ESERCITAZIONE TIGA: Architettura *Applet–RMI–Database*

Relativamente a quanto specificato nelle esercitazioni su *Swing*, *RMI*, e *JDBC*, si realizzi un servizio di conto corrente remoto strutturato secondo un'architettura a tre livelli *JavaApplet–RMI–Database* (Fig.4).

Il *front-end*[1] del sistema è costituito da una applet `BancaClient`, che realizza la *GUI* di Fig.1[2] ed implementa anche la funzione di client RMI invocando i metodi *accredita* (una quantità positiva o negativa) o *saldo* (che può anche essere negativo). Il *middleware*[3] è composto dalle classi `BancaServer`, il server RMI gestore degli oggetti `BancaImpl`, e `BancaImpl` medesima che implementa i servizi ricevendo richieste dei client ed accedendo al database mediante JDBC per soddisfarle. Infine, il *back-end*[4] è il DBMS *MySQL*© (a) oppure *Microsoft Access*© (b), che riceve le richieste da parte degli oggetti `BancaImpl` come statement *SQL* e restituisce i risultati, i quali riattraversano in senso inverso gli strati di software sino alla *GUI*.

Il database `banca` contiene i conti correnti in una tabella `CONTI`, dagli attributi `NOME` e `SALDO` (Fig.2). La classe `BancaImpl` (Fig.3) implementa una interfaccia locale (`BancaLocale`) oltre a quella remota (`Banca`).

Fig. 1 – *Graphical User Interface* del Cliente

CONTI

NOME	SALDO
pippo	100.50
pluto	3200.43
…	…

Fig. 2 - Tabella `CONTI` del database `banca`

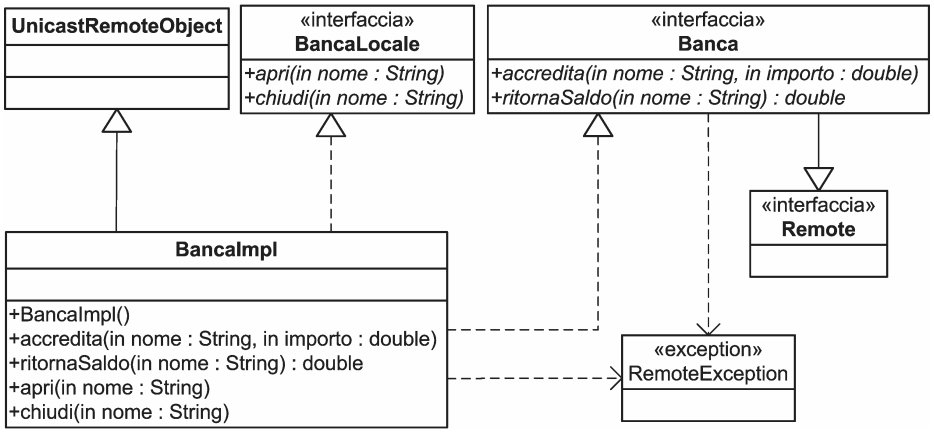

Fig. 3 – Diagramma delle Classi

```
public interface Banca extends Remote {
  void   accredita(String nome, double importo) throws RemoteException;
  double ritornaSaldo(String nome)              throws RemoteException;
}
```

```
public interface BancaLocale {
  void apri(String nome);
  void chiudi(String nome);
}
```

PROGETTARE E REALIZZARE:

Tutti i moduli descritti, semplificando al minimo la gestione delle eccezioni. Tralasciare l'implementazione dell'interfaccia utente locale creando staticamente due conti "pippo" e "pluto" all'interno del server RMI. Per semplicità il database si trova sul medesimo host di residenza di server e registry RMI (Fig.5) i quali devono necessariamente risiedere sul medesimo HTTP server da dove l'applet verrà scaricata[5]. Quindi il client accede ad un unico server, prima da browser mediante `http` per scaricare l'applet e le classi relative (comprese stub ed interfaccia) e poi attraverso `rmi` per comunicare con il registry ed il server.

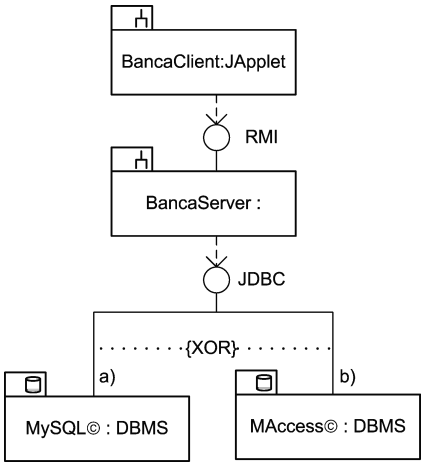

Fig. 4 – Sottosistemi funzionali

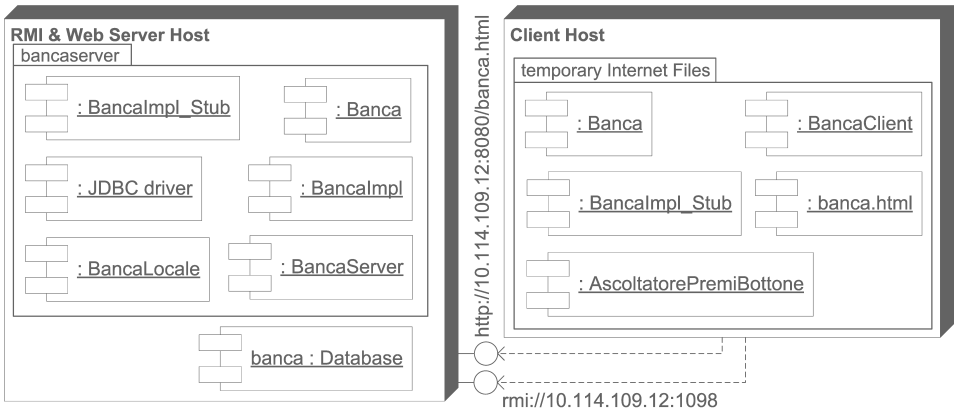

Fig. 5 – Diagramma di distribuzione dei componenti fisici

[1] *Front-end*, letteralmente "frontale", "all'estremità più vicina a chi usa l'applicazione" e quindi "che fornisce un'interfaccia" o "che esegue funzioni di comunicazione con dispositivi esterni".

[2] È possibile avere importi e saldi negativi. Premendo *accredita* viene sommato l'importo inserito e restituito il saldo nella medesima casella di inserimento.

[3] *Middleware*, letteralmente "parte intermedia" , in generale é uno strato di software tra la rete e le applicazioni, che provvede a servizi come identificazione, autenticazione, autorizzazione, direttorio, sicurezza.

[4] *Back-end*, letteralmente "posteriore", "all'estremità più lontana da chi usa l'applicazione", o anche "riservato per compiti determinati o secondari che non occorre mostrare all'utente".

[5] Per motivi di sicurezza un'applet non può stabilire connessioni con altri host diversi da quelli da cui è stata scaricata, a meno che essa non venga "firmata" e quindi acquisisca maggiori privilegi sugli host che ne riconoscono la firma. Dunque nel presente esercizio si trascuri il caricamento dinamico di classi da un terzo http server.