creare una connessione stabile;
  - ✓ StopConnectionCommand: per chiudere una connessione;
  - ✓ TransportCommand: serve per incapsulare un comando; questo comando viene trasportato fino a una stazione intermedia e da lì il comando contenuto viene inviato alla vera destinazione;
  - ✓ QuickCommand: comando che non viene eseguito in un thread separato, in modo da avere maggior efficienza.
  - ✓ SendMessageCommand: invio di un "Message" per un agente.

1.10.2 Comunicazione tra Agenti

Affinché gli agenti possano cooperare tra loro, occorre predisporre dei meccanismi. Abbiamo visto nel paragrafo **Errore. L'origine riferimento non è stata trovata.** che si può distinguere tra comunicazione "stretta" e "lasca".
Vediamo ora cosa mette a disposizione SOMA.

1.10.2.1 Interazione "stretta": solo oggetti condivisi

La si usa quando si vuole avere una forte collaborazione tra agenti.
Nella versione corrente di SOMA, un agente non può creare un nuovo agente. Tutt'al più può creare un "worker" che si occupa di mandare in esecuzione un agente, ma in ogni caso **un agente non può mai fare riferimento direttamente a un altro agente**.
Per questo motivo un agente **non può invocare i metodi di un altro agente!** non è permesso questo tipo di interazione "stretta".
Invece viene permesso l'uso di "**oggetti condivisi**".
Come abbiamo accennato nel paragrafo 1.8, all'interno dell'oggetto "**AgentSystem**" è presente una variabile così dichiarata:

```
public IndexHashtable sharedObjects;
```

Sappiamo già che in ogni Place c'è una sola istanza di "AgentSystem" e che questa viene di volta in volta "**passato per riferimento**" all'agente (tramite le variabili pubbliche dichiarate in "Agent").
Quindi ogni agente può riferirsi alla variabile "**sharedObjects**" che quindi rappresenta un riferimento a un oggetto condiviso.
Poiché "sharedObjects" non è altro che una tabella di hash, essa può memorizzare delle coppie "(chiave, valore)" in cui il campo "chiave" è rappresentato da un valore numerico mentre "valore" è un riferimento a un qualsiasi "Object".
Un agente che "deposita" qui un oggetto lo renderà "visibile" a tutti gli altri agenti (a patto che questi ne conoscano il corrispondente valore numerico).
In Figura 1.12 c'è uno schema di accesso a oggetti condivisi.

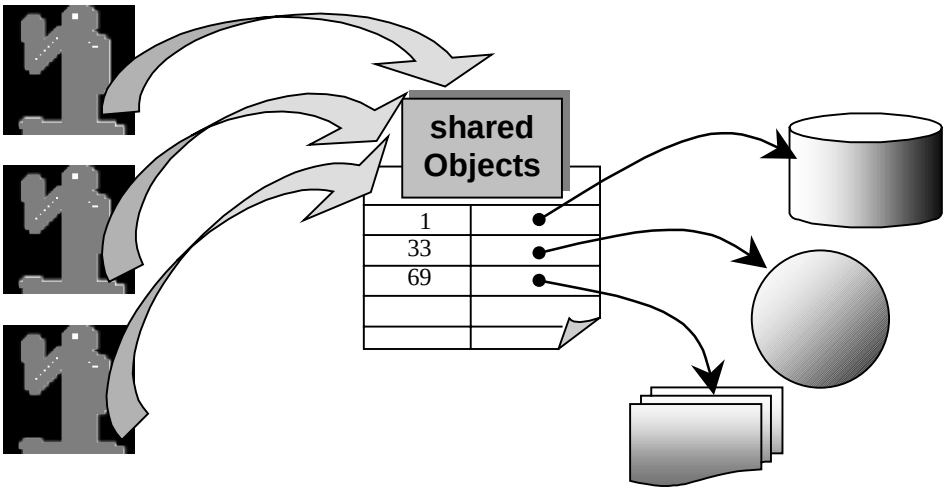


Figura 1.12 - agenti che accedono all'oggetto "sharedObjects".

1.10.2.2 Interazione "lasca": solo scambio di messaggi

Questo tipo di interazione può essere realizzata mediante uno **scambio di messaggi** ("message passing") tra agenti.
Come visto nel paragrafo 1.7, ogni agente ha al suo interno (poiché viene definito nella classe "Agent") la variabile:

```
public Mailbox mailbox;
```

La classe "**Mailbox**" rappresenta l'astrazione di "casella di posta" di un agente. Ogni agente ha una sua istanza di "Mailbox" che si porta dietro durante le migrazioni.
In essa vengono memorizzati i messaggi che arrivano – in modo **asincrono** – dagli altri agenti.
In ogni momento l'agente può controllare se ha ricevuto un messaggio oppure spedirne uno a qualcuno.
La classe "Mailbox" mette a disposizione i seguenti metodi (tutti "sincronizzati") che permettono di inserire e togliere messaggi:

```
public synchronized void storeMessage
                                        (Message message)
public synchronized Message getMessage()
public synchronized boolean isMessage()
```

Il costruttore della classe "**Message**" ha la seguente sintassi:

```
public Message(Object message,
               AgentID from, AgentID to)
```

Un messaggio ha quindi un agente mittente (“from”) e un destinatario (“to”) e il contenuto informativo è rappresentato dall’oggetto “message” (di classe “Object”, quindi può essere un suo qualunque discendente).
Si noti che ogni agente deve **conoscere il “nome” del destinatario**, cioè il suo identificatore unico.
Una volta creato il messaggio, per spedirlo bisogna consegnarlo al Place che provvederà a farlo arrivare all’agente destinatario.
L’agente non dovrà far altro che invocare il metodo “sendMessage()” del suo oggetto “AgentSystem”. Esso ha la seguente sintassi:

```
public void sendMessage(final Message message)
```

È compito del sistema **rintracciare l’agente e fare la consegna**.
Si noti che la spedizione del messaggio avviene in modo **trasparente alla locazione**: infatti si specifica il nome dell’agente di destinazione e non quello del Place su cui l’agente risiede.
Inoltre, la **spedizione** del messaggio è **asincrona bloccante**, cioè il sistema si occuperà di recapitare il messaggio a destinazione; nel frattempo l’agente non dovrà aspettare. Pertanto l’agente rimarrà “bloccato” per il solo tempo necessario a copiare il messaggio in un’area di sistema, dopodiché l’agente potrà proseguire nella sua esecuzione. Quindi **non c’è garanzia** che il messaggio arrivi a destinazione.
Anche la **ricezione** di un messaggio è “**bloccante**”; quando l’agente chiede la lettura di un messaggio, se la “mailbox” è vuota rimarrà bloccato fintantoché non ne arriverà uno.
Si noti che **solo il Place che ha creato l’agente** sa dove egli si trovi in ogni momento; perciò il messaggio dovrà seguirà la seguente strada:

AgMitt → Place → PlaceOrigine →PlaceCorrente →AgDest

Cioè il Place spedisce il messaggio al Place in cui è stato creato l’agente (lo si conosce poiché il suo nome è dato dalla parte iniziale del nome dell’agente) e quest’ultimo lo invierà al Place su cui si trova al momento l’agente.
Ovviamente, se si fa riferimento a Place “esterni” al dominio, tutti i messaggi dovranno passare per il Place di Default in modo che possano essere “instradati” verso il dominio corretto.
In Figura 1.13 ho schematizzato un possibile scenario di scambio di messaggi tra agenti che si trovano in Place anche diversi.
Ci sono quattro agenti che inviano messaggi; ognuno lo invia a uno specifico agente.
L’agente “D1 P1 2” – che si trova sul Place “D1 P2” – invia un messaggio all’agente “D1 P2 2” che si trova sullo stesso Place.: la consegna avverrà in modo diretto.
Sul Place “D1 P1” l’agente “D1 P1 1” invia un messaggio all’agente “D1 5” che si trova sul Place “D1” all’interno dello stesso dominio.
Il messaggio verrà trasportato dal Place “D1 P1” al Place di Default “D1” e lì recapitato al suo agente (che si trova ancora su di lui).
Il terzo messaggio è inviato da “D1 P2 2” a “D1 P2 33” che però si trova in un altro dominio. Ecco allora che occorre inviarlo al Place di Default del proprio dominio (cioè “D1”), il quale

provvederà a inviarlo al Place di Default del dominio corretto (ossia “D2”) che a sua volta lo invierà al Place su cui si trova l’agente (che è “D2 P8”) e da lì verrà consegnato all’agente.

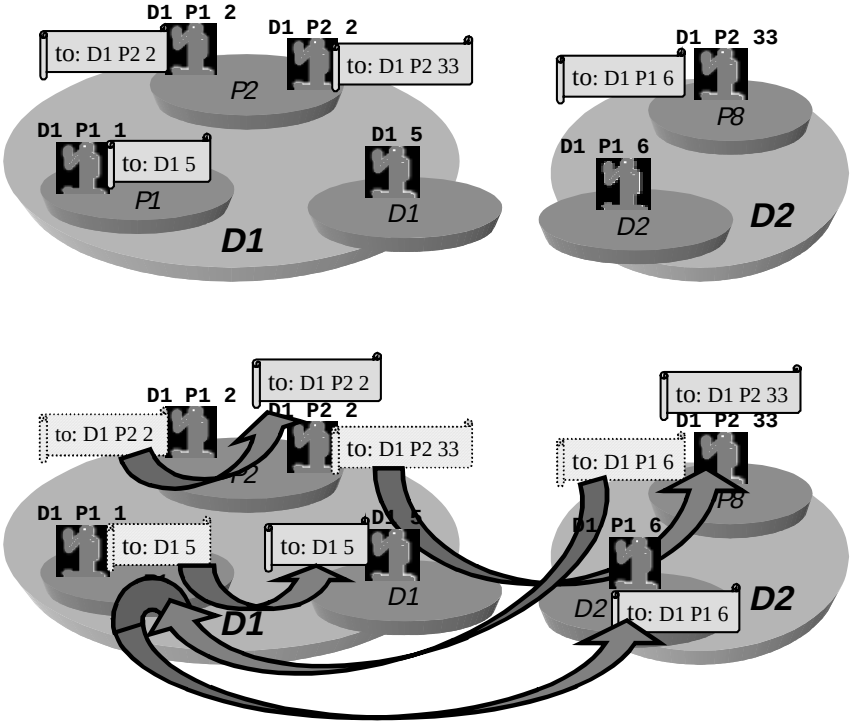


Figura 1.13 - scambio di messaggi fra agenti.

Può anche accadere che il messaggio compia un lungo percorso.
Per esempio, il messaggio inviato dall’agente “D1 P2 33” (sul Place “D2 P8”) a “D1 P1 6”. Sebbene l’agente sia nello stesso dominio, il messaggio va recapitato innanzitutto al Place di origine dell’agente, cioè a “D1 P1”. Per farlo però occorre fare due tappe intermedie (sui Place di Default dei due dominii). Il Place di origine ripasserà indietro il messaggio, fino al Place di Default “D2”, quello cioè su cui si trova al momento l’agente.
Un'altra forma di comunicazione che abbiamo visto è la quella “**anonima**”, ossia quella che viene realizzata tramite l’uso di una “blackboard” o di uno “spazio delle tuple”.
La corrente versione di SOMA **non** mette a disposizione nulla di tutto ciò (sebbene in una precedente versione fosse disponibile una “blackboard”).

1.11 GESTIONE DEGLI AGENTI: AGENTMANAGER

L’oggetto “**AgentManager**” è uno dei costituenti dell’Environment di un Place. Esso rappresenta il **gestore degli agenti** contenuti nel Place.
Tra le variabili che contiene, le più importanti sono:

```
Environment env;
```
il riferimento allo stesso Environment del Place

`AgentSystem` **`agentSystem`**`;`

il riferimento all'oggetto di interfaccia tra gli agenti e il sistema; questo viene copiato nella variabile "agentSystem" di ciascun agente che viene creato in questo Place.

`public ClassManager` **`agentClassManager`**`;`
`public ClassManager` **`cacheClassManager`**`;`

sono i caricatori di classi che permettono di caricare classi dal file-system locale; si riferiscono rispettivamente alla directory in cui risiedono le classi degli agenti e a quella che serve da memoria cache degli agenti.

`public int` **`AgentIDCounter`**`;`

è il contatore numerico progressivo per la numerazione degli agenti; a ogni agente creato da questo Place viene assegnato un nome costituito dal nome del Place seguito dal valore di questo contatore (che viene successivamente incrementato).

`public AgentWorkerStore` **`agentWorkerStore`**`;`

è il "contenitore" dei "worker" degli agenti che sono al momento in esecuzione;

`public AgentPositionStore` **`agentPositionStore`**`;`

è una struttura per memorizzare la posizione (cioè il Place) su cui si trovano al momento tutti gli agenti che sono stati creati dal Place. Serve per poter rintracciare sempre i propri agenti.

Inoltre mette a disposizione dei metodi per:

- creazione di agenti e dei "worker" associati;
- invio di messaggi a un agente;
- assegnazione del nome a un nuovo agente;
- costruzione di un "pacchetto" contenente un agente in modo che possa essere trasportato su un altro Place;
- altre funzionalità di manipolazione di agenti.

1.12 LA CREAZIONE DI UN PLACE (ENVIRONMENT)

Abbiamo visto che l'oggetto "**Environment**" racchiude tutti gli oggetti che costituiscono un Place; infatti per creare un Place bisogna creare un oggetto "Environment".
Il suo costruttore principale ha la seguente sintassi:

```
public Environment (final PlaceID placeID,
                    final DirExplorerItem dir,
                    final int port,
                    final InputStream in,
                    final PrintStream out,
                    final PrintStream err)
```

La differenziazione tra Place “normale” e “di Default” sta nel parametro di classe “**PlaceID**” passato.
Il parametro intero “**port**” contiene il numero di porta locale su cui il “demone” del Place attenderà l’arrivo di comandi da parte di altri Place.
Il parametro di classe “**DirExplorerItem**” corrisponde a un direttorio del menu a linea di comando di SOMA (che vedremo meglio nel capitolo successivo); in esso verranno inserite tutte le voci di menu relative al Place che si sta per creare.
I tre flussi contenuti nei parametri “in”, “out” e “err” corrispondono ai tre flussi di Input/Output/Error su cui il Place interagisce con l’utente.
Essi verranno usati soprattutto da “**ExplorerThread**” (il thread che permette di “navigare” nel menu a linea di comando) per ricevere input dall’utente e stampare i messaggi a video.
Il costruttore di “Environment” creerà tutti gli oggetti che costituiscono un Place.
Ma non basta, per creare un **Place di Default** si dovrà:

- **creare** il suo oggetto **Environment** (col PlaceID di un Place di Default);
- “**registrare**” il suo **DNS** presso un DNS di un altro Place di Default, in modo da inserirlo nella gerarchia dei DNS.

Quest’ultima operazione **non** va fatta se si tratta del DNS “radice” della gerarchia. Quindi il codice fare tutto questo è il seguente:

```
PlaceID domID = new PlaceID (dom, "");
DirExplorerItem dir = new
                              DirExplorerItem (domID.domain);
Environment env = new Environment (domID, dir, porta,
                           System.in, System.out, System.err);
boolean ok = env.domainNameService.register
                                        (regIndIP, regPorta);
```

Invece, per creare un **Place** all’interno di un dominio:

- **creare** il suo oggetto **Environment** (col PlaceID di un Place);
- “**registrare**” il suo **PNS** presso il PNS del Place di Default del dominio a cui il Place appartiene.

La registrazione presso il PNS dev’essere sempre fatta perché il proprietario della tabella di PNS è il Place di Default; tutti i Place ne hanno una “copia”.

Il codice per la creazione di tutto ciò è il seguente:

```
PlaceID plaID = new PlaceID (dom, place);
DirExplorerItem dir = new
                               DirExplorerItem (plaID.place);
Environment env = new Environment (plaID, dir, porta,
                           System.in, System.out, System.err);
boolean ok = env.placeNameService.register
                                        (regIndIP, regPorta);
```

In Figura 1.14 ho disegnato uno schema che mostra la sequenza di costruzione di ciascun oggetto coinvolto nella costruzione di un Place.
Si noti che non vengono costruiti tutti dall'oggetto Environment ma spesso è la creazione di un oggetto da parte sua che a sua volta crea ulteriori oggetti.
Per seguire la "sequenza" di chiamate bisogna "saltellare" da un file di classe a un altro.
È importante notare come il "**direttorio a menu**" sia **costantemente presente in ogni oggetto** che viene creato.
Questo aspetto si ripercuoterà molto sulla GUI che progetteremo nel prossimo capitolo; infatti non potendo "eliminare" i direttori (occorrerebbe modificare il codice di quasi tutto il sistema) si dovrà portarsi dietro questo menu, anche se non verrà fatto vedere all'utente.

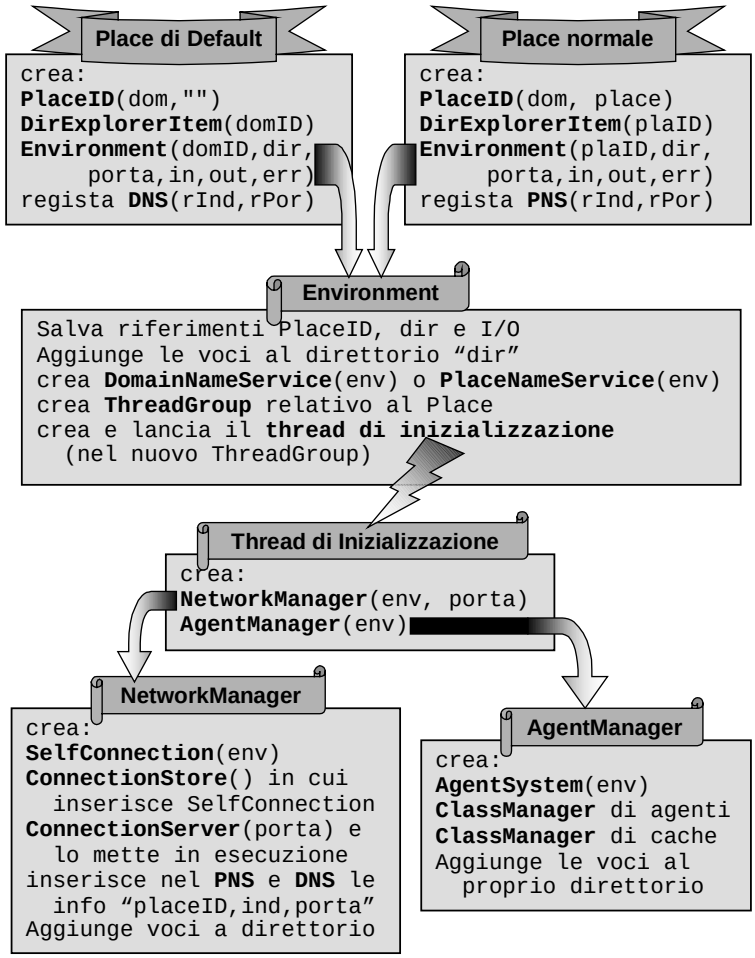


Figura 1.14 - sequenza di costruzioni degli oggetti di un Place.

1.13 PLACE MOBILI

Dalla versione 1.6 (col lavoro di **Errore. L'origine riferimento non è stata trovata.**) viene introdotto in SOMA il concetto di "**Place Mobile**" (in inglese, "Mobile Place").
A grandi linee, come il "Place" rappresenta l'astrazione di "nodo", così il "Place mobile" rappresenta quella di un **computer portatile**.

L’idea è quindi di creare un “Place mobile” all’interno di un computer portatile e di permetterne l’inserimento all’interno di un certo dominio. In ogni momento però l’utente può scegliere di “**disconnettere**” il Place, spegnere il computer, muoversi in un punto geograficamente anche lontano e lì collegare il computer all’interno di un’altra rete, riaccenderlo e far “**connettere**” il Place al di sotto di un altro dominio.
Il “**nome**” del Place mobile non verrà modificato dal fatto di essere sotto un particolare dominio, ma dipenderà solo dal nome del dominio che è stato usato al momento della sua creazione.
In questo modo il Place sarà sempre **identificabile univocamente** all’interno del sistema SOMA e quindi un agente mobile può essere opportunamente “condotto” dal sistema verso la nuova posizione fisica del Place.
Infatti, l’utilità di definire un “Place mobile” sta nel poter lanciare in esso un agente e, mentre questo sta migrando tra i vari Place del sistema, eseguire la “disconnessione”, spostarsi in un altro luogo geografico e “connettersi”, in modo da permettere al proprio agente di ritornare “a casa”.
In questo modo si può lanciare un operazione, lasciarla eseguire e **recuperarne i risultati in un secondo tempo**.
Quindi è il sistema sottostante che deve controllare se un agente mobile sta cercando di migrare su un Place mobile “disconnesso” e in tal caso “sospendere” l’agente fintantoché il Place non si riconnette.

1.13.1 La creazione di un “MobileEnvironment”

Il supporto alla mobilità dei Place è contenuto nel pacchetto “**SOMA.mobilePlace**”.
In esso viene definita la classe “**MobileEnvironment**” che non è altro che un estensione di “**Environment**” (cioè si eredita da essa).
Viene aggiunto un solo attributo:

```
public PlaceID currentDomainID;
```

In esso è contenuto il nome del dominio (sottoforma di “PlaceID”) a cui il Place mobile è attualmente “connesso”.
Il valore di default è “MobilePlaceManager.DISCONNECTED”, che indica che si è nello stato di disconnessione.
Il costruttore ha gli stessi parametri di quello della superclasse, però creerà differenti “servizi di nomi” rispetto a un Place “normale”.
Infatti, sebbene un Place mobile sia costruito alla stessa maniera di un Place “normale” (cioè ha bisogno di un proprio nome e di un dominio iniziale sotto il quale verrà creato) ha in più la tabella di DNS, poiché deve conoscere i dominii ai quali può chiedere di collegarsi.
Tale tabella viene “copiata” (al momento della creazione del Place) da quella contenuta nel dominio di iniziale d’appartenenza.
Inoltre, la tabella di PNS sarà di volta in volta ottenuta dal Place al quale ci si connette (infatti dovrà contenere l’elenco di tutti i Place che appartengono al dominio corrente, cioè alla specifica “località” in cui di volta in volta si trova il Place mobile).
Il “MobileEnvironment” mette a disposizione i metodi:

```
public void connect (PlaceID domainID)
public void disconnect ()
```

che servono appunto per eseguire la connessione a un dominio (datone il nome) e la disconnessione da quello attuale.

1.13.2 Il nuovo nome: un "MobilePlaceID"

Per distinguere i Place mobili da tutti gli altri è stata creata la classe "**MobilePlaceID**" che eredita da "**PlaceID**".
Il nome di un Place mobile è identico a quello di un Place "normale" (cioè è composto dai due soliti identificatori di "dominio" e di "place" al suo interno) ma preceduto da un altro identificatore: "**(M)**".
Tale prefisso viene aggiunto in modo automatico dalla classe "MobilePlaceID" e in questo modo lo si distingue anche visivamente dagli altri. Per es., se si crea il Place mobile "Mob" sotto il dominio iniziale "Dom", il suo nome risulterà essere "(M) Dom Mob".
Si noti che anche se si disconnette il Place e lo si riconnette sotto un altro dominio, il nome non viene alterato in nessun modo.

1.14 SICUREZZA

Come abbiamo già accennato nel paragrafo 1.2.3, la sicurezza di SOMA è attualmente in fase di progettazione.
Il problema della sicurezza è duplice e deriva dal "**mutuo sospetto**" tra i Place e gli agenti; infatti occorre proteggere:

- i **Place**, poiché gli agenti possono "attaccarli" e distruggere informazioni per loro vitali;
- gli **agenti**, che a loro volta possono essere attaccati da Place "malfidati" su cui si trovano a dover transitare; in particolare, da un agente non devono poter essere "estratte" le informazioni che trasporta né deve poter essere "alterato" o "rimpiazzato" con un agente "malfidato" (che finge di essere quello originale).

1.14.1 Protezione del Place

Per proteggere un Place da agenti "malfidati" occorre innanzitutto restringere il "campo di azione" di ciascun agente alle sole operazioni necessarie per lo svolgimento del proprio compito. Si deve associare a ogni agente un insieme di "**permessi**" per l'accesso alle risorse locali.
Inoltre, affinché un Place possa "fidarsi" di un agente è necessario che possa risalire al "creatore" (detto anche "**principal**") di quest'ultimo.

1.14.1.1 Politiche di sicurezza (i permessi degli agenti)

Per proteggere il Place da agenti "malfidati" si introduce il concetto di "**politica di sicurezza**" sfruttando le astrazioni di Place e di Place di Default (istanziarti non più da "Environment" ma dalla sua sottoclasse "**SecurityEnvironment**").

Sarà il Place di Default a stabilire la precisa "politica" da adottare in tutti i Place che fanno parte del suo dominio. Questi, a loro volta, potranno stabilire precise politiche di accesso alle loro risorse locali, ma sempre nel rispetto di quelle definite dal loro Place di Default (in pratica, andranno a fare l'intersezione tra la politica "di dominio" e quella da loro stabilita). Il Caricatore-di-Classi per il caricamento degli agenti si chiama "**AgentClassLoader**" e estende la classe "**SecureClassLoader**" (del pacchetto "java.security"). Infatti è lui che stabilisce i "**permessi**" (che fisicamente sono oggetti di classe "**Permission**") da dare o ogni agente in esecuzione sul Place.
Attualmente sono definiti in SOMA solo i seguenti "permessi":

- **PlaceAccessPermission**: prima che venga creato il "worker" per un agente, si verifica questo permesso, cioè si controlla che **all'agente sia permesso l'accesso al Place** su cui arriva. Questo permesso è controllato dal metodo "go()" di "AgentSystem" (ma non dall'omonimo metodo di "AgentWorker").
- **AgentPermission**: quando l'agente invoca il "getEnvironment()" di "AgentSystem", viene eseguito un controllo su questo permesso in modo da stabilire se all'agente può essere reso l'oggetto "Evironment" (che contiene tutte le strutture dati del Place).

I permessi assegnati a ogni agente si basano sull'identità di chi ne è il creatore (il "**principal**"), il quale è a sua volta identificato da un oggetto di classe "**CodeSource**", cioè dall'unione di un indirizzo URL e di alcune "chiavi crittografiche pubbliche" (usate per la verifica delle credenziali dell'agente).
L'**indirizzo URL** di un agente ha la seguente sintassi:

```
http://Dominio/Place/ClasseAgente/NumeroIdentif
```

L'AgentClassLoader, dopo aver caricato il codice dell'agente (da disco o attraverso la rete), tramite il "CodeSource" ne verifica le credenziali e, usando i "permessi" richiesti, crea (facendone l'intersezione con la politica definita dal Place) l'insieme dei permessi per l'agente (cioè il suo Dominio di Protezione) che provvederà a memorizzarsi internamente.

1.14.1.2 Autenticazione (identificazione del "principal")

È molto importante che dato un agente in esecuzione nel Place se ne possa risalire al "**principal**", inteso come l'utente (e non il Place) che l'ha creato. Infatti un agente esegue un lavoro per conto di qualcuno.
Si tratta quindi di un problema di "paternità" che può essere facilmente risolto usando un algoritmo di "**firma digitale**" (cioè usando uno degli algoritmi crittografici "asimmetrici"). Al momento della creazione dell'agente, il suo codice viene "firmato" usando la chiave privata dal suo "principal". Occorre però che ogni Place sia in grado di verificare tale "firma" e perciò deve avere (o poter ottenere da qualcuno) la chiave pubblica del "principal" che l'ha creato. A tale scopo è stata creata un'**infrastruttura per la gestione delle chiavi** (che vedremo).

1.14.2 Protezione dell'agente

Un Place può fare qualsiasi cosa voglia su un agente, perciò non è possibile impedirgli di alterare un agente. Non potendo fare altrimenti, ci si accontenta di una forma minore di protezione: tramite la "**firma digitale**" (descritta poc'anzi) posta sull'agente (fatta con la chiave privata del suo "principal") si può verificare l'**integrità** di un agente e di conseguenza riconoscere se ha subito manomissioni.
Un altro problema è la "**riservatezza**" delle informazioni che l'agente trasporta, cioè bisogna impedire che un Place "malfidato" (o un qualsiasi computer che intercetti il "flusso" di dati su cui viene trasferito l'agente durante una migrazione) possa leggere tali informazioni.
L'unica soluzione è fare una "**cifratura**" **dell'agente** prima che venga trasmesso. Quindi anche i Place avranno una loro coppia di chiavi (pubblica e privata) da usare per il trasferimento di agenti e dati.
Il Place sorgente cifrerà l'agente con la chiave pubblica del Place di destinazione (la può ottenere accedendo all'infrastruttura delle chiavi); una volta a destinazione il Place eseguirà la decifratura con sua chiave privata. In aggiunta, prima della cifratura, il codice viene "firmato" con la chiave privata del Place mittente, in modo che il destinatario possa essere sicuro che i dati arrivano effettivamente dal Place mittente. Si confronti il procedimento con lo schema riguardante la "firma digitale con doppia cifratura".

1.14.3 Infrastruttura per la gestione delle chiavi

In definitiva, SOMA usa degli algoritmi crittografici per proteggere sia il Place sia gli agenti e risolvere i problemi di riservatezza, integrità e paternità. Si tratta di algoritmi di "**crittografia forte**" e come tali non sono inclusi nelle classi di Java. Per usare la "sicurezza" in SOMA, occorre affiancare alle classi di Java quelle specifiche per la crittografia che sono contenute nei file:

- *iaik_jce.jar* distribuito da **Errore. L'origine riferimento non è stata trovata.**;
- *entrust.jar* e *entapplet.jar* reperibili da **Errore. L'origine riferimento non è stata trovata.**;
- *jndi.jar*, *ldap.jar*, *ldapbp.jar* e *providerutil.jar* che sono di Sun e possono essere scaricati da **Errore. L'origine riferimento non è stata trovata.**;
- *jaas.jar* (di Sun) scaricabili da **Errore. L'origine riferimento non è stata trovata.**;

Affinché gli algoritmi siano usabili, occorre che ogni Place abbia (o possa reperire da qualcuno) le "chiavi".
Se da un lato c'è bisogno delle chiavi "pubbliche" (per es., per la verifica delle firme digitali), dall'altro c'è bisogno anche delle chiavi "private" (sia dei Place sia dei Principal) poiché non si vuole vincolare l'utente a usare una "postazione" fissa.
Per la distribuzione delle chiavi pubbliche ogni Place si rifà a un "**Certification Authority**" (**CA**), la quale è un'infrastruttura che serve per mantenere i "certificati", ossia l'associazione tra ciascun utente (o Place) e la propria chiave pubblica.

Cap.2 - PROGETTO DELLA GUI PER SOMA

2.1 L'INTERFACCIA A LINEA DI COMANDO

Al tempo in cui mi fu assegnato l'incarico di progettare una GUI per il sistema ad agenti mobili SOMA l'unica interfaccia con l'utente era costituita da una "**linea di comando**".
L'interazione avveniva tramite una finestra a caratteri (in modo testo) andando a leggere e scrivere dai flussi di Input/Output standard.

2.1.1 I Menu e i Direttori

Poiché le funzionalità che si volevano fossero accessibili erano in gran numero (sebbene la maggior parte servissero solo allo scopo di controllo – in gergo si dice, "per debug") ne seguì l'idea di strutturarle in una **gerarchia di menu**: ogni voce o corrisponde a un'azione eseguibile o conduce a un sotto-menu (in maniera "ricorsiva").
Per "navigare" nel menu l'utente deve scrivere – con la tastiera – il nome della voce del menu (seguita da eventuali parametri).
Tutto ciò che viene scritto apparirà in una particolare linea dello schermo (preceduta dal simbolo identificativo di "maggiore").
La sintassi dei comandi che possono essere scritti dall'utente è molto "evoluta" e si basa sulla corrispondenza tra "menu" e "**direttorio**".
Ogni menu viene "visto" come un direttorio di un file-system e questo permette di invocare i comandi di un qualsiasi menu anche se al momento si è posizionati all'interno di un altro.
Per entrare in un sotto-menu basta usare il comando "**cd**" seguito dal nome del menu, mentre per tornare a quello precedente si usa "**cd ..**".
Il riferimento al menu principale – detto anche "radice" o "root" – è la barra "/" (in inglese, "slash").
Per esempio, per ottenere l'elenco dei thread che sono in esecuzione bisogna accedere al sotto-menu "util" (presente nel menu principale) e lì eseguire la voce di menu "threads". Allora si può scrivere la sequenza di comandi (che usano un riferimento relativo):

```
cd /
cd util
threads
```

oppure fare il tutto in un colpo solo (tramite un riferimento assoluto):

```
/ util threads
```

Inoltre, ogni volta che si cambia direttorio non ne viene visualizzato il contenuto automaticamente, ma per farlo basta semplicemente dare un "comando vuoto", cioè premere il tasto "invio" senza scrivere nulla.
Inoltre, per ogni nuovo Place creato viene inserita nel menu principale una voce di menu col suo stesso nome in cui sono presenti tutta una serie di funzionalità specifiche per il Place (non

viene fatta differenza tra Place normale e Place di Default, eccetto che il primo non avrà il comando relativo al DNS).

2.1.2 Difficoltà d'uso

Sebbene l'idea di strutturare il menu in direttori sia buona, è anche ovvio che non è semplice da usare per l'utente, soprattutto se questo non ha mai visto in vita sua una linea di comando (come ad es. il "prompt di MS-DOS") oppure se non è pratico a usare in modo "intensivo" la tastiera per scrivere i comandi.
È ovvio che è più "comodo" muovere il mouse su una voce di menu e premere il bottone piuttosto che scrivere l'intera sequenza di voci di menu via tastiera.
Senza contare che se ci si sbaglia a scrivere il comando, non è sempre possibile muovere il cursore sul punto errato o premere il tasto "cursore su" per riottenere il comando precedentemente impartito (sotto Windows ciò avverrà solo se l'utente ha caricato in memoria il programma "doskey"). Inoltre può accadere che l'output visualizzato ecceda le venticinque linee e quindi la sua parte iniziale non è più visibile per l'utente (cosa che accade immancabilmente sotto Windows 95, mentre in Windows NT un utente esperto può aumentare il "buffer" delle finestre DOS). Infine, la gerarchia di menu comprende molte voci per cui è possibile "perdersi", cioè occorre un certo tempo (e molta pratica) per abituarsi a com'è organizzata la gerarchia e di conseguenza a ritrovare rapidamente i comandi.
Siamo di fronte a un classico caso di **un'applicazione la cui diffusione viene enormemente limitata dalla sua interfaccia utente**.

2.2 LE FUNZIONALITÀ DA "FAR VEDERE"

Poiché nel menu a linea di comando sono già presenti tutte le funzionalità possibili, il primo passo da fare è di spulciarle una a una e depennare tutte quelle "inutili", cioè quelle che sono state inserite solo a scopo di "debug" e quelle che non servono assolutamente a un normale utente del sistema.

2.2.1 La struttura del Menu a Linea di Comando

Riporto nella Tabella 2.1 tutta la struttura a direttori in cui compare anche una voce relativa a un Place di Default creato.

Voce del Menu	Parametri & Significato
root	Il menu principale
security	Attiva il gestore della sicurezza
newDomain	`DomainName Port` Crea un nuovo dominio (un Place di Default) in base al nome e al numero di porta dati
newPlace	`DomainName PlaceName Port` Crea un nuovo Place all'interno di un certo dominio (dato) e con un nome e num.di porta

load	`FileTxt` Carica e eseuge un file di “script”
end	Termina la Macchina Virtuale Java
exit	Termina il thread di esplorazione dei manù (che si chiama “ExplorerThread”)
socket	Comunicazioni attraverso le socket
threads	Mostra tutti i thread
localHost	Mostra l’indirizzo dell’host locale
NewServer	`Port` Crea un nuovo server che accetta richieste
NewClient	`Host Port` Crea un client che comunica con un server posto sull’host e porta passati
util	Direttorio delle utilità di sistema
threads	Mostra tutti i thread
localHost	Mostra l’indirizzo dell’host locale
gc	Chiama il garbage-collector di Java
finalize	Invoca il metod o “finalize” di ogni oggetto che ha la finalizzazione “pendente”
freeMem	Mostra la quantità di memoria libera
totalMem	Mostra la quantità di memoria totale
usedMem	Mostra la quantità di memoria usata
prop	`{PropertyKey {PropertyValue}}` imposta una proprietà: chiave e valore
impl	`A B` Mostra l’implicazione dei permessi (cioè se A implica B); è usato per la sicurezza
refresh	Rinfresca (ricarica) la configurazione delle politiche di sicurezza
policy	Mostra il Dominio di Protezione per SOMA
<Place>	Direttorio di un Place (normale o di Default); ha lo stesso nome del Place
env	Mostra la struttura dati “Environment”

window	`{start | stop | status}` apre/chiude la finestra del Place (l’unica finestra grafica prevista in SOMA 1.5)

dns	`{list | put Place Host Port |`
	`del Place | register Host Port |`
	`refresh {Host Port}}`

	Esegue delle operazioni sul Domain Name Service (DNS) del Place (solo Place di Default).
pns	`{list | put Place Host Port | del Place | register Host Port | refresh {Host Port}}` Esegue delle operazioni sul Place Name Service (PNS) del Place.
launch	`[-s] [-nt] [-ns] "AgentID" {"Arg1" "Arg2"...}` Lancia un agente (la cui classe è specificata nella stringa passata) coi suoi argomenti; in più ci sono dei parametri che specificano il tipo lancio. -s = usa il caricatore di classi di sistema; -nt = agente non rintracciabile; -ns = crea ma non lancia l'agente
threads	Mostra tutti i thread appartenenti al gruppo relativo al Place (cioè al suo "threadgroup")
text	Stampa del testo sui tre possibili flussi di output (serve solo per scopi di debug)
<Place> / agentManager	Accesso all'AgentManager di un Place
path	`NewAgentsPath` Mostra/definisce la nuova directory su disco in cui risiedono gli agenti eseguibili
cache	`NewCachePath` Mostra/definisce la nuova directory su disco in cui mantenere la cache degli agenti
store	Mostra lo stato dell'oggetto "sharedObjects", cioè il contenitore degli oggetti condivisi
list	Mostra tutti i "worker" degli agenti in esecuzione
a	`"AgentID" {start | stop | status | dom | remove | kill | go "PlaceID"}` Comando per le azioni su uno specifico agente. Lo blocca e fa ripartire, ne mostra lo stato, lo rimuove e lo fa perfino "migrare" in un altro Place.
pos	Mostra la posizone corrente (il Place) di ogni agente che è stato creato dal Place
death	`"AgentID"` Notifica la morte di un agente al suo Place di creazione.
launch	`[-s] [-nt] [-ns]`

	`"AgentID" {"Arg1" "Arg2"...}` Lancia un agente (la cui classe è specificata nella stringa passata) coi suoi argomenti; in più ci sono dei parametri che specificano il tipo lancio. -s = usa il caricatore di classi di sistema; -nt = agente non rintracciabile; -ns = crea ma non lancia l'agente
<Place> / netManager	Accesso al NetworkManager di un Place
placeInfo	Mostra le informazioni sul Place (indirizzo di IP e numero di porta del proprio "demone")
server	`{start | stop | status}` Blocca e fa ripartire il server (il "demone") del Place.
connList	Mostra la l'elenco delle connessioni
send	`PlaceID Message` Invia un messaggio da un Place a un altro Place (per il debug dei comandi tra Place)
perm	`["PlaceID" [start | stop]]` Creazione di connessioni permanenti tra il Place e un altro che è esterno al dominio.

<Place> / netManager / connections	Accesso alle Connessioni tra il Place e un altro Place

SelfConnection	`{start | stop | status |`
	`send Message}`
	Connessione con se stesso

ServConnX	`{start | stop | status |`
	`send Message}`
	Connessione con un altro Place del dominio

Tabella 2.1 - struttura del menu a linea di comando.

2.2.2 Le Funzionalità "rimaste"

Dopo aver depennato dai menu tutte le voci "inutili", abbiamo raggruppato (in Tabella 2.2) le rimanenti in nuove categorie.

Funzionalità	Voce di Menu
Funzionalità del Sistema	newDomain, newPlace, exit
Funzionalità del Place	window, dns (solo per Place di Default),

	pns, launch, a (gestione agenti), pos, path, cache
Informazioni sul Sistema	threads, localHost, gc, *Xxx*Mem (free, total & used)
Funzionalità di Agente	go (migrazione), send (invia messaggio a agente)

Tabella 2.2 - funzionalità "utili" di SOMA

2.3 DIAGRAMMA DI FLUSSO D'INTERFACCIA

2.3.1 Interfaccia per un sistema multi-processo

Innanzitutto occorre notare che il sistema ha di per sé una natura **multi-processo**, cioè è possibile che in ogni momento ci possa essere più di un'entità in esecuzione. Infatti, non c'è l'obbligo che su ogni computer venga creato al massimo un Place, anzi è possibile addirittura una marea. Questo aspetto di "**parallelismo**" dovrà ritrovarsi anche nell'interfaccia di controllo del sistema.
Non si dovrà costruire un diagramma di flusso di tipo "sequenziale", dove cioè il passaggio da una finestra all'altra avviene con la distruzione (o la scomparsa dal video) di quella precedente in modo tale che sul video sia mostrata sempre e solo una finestra.
Occorre fare in modo che ogni finestra abbia una "vita propria" e che sia l'utente il solo (per lo meno nella maggior parte dei casi) a decidere quando aprirla e quando chiuderla.

2.3.2 Le funzionalità di base

Guardando la Tabella 2.2 ci si accorge che le azioni che l'utente può fare sono di due tipi:

- **azioni sul Sistema**: creare un Place, visualizzare informazioni;
- **azioni sui Place**: lanciare gli agenti, modificare la configurazione, manipolare gli agenti, visualizzare informazioni;

Di sicuro – per avere un'interfaccia "parallela" – occorrerà prevedere **una finestra particolare per ogni possibile Place**.
In più serviranno alcune **finestre "di sistema"** per la creazione dei Place (e l'apertura delle relative finestre).
Il caso "tipico" però prevede che sia creato un solo Place per ogni macchina, quindi si deve prendere in considerazione quest'eventualità.

2.3.3 Il Diagramma di Flusso

In Figura 2.1 riporto lo schema del diagramma di flusso d'interfaccia.

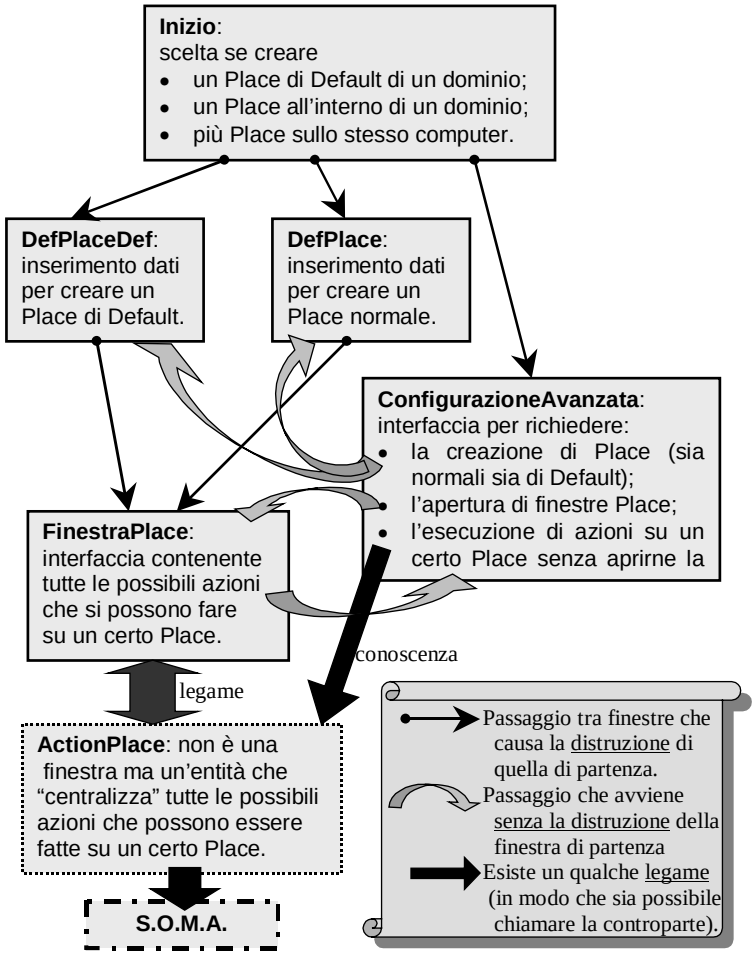


Figura 2.1 - il diagramma di flusso d'interfaccia

Come si può vedere si inizia sempre dalla schermata "**Inizio**", che chiede all'utente cosa vuole fare, se creare un singolo Place (normale o di Default) o accedere alla finestra "**ConfigurazioneAvanzata**" nella quale è possibile creare tutti i Place che si vuole, aprire le finestre dei Place (nel caso non fossero già aperte) e anche eseguire azioni (per es., il lancio di un agente) su un particolare Place (scelto al momento) senza dover aprirne la finestra.
Si noti la differenza dei tipi di frecce (che, tra l'altro, non rispecchia un particolare standard poiché si tratta di una mia simbologia).
Con essa ho voluto evidenziare la differenza tra le "navigazioni" che causano o meno la distruzione della finestra di origine.
Partendo da "Inizio", tutte le direzioni comportano una distruzione di tale finestra. Infatti essa assolve lo scopo di scelta se creare uno (e un solo) Place oppure più di uno, perciò una volta che quest'uno è stato creato (con "**DefPlaceDef**" o "**DefPlace**") non ha più senso ritornare a "Inizio".
D'altra parte, se si sceglie di andare in "ConfigurazioneAvanzata", si può ritornare in "DefPlaceDef" o "DefPlace" senza che la prima finestra venga distrutta.

Si noti che le finestre “DefPlaceDef” e “DefPlace” verranno in ogni caso distrutte nel momento in cui si deve creare il Place (cioè quando avranno finito di assolvere la loro funzione: l’inserimento dei dati necessari per creare un Place).
La finestra “**FinestraPlace**” costituisce un’interfaccia che permette all’utente di eseguire le funzionalità dei Place specificatamente su un particolare Place.
Ogni Place avrà una sua “**FinestraPlace**” associata, ma non è detto che questa sia sempre mostrata a video (si deve prevedere che possa essere “chiusa” quando non serve e “riaperta” successivamente, senza che venga alterato il funzionamento del Place).
Ovviamente non si può pensare che una volta creato un **unico** Place nel sistema non si voglia (in futuro) aggiungerne un altro.
Poiché siamo nel caso di un Place unico, secondo il diagramma l’unica finestra aperta è quella del Place; ecco allora che occorre prevedere che da tale finestra sia possibile **richiamare** quella di “ConfigurazioneAvanzata” (visto che da lì è possibile fare ogni cosa).
Nel diagramma è presente anche un oggetto chiamato “**ActionPlace**” che **non è una finestra**. Come si vedrà meglio più avanti, tale oggetto è introdotto con lo scopo di “centralizzare” tutte le azioni che possono essere fatte su un Place (e quindi è lui che va a invocare i metodi degli oggetti di SOMA).
Nel prossimo capitolo (dedicato alla costruzione dell’interfaccia) vedremo che esisterà una stretta relazione tra gli oggetti “**FinestraPlace**” e “**ActionPlace**”.
In ultima analisi, affinché da “ConfigurazioneAvanzata” sia possibile eseguire “**azioni**” su un particolare Place **senza andare a aprirne la finestra** occorre che:

- sia possibile conoscere l’**elenco di tutti i Place** creati;
- si riesca a rintracciare l’oggetto “**ActionPlace**” associato al Place.

Possiamo già intuire una cosa (che vedremo nel paragrafo 2.5.4): perché ogni volta che si lancia il sistema bisogna creare manualmente i Place? E se volessi “riusare” le impostazioni che erano state fatte nella precedente esecuzione? Occorrerà prevedere la possibilità di **salvare su disco e poi ricaricare le “definizioni” dei Place** (cioè tutti i dati che vengono inseriti nelle finestre “DefPlaceDef” e “DefPlace”).

2.4 BOZZETTI DELLE SCHERMATE

Esaminiamo adesso – una a una - tutte le finestre (con le loro funzionalità) che sono state mostrate nella precedente Figura 2.1.
Vedremo anche altre finestre che saranno necessarie alle prime (per inserimento e/o visualizzazione dati).
Per ciascuna finestra verrà fatto un “bozzetto” che schematizza come si vorrà che appaia a video.

2.4.1 Bozzetto di “Inizio”

Con questa finestra l’utente può scegliere solo una di queste opzioni:

- creazione di **un solo**:
  - **Place di Default**, cioè il Place che gestirà un nuovo dominio;

- **Place** all'interno di un Place di Default (che dovrà essere già stato creato su di un altro computer);

- creazione di **più Place sullo stesso computer** (cioè di una "configurazione avanzata", poiché il funzionamento "normale" del sistema prevede la creazione di un solo Place su un singolo computer);
- **uscita** dal sistema.

In Figura 2.2 ho riportato il bozzetto della finestra "Inizio".

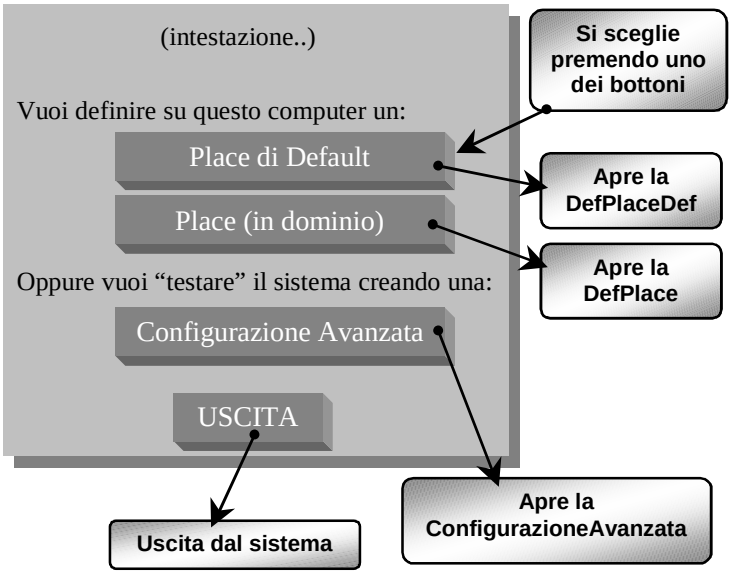


Figura 2.2 - bozzetto di Inizio

2.4.2 Bozzetto di "DefPlaceDef"

Lo scopo di questa finestra è di permettere all'utente di inserire i dati che servono per la "**Def**inizione di un **Place** di **Def**ault".

Poiché viene usata sia da "Inizio" sia da "ConfigurazioneAvanzata", occorre prevedere diverse possibilità di registrare il DNS:

- **nessuna registrazione**: il Place di Default sarà la "radice" della gerarchia dei DNS.
- registrazione presso il DNS di un Place di Default creato "**localmente**", cioè sullo stesso computer (il nome e i dati del Place di Default genitore dovranno essere ottenuti in modo automatico dalla procedura di registrazione una volta che l'utente specifica soltanto il **nome del genitore**).
- registrazione presso il DNS di un Place di Default "**remoto**", cioè che risiede su un altro computer; per riferirsi a un singolo Place occorrerà che l'utente specifichi:
  - l'indirizzo IP (logico o numerico) del nodo remoto;
  - il numero di porta associato al "demone" del Place remoto.

In sostanza viene fatto quello che nel menu a linea di comando corrisponde alla sequenza di comandi:

```
newDomain NomeDominio NumeroPorta
NomeDominio dns register RegIndIP RegNumPorta
```

Di cui il secondo non viene fatto nel caso del DNS “radice”.
Inoltre l’utente potrà volere o meno che venga aperta la finestra del Place (“FinestraPlace”). Ovviamente la finestra dovrà essere aperta in ogni caso se ci si trova in fase di creazione di un Place “unico” (poiché non rimarrebbe più alcuna finestra aperta con la quale l’utente può interagire).
Perciò, a seconda che a questa finestra ci si arrivi da “Inizio” o da “ConfigurazioneAvanzata”, si permetterà la modifica soltanto di alcuni dati (gli altri verranno **disabilitati ma non eliminati**).
Il bozzetto di questa finestra è mostrata in Figura 2.3.

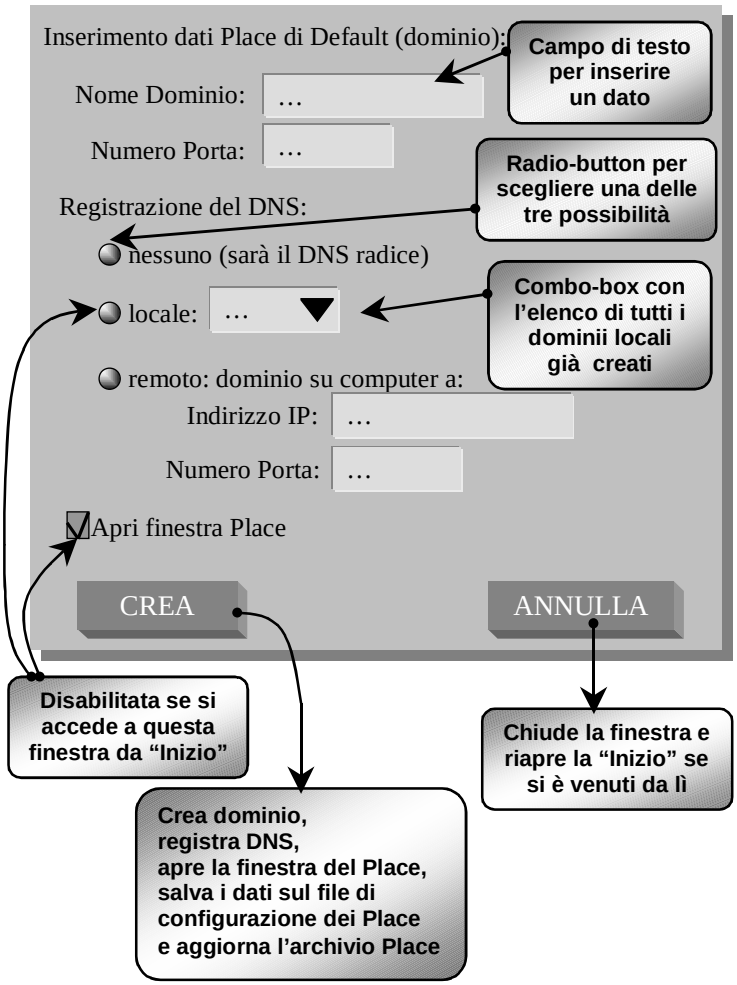


Figura 2.3 - bozzetto di DefPlaceDef

2.4.3 Bozzetto di “DefPlace”
Questa finestra è molto simile alla precedente, ma i suoi dati servono per la “**Def**inizione di un **Place**” all’interno di un dominio già creato.

Anche qui c'è una registrazione da fare, ma si tratta del PNS (cioè occorre registrarsi presso il Place di Default del dominio di appartenenza in modo da farsi dare – e farsi mantenere aggiornata – la tabella del servizio di nomi di Place).
La registrazione qui può essere solo di due tipi:

- registrazione presso un Place di Default creato "**localmente**", cioè sullo stesso computer (il nome e i dati del Place di Default saranno ottenuti attraverso il **nome** che viene dato dall'utente);
- registrazione presso un Place di Default "**remoto**", cioè che risiede su un altro computer; per ritrovare il "demone" associato al Place remoto occorrerà che l'utente specifichi:
  - l'indirizzo IP (logico o numerico) del nodo remoto;
  - il numero di porta associato al "demone" del Place remoto.

  Inoltre bisogna inserire anche:
  - il **nome del dominio** a cui si vuole appartenere.

Quello che viene fatto equivale alla seguente sequenza di comandi tramite il menu a linea di comando:

```
newPlace NomeDominio NomePlace NumeroPorta
NomePlace pns register RegIndIP RegNumPorta
```

In questo caso **la registrazione deve sempre essere fatta**.
In Figura 2.4 ho riportato il bozzetto di questa finestra.

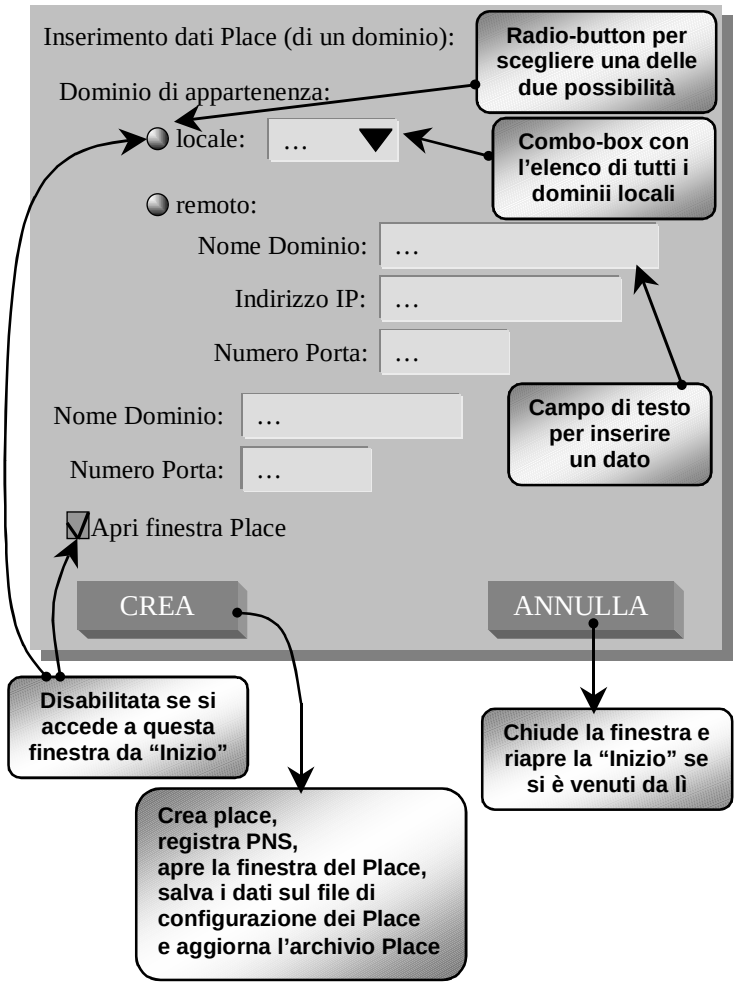


Figura 2.4 - bozzetto di DefPlace

2.4.4 Bozzetto di "AdvConfig" (ConfigurazAvanzata)

Tramite questa finestra dev'essere possibile:

- ✓ Creare nuovi Place (di Default o normali) all'interno dello stesso computer (e quindi per la "registrazione" si userà prevalentemente il campo "locale" delle finestre viste in precedenza);
- ✓ Fare delle azioni (quelle più "utili") su un certo Place (scelto dall'utente di volta in volta) che è stato creato all'interno del computer locale.

Per la prima funzionalità non c'è molto di nuovo: si richiamano le finestre "DefPlaceDef" e "DefPlace" senza distruggere la finestra corrente.

L'altra funzionalità pone invece qualche problema.

Per far scegliere all'utente uno dei Place locali occorre averli messi precedentemente tutti in una lista. Il sistema infatti non prevede la "ricerca" dei Place presenti nel computer locale.

Occorre predisporre un elenco di Place (che va aggiornato ogni volta che se ne crea uno nuovo) da mantenere come un'entità a uso e consumo della sola interfaccia grafica.

Poiché il sistema non tiene traccia dell'elenco dei Place locali, non bisogna che sia accessibile a nessuno fuorché all'interfaccia.
Un altro problema è come fare a eseguire le azioni del Place anche quando la sua finestra è chiusa.
Come vedremo nel prossimo capitolo, l'introduzione dell'oggetto "**ActionPlace**" permette la "centralizzazione" (e quindi la gestione) delle più comuni azioni che possono essere fatte dall'utente su un Place. In questo modo, basta essere in grado di ottenere tale oggetto – una volta noto il solo nome di un Place – per poter eseguire qualsiasi azione sul Place.
Come ho accennato nel paragrafo 1.6, il riferimento a tale oggetto è stato inserito nell'Environment del Place.
In tal modo può essere acceduto anche dagli agenti, a patto che abbiano i "permessi di accesso" all'oggetto Environment.
Per la selezione del Place su cui eseguire le azioni ho pensato di non usare una semplice lista ma permettere differenti "visioni" di questa, in modo da ottenerne una "strutturazione" che faccia capire il funzionamento del sistema.
Alla fine penso che le sole "viste" possibili siano:

- ✓ **albero gerarchico di appartenenza al dominio**: è un albero gerarchico che contiene al primo livello tutti e solo i Place di Default; da ognuno di questi discendono (al secondo livello) tutti i Place che sono contenuti nel dominio che lui rappresenta.
- ✓ **albero gerarchico dei DNS**: è un albero gerarchico a più livelli; la radice è costituita dal Place di Default il cui DNS è la "radice" della gerarchia dei DNS; i figli della radice sono i Place di Default il cui DNS sono "figli" del DNS radice; e così via per tutti gli altri.
- ✓ **elenco dei soli Place di Default**: contiene la lista ordinata di tutti e soli i Place di Default locali.
- ✓ **elenco di tutti i Place**: contiene la lista ordinata di tutti i Place locali (sia di Default sia normali).

Sulla finestra sarà mostrata solo una di queste "viste" (anche per ragioni di spazio) ma dovrà essere possibile passare da una all'altra.
La selezione del Place da usare avverrà nel modo più semplice possibile: si muove il puntatore del mouse sul nome del Place contenuto nella lista e si preme il tasto del mouse.
Una volta selezionato un Place si potranno fare le seguenti azioni:

- visualizzare/modificare la tabella di DNS (se c'è) e di PNS;
- lanciare un agente (si aprirà una finestra che vedremo più avanti);
- aprire la "FinestraPlace" (se non è già stata aperta).

Inoltre occorre prevedere un "bottone" per la chiusura della finestra.
Nota: per semplicità di scrittura questa finestra verrà poi ribattezzata col nome "**AdvConfig**".
Ecco allora in Figura 2.5 il primo bozzetto della finestra.

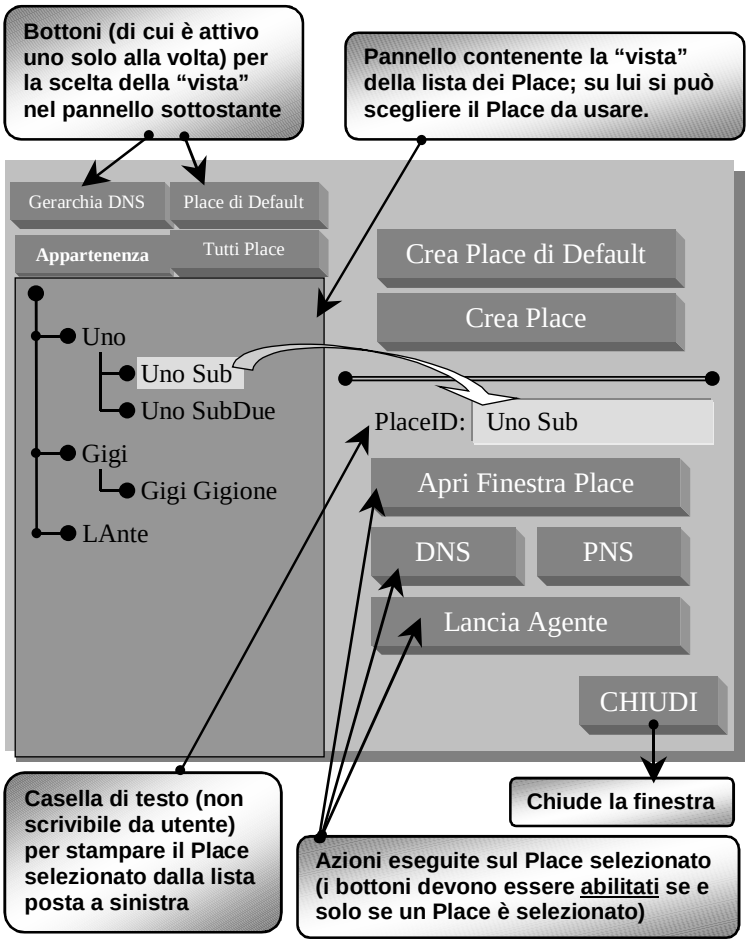


Figura 2.5 - bozzetto di ConfigurazioneAvanzata

2.4.5 L'oggetto Anagrafe

Al fine di "ricordarsi" tutti i Place che sono stati creati "localmente" (cioè sul computer che si sta usando) si è deciso di creare un oggetto che mantenga internamente una **lista di tutti gli** "**Environment**".

Tale oggetto si chiama "**Anagrafe**" ma **non** è "statico" perché si vuole (per ragioni di sicurezza) che **non sia accessibile da nessun altro** all'infuori delle classi della GUI. Di tale classe ne verrà creata una sola istanza all'avvio del sistema e l'unico riferimento a essa sarà mantenuto all'interno di "**AdvConfig**", in una variabile "**privata**".

Ogni volta che un nuovo Place viene creato tramite "Creatore", quest'ultimo provvederà a "battezzare" il Place sull'Anagrafe (la quale gli viene passata come parametro di creazione).

L'oggetto "Anagrafe" verrà usato per reperire i dati dei Place locali in modo che l'utente non debba inserirli ogni volta.

In questo modo si facilitano le azioni locali sul sistema. Per esempio, quando si vuole registrare un Place in sotto un Place di Default, basterà mostrare all'utente la lista di tutti quelli di Default creati localmente (ottenibile con "Anagrafe") e fargliene scegliere uno; una volta ottenuto dall'utente il PlaceID si possono ottenere i dati del Place di Default (per es., il numero di porta) sempre da "Anagrafe".

2.4.6 Bozzetto di “FinestraPlace”

Questa è la finestra di un Place perciò da qui devono poter essere eseguite tutte le possibili azioni sullo specifico Place.
Poiché le funzionalità sono molte ho pensato di metterle come voci di un **menu a cascata** posto nella parte alta della finestra.
Tramite esso l’utente potrà eseguire le azioni sul Place.
Inoltre ho “tirato fuori” le azioni che – si presume – vengono fatte molto di frequente e le ho riportate sottoforma di “bottoni” (posti nella parte bassa della finestra).
Ho inoltre aggiunto un’area di testo nella parte centrale della finestra e, sotto di lei, una linea di testo; la prima serve per l’output mentre la seconda per l’eventuale (ma improbabile, come vedremo poi) input.
In Figura 2.6 ho riportato il bozzetto della “FinestraPlace”.

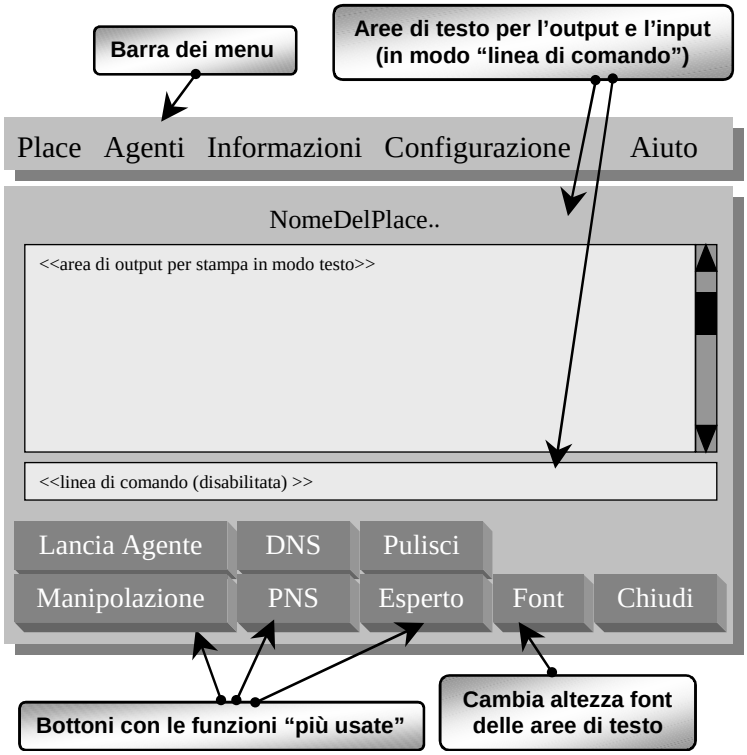


Figura 2.6 - bozzetto di FinestraPlace.

Come ho già detto, già la versione 1.5 di SOMA rendeva disponibile una “finestra di Place”; però (escludendo la barra di menu) tutta l’interazione con l’utente avveniva tramite una linea di comando (per eseguire le azioni del solito “menu a linea di comando”).
Inoltre sappiamo che ogni Place ha un suo flusso di Input/Output (si veda l’oggetto Environment al paragrafo 1.6).
Per questi motivi la “vecchia” finestra di Place aveva un’area di testo (per l’output) e una linea di testo (per l’input utente).

Per default i flussi di I/O sono diretti sui flussi di I/O standard, ma quando si apre la finestra vengono ridiretti sui componenti di testo grafici appena descritti.
Nel mio progetto **avrei voluto evitare l'uso di queste due aree di testo**. Infatti lo scopo principale dell'interfaccia grafica è di **fare in modo che l'utente usi la tastiera il meno possibile**.
Infatti seguirò questa regola e l'unica interazione tra utente e tastiera avverrà solo per introdurre dei dati che non sono ancora noti (per es., per scrivere il nome di un **nuovo** Place, ma **mai** per la scelta di un nome di un Place già esistente o noto al sistema).
Con il "bottone" chiamato "Font" si potrà scegliere l'altezza del font usato nelle aree di testo, in modo da adattare la dimensione del testo stampato all'attuale risoluzione del video.
Come ho già anticipato, avrei voluto "disfarmi" del "menu a linea di comando" ma **non** ho potuto perché la sua esistenza è legata in modo molto stretto al codice di SOMA:
Basta riguardare la Figura 1.14 in cui viene descritta la sequenza di creazione degli oggetti in seguito alla creazione di un "Environment" per vedere che la dicitura "**aggiunge le voci al direttorio**" è presente quasi ovunque.
Poiché ho dovuto portarmi dietro il "menu a linea di comando" ho pensato di mantenermi "compatibile col vecchio" e di conseguenza nella mia "finestra di Place" ci sono ancora le due aree di testo.
Soltanto che, per default, la linea di comando è disabilitata (ma può essere abilitata eseguendo una certa azione del menu a cascata).
In questo modo tendo a far usare la sola interfaccia grafica.
In oltre, ogni azione del menu andrà a aprire (se necessario) un ulteriore finestra tramite la quale mostrare o richiedere dati all'utente.
Ogni finestra vivrà di vita propria, cioè non sarà necessario distruggerla per poter tornare alla precedente.
Questo però non è sempre vero: non ha senso tenere aperta una finestra di inserimento dati. Ecco allora che si è pensato di "**disabilitare**" la finestra precedente (chiamata "genitore") fintantoché la finestra "figlia" non viene distrutta (chiudendola o eseguendo l'operazione per la quale era stata creata).
Ma torniamo alle voci del **menu a cascata**.
Ogni funzionalità è stata raggruppata in una voce di menu generale.
Ecco tutte le funzionalità che mi aspetto che siano realizzate:

- **Place**
  - Visualizzare l'identificatore del Place (il PlaceID)
  - Aprire la "**FinestraDNS**" relativa al Domain Name Service (DNS) del Place (ma solo se il Place è un Place di Default)
  - Aprire la "**FinestraPNS**" relativa al Place Name Service (PNS) del Place
  - Abititare/disabilitare la comunicazione con le **Applet** remote
  - Visualizzare l'elenco dei thread appartenenti allo stesso gruppo di thread (ThreadGroup) del Place
  - Cancellare l'area di testo per l'output
  - Chiudere questa finestra di Place
  - Uscire da SOMA (chiude tutto e esce)
- **Agenti**

- Aprire la finestra "**LancioAgente**" con la quale si può scegliere e lanciare un nuovo agente
- Aprire la finestra di "**ManipolaAgenti**"
- Aprire la finestra di "**PosizioneMieiAgenti**"

☐ **Informazioni**
- Visualizzare l'indirizzo IP del computer locale
- Mostrare la quantità di memoria (totale/libera/usata)
- Chiamare il Garbage-Collector di Java
- Visualizzare l'elenco di tutti i thread presenti nella Macchina Virtuale Java.
- Elencare le "proprietà" del sistema

☐ **Configurazione**
- Modificare la lingua usata dall'interfaccia grafica
- Cambiare l'aspetto grafico (che chiamerò – di mia iniziativa – "apparenza") della GUI
- Modificare le opzioni di configurazione della GUI
- Definire il percorso del file-system locale da cui vengono caricati le classi degli agenti
- Definire il percorso della cache degli agenti
- Cancellare il contenuto della cache degli agenti (questa funzione non c'è in SOMA, l'ho aggiunta io)
- Passare da modalità "Base" a "Esperto" e viceversa: ossia abilitare o disabilitare la "linea di comando" (e lanciare il thread chiamato "ExplorerThread")
- Aprire la finestra di "**ConfigurazioneAvanzata**" (nel caso sia stata chiusa precedentemente può essere utile riaprirla per poter creare nuovi Place)

☐ **Aiuto**
- Apre la finestra di informazioni su SOMA

Come si può intuire da quanto detto prima, ogni volta che nel menu c'è un "visualizzare qualcosa" significa che bisogna aprire una finestra dedicata alla visualizzazione di un particolare dato.
Nulla verrà stampato sull'area di testo (a eccezione dei messaggi interni di sistema che – chissà perché – qualcuno ha scelto di stamparli a video).

2.4.7 L'oggetto "ActionPlace"

L'oggetto "**ActionPlace**" rappresenta l'esecutore di tutte quelle azioni che possono essere invocate da un oggetto "**FinestraPlace**" Quest'ultima **non sa** "**come**" fare le azioni che mette a disposizione, **ma sa** "**chi**" è incaricato di farle, cioè conosce il suo oggetto "ActionPlace". Questo definisce (sottoforma di **metodi**) tutte le azioni eseguibili su un determinato Place.
Infatti questi due oggetti lavorano in maniera molto stretta tra loro e realizzano una sorta di "**Model-View-Controller**": il modello è costituito da "ActionPlace" mentre vista e controllore sono contenuti in "FinestraPlace" (che quindi assomiglia a un "delegate").
Per questo motivo **ognuna** delle due controparti **conterrà in una variabile un riferimento all'altra**, come mostrato in Figura 2.7.

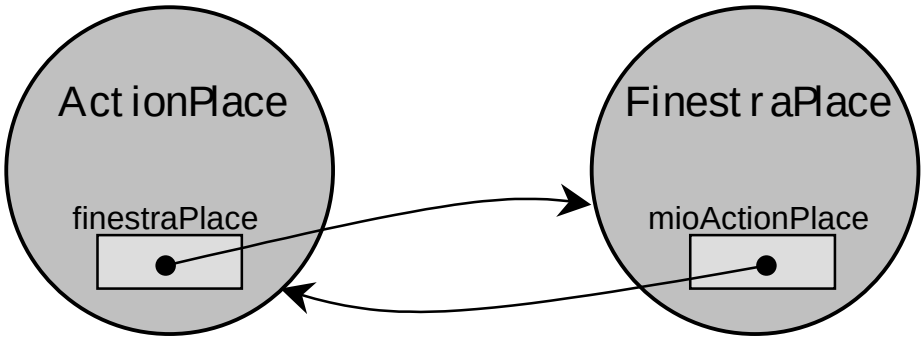


Figura 2.7 - duplice riferimento tra ActionPlace e FinestraPlace

Infatti la finestra è una “vista” con la quale l’utente può richiedere le azioni a un Place. È vero che l’utente non ha bisogno di più “viste” per lo stesso Place, ma più avanti vedremo che la “**centralizzazione**” delle azioni in un solo posto risulterà molto comoda (ciò si verificherà quando permetteremo la comunicazione tra il Place e delle Applet remote ma soprattutto quando inseriremo la “sicurezza”).
Poiché ogni Place avrà un suo “ActionPlace”, si è pensato di inserirne il riferimento all’interno della classe “**Environment**” (che, come già detto nel paragrafo 1.6, assolve proprio il compito di contenere tutti gli oggetti che costituiscono un Place).
D’altra parte, anche “ActionPlace” dovrà mantenere un riferimento all’oggetto “Environment” che l’ha creato, poiché è a lui che si deve rivolgere per fare effettivamente le azioni sul Place.
Ecco allora la “catena” di riferimenti riportata in Figura 2.8.

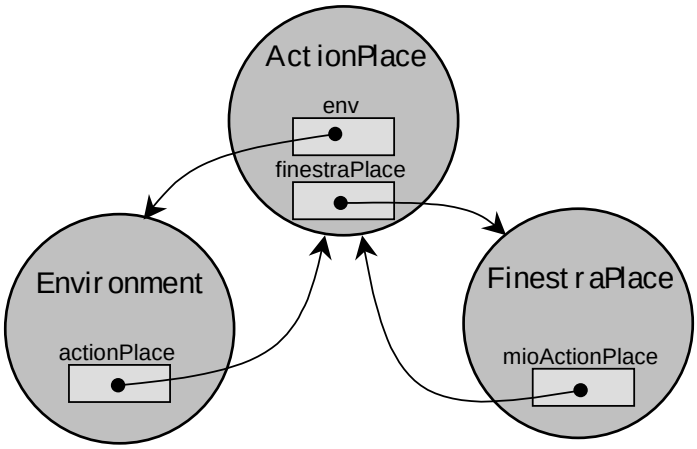


Figura 2.8 - riferimenti tra Environment-ActionPlace-FinestraPlace.

Poiché “FinestraPlace” può essere o meno mostrata a video, sarà compito di “ActionPlace” tenere traccia dello stato della propria finestra e mettere a disposizione dei metodi per aprirla o chiuderla.
Quindi il secondo compito di “ActionPlace” è di gestire l’apertura della “FinestraPlace” in modo che ne venga creata **al più una sola istanza** relativa a uno stesso Place. Si noti che non viene mai distrutto l’oggetto “ActionPlace” ma solo la sua finestra;

Per realizzare le funzionalità che mette a disposizione, “ActionPlace” andrà a “usare” l’oggetto “Environment” a cui si riferisce.
Al fine di ottenere delle buone prestazioni, si è scelto di **non andare a richiamare le funzionalità del menu a linea di comando ma di andare a agire direttamente sugli oggetti contenuti in “Environment”** (che sono definiti “pubblici”) **invocandone i metodi.**
In questo modo si “scavalca” la gestione del menu a linea di comando e ci si “svincola” dallo specifico “Environment” poiché la GUI potrà usare solo i metodi di “ActionPlace” e non di “Environment”.
In fin dei conti, “ActionPlace” rifarà (in modo quasi identico) quello che viene fatto dai vari “*Xxx*ExplorerItem” sparsi per i direttori di SOMA.
Tra le funzionalità importanti ci saranno quelle per la richiesta dei seguenti dati:

- l’Environment di riferimento;
- il PlaceID relativo a tale Environment;
- l’indirizzo IP locale;
- le quantità di memoria (totale, usata e libera);
- l’elenco delle proprietà del sistema;
- l’albero con l’elenco di tutti i thread della JVM;
- come la precedente, ma coi soli thread relativi al Place;
- l’elenco dei PlaceID contenuti nel PNS del Place;
- simile alla precedente, ma preleva i dati dal DNS (se esiste);
- il percorso da cui vengono caricati gli agenti;
- il percorso per la “cache” degli agenti;
- l’elenco degli agenti che possono essere messi in esecuzione (cioè quelli che sono contenuti nel direttorio degli agenti).

Le azioni eseguibili saranno:

- lancio di un agente;
- definizione dei direttori di caricamento agenti e di “cache”
- invocazione del “garbage-collector” della JVM.

Soltanto l’azione di “lancio agente” non userà direttamente l’Environment ma delegherà il lancio a un oggetto particolare della GUI chiamato “**Creatore**” (che vedremo nel paragrafo 2.4.9) poiché tale oggetto a sua volta servirà da punto di “centralizzazione” per il lancio di agenti (oltre a centralizzare, come ne suggerisce il nome, le azioni di creazione di nuovi Place).

2.4.8 Bozzetto di “LancioAgente”

Tramite questa finestra (il cui bozzetto è riportato in Figura 2.9) l’utente può scegliere l’agente da mettere in esecuzione nel Place.

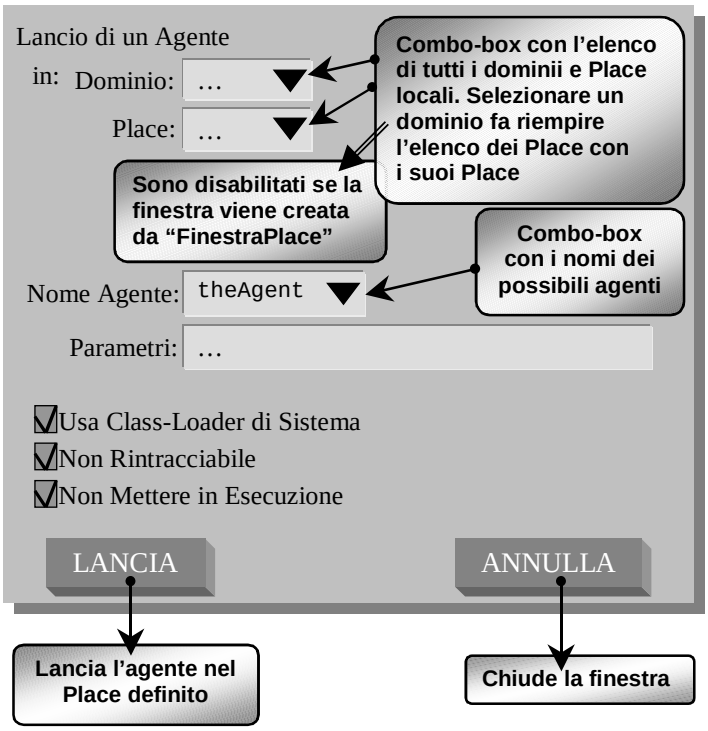


Figura 2.9 - bozzetto di LancioAgente

Poiché gli agenti "lanciabili" sono tutti e solo quelli presenti in un direttorio del file-system locale (in genere è chiamato "agents" ma può essere ridefinito da ciascun Place) è inutile far scrivere all'utente il nome dell'agente da lanciare: basta mostrargli l'**elenco degli agenti** presenti nel direttorio e fargliene scegliere uno (così si evitano anche gli eventuali errori di scrittura del nome).
Inoltre bisogna prevedere una "linea" di testo in cui l'utente può scrivere i parametri da passare all'agente (come se venisse lanciato da linea di comando).
Dato che al metodo di creazione di un agente in SOMA possono essere passati alcuni parametri "particolari", è bene che possano essere inseriti tramite questa finestra. Ecco allora che servono tre opzioni di tipo "booleano" (cioè coi soli due valori: vero o falso):

- Usare il caricatore di classi di sistema;
- Creare un agente "non rintracciabile";
- Creare l'agente ma non metterlo in esecuzione.

Il valore predefinito di tutte e tre è "falso".
Un accorgimento: poiché questa finestra verrà chiamata anche da "ConfigurazioneAvanzata", si può pensare di permettere la modifica del Place in cui lanciare l'agente (scegliendolo tra quelli "locali").

2.4.9 L'oggetto "Creatore"

L'oggetto "**Creatore**" (che sarà una classe "statica") rappresenta il punto di "centralizzazione" delle azioni di:

- creazione di nuovi Place (normali o di Default);

- caricamento e salvataggio dei dati di creazione di tali Place;
- lancio di agenti.

Per poter creare un Place occorrono i “dati” che sono contenuti nelle finestre di “DefPlaceDef” e “DefPlace” (paragrafi 2.4.2 e 2.4.3).
Poiché si vuole che in esecuzioni successive del sistema ci si “ricordi” i Place che sono stati creati, occorre prevedere il salvataggio dei “dati di creazione” in un file su disco.
Quindi abbiamo due sorgenti di tali dati: le finestre e il file.
Al fine di “standardizzare” i metodi di creazione, si è pensato di introdurre una “forma intermedia” per essi: una stringa che è stata battezzata “**stringa d’invocazione**”.
La seguente figura ne schematizza il funzionamento.

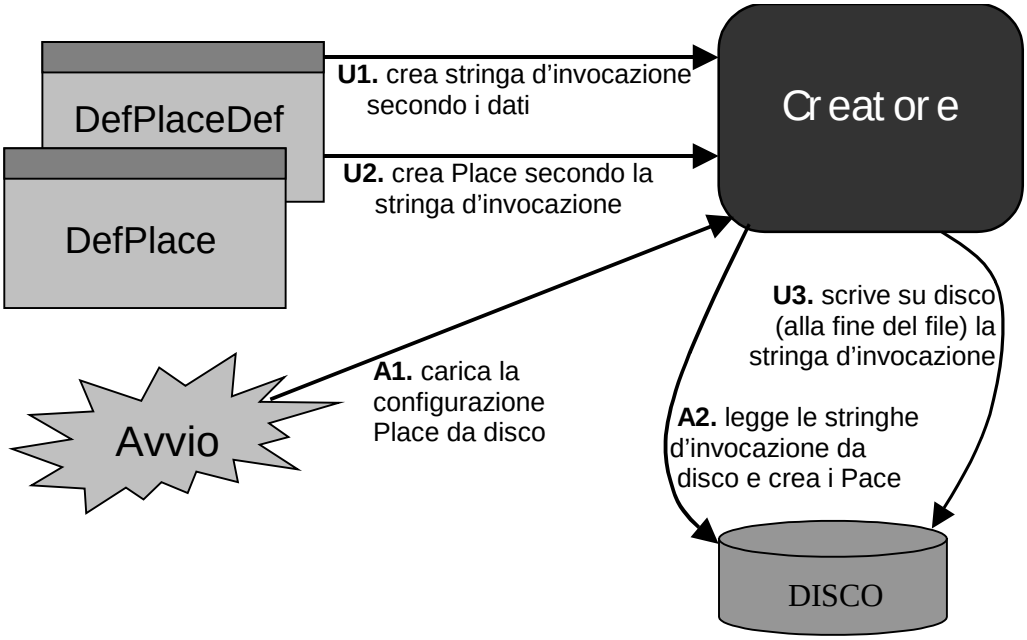


Figura 2.10 - schema di creazione con "Creatore"

Esistono quindi vari metodi che permettono di traslare il blocco dei dati di creazione in una stringa d’invocazione che dovrà essere poi passata all’**unico metodo di creazione**. Tali stringhe possono essere facilmente “accodate” in un file su disco e rilette al momento del lancio del sistema.
Il metodo per il **lancio di un agente** dovrà essere “parametrico” a un certo Place, e quindi se ne dovrà dare l’Environment. Infatti quando “ActionPlace” lo chiamerà gli passerà quello a cui lui si riferisce.
Infine, “Creatore” sarà l’unica classe della GUI a tenere memorizzato (e accessibile a tutte le altre classi) il “**direttorio radice**” del menu a linea di comando (che è creato all’avvio e dato a questa classe).

2.4.10 Bozzetto di “FinestraXNS” (DNS & PNS)

Tramite queste finestre dev’essere possibile mostrare all’utente le tabella dei “servizi di nomi” del Place. In realtà le finestre sono due: una per il DNS e l’altra per il PNS; perciò la chiamo “**FinestraXNS**”.

Ma poiché le “azioni” da fare sono sostanzialmente le stesse, si può pensare di usare il paradigma **Model-View-Controller** e di creare una stessa finestra “generica” per il servizio di nomi che andrà a “usare” il DNS o il PNS del Place a seconda di quanto specificato al momento della sua creazione.

Quindi, in entrambi i casi, nella finestra ho:

- la **visualizzazione della tabella** del servizio di nomi
- dei **bottoni**, ciascuno relativo a un’**operazione del servizio**.

Come vedremo più avanti, anche un agente potrebbe voler accedere alla tabella di DNS o PNS, ma solo per “visualizzarla” e non per modificarla. Ecco che occorre prevedere la possibilità di “**eliminare**” **i bottoni** (non “disabilitarli”, poiché chi usa l’agente non potrà mai voler modificare la tabella).

In aggiunta, la finestra del DNS può prevedere la visualizzazione del nome del Place di Default “genitore” (nella gerarchia di DNS) e dell’elenco di quelli “figli”.

Si noti che bisogna rendere possibile la “**selezione**” di una riga della tabella (affinché si possa scegliere quella da cancellare).

Le “azioni” su un DNS o PNS sono le seguenti:

- **PUT**: serve per inserire una nuova “riga” nella tabella; i dati (nome, indirizzo IP e numero di porta) verranno chiesti all’utente attraverso l’apertura di un’apposita finestra.
- **DEL**: viene tolta dalla tabella la riga che l’utente ha “selezionato” sulla tabella; ovviamente, occorre che ne sia stata selezionata una per poter invocare questa funzionalità (il bottone dev’essere disabilitato fintantoché non se ne seleziona una);
- **REFRESH**: si chiede al “proprietario” della tabella – il Place di Default “genitore” per il DNS e quello del proprio dominio per un PNS – di farsi ridare la tabella.

Il bozzetto della generica finestra di “XNS” è riportato in Figura 2.11.

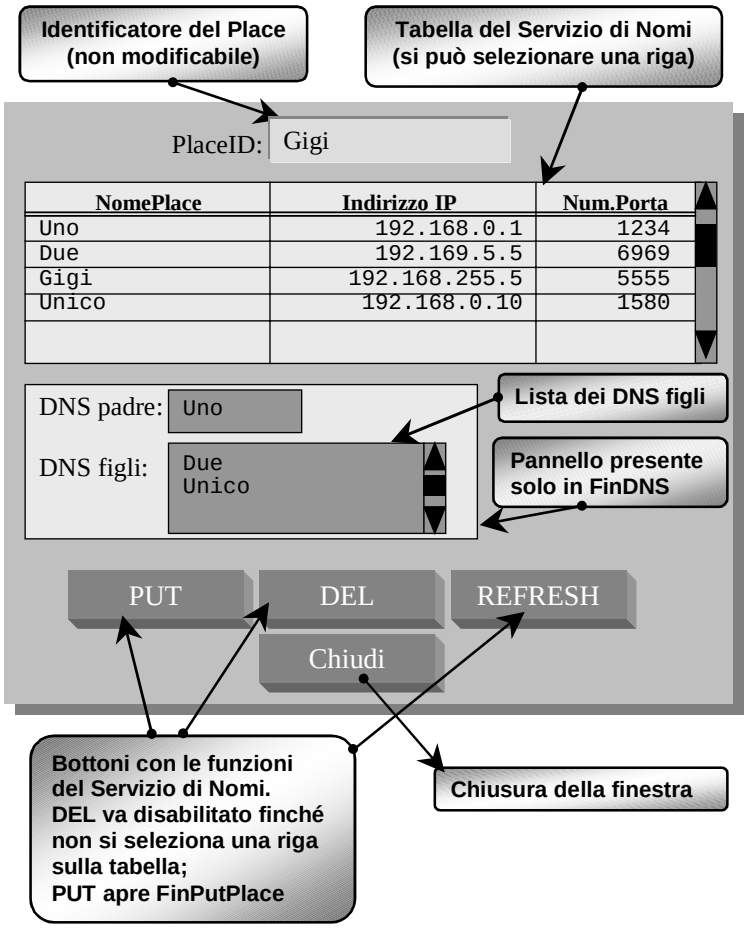


Figura 2.11 - bozzetto di FinestraXNS (per DNS e PNS)

2.4.10.1 Bozzetto di “FinPutPlace”

Per chiedere all’utente l’inserimento dei dati di un Place occorre distinguere tra Place di Default e Place all’interno di un dominio.
Nel primo caso il nome è composto da una sola stringa alfanumerica e numerica (senza bianchi) mentre nel secondo caso è composto da due stringhe di quel tipo.
In Figura 2.12 è riportato il bozzetto di questa finestra d’inserimento.

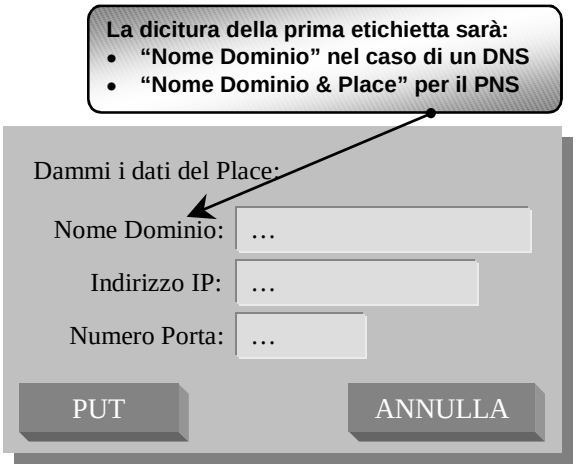


Figura 2.12 – bozzetto di FinPutPlace.

2.4.11 Bozzetto di "ManipolaAgenti"

Attraverso questa finestra (abbozzata in Figura 2.13) l'utente può fare delle azioni sugli agenti che sono al momento nel Place.
Ci sarà quindi un elenco di tutti gli agenti tramite il quale l'utente può sceglierne uno ("**selezionandolo**").
Sull'agente selezionato possono essere fatte le seguenti azioni:

- **STOP**: sospende l'esecuzione dell'agente (ossia del suo "worker");
- **START**: rimette in esecuzione un agente (che dev'essere "fermo");
- **REMOVE**: rimuove l'agente facendo terminare il suo "worker";
- **GOTO**: forza la l'agente a migrare su un altro Place (occorrerà aprire una finestra dedicata all'inserimento da parte dell'utente del Place verso cui far migrare l'agente);

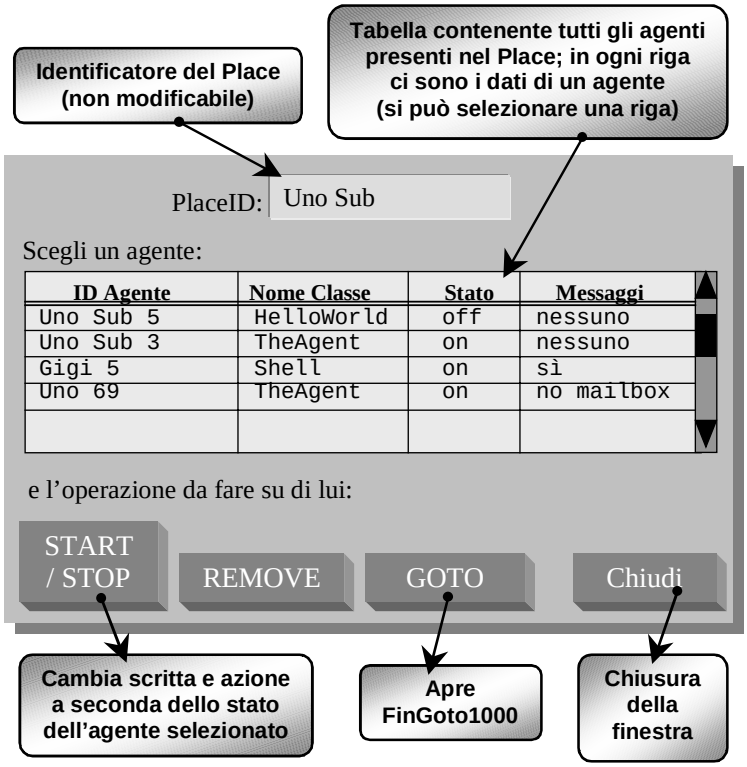


Figura 2.13 - bozzetto di ManipolaAgenti

Le due azioni di START e STOP sono una l'opposto dell'altra, perciò le si può inglobare in un **unico "bottone"** che eseguirà una o l'altra azione **a seconda dello stato dell'agente selezionato**.
La REMOVE può essere fatta solo quando l'agente è "fermo", perciò bisogna che il tasto sia disabilitato se si seleziona un agente che è in esecuzione.

2.4.11.1 Bozzetto di "FinGoto1000"

Come abbiamo detto prima, per l'azione GOTO occorre predisporre una particolare finestra per la scelta del Place di destinazione.
All'utente va richiesto soltanto il **nome del Place** verso cui si vuole migrare. Infatti quando si invoca il metodo "go()" per far migrare un agente basta specificare il "PlaceID", cioè il solo nome completo del Place di destinazione (ossia "Dominio" oppure "Dominio Place").
Quindi occorrerà predisporre una linea di testo in cui l'utente andrà a scrivere per intero il nome della destinazione. Il suo contenuto dovrà essere successivamente "convertito" in un oggetto "PlaceID" da dare al metodo "go()".
Ma è probabile che ci si voglia spostare in un Place **all'interno del proprio dominio**. Ma l'elenco di tali "PlaceID" sappiamo essere contenuto nella tabella di PNS.
Analogamente, tutti i Place di Default raggiungibili sono contenuti nella tabella di DNS (che però ha solo il Place di Default del dominio).
Allora si può pensare di includere nella finestra entrambe le possibilità:

- tramite un campo di testo si può inserire una stringa che rappresenta il PlaceID di un qualunque Place (anche non conosciuto dal DNS o PNS del Place attuale).
- con due "combo-box" si può scegliere il PlaceID del dominio e di un Place al suo interno; poiché tali informazioni sono prelevate dal PNS e dal DNS (se presente) del Place corrente:
  - se questo è un **Place di Default**, il "combo-box" dei dominii verrà riempito coi dati del DNS mentre quello dei Place conterrà i dati del PNS solo nel caso in cui nel "combo-box" precedente sia selezionato il Place attuale; in tutti gli altri casi conterrà come unica voce il Place selezionato dal "combo-box" dei dominii;
  - se invece ci si trova su un Place "normale", non avendo il DNS il "combo-box" dei dominii conterrà come unica voce il nome del Place attuale mentre quello dei Place conterrà i dati del PNS.

In Figura 2.14 ho fatto il bozzetto di questa finestra.

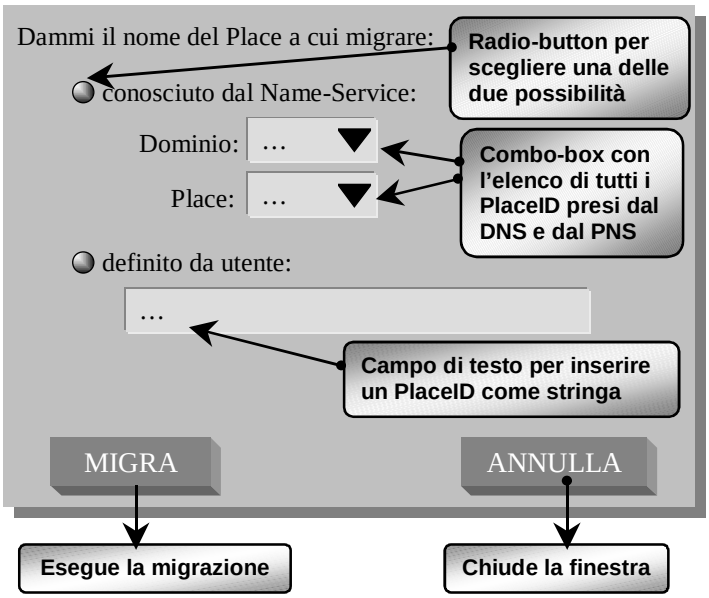


Figura 2.14 - bozzetto di FinGoto1000

Questa finestra risulterà molto utile anche agli agenti. Poiché alla fine diventerà una finestra di sistema, potrà essere usata anche da un agente di SOMA per chiedere all'utente dove si vuole andare.

2.4.12 Bozzetto di "PosizioneAgenti"

Questa è una semplicissima finestra (abbozzata in Figura 2.15) che serve per mostrare la posizione corrente (intesa come PlaceID) di tutti gli agenti che sono stati creati da un Place. Tali informazioni sono contenute nel Place di creazione; questa finestra esegue una pura "visualizzazione dati".
Una volta aperta, l'unica azione che si può fare e la chiusura.

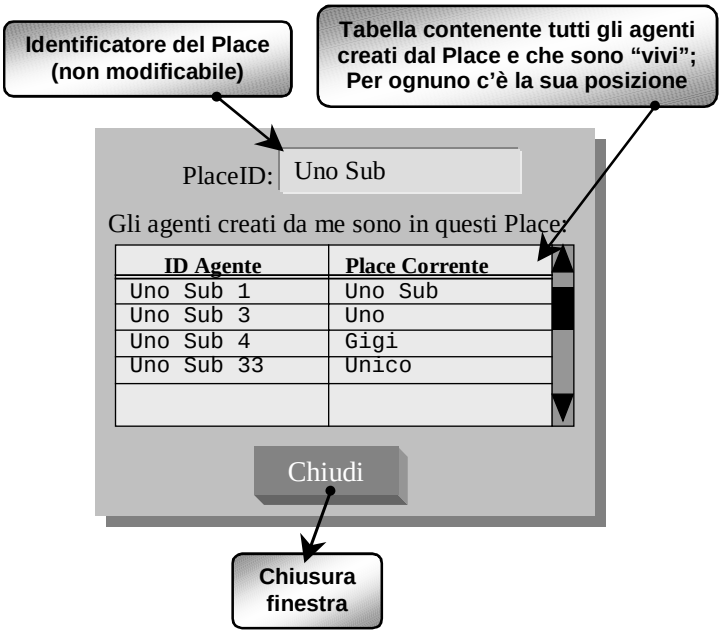


Figura 2.15 - bozzetto di PosizioneAgenti.

2.5 PARTICOLARITÀ DA REALIZZARE

Nella realizzazione dell'interfaccia utente grafica si vogliono ottenere alcune "particolarità" che possono agevolare anche in modo notevole l'uso del programma (e quindi agevolarne la "diffusione") .

2.5.1 Apparenza

Col termine "**apparenza**" intenderò l'aspetto grafico della GUI (cioè quello che in Java Swing viene chiamato "**Look & Feel**").
Si vuole che – in ogni momento – sia possibile cambiare l'apparenza della GUI secondo uno degli "stili" predefiniti.
Da notare che non si sta dicendo che si vuole cambiare l'aspetto di una sola finestra ma che sia possibile cambiare l'aspetto dell'intera GUI, cioè della finestra corrente, **di tutte le finestre** che verranno create ma anche di tutte quelle che sono già mostrate su video.

2.5.2 Lingua

Si può pensare di rendere la **GUI indipendente dalla lingua**, cioè che tutte le frasi e i messaggi che appaiono nell'interfaccia siano scritti in una lingua che non è fissa ma può essere cambiata (anche dinamicamente) dall'utente.
Quindi alla prima esecuzione il programma dialogherà con l'utente in inglese (che è una delle lingue più diffuse); in ogni momento però l'utente può decidere di usare un'altra lingua (per es., l'italiano).

In pratica, ogni stringa del programma deve essere disponibile in più “versioni”, una per ogni lingua.
Ovviamente, **non è pensabile** che si vada a creare **tante “versioni”** di una stessa classe quante sono le “lingue” che si usano (cioè di fare una “copia” di ogni classe che viene poi “tradotta” e resa disponibile con un nome leggermente diverso).
Si vuol far in modo che qualsiasi “classe” (che costituisce il programma) possa **ottenere le frasi (**che le servono) nella lingua correntemente impostata. Ecco allora l’idea di “**centralizzare**” in un particolare oggetto la **distribuzione delle “frasi in lingua”**.
Essendo un oggetto “distributore” ne basterà una sola “copia” ma occorrerà che tutte le classi possano “vederlo”. Il metodo più semplice per fare ciò è renderlo un oggetto “**statico**”.
Sappiamo che un programma scritto in un linguaggio orientato agli oggetti è strutturato in “classi” (che rappresentano – in un certo senso – il “mondo delle idee” di cui i singoli oggetti ne sono un’istanza).
La stessa strutturazione dev’essere mantenuta anche nella creazione delle frasi, cioè le singole frasi non sono di “proprietà” del programma ma delle particolari classi che lo compongono. Quindi le frasi non saranno memorizzate in un unico file ma in tanti file separati, ciascuno corrispondente alla classe che lo usa; **per ogni classe** occorre prevedere **un “file di lingua” per ogni possibile lingua**!
Ho stabilito che tale “file” dovrà essere memorizzato nello stesso direttorio in cui risiede il file di classe e dovrà avere lo stesso nome della classe ma con estensione “.it” (per la lingua italiana) o “.eng” (per quella inglese) o qualsiasi altra estensione che rappresenta una “nuova lingua”. Con questa strutturazione sarà facilissimo rintracciare il file di lingua (anche “dinamicamente”) nella lingua “attuale” per ogni classe del programma.
Inoltre non si altera di molto il normale “funzionamento” delle classi e si permette che possano ancora essere “riusate” in diversi programmi.
Per comodità realizzerò la traduzione nelle sole lingue “**italiano**” e “**inglese**”, ma predisporrò il tutto in modo che l’**aggiunta di una nuova lingua** (e di tutte le frasi “tradotte” con essa) sia il più facile possibile e, soprattutto, non necessiti di una modifica del codice.
Affinché sia possibile **recuperare una singola frase** all’interno del file di lingua ho pensato di associare a ciascuna un “**identificatore**”; in questo modo la classe può riferirsi in maniera “astratta” a ogni singola frase.
Ogni classe dovrà chiedere innanzitutto il caricamento del proprio “file di lingua”, dopodiché non dovrà far altro che richiedere una particolare frase tramite il corrispondente identificatore.
Affinché la lingua sia modificabile in modo dinamico bisogna permettere agli oggetti di essere **avvisati di un cambio di lingua**.
Sarà quindi necessario – attenendosi al modello a delegazione di eventi di Java – scrivere un proprio evento di “cambio di lingua”.

2.5.3 Salvataggio Configurazione

Una delle noie più grandi per l’utente è il dover fare delle azioni per riportare il sistema nello “stato” che vuole lui. Si deve quindi prevedere di salvare tutte le possibili modifiche che l’utente può fare durante l’uso della GUI. Possiamo elencare già le più importanti:

- **Lingua** in uso

- **Apparenza** in uso
- **Posizione e dimensione di ogni finestra**

Bisogna predisporre un "**file di configurazione**" in cui salvare tutte queste informazioni. Il salvataggio non deve necessariamente avvenire alla chiusura del programma ma non appena viene fatta una **modifica** (perché ci si deve ricordare la posizione di una finestra quando viene riaperta una seconda volta anche durante la stessa esecuzione).
L'idea è molto semplice: la configurazione del programma può essere rappresentata come un **insieme di proprietà**; a ognuna di queste viene attribuito un "**identificatore**" (cioè, un nome univoco all'interno del programma) al quale viene assegnato un "**valore**" il quale specifica l'attuale impostazione della proprietà.
Su una generica proprietà possiamo identificare tre possibili azioni:

- **Definizione**: l'assegnazione del valore alla proprietà viene fatto dal programma, cioè si sta impostandone il valore "di default"; in questo momento viene creata la coppia "identificatore-valore";
- **Modifica**: a una proprietà (precedentemente "definita") viene modificato il valore in seguito a un'azione fatta dall'utente; questo nuovo valore "scavalcherà" il default ma non lo andrà a cancellare (poiché si potrebbe volere che venga ripristinato quello di default);
- **Lettura**: dato l'identificatore di una proprietà viene reso il valore a esso associato; se c'è stata una modifica verrà reso il valore "modificato" altrimenti verrà reso quello "di default".

Poiché le modifiche devono essere "ricordate" anche dopo la terminazione del programma, bisogna far in modo che possano essere salvate su disco e quindi caricate successivamente.
Però le scritture su disco sono azioni "costose" (in termini di tempo), perciò si può pensare di fare le seguenti "ottimizzazioni":

- scrivere su disco **le sole proprietà** che sono state "**modificate**" dall'utente (e non quelle che sono ancora al valore di "default");
- scrivere su disco tali proprietà **soltanto quando è avvenuta almeno una modifica**, cioè quando le proprietà in memoria sono differenti da quelle memorizzate su disco.

Poiché si vuole che le proprietà siano "del programma" e non "della singola classe" che lo costituisce, si creerà una classe "**statica"**.

2.5.4 Salvataggio Configurazione dei Place

Allo stesso modo, tra un'esecuzione e l'altra è bene che non vadano perse neppure tutte le **definizioni dei Place** che l'utente ha dovuto inserire per crearli. Occorre prevedere un file di salvataggio della **configurazione dei Place** (separato da quello di configurazione della GUI) in cui memorizzare tutti i dati che l'utente inserisce con "**DefPlaceDef**" e "**DefPlace**".
Non appena il programma viene lanciato si deve controllare se esiste questo "salvataggio" e in caso affermativo chiedere all'utente se lo si vuole usare oppure no. Si noti che l'unica cosa che si salvano sono i dati inseriti da utente, quindi la creazione può avvenire "**come se**" l'utente gli avesse nuovamente inseriti tramite le finestre.

2.6 SCHEMA RIASSUNTIVO

In quest'ultimo paragrafo riproponiamo tutte le finestre e gli oggetti visti precedentemente in uno schema d'interazione (che è simile a quello di navigazione, visto in Figura 2.1, ma con l'aggiunta dei riferimenti agli oggetti che vengono usati).
Abbiamo quindi le seguenti entità:

- **Inizio**: finestra iniziale per scegliere cosa fare; si può creare un Place, passare a AdvConfig o caricare la vecchia configurazione (cosa che verrà fatta da Creatore).
- **DefPlace** e **DefPlaceDef**: inserimento dati di creazione di un Place; la creazione viene delegata a Creatore.
- **AdvConfig**: creazione di più Place (sempre richiamando Creatore), apertura di FinestraPlace di Place già creati e esecuzione "rapida" di azioni su un qualsiasi Place.
  Usa un oggetto Anagrafe per memorizzare la configurazione locale del Place (cioè gli Environment) che possono essere reperiti e usati; da questi si risale all'ActionPlace a cui si possono richiedere azioni (come l'apertura della FinestraPlace o il lancio di agenti).
- **FinestraPlace**: la finestra del Place da cui si possono eseguire le azioni su un Place; non le esegue direttamente, ma le fa fare al suo ActionPlace (quindi è limitata dalle azioni che lui può fare);
- **ActionPlace**: definisce e centralizza tutte le azioni che possono essere fatte su un Place; contiene anche lo "stato" della sua FinestraPlace (aperta/chiusa).
- **Creatore**: il responsabile di creare un Place tramite opportune "stringhe d'invocazione"che possono quindi essere salvate su disco e recuperate al successivo lancio del sistema.
- **Anagrafe**: mantiene un elenco di tutti gli Environment creati "localmente" (con i relativi dati di creazione) in modo che possano essere facilmente reperiti tramite il solo PlaceID. Serve per evitare che l'utente inserisca dei dati di creazione che ha già inserito.

In Figura 2.16 è riportato lo schema d'interazione.

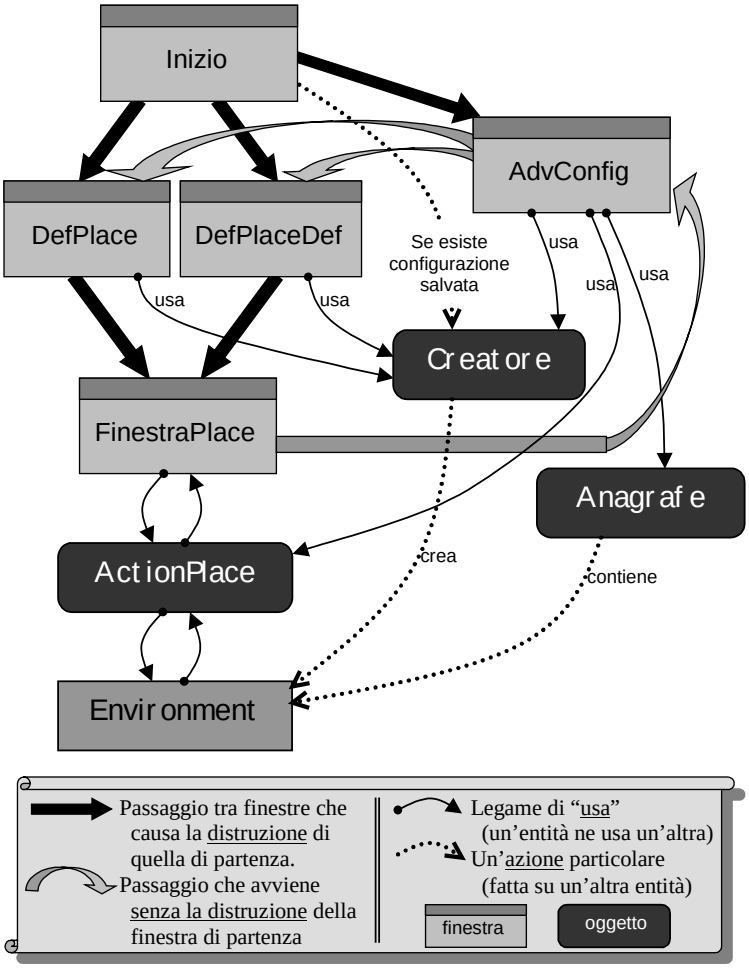


Figura 2.16 - schema riassuntivo di navigazione e interazione

Cap.3 - UN AGENTE & LE APPLET REMOTE

3.1 PROLOGO

In questo capitolo si descriverà come si sia realizzato un agente mobile "di esempio" e come si sia modificato il sistema per prevedere la comunicazione tra un Place "locale" e una particolare "applet" remota (da noi realizzata) che viene scaricata dal computer su cui risiede il Place ma che sarà eseguita su un computer "remoto".

3.2 L'AGENTE *THEAGENT*

Oltre alla realizzazione delle finestre della GUI, si è previsto anche la costruzione di un **agente mobile** che sfrutti tali finestre (per es., quella di inserimento del Place verso cui migrare o quella per visualizzare i dati del DNS o PNS).
Prendendo spunto dall'agente *Shell* (già disponibile nella versione 1.5 di SOMA) abbiamo deciso di realizzare l'agente "TheAgent" (cioè "L'Agente", con la L e la A maiuscola).
Lo scopo è di avere uno **scheletro** d'agente mobile da cui si possa facilmente un agente più complesso e specifico per un campo applicativo.
L'unica cosa importante è come venga "**mantenuto in vita**" l'agente in modo che aspetti le azioni fatte da un utente locale. Infatti sappiamo che ogni agente ha un worker associato che lo mette in esecuzione invocandone un particolare metodo. Il problema è che quando questo metodo termina e il controllo ritorna al worker, questo distruggerà l'agente perché è convinto che sia terminata l'esecuzione. Per realizzare una sorta di "**programmazione guidata dagli eventi**" occorre mantenere in vita l'agente.
Il metodo più facile per far questo è di bloccare l'agente su un semaforo (più precisamente, sulla coda di un **monitor** di un particolare oggetto). Lo si farà nell'esecuzione del suo metodo di avvio. Anche se è bloccato, la sua interfaccia grafica potrà lo stesso (a seguito di azioni dell'utente) invocarne i metodi.
Inoltre, per vedere che la migrazione coinvolge anche i dati dell'agente, abbiamo predisposto una struttura dati per memorizzare i nomi dei Place visitati (quindi è una struttura che cresce man mano che l'agente migra).
Un problema sarà la possibilità del "**cambio di lingua**"; infatti l'agente può migrare ma i *file di lingua* (previsti per l'agente) no.
Si capisce subito che i file di lingua sono delle **risorse non trasferibili** e quindi possono essere acceduti solamente quando l'agente si trova sul **Place di creazione**.
A questo punto si possono pensare varie soluzioni:

- Quando l'agente migra si deve portare dietro anche tutti i file di lingua: è una soluzione molto costosa, soprattutto se si hanno un certo numero di lingue;
- L'agente migrerà portandosi dietro solamente il file di lingua corrispondente alla lingua attuale: è la soluzione migliore perché l'agente non viene appesantito da un bagaglio inutile di frasi; in più si potrebbe pensare che:
  - L'agente possa successivamente instaurare una comunicazione con l'oggetto "Lingua" presente sul suo Place di origine in modo da farsi spedire un file di lingua di cui può avere bisogno;

- L’agente non faccia nessuna comunicazione col Place di origine perciò la lingua non potrà essere cambiata quando lui si trova su altri Place.

Tra queste possibilità ho optato per l’ultima perché ho preferito mantenere il più semplice possibile il modello e non eseguire nessuna connessione in più del necessario.
Quindi:

- L’agente caricherà al suo interno tutte le frasi di lingua in modo che vengano trasportate insieme a lui quando viene fatto migrare su un altro Place, cioè quando viene serializzato; in questa maniera non è richiesta nessuna connessione in più col Place di origine;
- La lingua potrà essere cambiata solamente quando l’agente si trova nel Place di creazione e tutte le nuove frasi dovranno essere caricate all’interno dell’agente;
- Sui Place diversi da quello di origine bisogna disabilitare all’agente la possibilità di cambiare la lingua.

3.3 COMUNICAZIONE CON APPLET REMOTE

Una volta realizzata la GUI del Place, si è pensato di creare un’interfaccia che non sia solo locale. Si è deciso di costruire un’**Applet Java** che può essere scaricata (secondo il classico paradigma COD) tramite un **browser** e messa in esecuzione su un qualsiasi computer. Il codice dell’Applet sarà memorizzato all’interno del computer in cui risiede il Place, però verrà scaricato e eseguito su un computer **remoto**.
L’Applet dovrà perciò **dialogare col Place attraverso la rete** in modo da eseguire certe azioni o richiedere dei dati.
Per come è stato strutturato il sistema SOMA, le uniche possibilità di “scambio di messaggi” previste sono tra Place e Place; occorrerà quindi realizzare un nuovo tipo di comunicazione.

3.3.1 Schema Concettuale

Nella progettazione della GUI si è scelto di “**centralizzare**” tutte le azioni che possono essere eseguite su un Place nell’oggetto “**ActionPlace**” associatogli. (si veda il paragrafo 2.4.7).
È proprio a lui che le Applet andranno sia chiedere le informazioni sul Place sia a inviare le richieste di fare certe azioni.
Nelle seguenti figure “a doppia pagina” è mostrato lo schema concettuale del collegamento tra il lato *locale* e quello *remoto*.

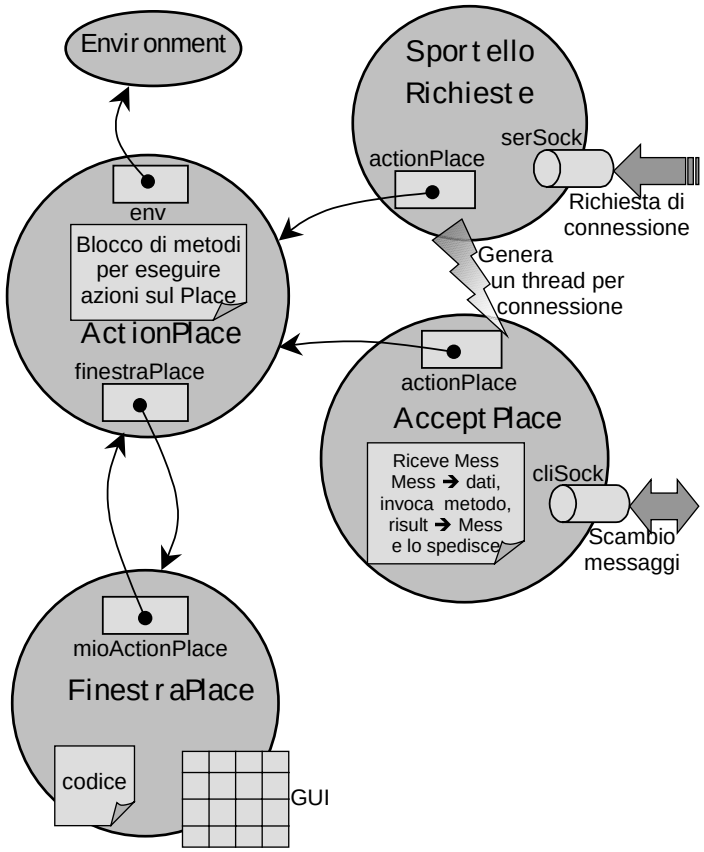


Figura 3.1 - schema di interazione Place-Applet, lato “locale”

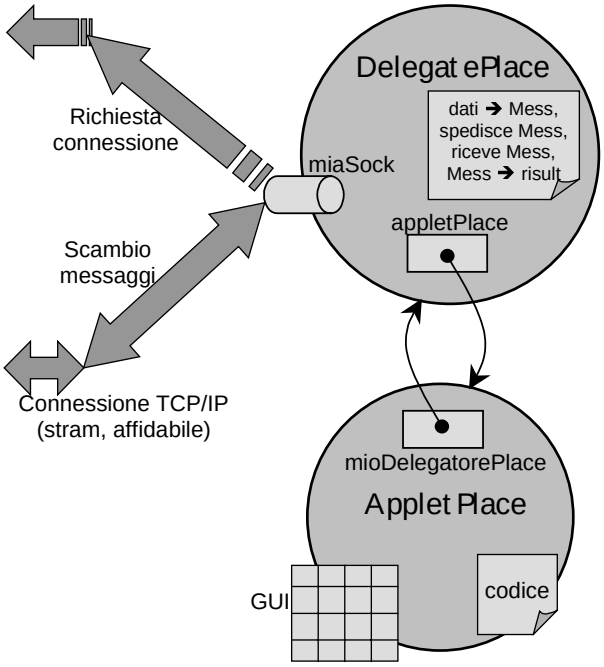


Figura 3.2 - schema di interazione Place-Applet, lato “remoto”

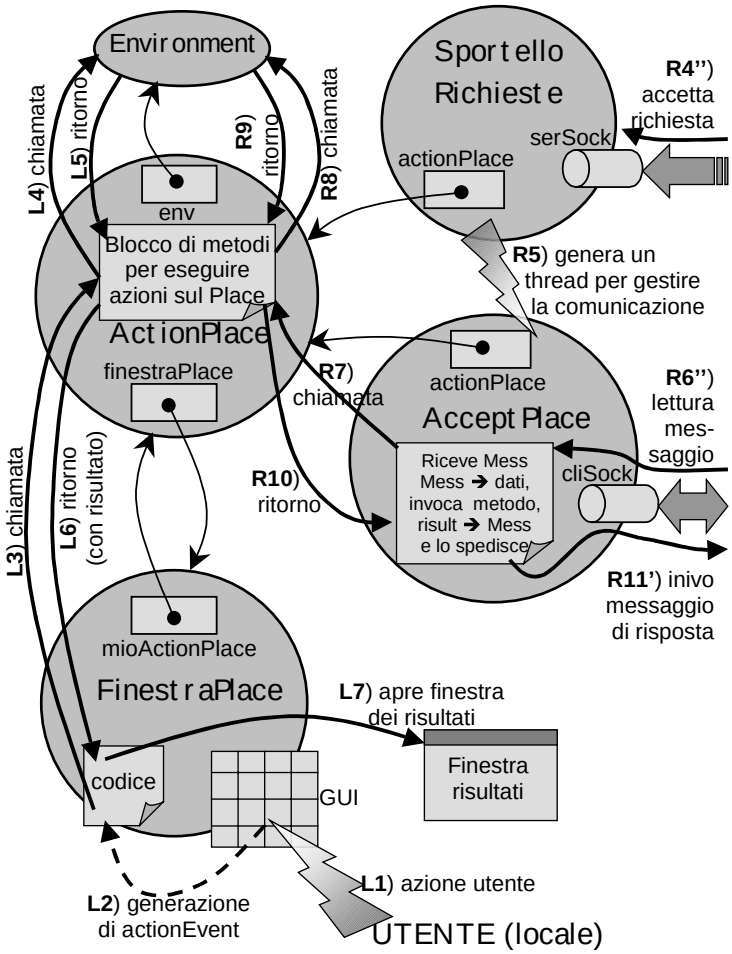


Figura 3.3 - uso dello schema di interazione Place-Applet, lato “locale”

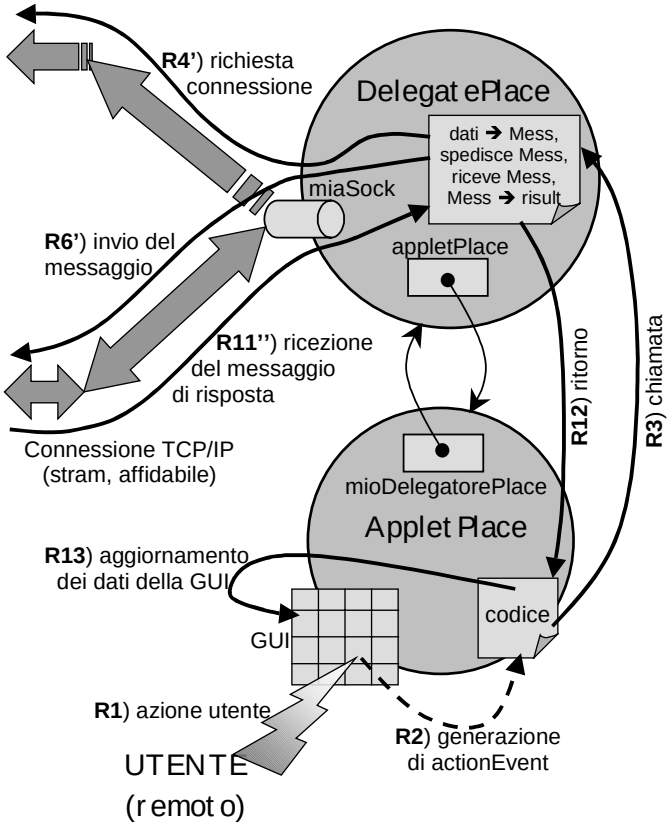


Figura 3.4 - uso dello schema di interazione Place-Applet, lato "remoto"

Nei primi due è riportato il solo schema mentre gli altri due mostrano la sequenza di azioni che viene fatta ("L" per un azione locale e "R" per una remota).
Si è pensato di dichiarare tutti i "metodi" che sono invocabili su un Place, tramite un'interfaccia chiamata "**ActionPlaceInterface**".
Gli oggetti "**ActionPlace**" e "**DelegatePlace**" implementano tale interfaccia perché devono permettere l'interazione col Place (rispettivamente dal lato "locale" e da quello "remoto"). In questo modo gli oggetti che li usano (FinestraPlace e AppletPlace) sanno quali sono i metodi che possono invocare.
Ovviamente, "DelegatePlace" dovrà instaurare una comunicazione via rete con un "demone" presente sul computer in cui risiede il Place.
Tale comunicazione sarà però "**nascosta**" all'AppletPlace, la quale non farà altro che invocare i metodi messi a disposizione dal suo DelegatePlace: il "come" vengano realizzati non gl'interessa.

3.3.2 Scambio di Messaggi

La coppia di oggetti "**DelegatePlace**" e "**AcceptPlace**" servono per trasportare i "**comandi**" (sottoforma di **messaggi**) dal computer remoto a quello locale e viceversa (per i messaggi di risposta).
Per la comunicazione si è scelto di usare un trasporto "**affidabile**", cioè **TCP/IP** il quale realizza l'astrazione di **flusso di dati**. Su tale canale di comunicazione passerà quindi un flusso di dati o, più in generale, di oggetti. Poiché "*everything is an object*", anche i messaggi che vengono scambiati saranno degli oggetti (e non dei semplici dati) e quindi dovranno essere "**serializzabili**".
I messaggi servono solo per **contenere i dati** che devono essere trasportati alla controparte; la **classe** del particolare messaggio indicherà l'**azione** a cui i dati si riferiscono.
Tutti i messaggi erediteranno da una classe astratta "**Mess**" la quale non dichiarerà nessun metodo particolare poiché il suo scopo è di realizzare **l'astrazione di "messaggio"**.
Da questa si è deciso di far discendere tre classi, ognuna delle quali rappresenta l'astrazione di un "**messaggio in una certa direzione**":

- **MessDel**: rappresenta l'astrazione di un messaggio che viene creato e inviato *dal DelegatePlace* verso l'AcceptPlace;
- **MessAcc**: rappresenta l'astrazione di un messaggio che viene creato e inviato *dall'AcceptPlace* verso il DelegatePlace;
- **MessBid**: è l'astrazione di un messaggio *bidirezionale*, cioè può essere creato e inviato da una qualsiasi delle controparti.

Da ognuna di queste discenderà la **realizzazione concreata** di uno o più messaggi. In questo modo s'è creata una distinzione che può risultare molto utile nel caso sia necessario in futuro aggiungere altri messaggi. Si noti che ogni sottoclasse è "battezzata" con un nome la cui **parte iniziale** è data dal **nome della sua superclasse**; per es., da "Mess" discende "MessDel" e da questo "MessDelDammi".
In questo modo già dal nome si capisce la classe di appartenenza.
Ecco l'elenco dei messaggi che si è ritenuto d'implementare:

- **MessDelLancia**: richiesta del lancio di un agente;
- **MessDelDammi**: richiesta di rendere un "qualche dato" (specificato tramite una valore contenuto nel messaggio); potrà essere reso uno tra i seguenti "oggetti":
  - il PlaceID;
  - l'elenco degli agenti che possono essere messi in esecuzione;
  - l'elenco (un "Vector") dei PlaceID contenuti nel DNS;
  - analogo, ma legge dal PNS;
  - l'albero dei thread;
  - un elenco contenente le terne "AgentID, NomeClasse, Stato" per tutti gli agenti in esecuzione nel Place;
  - un elenco contenente le coppie "AgentID, PlaceIDAttuale" per tutti gli agenti che sono stati creati dal Place;
- **MessDelAYA**: è la richiesta di "AYA" (abbreviativo di "Are You Alive") da parte del Delegatore per chiedere alla controparte se è "viva", cioè se può ricevere i messaggi. Infatti il Place può disabilitare la funzionalità di accettare connessioni dalle Applet.

- **MessAccIAA**: e questo è la “IAA” (“I Am Alive”), cioè la risposta del lato AcceptPlace al messaggio di “AYA”. L’Applet invierà l’AYA e si aspetterà (entro un certo “**time-out**”) l’arrivo di un IAA come conferma della disponibilità del Place.
- **MessBidMess**: serve solamente per trasportare un messaggio di testo alla controparte (è infatti un messaggio bidirezionale); viene usato principalmente da AcceptPlace per segnalare alla controparte che l’operazione richiestagli è stata fatta correttamente;
- **MessBidErrore**: mentre questo serve per rendere un errore, cioè un oggetto di classe “Exception”; infatti AcceptPlace usa questo messaggio per dire alla controparte che non è stato possibile eseguire l’operazione appena richiestagli.

Il seguente schema mostra la gerarchia di ereditarietà dei “messaggi”.

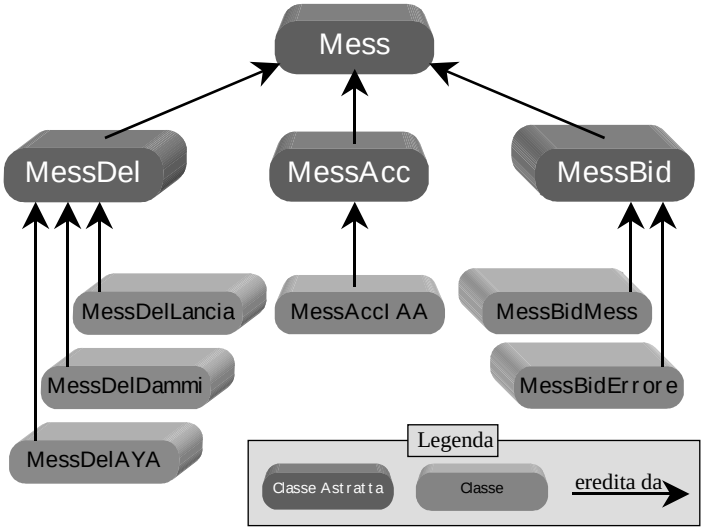


Figura 3.5 - gerarchia di ereditarietà dalla classe "Mess"

Per l’invio e la ricezione degli oggetti “messaggio” si sfrutta la “serializzazione”. In particolare, si “incapsulerà” la socket in due oggetti di classe “ObjectInputStream” e “ObjectOutputStream”, cioè si vedrà la connessione come se fosse un “**flusso di oggetti**”.
Per ricevere un messaggio (cioè un oggetto discendente di “Mess”) non si farà altro che scrivere qualcosa come:

```
ObjectInputStream  objIn  = new
                  ObjectInputStream(cliSock.getInputStream());
Mess mess = (Mess) objIn.readObject ();
```

e per inviarne uno:

```
ObjectOutputStream objOut = new
                ObjectOutputStream(cliSock.getOutputStream());
objOut.writeObject(mess);
objOut.flush();
```

L'AcceptPlace saprà che azione dovrà fare andando a testare (sfruttando il "late-binding" di Java) la classe di appartenenza del messaggio con l'operatore "**instanceof**". In base a essa si andrà a eseguire uno dei metodi di ActionPlace e se ne renderà il risultato al chiamante (sempre tramite un messaggio). Riassumendo:

- AppletPlace invoca un metodo di DelegatePlace;
- esso crea un messaggio (a seconda dell'azione da compiere) contenente i dati opportuni;
- dopodiché crea una connessione con SportelloPlace del Place (il cui indirizzo e N.di porta sono già stati definiti al suo interno);
- quindi invia il messaggio sul lato di "output" del flusso d'oggetti;
- e si mette in attesa (dal lato di "input") del messaggio di risposta (e in caso sia un messaggio di errore si provvederà a "rigenerare" l'eccezione in esso contenuta). L'attesa è "**bloccante**", cioè l'applicazione verrà sospesa fintantoché non arriva la risposta.
  Per non aspettare indefinitamente una risposta che (per qualsiasi motivo) potrebbe non arrivare, si è deciso di impostare un tempo massimo (**time-out**) di attesa. Allo scadere di tale tempo l'applicazione verrà sbloccata e potrà continuare l'esecuzione e, dato che non è arrivata una risposta, si può provare a inviare una seconda volta il messaggio di richiesta.

Un lettore "esperto" può aver già riconosciuto in tutto questo uno schema molto simile a quello di "chiamata di procedura remota". Infatti ci assomiglia molto, però **non** è stata usata la **RMI** (remote method invocation) ma il semplice **scambio di messaggi** perché si è voluto stare "dalla parte dei bottoni". In particolare è stato richiesto che fosse possibile da parte del lato locale accettare o negare l'esecuzione di azioni (come vedremo nel prossimo paragrafo).

3.3.3 SportelloRichieste

Il DelegatePlace non instaurerà una comunicazione direttamente con un AcceptPlace, ma invierà (per prima cosa) una "**richiesta** di connessione" al **demone** chiamato **SportelloRichieste** presente sulla macchina remota. Per ogni nuova richiesta il demone creerà e innescherà un **thread** separato (**AcceptPlace**, appunto) a cui affiderà l'intera comunicazione col DelegatePlace.
In sostanza stiamo realizzando un "**server parallelo**" poiché il demone dopo aver accettato una nuova richiesta **tornerà subito a accettare nuove richieste**, disinteressandosi completamente della comunicazione vera e propria dei messaggi. In questo modo possono esserci più oggetti "DelegatePlace" che inviano messaggi allo stesso Place, ma passando attraverso differenti istanze di AcceptPlace. Infatti, grazie all'uso di una comunicazione "**con connessione**", ogni volta che viene accettata una richiesta, il sistema ci creerà in modo automatico una **nuova socket** (diversa da quella su cui era arrivata la richiesta) su cui far avvenire lo scambio di messaggi.
Quindi ogni AcceptPlace avrà un **canale di comunicazione separato** da tutti gli altri e di conseguenza si possono avere più istanze di questa classe che eseguono in modo concorrente.
Un Place può quindi ricevere dei comandi **contemporaneamente** da più Applet (poste su dei computer che – usando un collegamento Internet – possono trovarsi in qualsiasi locazione geografica).
Inoltre SportelloRichieste può funzionare anche da **accettatore delle richieste** poiché può essere messo in grado di controllare quante connessioni sono al momento aperte (basta sapere

quante istanze di AcceptPlace sono in esecuzione) oppure chi è il mittente di una richiesta e a sua discrezione accettare o negare il servizio.
Un particolare molto importante è che qualsiasi richiesta (sia locale sia remota) passa sempre per l'oggetto **ActionPlace**. Se in esso alcuni metodi vengono definiti sincronizzati (**synchronized**) si può avere la certezza che eventuali contemporanee richieste di esecuzione verranno sequenzializzate.
Inoltre lo SportelloRichieste deve poter essere **chiuso** (anche solo temporaneamente) da parte del Place di riferimento.
Ma affinché possa essere "visibile" dal Place è necessario memorizzarne il riferimento all'interno dello stesso "**Environment**" (come fu preannunciato nel paragrafo 1.6).
Ricordiamo che abbiamo bisogno di due informazioni per poter identificare un processo all'interno di un computer posto in una rete:

- l'**indirizzo** di rete del computer (che può essere logico o numerico)
- il numero di **porta** a cui si è collegato ("bind") il processo.

Quindi le Applet devono conoscere entrambe queste informazioni per poter comunicare col Place. Ovviamente, l'indirizzo di rete sarà quello da cui sono state scaricate (si ricorda che, per ragioni di sicurezza, le Applet non possono comunicare via rete con nodi differenti da quelli da cui sono state scaricate).

3.3.4 AppletPlace

L'oggetto "**AppletPlace**" è solamente la GUI per il lato remoto.
Un qualsiasi utente del web può accedere col suo browser alla pagina "**AppletPlace.html**" nella quale è contenuto il "rimando" al codice dell'applet chiamata AppletPlace.
Il "browser" scaricherà tale codice e lo metterà in esecuzione sul suo computer mostrandolo in una zona all'interno di se stesso.
Per prima cosa andrà a creare e a inizializzare il suo DelegatePlace.
In fin dei conti l'Applet non farà altro che chiamare le funzionalità di quest'oggetto e aggiornare di conseguenza i dati mostrati a video.
Poiché lo "stato" del Place può variare in ogni momento ma esso non è a conoscenza delle possibili Applet che lo usano (cioè il Place è come se fosse un **servitore senza stato**), affinché l'Applet possa vedere lo stato corrente del Place bisogna che glielo chieda.
Quindi occorre predisporre dei "bottoni" per la richiesta di una particolare "parte" dello stato del Place.
L'unica azione che si può fare dall'esterno sul Place è il **lancio di un agente** (con parametri). Ovviamente, una volta lanciato non si può avere alcun controllo su di esso. Il motivo di questa scelta è che **tutte le altre azioni possono essere "pericolose" se fatte da qualche malintenzionato**. Infatti non è prevista nessuna autenticazione degli utenti e non viene fatta alcuna distinzione tra le azioni di più utenti.
Se si desse la possibilità di far migrare gli agenti (contro la loro volontà) o di fermarli e ucciderli, si andrebbe a interferire con il normale funzionamento del sistema.
Nessuno eccetto il creatore deve poter uccidere i propri agenti. Poiché l'utente dell'Applet può essere chiunque, è bene che non possa fare azioni che siano "dannose" per gli altri utenti.
Le altre funzionalità che sono state messe a disposizione riguardano il solo recupero di informazioni sullo stato del Place, cioè:

- il nome dei Place;
- la tabella di DNS e quella di PNS (ma solo l’elenco dei PlaceID);
- l’elenco di tutti gli agenti in esecuzione nel Place;
- la posizione di tutti gli agenti che sono stati creati dal Place;
- l’albero contenente tutti i thread in esecuzione all’interno del Place.

Cap.4 - USO DELLA GUI DI SOMA

4.1 AVVIO DEL SISTEMA

Come si è già detto, la prima finestra della GUI sarà “Inizio”, però tale classe si trova in “SOMA.gui”, quindi non è “facilmente” accessibile da utente. Chi ha progettato l’interfaccia con menu a linea di comando ha previsto una classe chiamata “**Main**” nel **package** “**SOMA**”, che è facilmente identificabile da utente perché si trova nel direttorio principale del sistema (che è l’unico posto in cui un utente “inesperto” si aspetta di trovare in punto di ingresso al programma).
Poiché il nostro scopo è di costruire un GUI senza modificare il codice già esistente di SOMA (o per lo meno, modificandolo il meno possibile) ho pensato di realizzare la classe “**MainGui**” sempre nel package “**SOMA**”. Essa contiene il metodo “main” col quale lanciare il sistema corredato dell’interfaccia grafica. In questo modo, chi preferisce l’interfaccia a menu può sempre lanciare la “vecchia” classe “Main”. Per metterle in esecuzione la GUI, occorre scrivere:

```
java -Djava.security.policy=SOMA.policy SOMA.MainGui
```

Se si vuole usare la “sicurezza” bisogna aggiungere al “classpath” le classi per la crittografia; per esempio:

```
java  -classpath .;lib\entrust.jar;lib\jndi.jar;
  lib\entrust.jar;lib\entapplet.jar
  -Djava.security.policy=somaSecurity.policy
  -Djava.security.auth.policy==soma_jaas.policy
  -Djava.security.auth.login.config=soma_jaas.config
  SOMA.MainGui
```

Premendo il tasto “invio” verrà mostrato il messaggio di Figura 4.1 e verrà aperta la finestra “**Inizio**”.

```
Z:\ANTENU~1\Soma>java -Djava.security.policy=SOMA.policy SOMA.MainGui
*** S.O.M.A. ***  by Faculty of Computer Science Engineering
                  of the University of Bologna (Italy)
use << MainGui -? >> for command line syntax
```

Figura 4.1 - avvio di SOMA con "MainGui".

Ho previsto la possibilità di passare alcuni parametri a “MainGui”: basterà aggiungerli in coda al precedente comando di lancio.
Si riconoscono perché sono preceduti da un carattere **meno** (-) o da uno di **barra** (/) e sono:

- -**menu** per non usare la nuova GUI ma il “vecchio” menu a linea di comando (come se si lanciasse “Main” anziché “MainGui”);
- **-cfg**: per aprire subito (prima ancora di “Inizio”) la “FinOpzioni”.
- **-?**: mostra una rapida spiegazione di questi parametri.

4.2 LA FINESTRA INIZIALE

La finestra iniziale del sistema apparirà così:

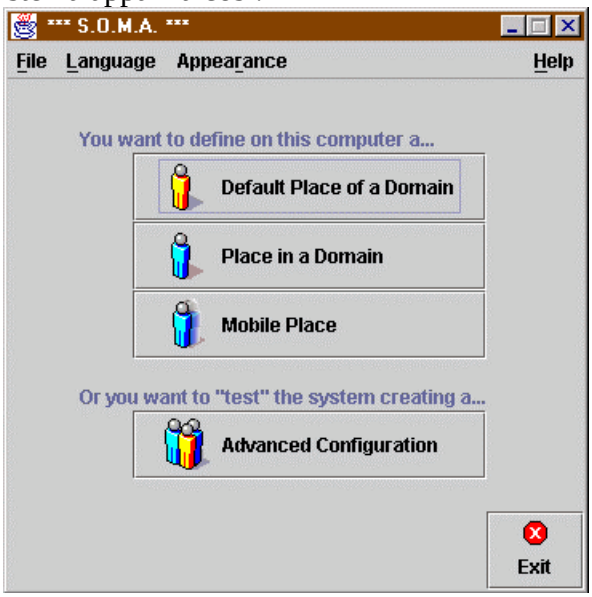


Figura 4.2 - finestra "Inizio"

Si nota subito la presenza di un menu a cascata nella parte alla della finestra; le voci più importanti sono “**file:options**” (in cui si possono impostare alcune opzioni specifiche per la GUI), “**language**” (tramite cui si può cambiare – anche durante l’esecuzione – la lingua usata dalla GUI) e “**appearance**” (tramite il quale si può scegliere una differente visualizzazione grafica delle finestre). Ecco le opzioni:

Figura 4.3 - la finestra delle "opzioni" della GUI

Per esempio, se si imposta la lingua “Italiana” con apparenza “Windows” la finestra “Inizio” si modificherà come la seguente:

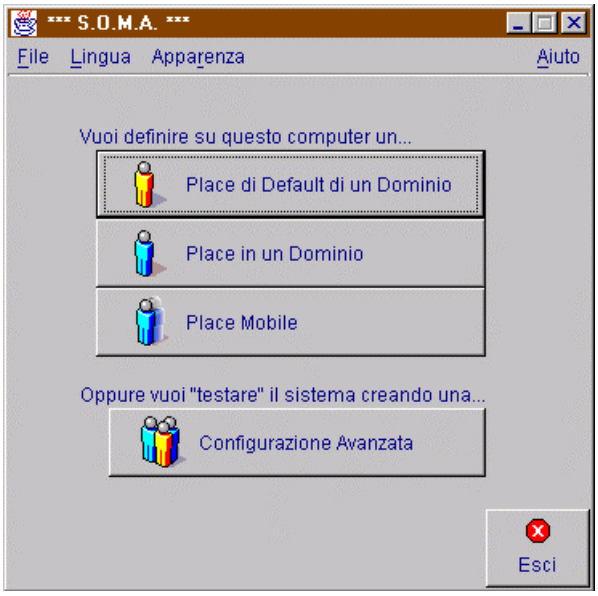


Figura 4.4 - finestra "Inizio" in Italiano con il Windows L&F

Andiamo subito nella "*configurazione avanzata*" premendo il bottone nella parte bassa della finestra.

4.3 LA "CONFIGURAZIONE AVANZATA"

La nuova finestra che ci apparirà è mostrata in Figura 4.5 e ci permette di creare più Place sullo stesso computer. Nella parte di destra della finestra sono presenti i bottoni di controllo; quelli nella zona alta servono per la creazione di un nuovo Place mentre quelli nella parte bassa svolgono delle funzioni specifiche per un certo Place (che tra breve vedremo come possa essere scelto). Nella parte sinistra della finestra un pannello mostra un elenco (al momento vuoto) di tutti i Place presenti nel sistema **locale**; la "vista" può essere cambiata tramite una delle "linguette" poste al di sopra del pannello.
Si noti che il menu a cascata è lo stesso di quello della finestra iniziale e sarà sempre presente in tutte le altre finestre (in particolare, solo nella "finestra di Place" gli verranno aggiunte molte altre voci).

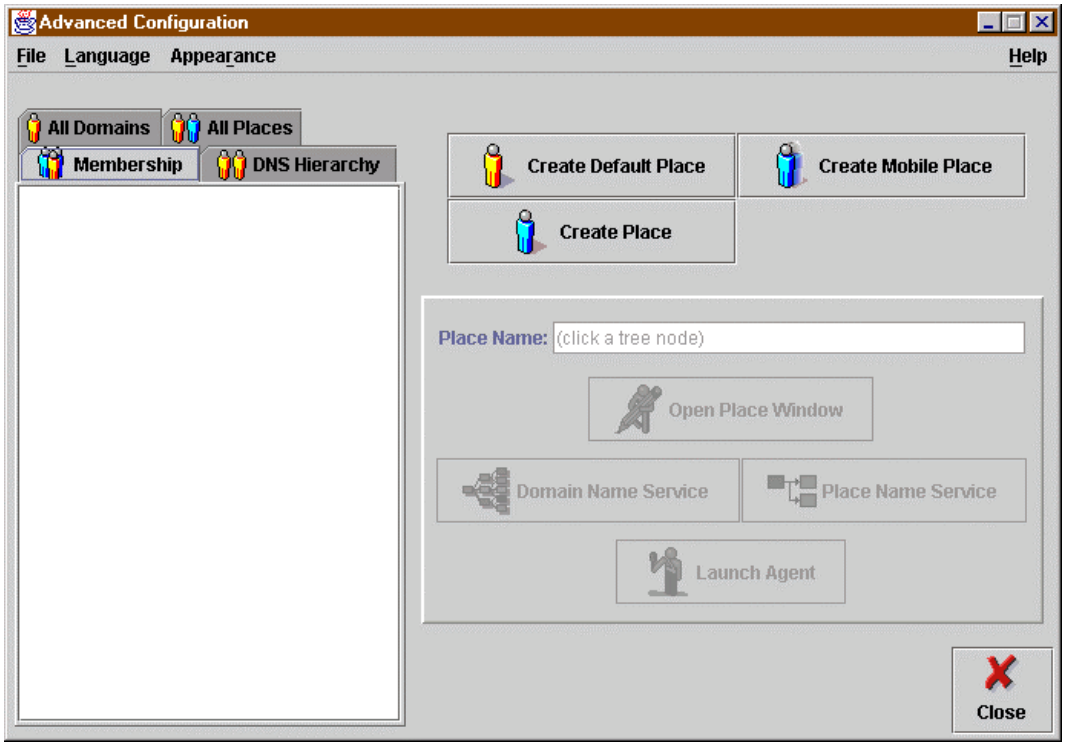


Figura 4.5 - la finestra "Configurazione Avanzata"

Tutti i bottoni nella parte bassa sono disabilitati (cioè colorati in grigio) perché non è stato selezionato un Place sul quale possano lavorare. Nel progettare la GUI abbiamo sempre seguito questo comportamento: tutto ciò che, in un certo momento, non può essere fatto viene "**disabilitato**" (in modo da impedire ogni possibile errore dovuto all'esecuzione di funzionalità su dati inesistenti o incompleti).
Inoltre ogni componente della GUI ha un suo messaggio di "**tip**", ossia se si muove il mouse su di esso apparirà vicino al puntatore una casella di testo contenente una breve descrizione dell'uso.

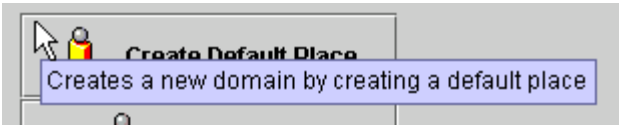


Figura 4.6 - un esempio di messaggio di "tip"

4.3.1 Creazione di un Place di Default

Come prima cosa creiamo un Place di Default premendo il primo bottone (della parte alta); apparirà la schermata di Figura 4.7.
Occorre inserire il nome del Place di Default (che di conseguenza sarà il nome del dominio) e il numero di porta attraverso cui comunicare con gli altri Place.
Quindi occorre inserire i dati per la "registrazione" nella gerarchia dei DNS. Se si sta creando la "radice" della gerarchia, basta lasciare attivato il primo "radio-button"; le altre voci indicano **a chi** registrarsi.

Se il Place presso cui registrarsi è stato creato sullo stesso computer basterà abilitare la voce "**local**" e sceglierne il nome sul combo-box.
Se invece il Place risiede su una macchina "**remota**", bisognerà selezionare l'ultima voce e indicare i dati del Place remoto (l'indirizzo IP del computer su cui risiede e il numero di porta a cui è stato collegato al momento della sua creazione).

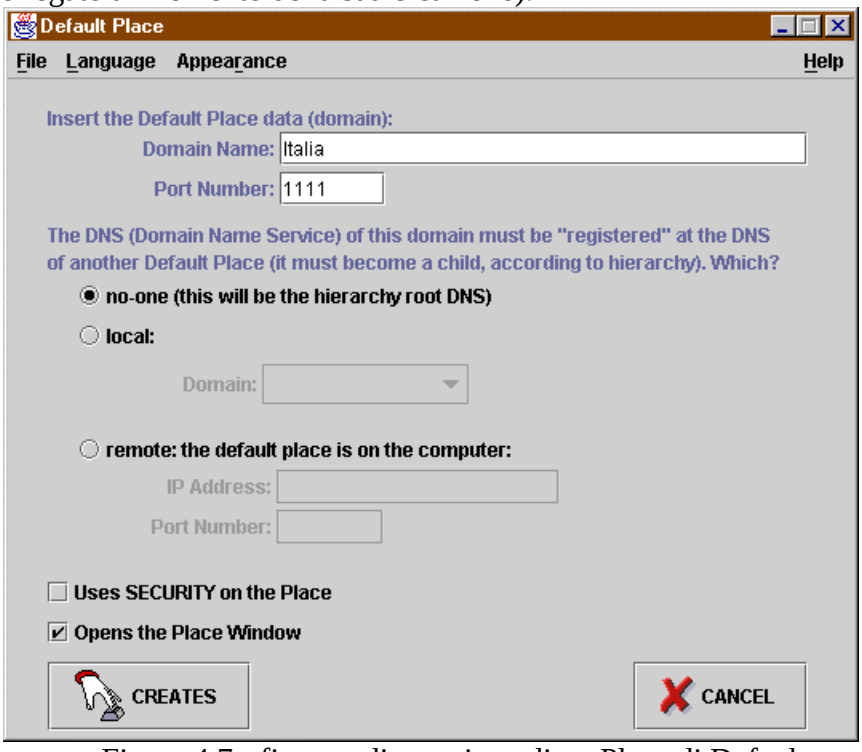


Figura 4.7 - finestra di creazione di un Place di Default

Tramite i due "check-box" nella parte bassa è possibile usare la "**sicurezza**" sul Place (cioè anziché creare un Place normale, verrà creato un "**Place sicuro**") e aprirne la finestra dopo la creazione.
Per esempio, creiamo un Place non sicuro chiamato "Italia" che comunica sulla porta 1111 (ma per il momento disabilitiamo l'apertura della finestra). Dopo un po' (dipendente sia dalla velocità del computer sia dalla latenza della rete) la finestra d'inserimento dati sparirà e si ritornerà a quella di "configurazione avanzata".

4.3.2 Creazione di un Place in un dominio

Ora creiamo un Place sotto il dominio appena creato; per farlo premiamo il secondo bottone; si aprirà la seguente finestra.

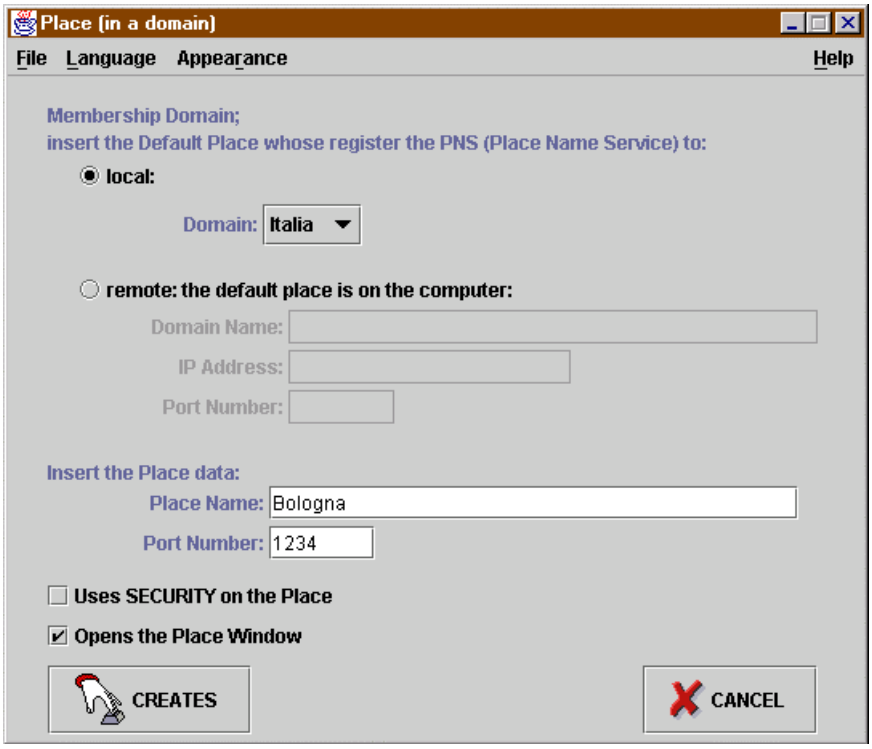


Figura 4.8 - finestra di creazione di un Place in un dominio

Come prima cosa vengono richiesti i dati per la “registrazione”, ma stavolta si tratta di registrarsi all’interno di un dominio; perciò bisognerà inserire i dati relativi a un Place di Default già esistente (per es., il dominio “Italia” creato poc’anzi sullo stesso computer).
Più sotto viene richiesto il nome del Place (ad es., Bologna) e il suo numero di porta (per es., 1234); i due check-box sottostanti sono identici a quelli già visti nella precedente finestra.

4.3.3 Creazione di un Place di Default “Sicuro”

Alla stessa maniera creiamo un Place di Default “**sicuro**” (basta attivare il check-box chiamato “sicurezza”) a cui diamo il nome di “Inghilterra” e che “registriamo” (nella gerarchia di DNS) sotto il dominio “Italia” (basta attivare la voce “local” e scegliere “Italia”).
Ecco la schermata d’inserimento dati:

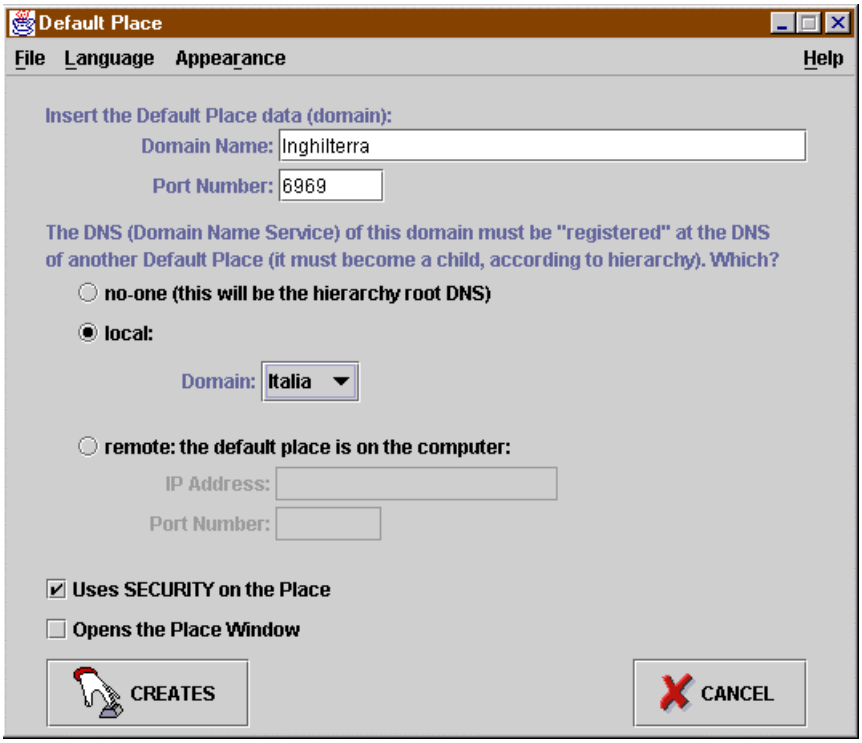


Figura 4.9 - finestra di creazione di un Place di Default "Sicuro"

4.3.4 Creazione di un “Place Mobile”

Sempre per scopo dimostrativo, proviamo a creare un “Place Mobile” premendo il terzo bottone nella finestra “Configurazione Avanzata”.
Apparirà una finestra molto simile a quella per la creazione di un Place all’interno di un dominio (Figura 4.10); chiamiamolo “Londra” e registriamolo sotto il dominio “Inghilterra”.

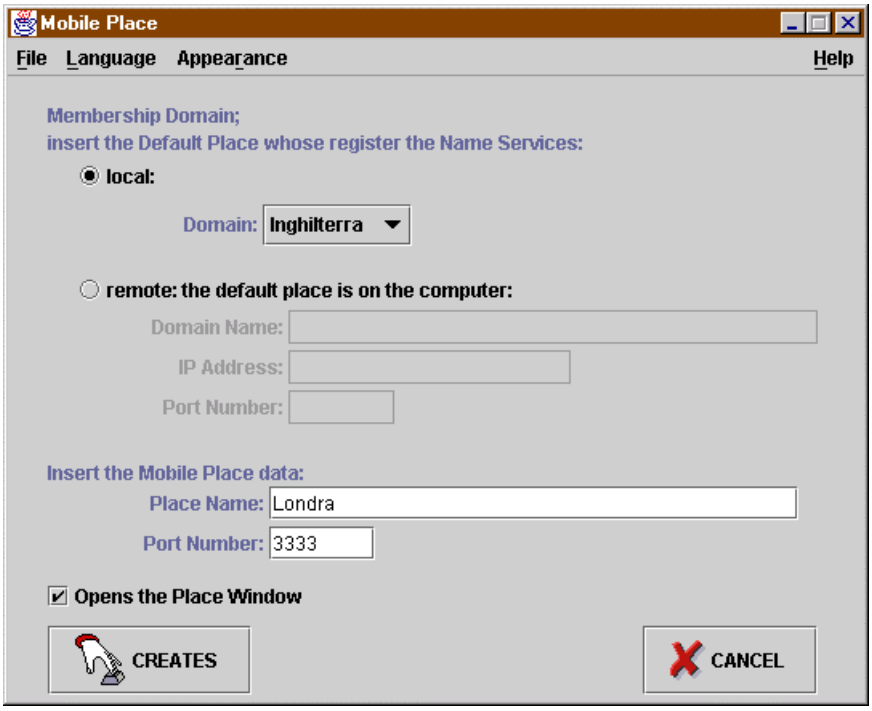


Figura 4.10 - finestra di creazione di un Place Mobile

4.3.5 I Bottoni di “Configurazione Avanzata”

Ora ci troveremo nuovamente nella finestra di “configurazione avanzata”, però nel pannello di sinistra saranno visualizzati i Place che sono stati creati localmente. Per “selezionarne” uno basta muovere il puntatore del mouse sul nome del Place e premere il bottone; il nome scelto verrà riportato in una casella di testo nella parte destra e tutti i bottoni sotto di essa verranno abilitati: è ora possibile fare un’azione “rapida” sul Place selezionato.

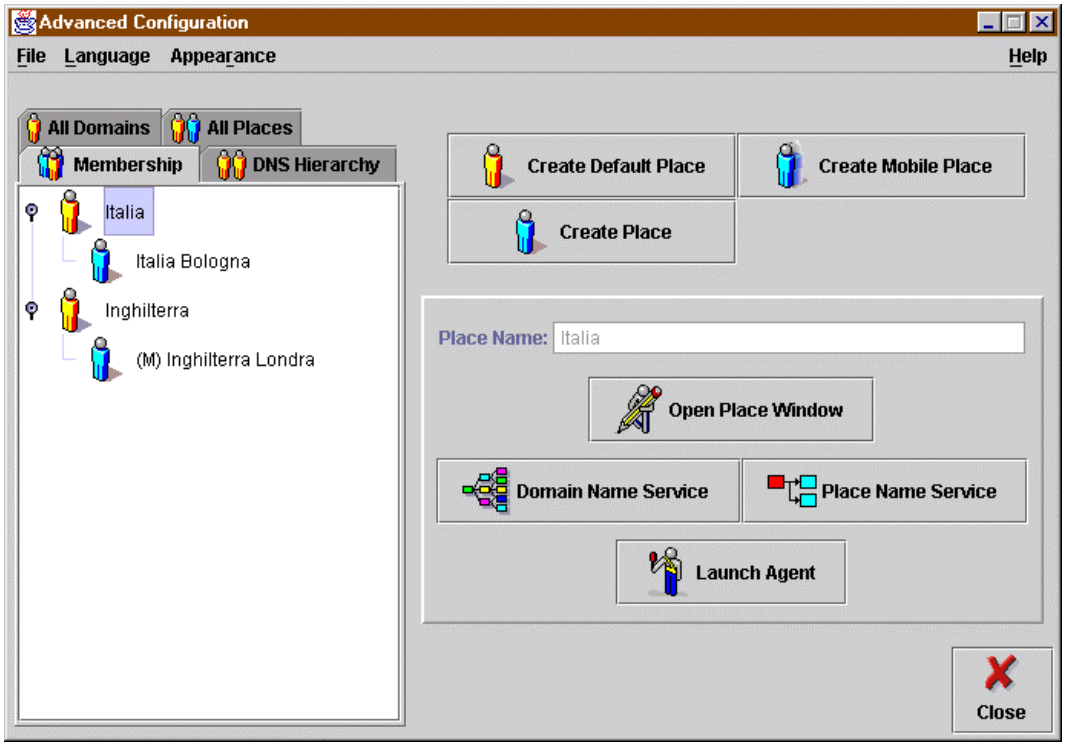


Figura 4.11 - la "configurazione avanzata" con i Place creati

I bottoni permettono di eseguire delle funzionalità che ritroveremo nella “finestra di Place” (e quindi le vedremo più avanti) eccetto la prima, che serve per aprire proprio la finestra di Place (nel caso tale finestra sia già aperta, verrà portata in primo piano).
Inoltre si può scegliere la “vista” usata dal pannello a sinistra:

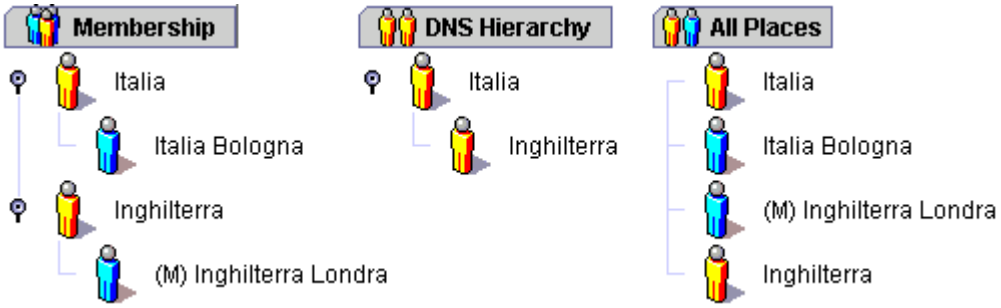


Figura 4.12 - alcune "viste" del pannello dei Place

4.4 LA FINESTRA DI PLACE

Tramite la “Finestra di Place” si possono eseguire molte azioni su un certo Place (quello a cui corrisponde la finestra). Selezioniamo per esempio il Place di Default “Italia”; ecco come appare la sua finestra:

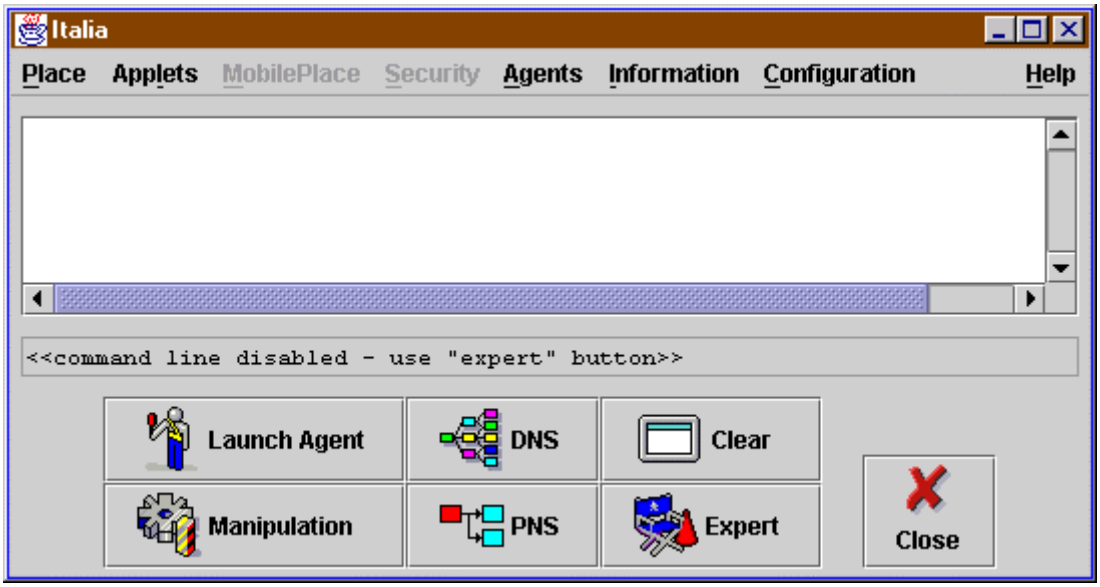


Figura 4.13 - la finestra di Place

Tutte le azioni che possono essere fatte sul Place sono accessibili attraverso la barra di menu posta nella parte alta della finestra; in compenso le funzionalità “più usate” sono state riportate nella parte bassa sottoforma di bottoni. Nella parte centrale c’è un’area di testo su cui verrà stampato lo “standard output” associato al Place (ogni Place ha dei suoi flussi di I/O) mentre lo “standard input” è stato ridiretto sulla sottostante linea di testo.
Se si usa lo standard input si andrebbe a usare il “menu a linea di comando” di SOMA; poiché si è scelto di privilegiare l’uso della GUI, per default lo standard è disabilitato (infatti la frase contenuta nella linea di testo ci avvisa proprio di questo). Per abilitare l’uso di tale linea (e quindi accedere tramite il menu a linea di comando) basta premere il bottone “**expert**”. La modalità “esperto” serve proprio per quegli utenti che hanno una buona familiarità col sistema; infatti il menu a linea di comando non è “user friendly” e contiene molte voci utili solo per il debug delle varie funzionalità).

4.4.1 Il menu a cascata

La seguente figura mostra tutte le voci presenti nel menu a cascata della finestra di Place; in esso vengono mostrate come “attive” tutte le voci, ma non è detto che lo siano sempre. Per esempio, per i Place non di Default la voce “apri finestra DNS” sarà disabilitata; il menu sicurezza è disabilitato per i Place “non sicuri” mentre per quelli “sicuri” sarà attivata all’inizio la sola voce “Certification Authority”.

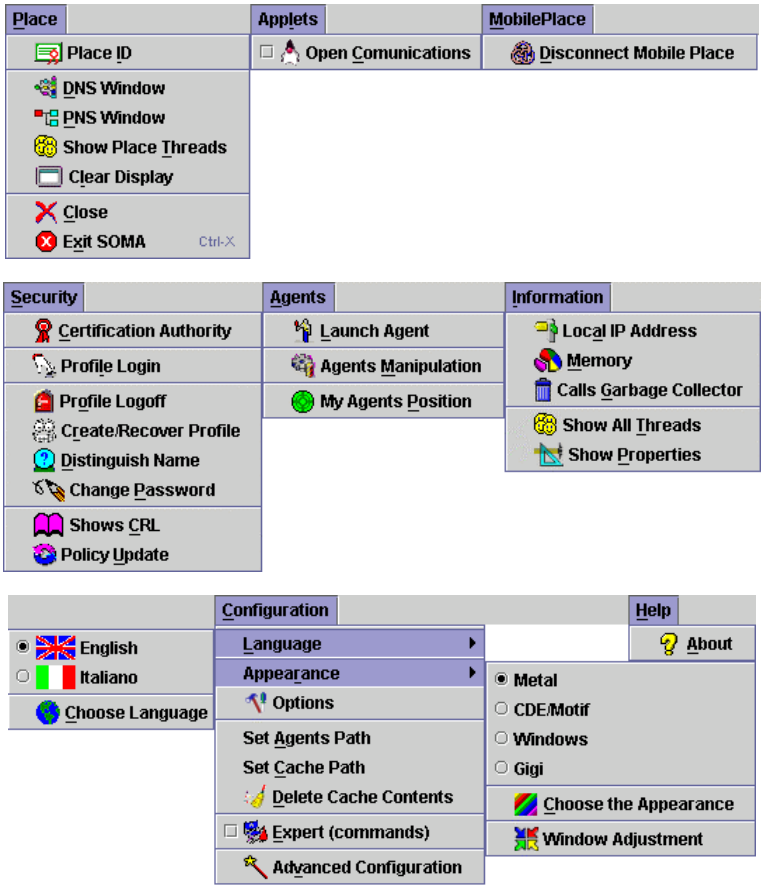


Figura 4.14 - tutte le voci dei menu delle finestre di Place

4.4.2 La finestra di DNS

A questo punto possiamo aprire la finestra di DNS (Domain Name Service) e eventualmente eseguirci delle operazioni. Ecco la finestra:

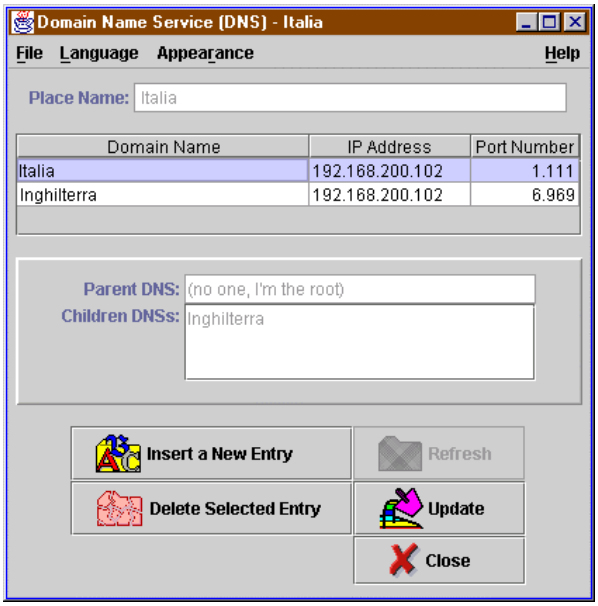


Figura 4.15 - la finestra di DNS

Nella parte centrale ci sono le informazioni di “parentela” del DNS.
Con i bottoni nella parte bassa si può inserire una nuova entrata, cancellare quella che si è selezionato nella tabella, rinfrescare la tabella (la si richiede al Place possessore) e aggiornare il contenuto della finestra.

4.4.3 Lancio di un agente mobile

Ma proviamo subito a lanciare un “agente mobile”. Possiamo farlo sia dalla finestra di Place sia da quella di “configurazione avanzata” (e nel secondo caso si può scegliere il Place di lancio); si aprirà la finestra mostrata in Figura 4.16. Nel “combo-box” al centro c’è il nome della classe dell’agente da lanciare e nella linea di testo sottostante si possono inserire i parametri da passare all’agente all’avvio.

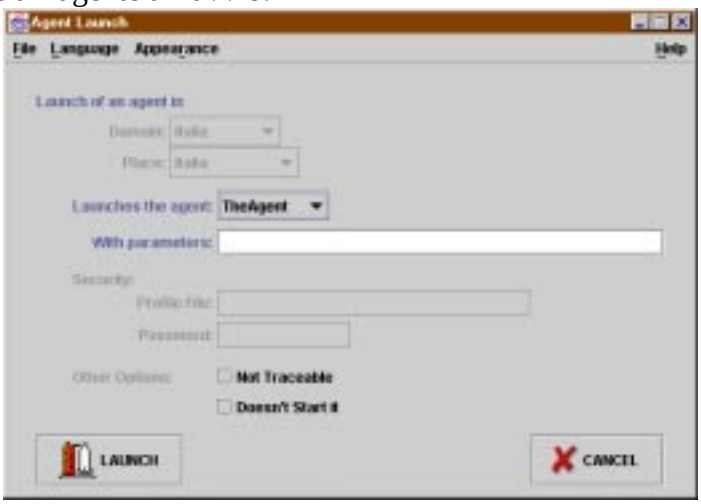


Figura 4.16 - finestra per il lancio di un agente

Il Place su cui lanciare l'agente può essere scelto solo se la finestra viene aperta da "configurazione avanzata" (è solo tale finestra che sa quali Place sono stati creati localmente). Invece, i parametri di sicurezza saranno abilitati solo se si sta usano un "Place sicuro". Ecco la finestra "tutta abilitata":

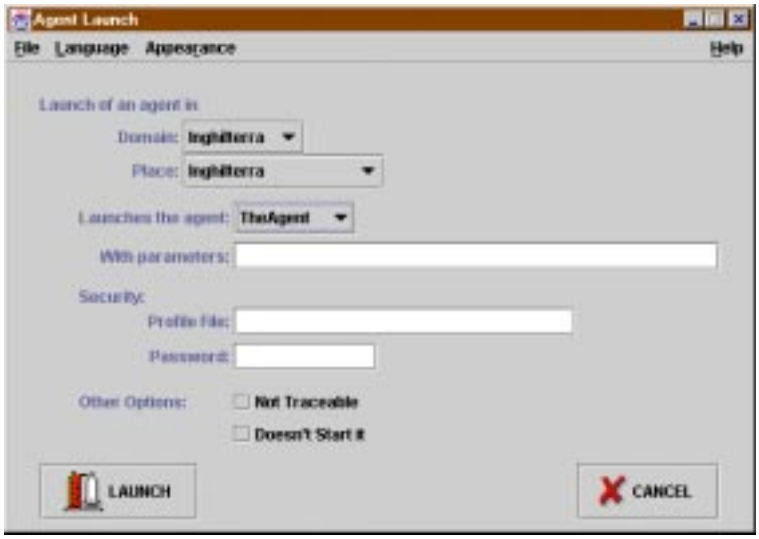


Figura 4.17 - lancio di un agente da "configurazione avanzata"

Proviamo a lanciare l'agente che abbiamo creato a scopo dimostrativo, chiamato "TheAgent". Ecco come ci apparirà (si noti che il nome dell'agente è composto da quello del Place su cui è stato creato).

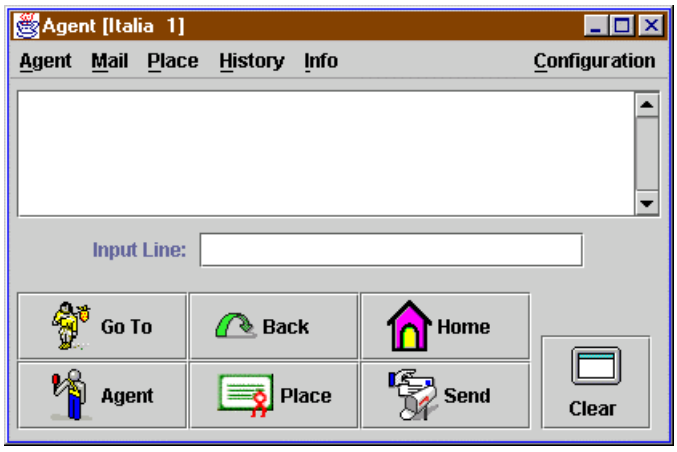


Figura 4.18 - l'agente "TheAgent"

4.4.4 Manipolazione degli agenti

Ora torniamo nella finestra di Place e apriamo la finestra chiamata "manipolazione agenti"; tramite essa si possono eseguire operazioni su un qualsiasi agente in esecuzione nel Place corrente.

Ecco come ci apparirà:

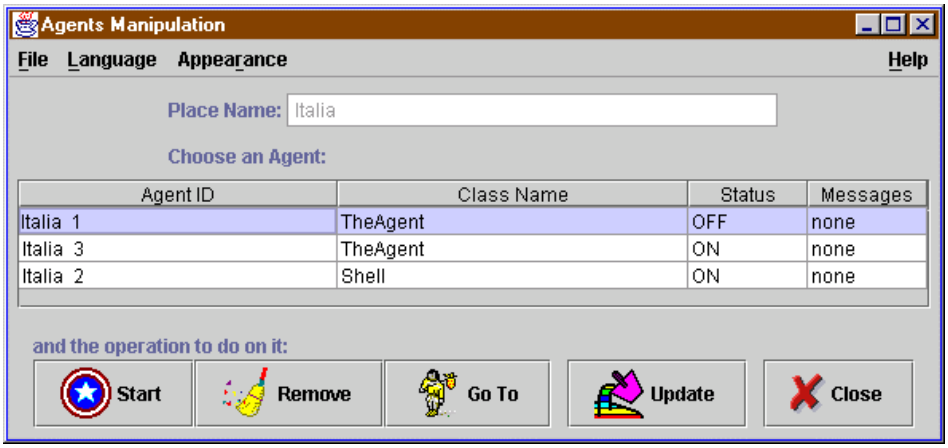


Figura 4.19 - finestra di "manipolazione agenti"

Basta selezionare un agente dalla tabella (semplicemente movendoci sopra il puntatore e premendo il tasto del mouse) e premere uno dei bottoni in basso. A seconda dello “stato” dell’agente alcuni bottoni possono o meno essere “disabilitati”; per es., se l’agente è in esecuzione, prima che possa essere rimosso occorrerà “fermarlo” (il bottone a sinistra commuta tra “start” e “stop”) e di conseguenza il tasto “rimuovi” sarà disabilitato.
Nel caso si scelta l’operazione di “go to”, si aprirà la seguente finestra, in cui si può scegliere il Place verso cui migrare.

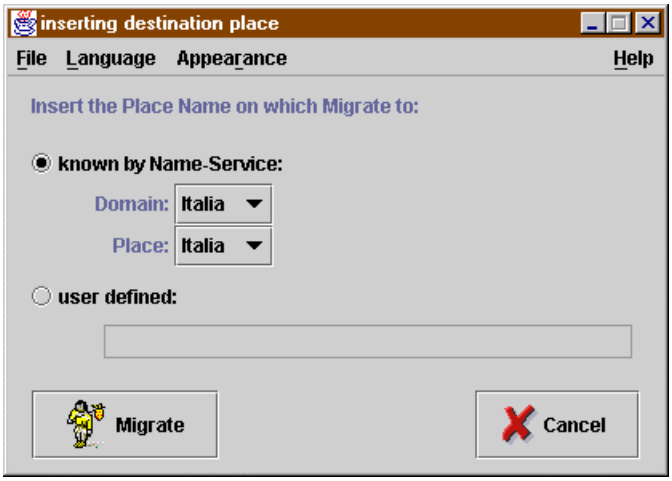


Figura 4.20 - finestrella di "go to"

Nei combo-box in alto si può scegliere un Place “conosciuto” (cioè il cui nome è contenuto nel DNS e/o nel PNS del Place corrente) mentre nella linea di testo in basso può essere indicato un Place non noto (per es., se il Place corrente non è di Default e si vuole fare una migrazione verso un altro dominio). Proviamo a farlo migrare in “Italia Bologna”.

4.4.5 Posizione degli agenti

Possiamo sempre sapere dove si trova un nostro agente aprendo la finestra di “posizione agenti” (accessibile dal menu “agenti” della finestra di Place). Si aprirà la seguente finestra:

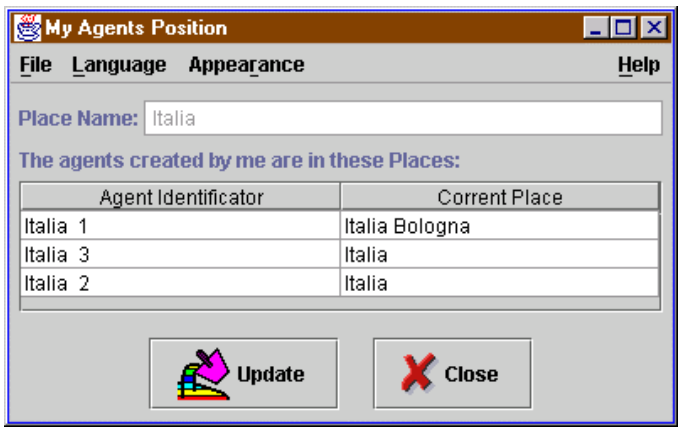


Figura 4.21 - finestra di "posizione agenti"

Come si vede, l’agente “Italia 1” si trova in “Italia Bologna”, cioè la migrazione che abbiamo forzato poc’anzi è andata a buon fine.

4.5 USO DI UN PLACE MOBILE

Per poter funzionare un “Place Mobile” ha bisogno di essere connesso a un Place di Default. Per farlo basta attivare la voce “connetti” del menu “MobilePlace” della finestra di Place; una volta fatto questo, apparirà una finestrella molto simile a quella vista per la migrazione di un agente ma che permette di inserire un nome di Place di Default.

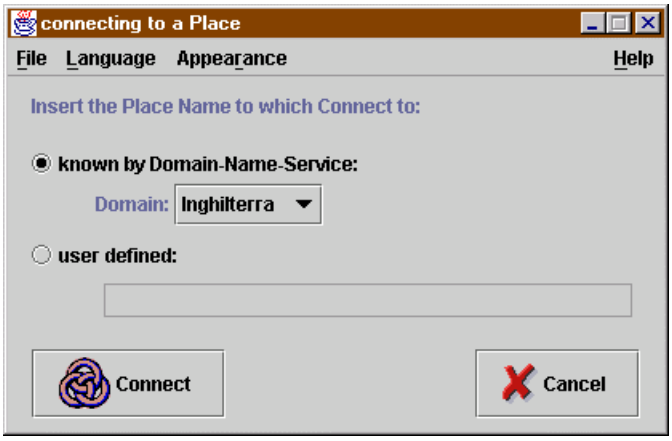


Figura 4.22 - richiesta di connessione per un Place Mobile

Una volta connesso, un “Place mobile” si comporta come qualsiasi altro Place, a eccezione del fatto che al momento della connessione può avvenire un “ritorno” degli agenti che aveva messo in esecuzione prima della precedente disconnessione.
Una particolarità del Place mobile è proprio quella di poter lanciare degli agenti (quindi, delle operazioni) e recuperare i risultati solo in un secondo momento, quando si riconnetterà (anche a un Place di Default diverso da quello precedente).

4.6 USO DI UN PLACE SICURO

Per un “Place Sicuro” (di Default o meno) è abilitato il menu “sicurezza” della finestra di Place. Ogni utente del sistema ha un proprio “profilo” e una “password” d’accesso; entrambi questi dati devono essere specificati prima che possa essere fatta qualsiasi azione.
Innanzitutto occorre collegarsi a un’autorità di certificazione; per farlo basta richiamare la voce “certification authority” del menu “security”.

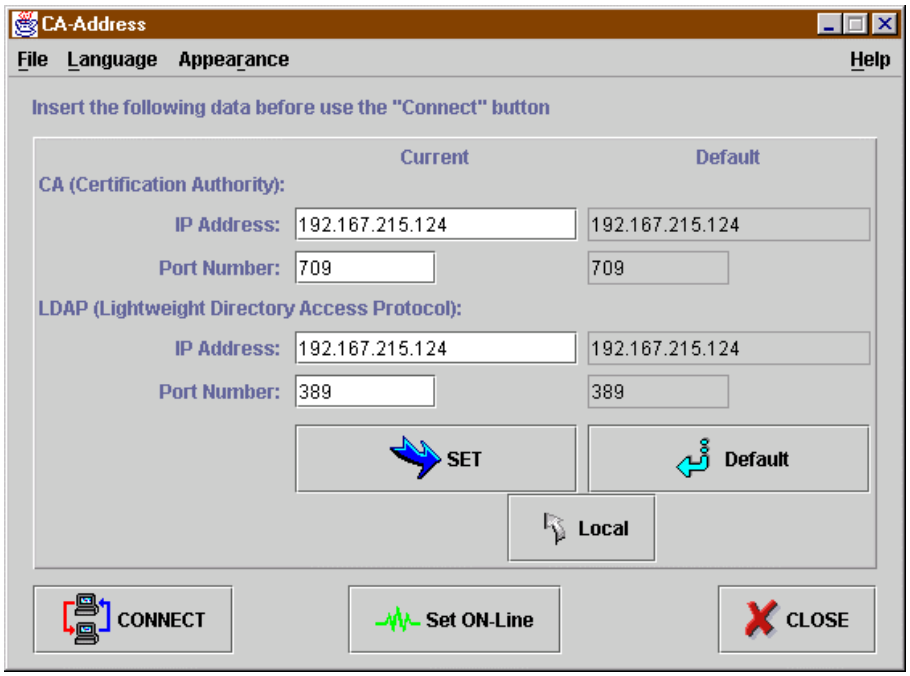


Figura 4.23 - finestra per la definizione del CA

Una volta definito il CA e il LDAP (per default c’è il riferimento a un PKI installato al DEIS di ingegneria di Bologna), si può premere il bottone “connect”: soltanto ora verrà abilitata la voce “login” del menu della finestra di Place. Chiamiamola e ci verrà richiesto qual è il profilo del Place (cioè il nome di un file su disco) e la password.

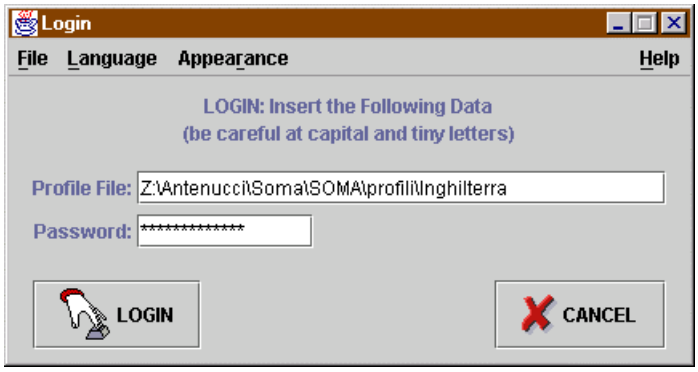


Figura 4.24 - finestra per il login del Place

Una volta fatto il “login”, il Place avrà aperto il proprio profilo crittografico e avrà in mano le chiavi e i certificati.
Solo adesso verranno abilitate le restanti opzioni del menu “sicurezza”, tra cui quella di “logoff” (che le disabiliterà nuovamente e riabiliterà quella di “login”).
Per esempio ora possiamo creare un nuovo profilo aprendo la finestra:

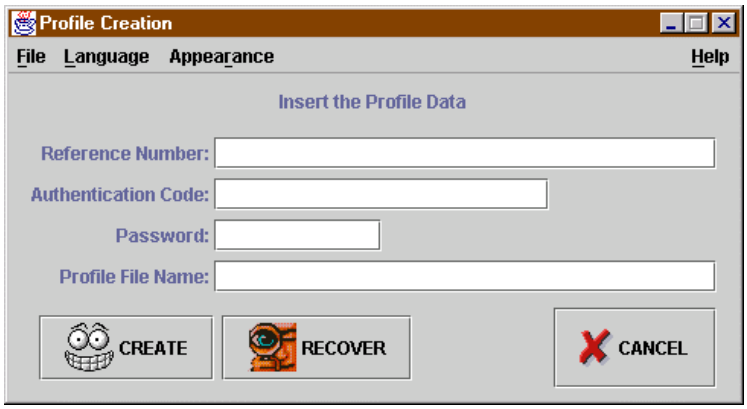


Figura 4.25 - finestra per la creazione di un nuovo profilo

Oppure si può cambiare la propria password con quest’altra finestra:

Figura 4.26 - cambio della password

Se ora proviamo a lanciare un agente, occorrerà dare il proprio profilo (dell'utente, questa volta, non del Place) e la propria password.
Per esempio, si veda la seguente schermata.

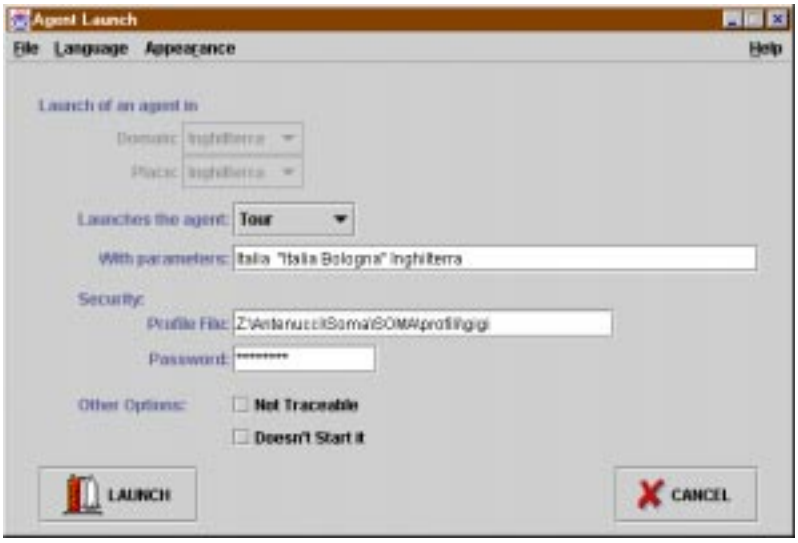


Figura 4.27 - lancio di un agente dando profilo e password

Una volta inserito, il nome del profilo verrà "ricordato" dalla GUI; cioè verrà automaticamente riportato in tutte le nuove finestre di lancio. Per ragioni di sicurezza, la password va inserita ogni volta.

4.7 L'AGENTE "THEAGENT"

Abbiamo creato un agente mobile a scopo dimostrativo; esso mette a disposizione dell'utente tutte le possibili funzionalità di un qualsiasi agente (compreso l'invio di messaggi a altri agenti).
Abbiamo già visto nella Figura 4.18 come ci appare a video.
Se proviamo a eseguire la migrazione (operazione "go to") ci apparirà la stessa finestra vista in "manipolazione agenti" (paragrafo 4.4.4).
Scelto il Place di destinazione, si può far migrare l'agente; una volta a destinazione verrà mostrato il tempo impiegato per la migrazione.

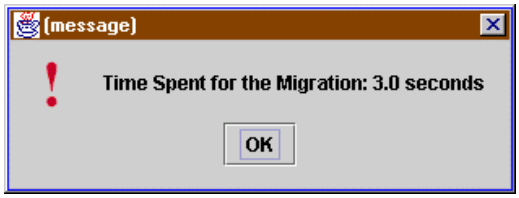


Figura 4.28 - tempo di migrazione

Supponiamo che (in un qualche Place) ci sia un agente chiamato "Inghilterra 5". Per mandargli un messaggio basta premere il bottone "send"; apparirà la seguente finestra:

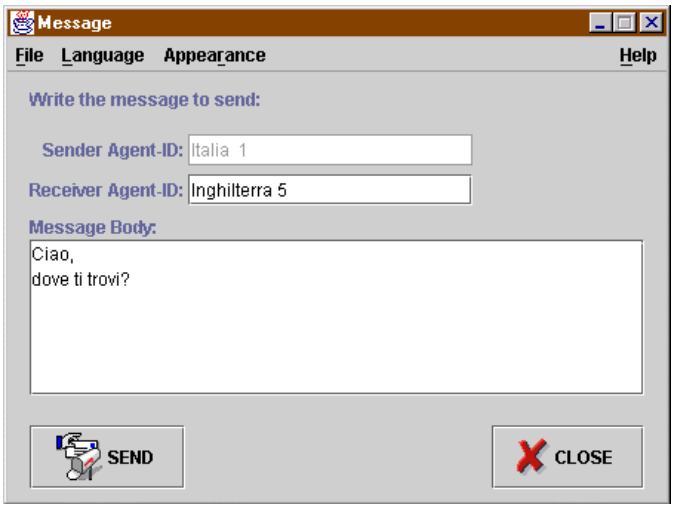


Figura 4.29 - invio di un messaggio a un agente

Il destinatario verrà avvisato dell'arrivo del messaggio e potrà leggerlo tramite la voce "check mail" del menu. Apparirà la finestra con il messaggio arrivato a cui potrà eventualmente "rispondere".

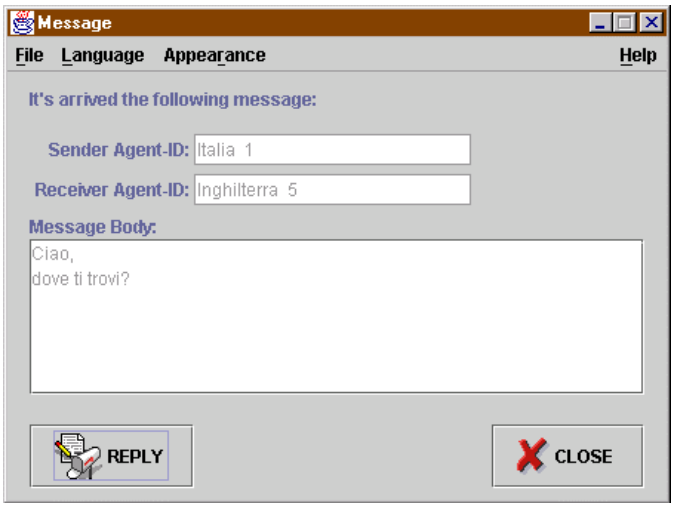


Figura 4.30 - lettura del messaggio arrivato

Se si decide di rispondere apparirà una finestra molto simile a quella di "invio" che però contiene il messaggio arrivato e permette di aggiungere del testo.

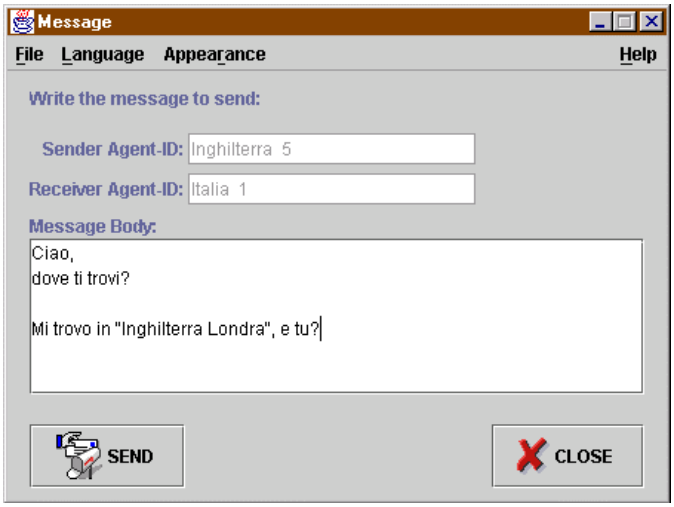
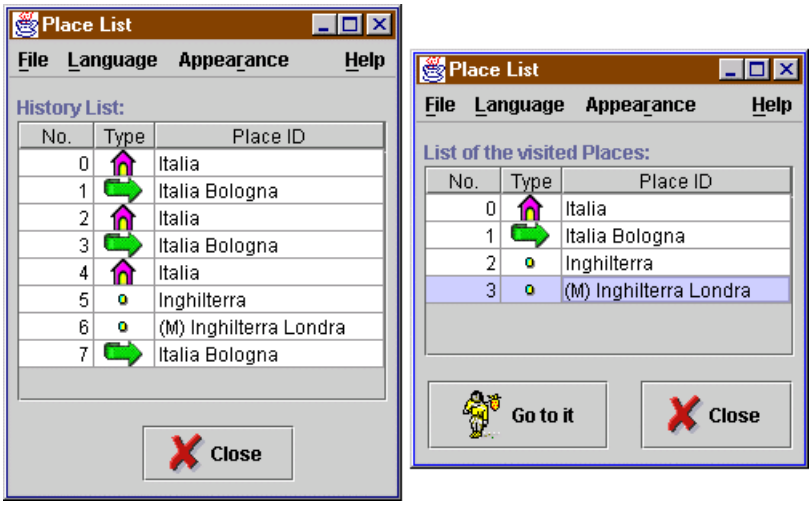


Figura 4.31 - scrittura della risposta a un messaggio

Tra le altre funzionalità dell'agente c'è la visualizzazione della "storia" di migrazione, cioè tutti i Place che ha attraversato. Oltre alla sola visualizzazione c'è anche la possibilità di ritornare su un Place visitato. Basta scegliere il Place e premere il bottone "go to it".



4.8 COMUNICAZIONI CON LE APPLET REMOTE

Sappiamo già che a un Place possono arrivare dei comandi da particolari applet: le "AppletPlace". Per default però il Place non accetta tali comandi; per permettere la comunicazione con queste Applet occorre attivare la voce "apre comunicazioni" nel menu della finestra di Place. Ci apparirà una finestra in cui viene richiesto il numero di porta sulla quale il Place si metterà a ascoltare messaggi (il valore predefinito è 5555).

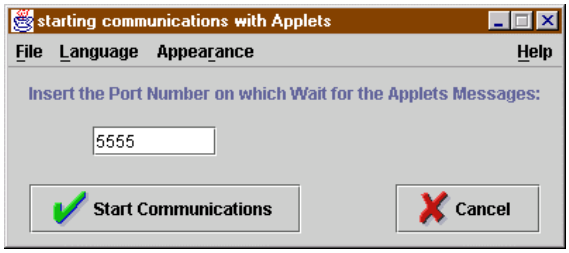


Figura 4.32 - scelta della porta su cui comunicare

Su un qualsiasi computer collegato in Internet può essere lanciata l'applet "AppletPlace" (contenuta in "SOMA.gui.remoteapplet") semplicemente caricando la pagina "AppletPlace.html".
La seguente figura mostra come appare l'applet.

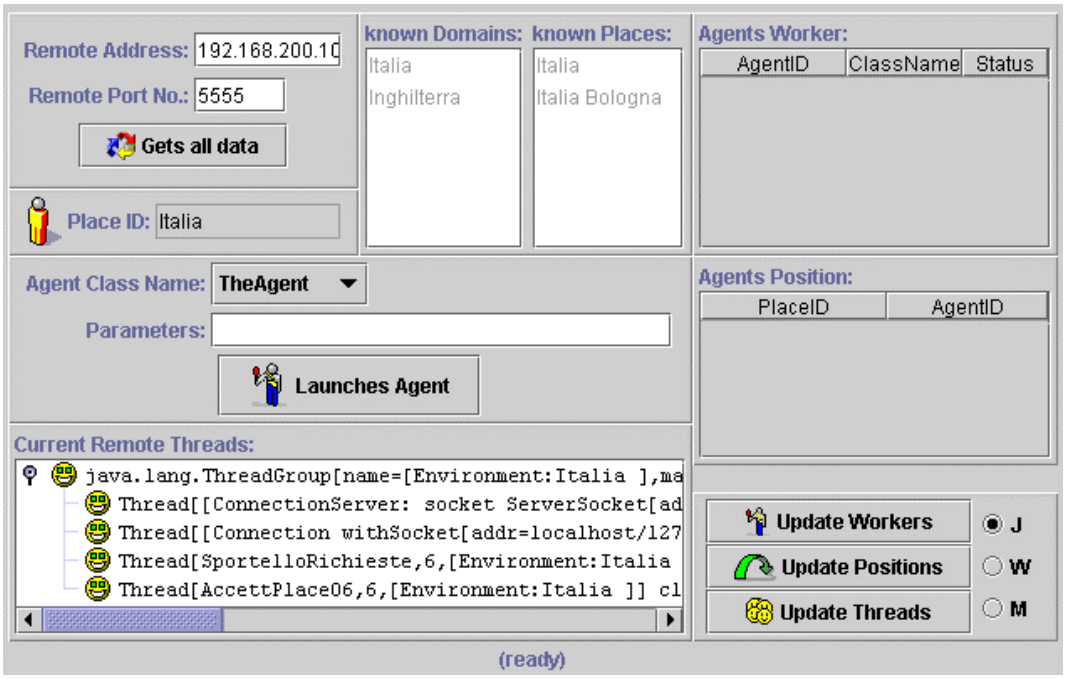


Figura 4.33 - l'applet AppletPlace

Nella parte in alto a sinistra è possibile modificare l'indirizzo IP e il numero di porta del Place a cui l'applet invia i messaggi (per default, viene usato l'indirizzo da cui è stata scaricata l'applet e la porta 5555, ma è possibile specificarli anche tramite parametri passati all'applet all'interno del file "html"). Il bottone sottostante serve per "rinfrescare" tutti i dati che sono mostrati sul video (in sostanza, vengono richiesti tutti in blocco al Place).
Nella parte centrale della finestra viene data la possibilità di lanciare un agente nel Place. Ovviamente, una volta lanciato non è possibile "controllarlo" (cioè forzarlo a migrare o ucciderlo) perché non si ha alcuna garanzia che l'utente dell'applet sia un personaggio "fidato" e che quindi non faccia azioni a danno di altri agenti del Place.
Il resto della finestra serve per mostrare le informazioni sullo "stato" del Place:

- i Domain & Place Name Service, nella parte centrale alta;
- l'elenco dei "worker" (le entità che mettono in esecuzione gli agenti) in alto a destra;
- la posizione corrente degli agenti creati dal Place, poco sotto;
- l'albero con l'elenco dei thread appartenenti al "gruppo" di cui fa parte il Place, nella parte bassa.

In basso a destra ci sono dei bottoni che permettono l'aggiornamento di una sola componente per volta dello "stato" visualizzato (elenco worker, posizione agenti e elenco thread).
Poco più a destra ci sono tre "radio-button" che servono per impostare dinamicamente il "look & feel" (java, windows e motif).
Proviamo a lanciare un agente (per es., "TheAgent") e vedremo che lo "stato" del Place cambierà; la finestra riporterà dei dati simili a questa:

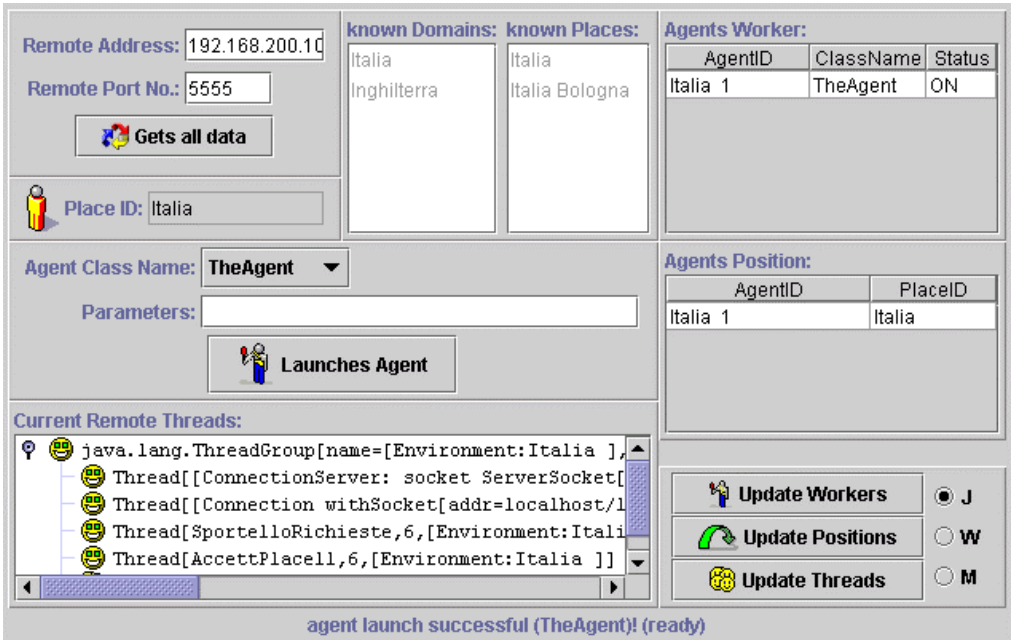


Figura 4.34 - modifica della schermata dopo il lancio di un agente

4.9 SALVATAGGIO DELLA CONFIGURAZIONE

Chiudiamo l'applet e torniamo al sistema su cui sono stati creati i Place. Ogni modifica che l'utente fa sulla GUI viene "memorizzata" in un file su disco (cioè viene mantenuta in modo permanente) così che possa essere "recuperata" in un successivo momento (anche dopo la chiusura del sistema).
In particolare, in ogni momento il sistema si ricorda la *lingua* e l'*apparenza* che sta usando la GUI. Inoltre, vengono "ricordati" anche tutti i *dati inseriti* da utente (come l'ultimo nome selezionato sul combo-box di lancio di un agente o i parametri passati all'ultimo agente lanciato). In più, anche la *posizione di ogni finestra* viene "memorizzata", in modo che possa essere successivamente riaperta nello stesso punto (e con le stesse dimensioni).

Tutti questi dati sono creati e gestiti internamente alla GUI; l'utente non ha bisogno di conoscere dove sono memorizzati, però può volere "ripristinarli" al valore di default e può farlo tramite la finestra di "opzioni della GUI".

4.10 CAMBIO DI LINGUA E DI APPARENZA

Tutte le finestre sono state progettate in modo da mantenere separati i componenti grafici e le frasi (in una particolare "lingua") che essi possono mostrare a video. Inoltre, ogni finestra viene "avvisata" (usando il meccanismo di delegazione degli eventi di Java) quando si verifica un evento di "*cambio di lingua*".
Abbiamo progettato tutta la GUI in modo che il cambio sia di lingua sia di apparenza possa avvenire **dinamicamente**: quando l'utente decide di fare un cambiamento, esso si ripercuoterà non solo su tutte le finestre future (ancora da creare) ma anche su tutte quelle già create (e mostrate sul video)!
Basta aprire un po' di finestre e su una di queste chiedere un cambio di lingua (tramite l'apposita finestra o meno); su tutte le finestre verrà eseguito il cambio di lingua (allo stesso modo funziona il cambio di apparenza). Le seguenti figure mostrano il "prima" e il "dopo".

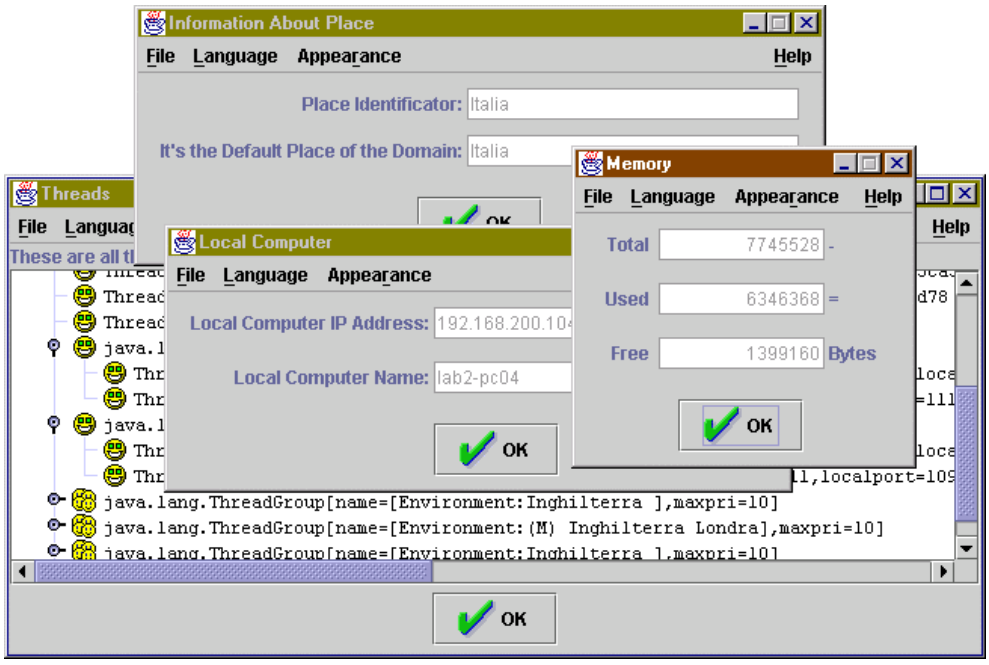


Figura 4.35 - "prima" del cambio di lingua

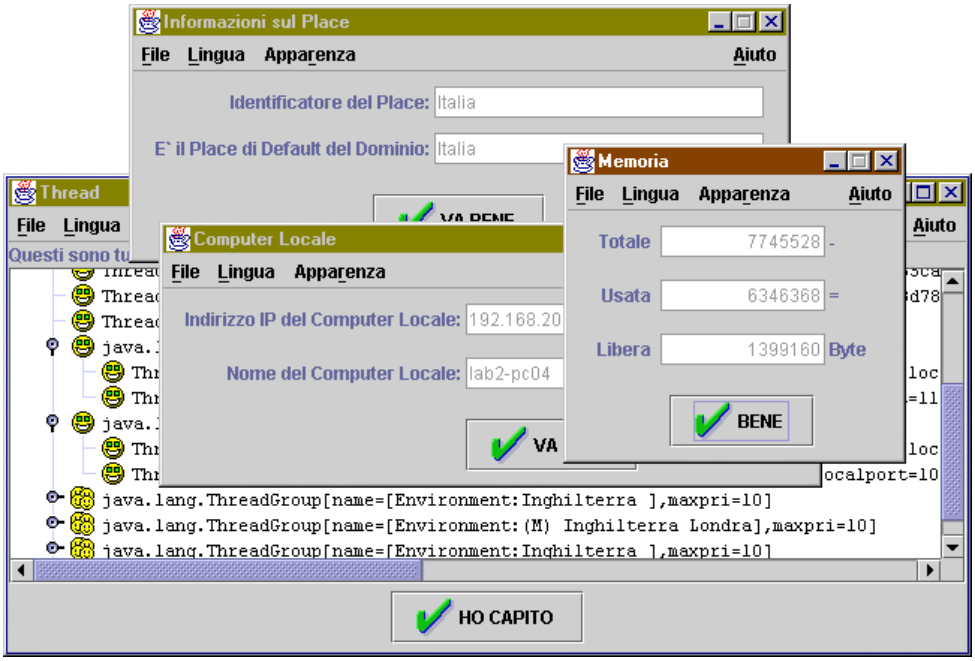


Figura 4.36 - "dopo" il cambio di lingua

Sia il cambio di lingua sia quello di apparenza avvengono tramite le seguenti finestrelle:

Figura 4.37 - richiesta della lingua da usare

Figura 4.38 - richiesta dell'apparenza da usare

4.11 CHIUSURA E RIAVVIO DEL SISTEMA

Ogni volta che sul computer viene creato un nuovo Place, la GUI salverà automaticamente su disco i dati relativi alla creazione (quelli che l'utente inserisce tramite le apposite finestre di creazione).

In questo modo possono essere "recuperati" al momento di un nuovo avvio del sistema. Proviamo a uscire dal sistema (basta chiudere tutte le finestre o usare la voce di menu "esci da SOMA") e a rientrarvi; apparirà subito la seguente finestrella:

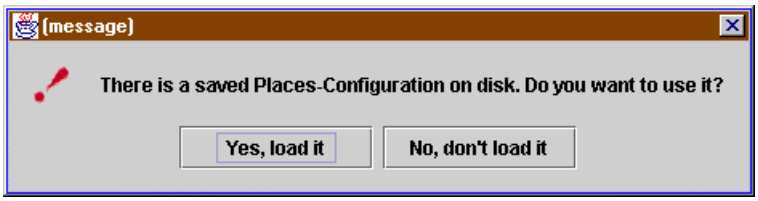


Figura 4.39 - richiesta di caricare la configurazione salvata

Se si risponde “no”, si procederà come se fosse la prima volta che si lancia il sistema; altrimenti apparirà una “barra progressiva” che indica l’avanzamento del processo di creazione di tutti i Place.

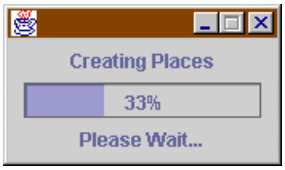


Figura 4.40 - creazione in corso

Quindi apparirà una nuova finestrella che ci chiederà se vogliamo che vengano aperte automaticamente tutte le finestre dei Place creati; a meno che non sia creato un solo Place, verrà aperta anche la finestra di “configurazione avanzata”.
Tutte queste finestrelle di scelta sono “scavalcabili” tramite la definizione di particolari voci nella finestra di “opzioni” (della GUI).

INDICE ANALITICO