Università della Calabria – Facoltà di Ingegneria – Corso di laurea in Ingegneria Informatica

Ingegneria del Software

aa 2006/2007

Aste On Line

Relazione di Progetto

Studente

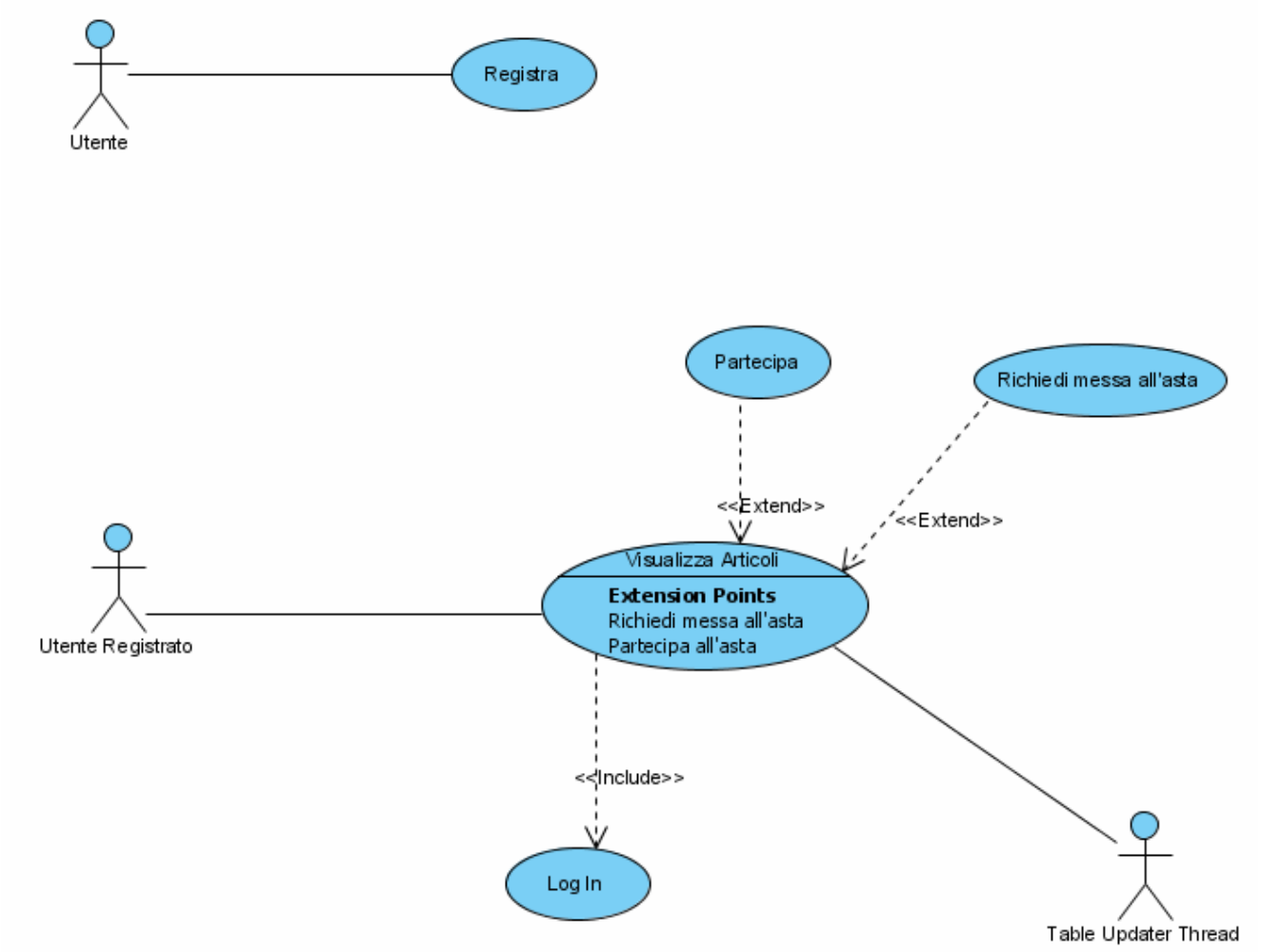
Falace Gabriele

matricola 74956

Indice

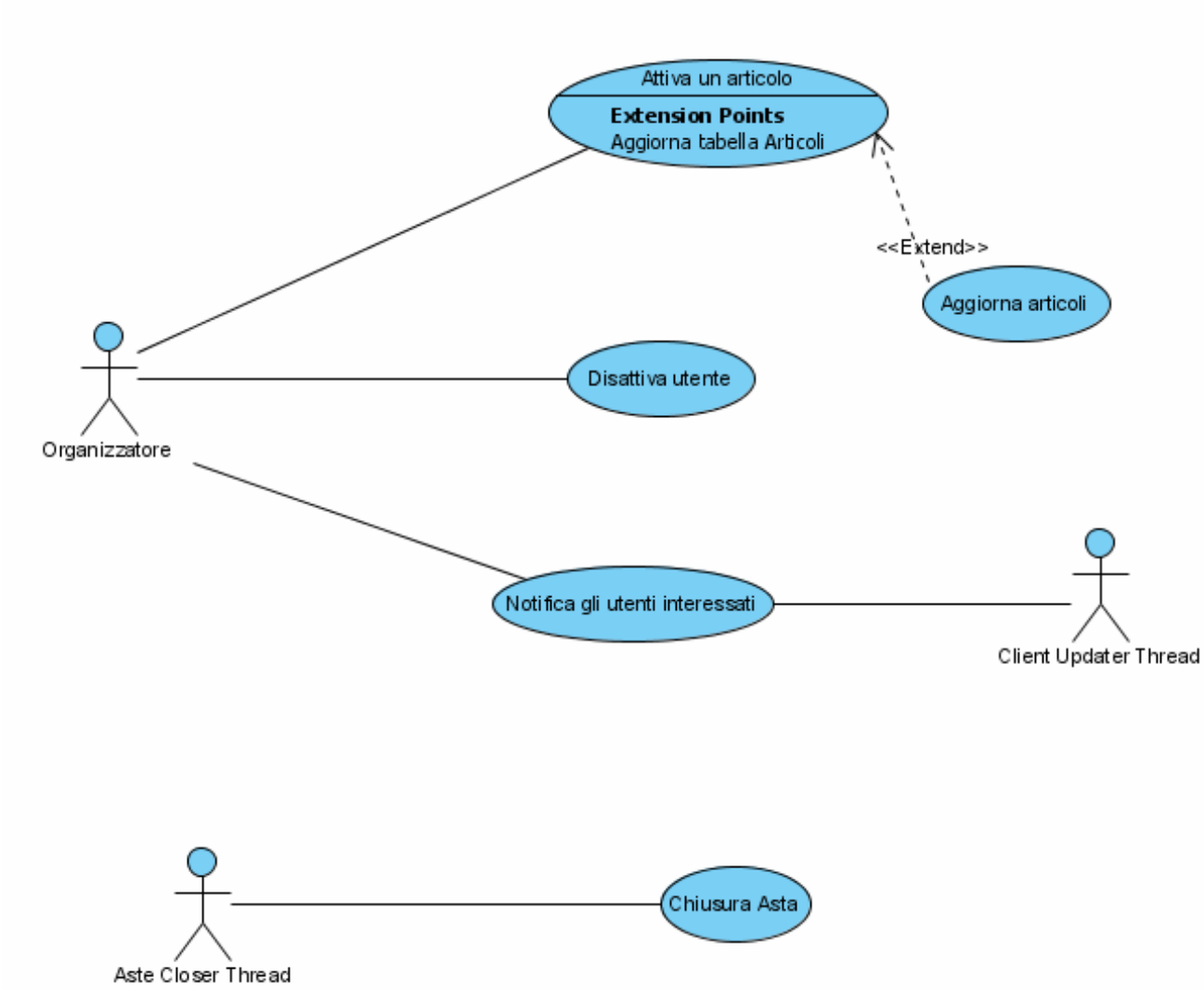
Progettazione dei casi d’uso

Utilizzo da parte dell’ utente



Come si evince da questi use case, è previsto che l’utente si registri prima di poter accedere al sistema. Qualora un utente sia registrato, può accedere al sistema effettuando la log-in. L’utente potrà visualizzare un elenco degli articoli di suo interesse attualmente messi all’ asta. Una volta acceduto, l’utente può scegliere un articolo per cui fare una offerta o può compilare un form per richiedere la messa all’asta di un proprio articolo.
Viene messo in evidenza che un’ altra entità (un apposito Thread) deve mantenere aggiornata automaticamente la tabella degli articoli presso l’utente, ogni volta che un articolo viene aggiunto e il server lo notifica.

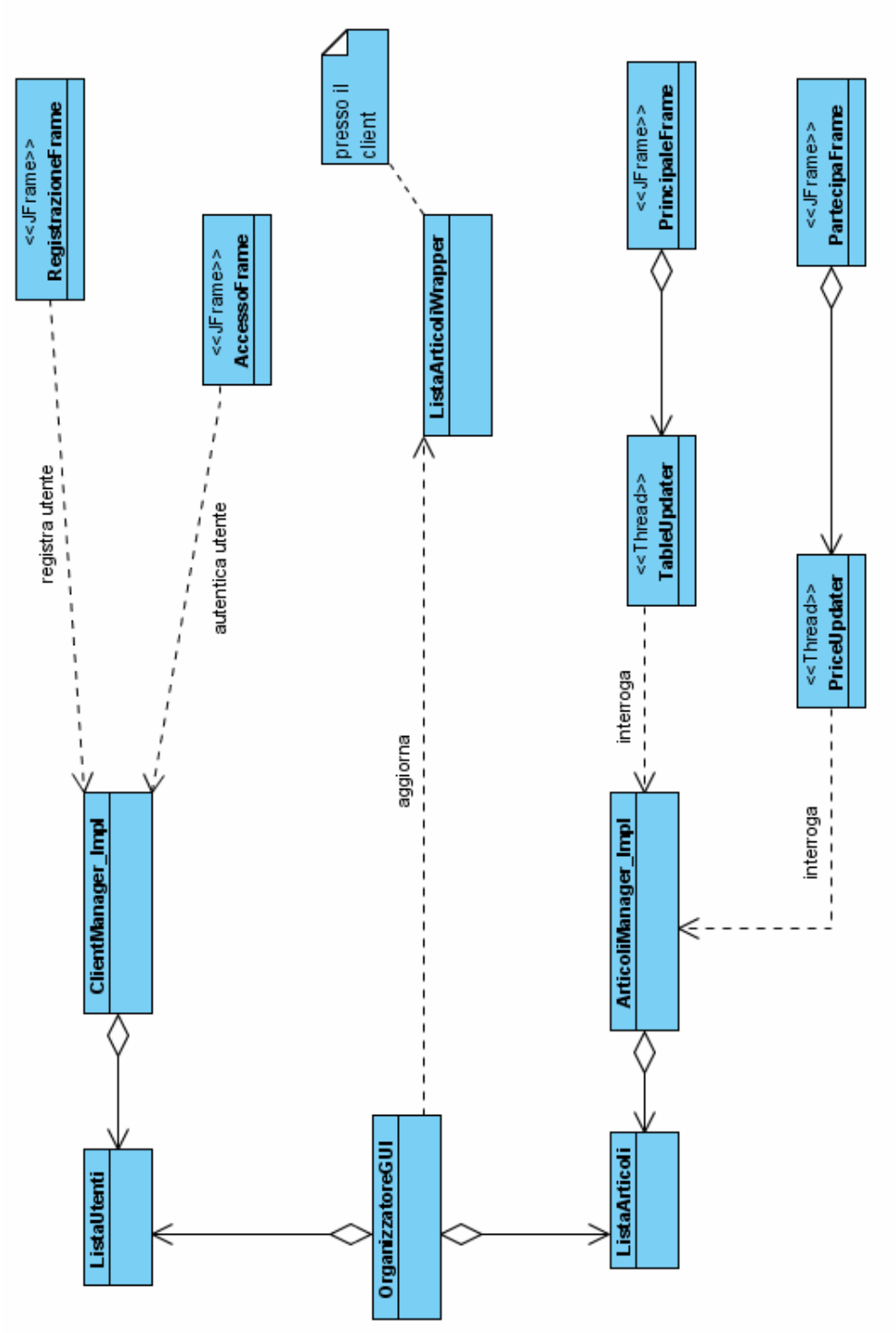
Utilizzo da parte dell' organizzatore



L' organizzatore si occupa di attivare e disattivare articoli, di disattivare eventualmente gli utenti e di notificare gli utenti di eventuali aste di loro interesse. Per l'organizzatore si è scelto di non effettuare gli aggiornamenti delle tabelle di utenti e articoli in modo automatico come avviene nel client, ma è l'organizzatore, appunto, che effettua l'aggiornamento. Questa scelta è dovuta al fatto che l'organizzazione di un' asta non avviene molto di frequente, per cui l'organizzatore può, senza particolare fatica, fare gli aggiornamenti solo a tempo di preparazione dell' asta. In alternativa si sarebbe potuto usare lo stesso schema usato sul client, per fornire un aggiornamento automatico delle tabelle.

Progettazione concettuale

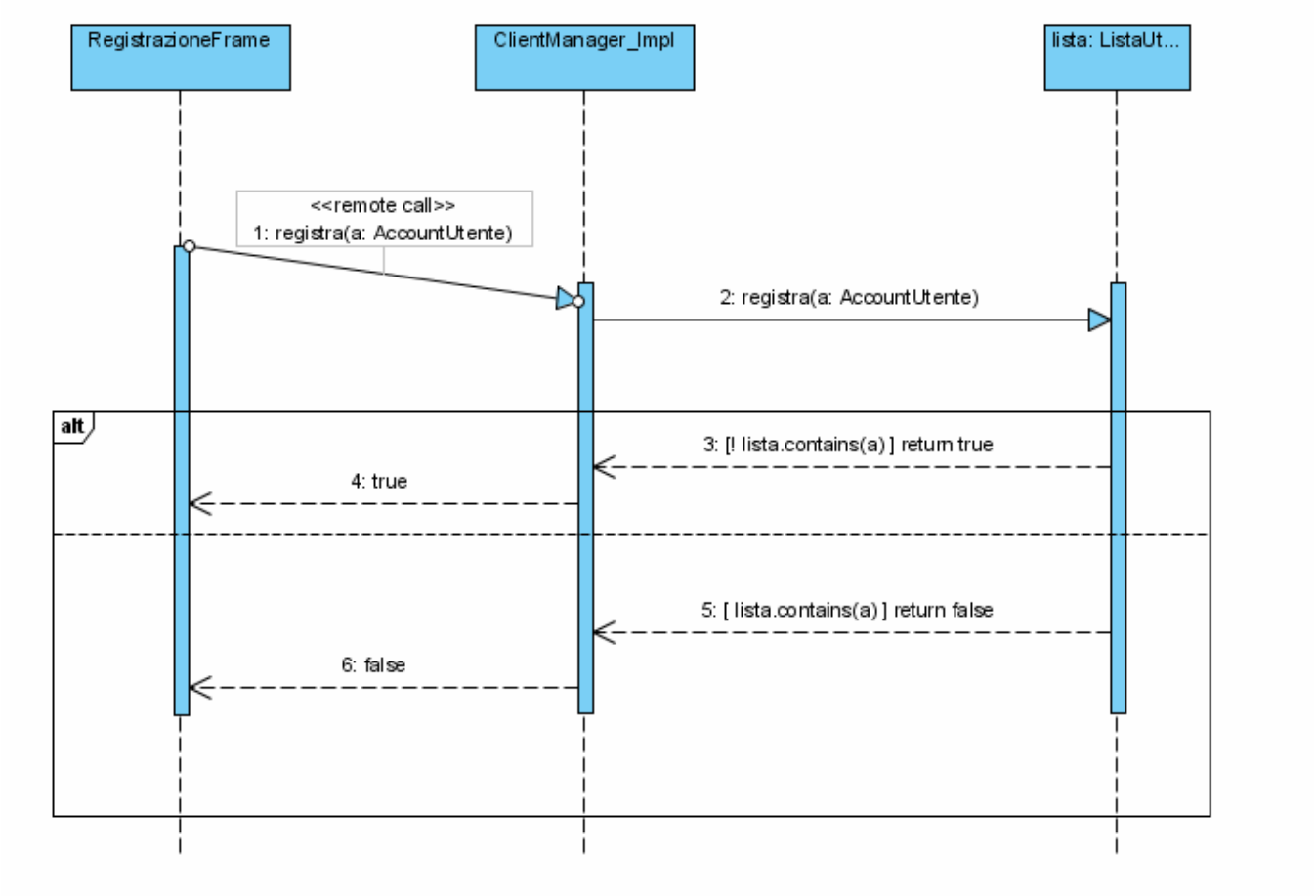
Diagramma delle classi preliminare



L’idea principale del sistema è la seguente:
Il componente server mantiene una lista di utenti ed una di articoli opportunamente rese thread-safe. Le due liste sono “esposte” ai client tramite un ArticoliManager ed un ClientManager, che fondamentalmente fanno da PROXY verso le liste del server. Il server, dunque, effettua le sue operazioni direttamente sulle liste, mentre i client accedono alle operazioni tramite i PROXY che, talaltro, non consentono tutte le operazioni effettivamente presenti sulle liste (alcune sono riservate per l’organizzatore).

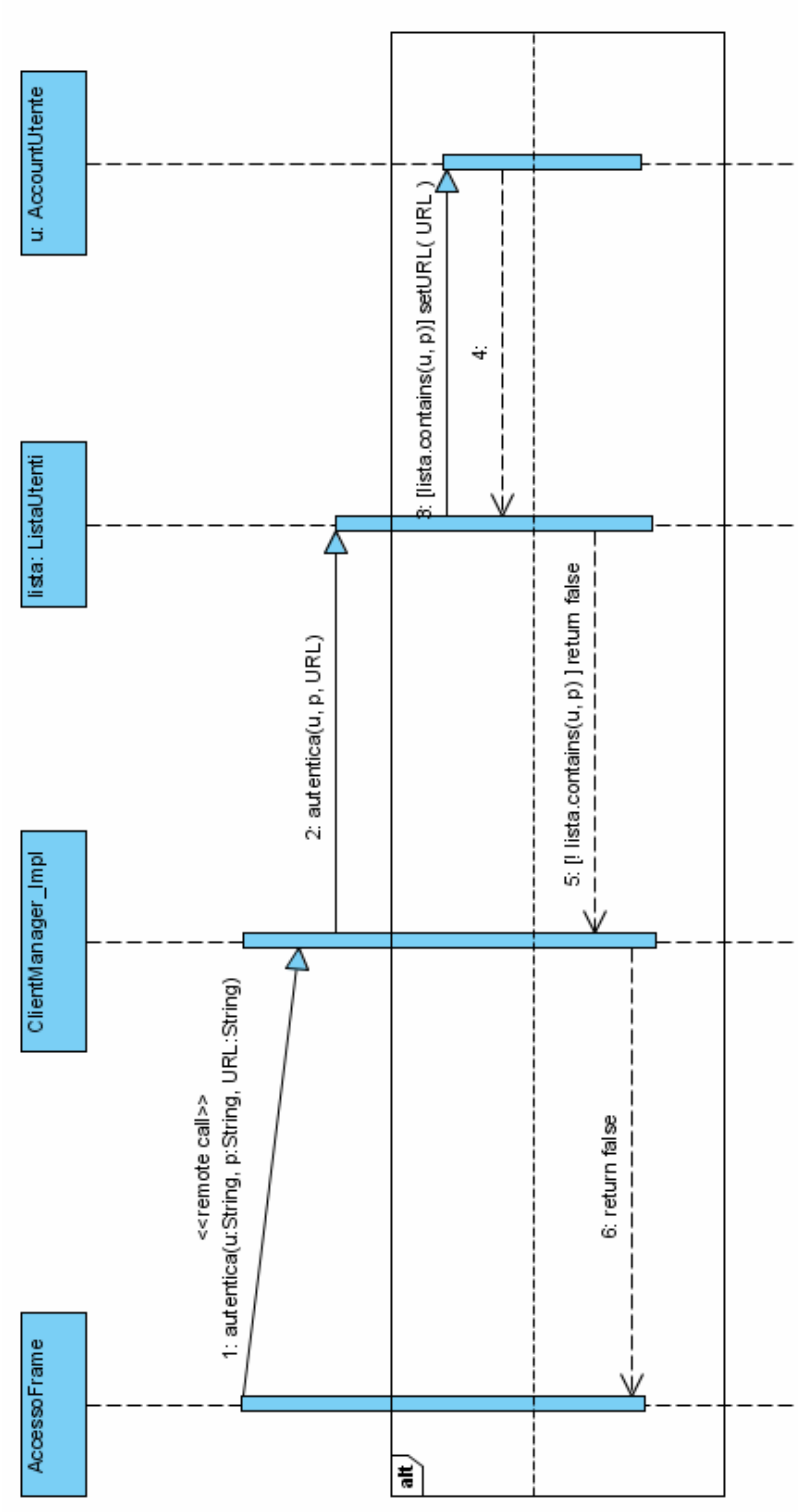
Progettazione delle operazioni

Registrazione utente



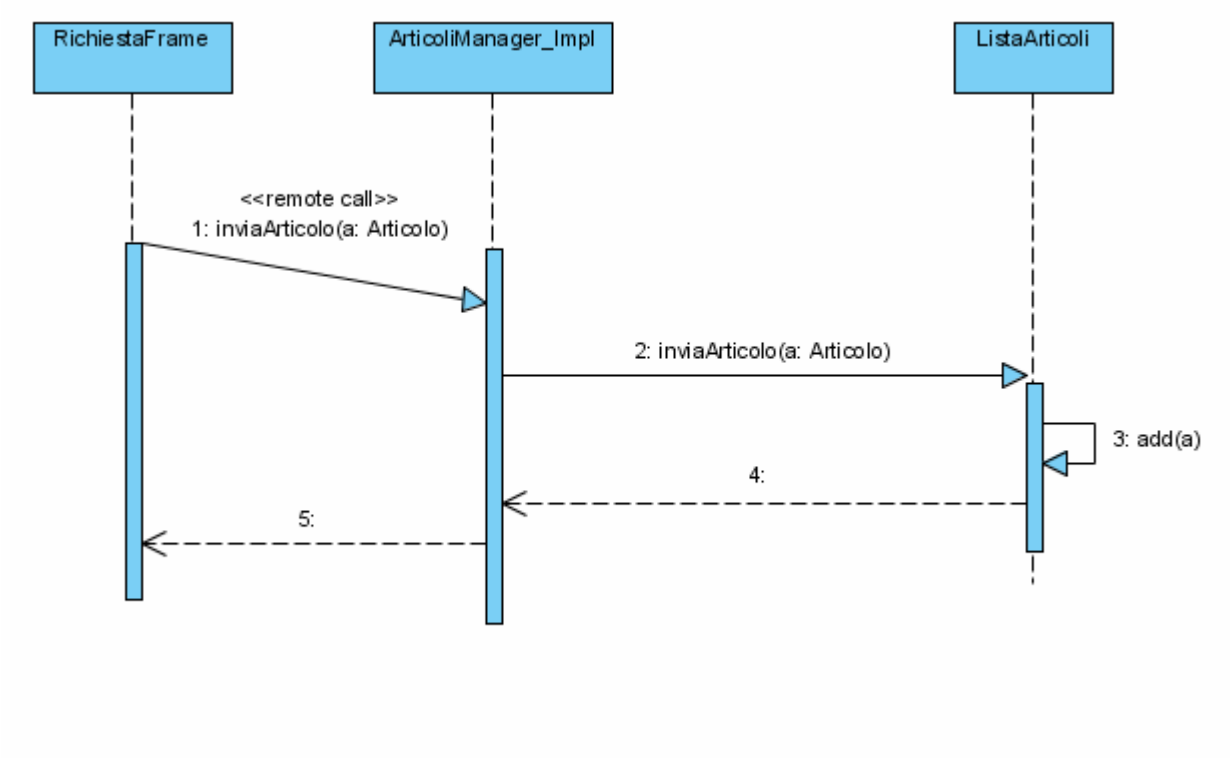
per effettuare la registrazione, il RegistraFrame richiede di specificare userID, password e una lista di nomi generici di articoli di cui si è potenziali acquirenti ha un riferimento verso il ClientManager e consente la registrazione all'utente invocando il metodo remoto registra(), che fondamentalmente aggiunge l'utente alla lista e ne permette l'accesso.

Autenticazione utente



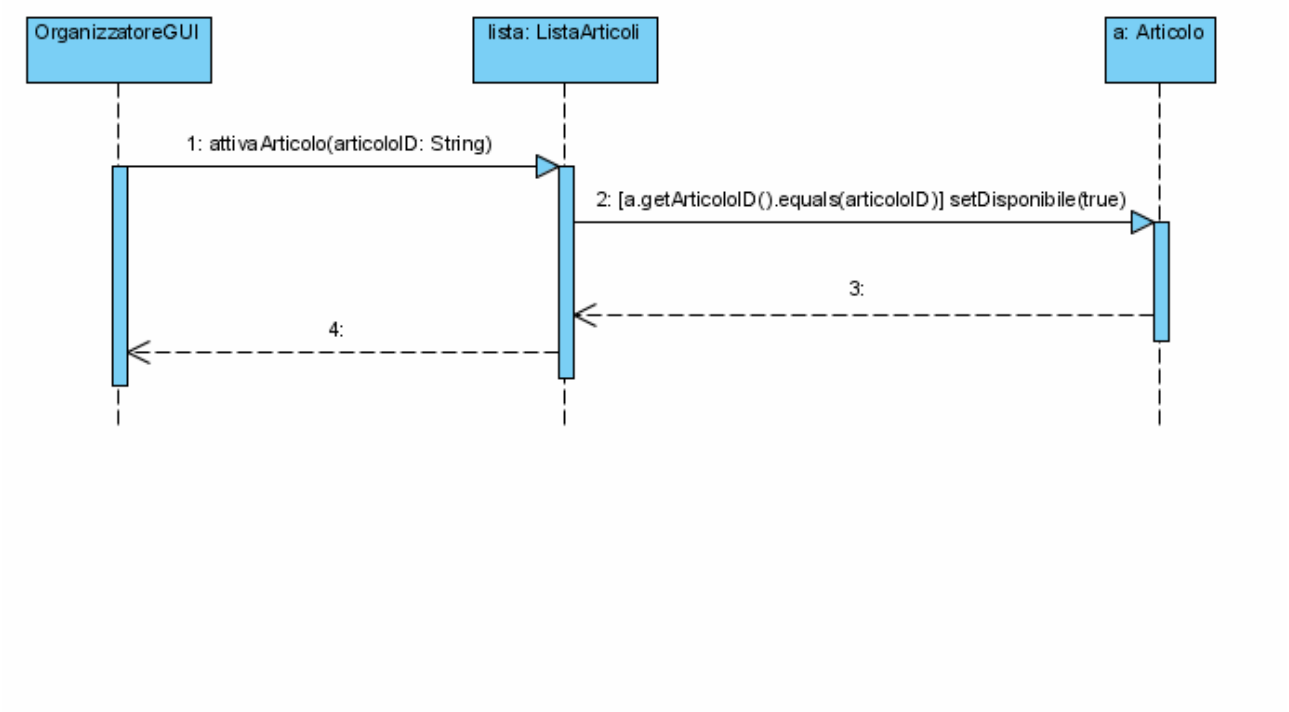
in modo analogo alla registrazione, il frame di accesso ha un riferimento al ClientManager di cui invoca il metodo autentica() passando i dati dell'utente, cioè userID e password per verificare l'identità dell'utente. Se il metodo ritorna true, il client potrà visualizzare il frame principale dell' applicazione con la lista di articoli.

Invio di una richiesta di messa all'asta



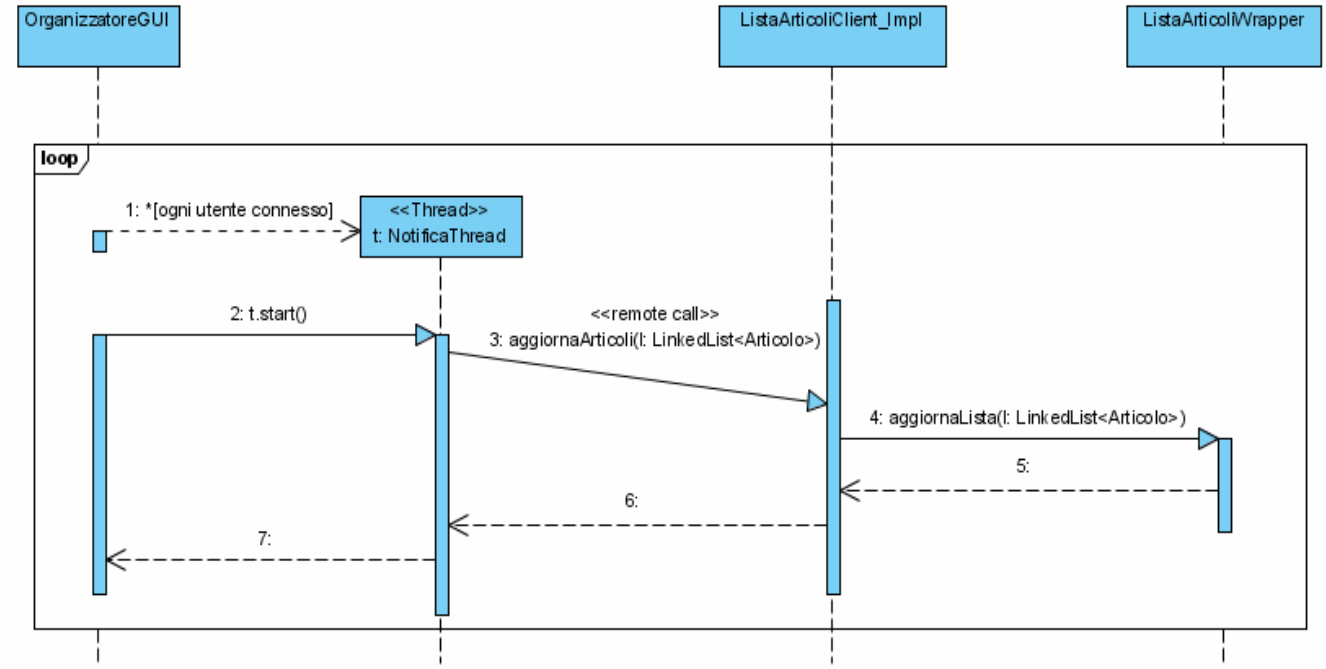
l'invio di un articolo da mettere all'asta avviene attraverso un apposito frame, che consente di specificare le caratteristiche dell'articolo. Questo frame, contattando remotamente l' ArticoliManager, richiede l'inserimento dell' articolo nella lista. Da notare che l'articolo appena inserito nella lista risulta inattivo e la sua data di scadenza è null, finché l'organizzatore dell' asta non lo attiva settando la data di chiusura.

Attivazione di un articolo



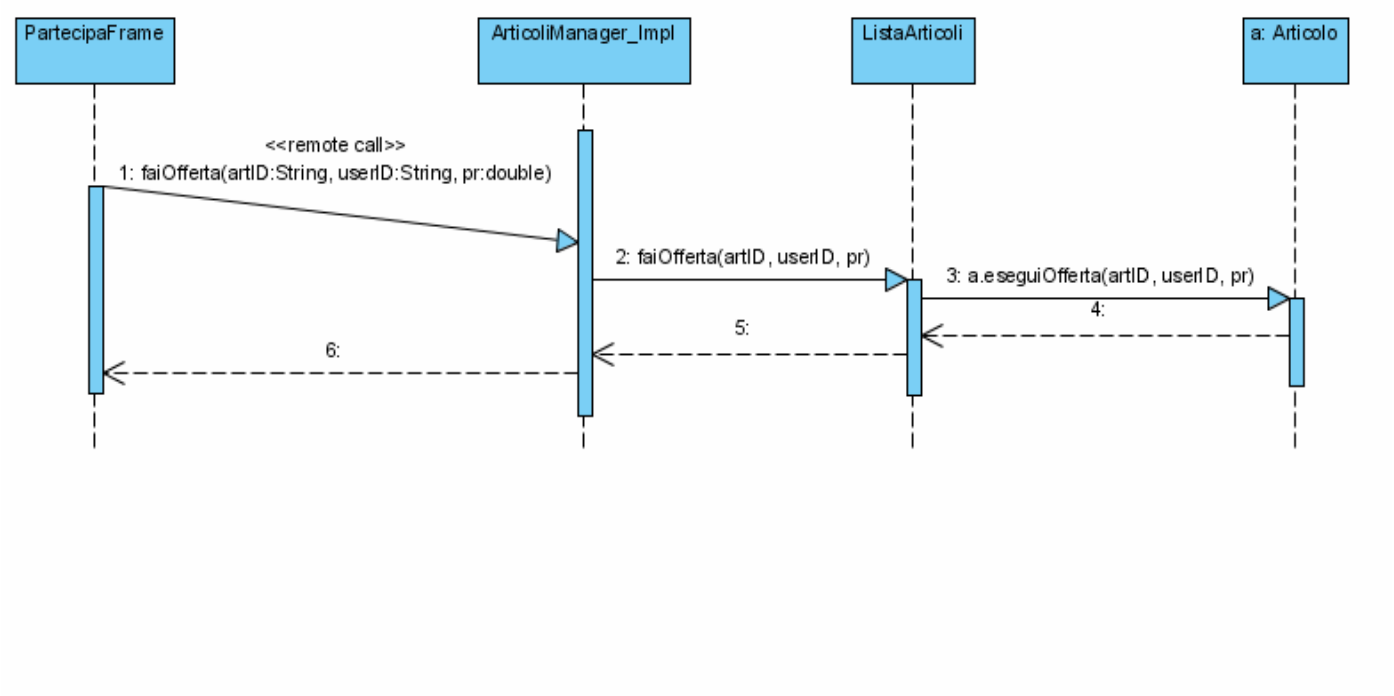
quando l'organizzatore vuole attivare un articolo usa un apposito comando; la GUI sfrutta un metodo apposito della ListaArticoli, cui ha un riferimento, per modificare il campo "disponibile" del particolare Articolo.
A quel punto può premere "notifica" e gli interessanti verranno avvisati del nuovo Articolo disponibile.

Notifica degli utenti di prossime aste



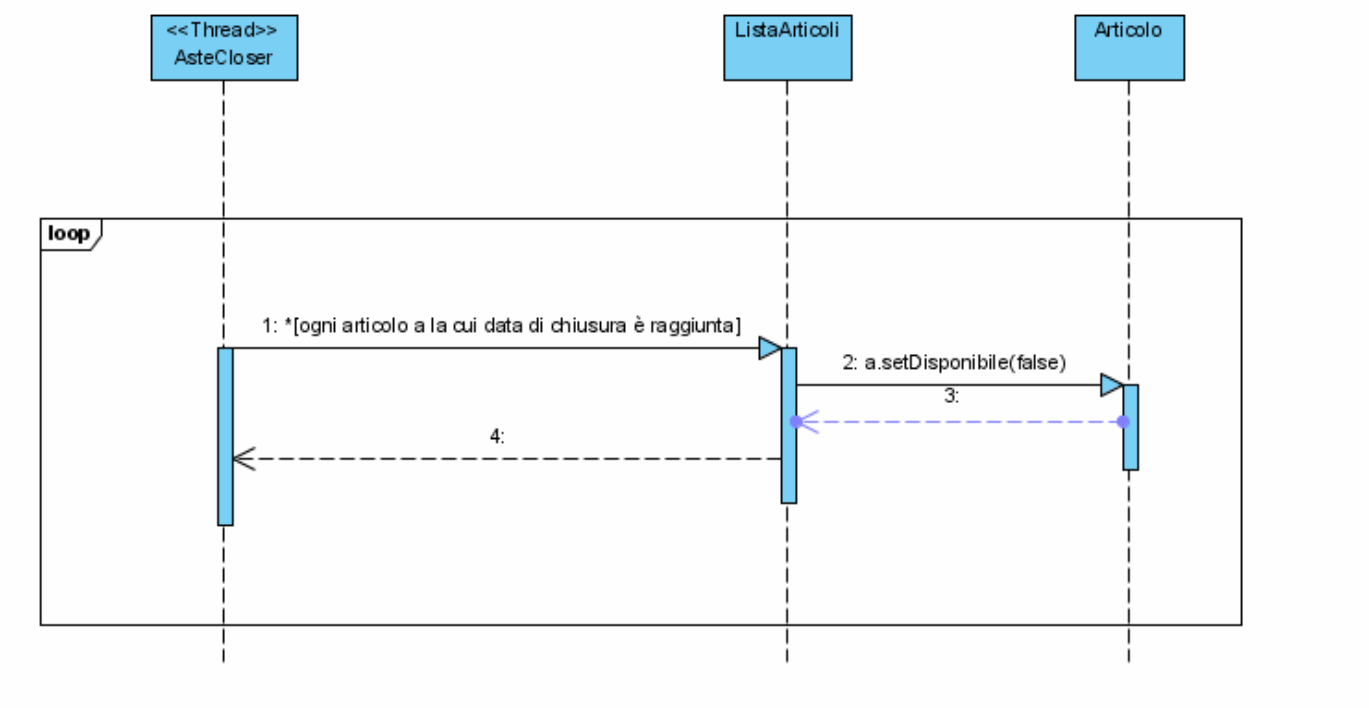
una volta che l'organizzatore ha attivato un articolo, può notificare agli utenti interessati utilizzando un comando esplicito della GUI. In particolare viene avviato un Thread notificatore per ogni utente connesso; ogni thread inserisce una copia della lista aggiornata nella lista locale del client tramite l'interfaccia remota dell'oggetto del client "ListaArticoliClient", (che deve essere pubblicato sul registry locale del client).

Esecuzione delle offerte per gli articoli all'asta



l'offerta per un particolare articolo avviene tramite un frame apposito che consente di visualizzare le variazioni del prezzo (dovute ad altre offerte) in "tempo reale".
L'offerta viene inoltrata al sistema tramite un metodo dell' ArticoliManager che modifica il prezzo dell' articolo nella lista.
Ovviamente la precondition che l'articolo sia attivo è banalmente sempre verificata in quanto un utente può visualizzare solo articoliattivi e che rispondono alle sue preferenze.

Chiusura di un' asta



la chiusura delle aste è effettuata da un apposito thread che periodicamente per ogni articolo verifica la data di chiusura e, qualora la data attuale superi quella di chiusura dell'articolo, questo viene disabilitato.

Vincoli di sistema in OCL

```
  /*
   * Controllo che l'utente non sia già registrato, in fase di registrazione
   */
Context
                ListaUtenti::registra(a: AccountUtente): boolean
Pre:
        not self.listaUtenti->includes(a)
Post:
       Self.listaUtenti->includes(a)
```

```
/*
 * Controllo di userID e password in fase di autenticazione
 */
Context
                ListaUtenti::autentica(userID:String, password:String, URL: String): Boolean
Pre:
        self.listaUtenti->includes(a | a.getUserID() = userID and
                                        a.getPassword() = password and a.getURL() = null)

Post:
        self.listaUtenti->includes(a| a.getUserID() = userID and
                                        a.getPassword() = password and a.getURL() = URL)
```

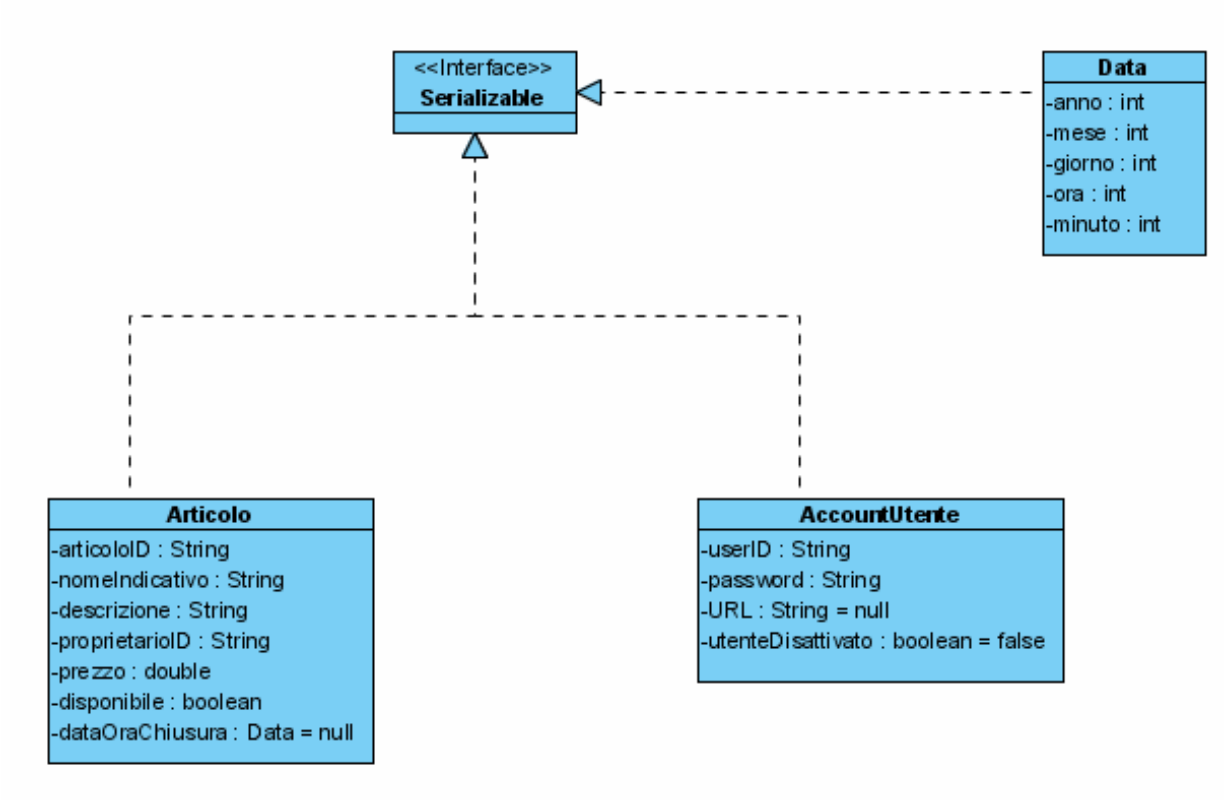
```
/*
 * Controllo validità offerta
 */
Context
           Articolo::eseguiOfferta(uID: String, p: double): boolean
Pre:
        self.abilitato = true and self.prezzo < p

Post:
        self.proprietarioID = a.getUserID()  and
        self.prezzo = p
```

/*
 * Controllo di vallidità di un articolo inviato per la messa all' asta
 */
Context
ListaArticoli::inviaArticolo(a: Articolo): boolean
Pre:
not self.listaArticoli->includes(a)
Post:
self.listaArticoli->includes(a)

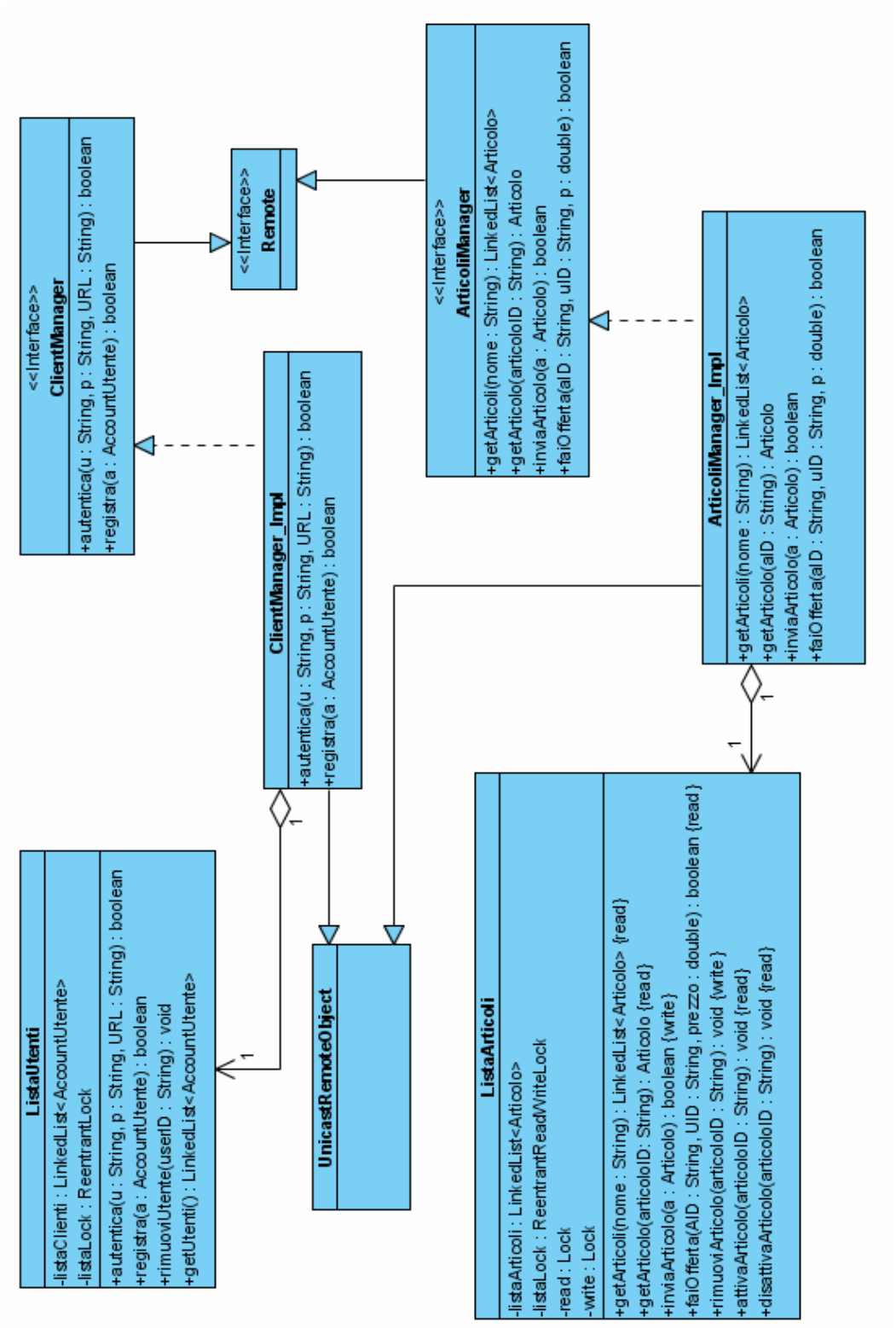
Diagrammi di classe dettagliati

Oggetti fondamentali



Anche se non è visualizzato, la classe Articolo ha un metodo eseguiOfferta(), come si può evincere dai precedenti sequence diagrams.

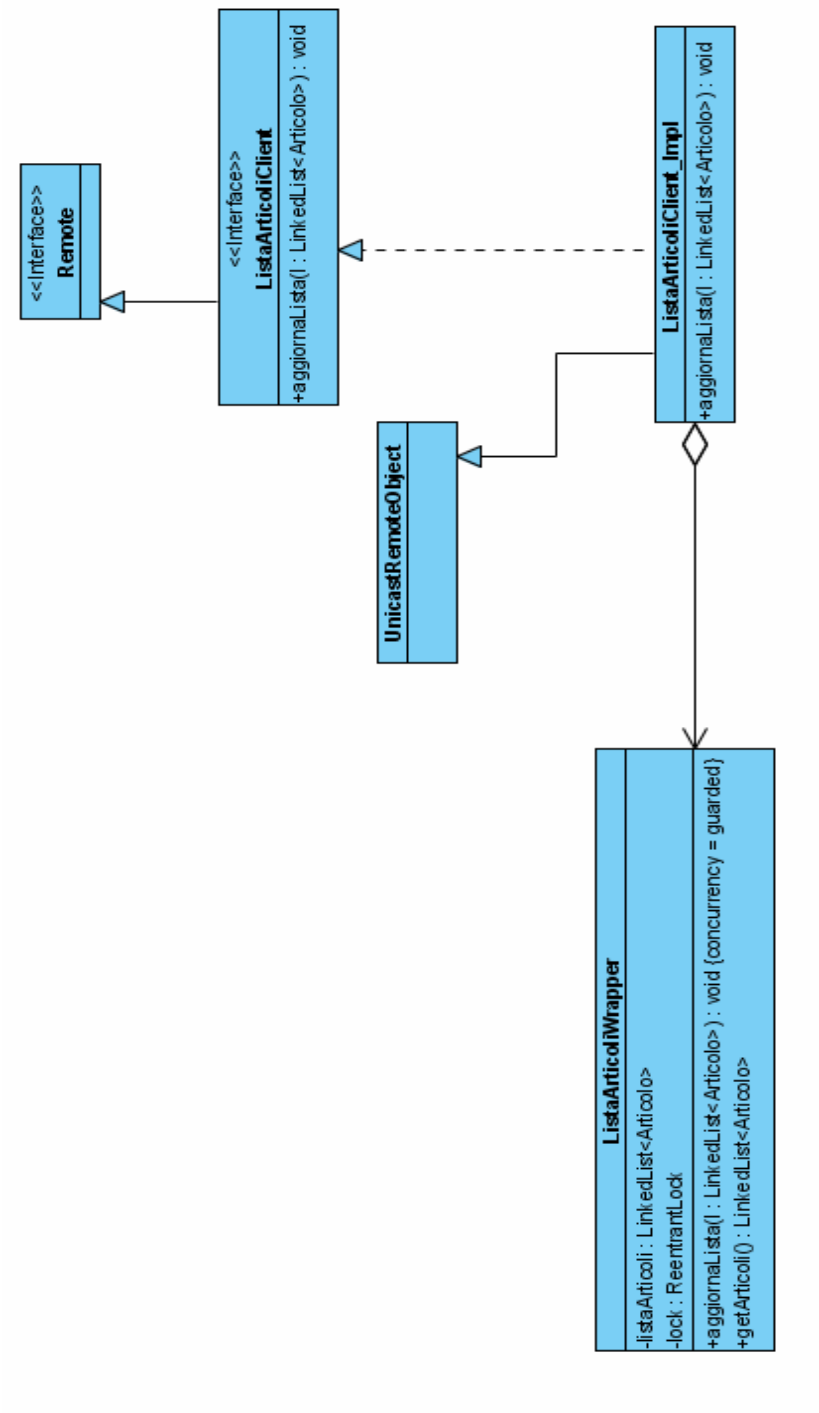
Business logic del componente Server



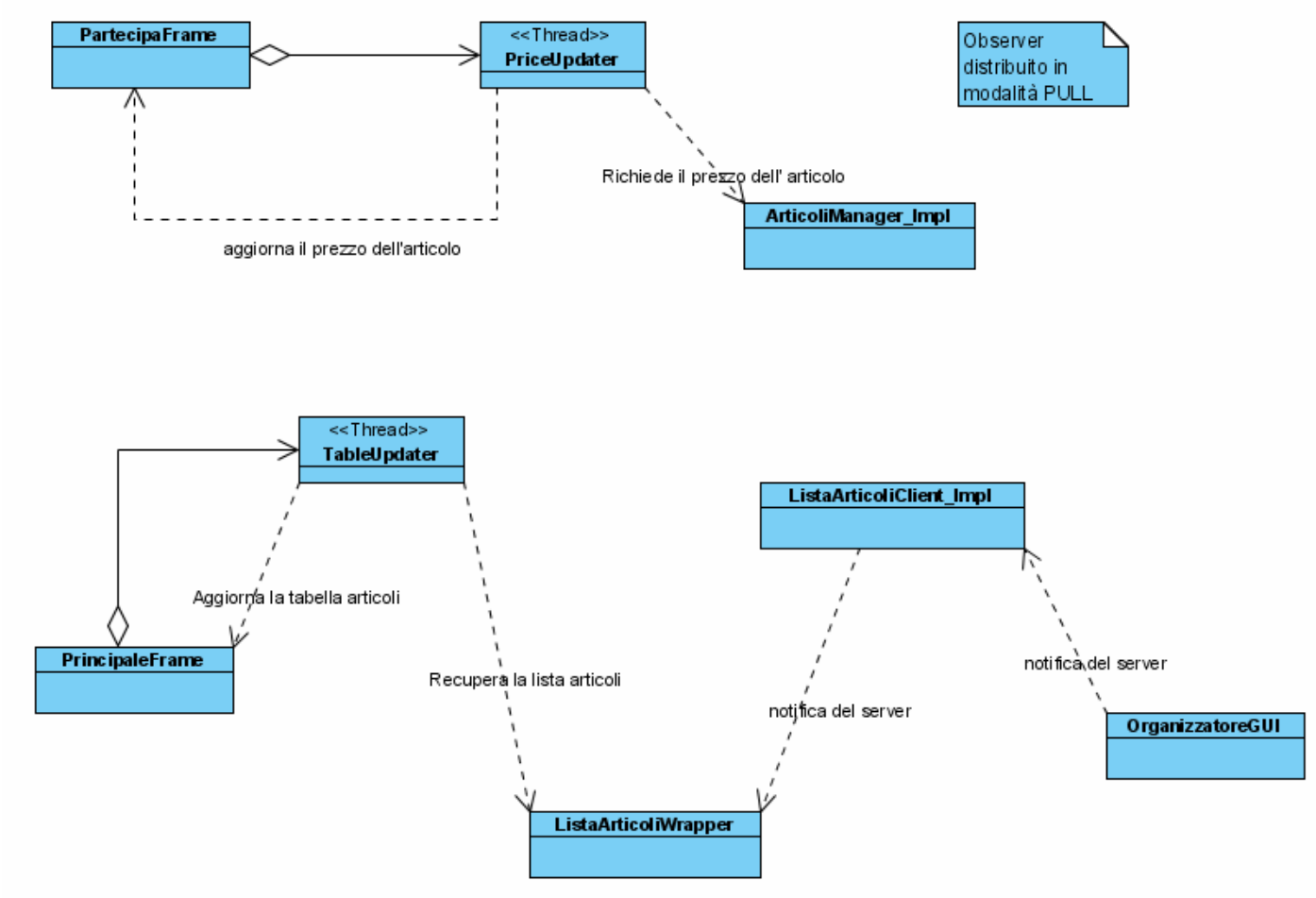
La ListaArticoli è sincronizzata mediante READ WRITE LOCK; è da notare che le operazioni sono di “read” o di “write” secondo che modifichino o meno la struttura

della lista; pertanto anche il metodo che esegue le offerte è di lettura perché non altera la struttura della lista. Sull'oggetto è però comunque garantita la SINGLE THREADED EXECUTION tramite l'uso del lock intrinseco dell' oggetto Articolo.

Business logic del componente Client

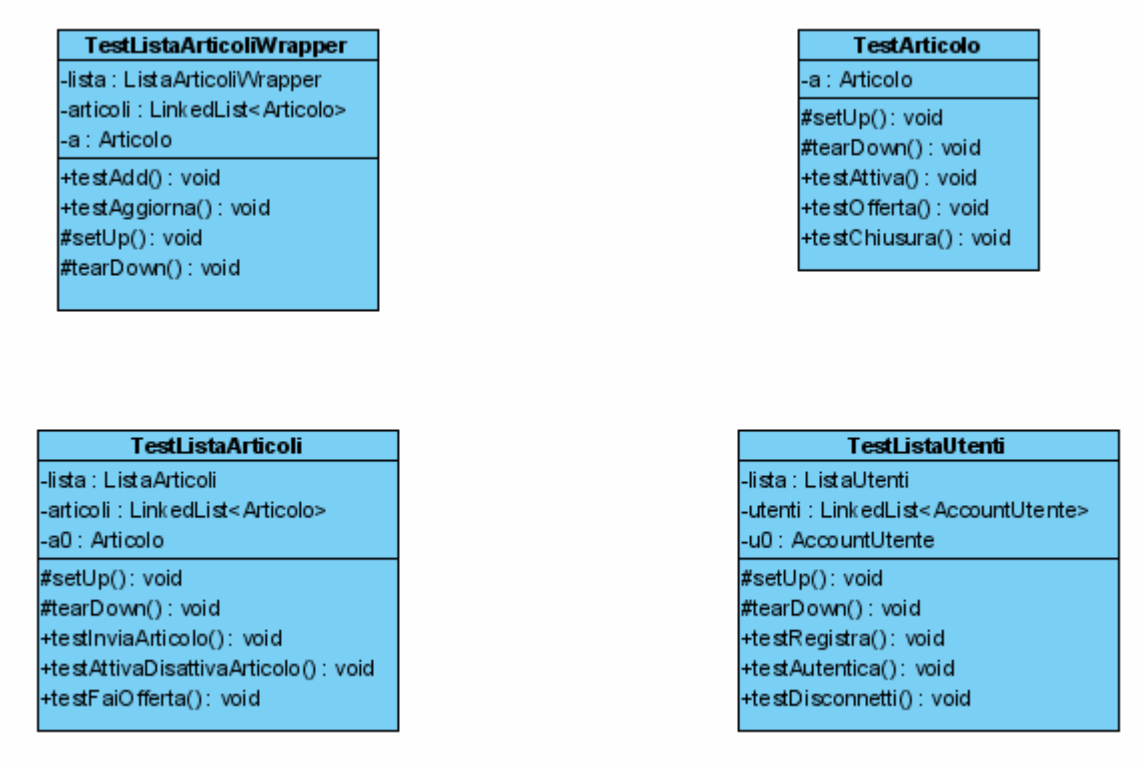


GUI del componente Client



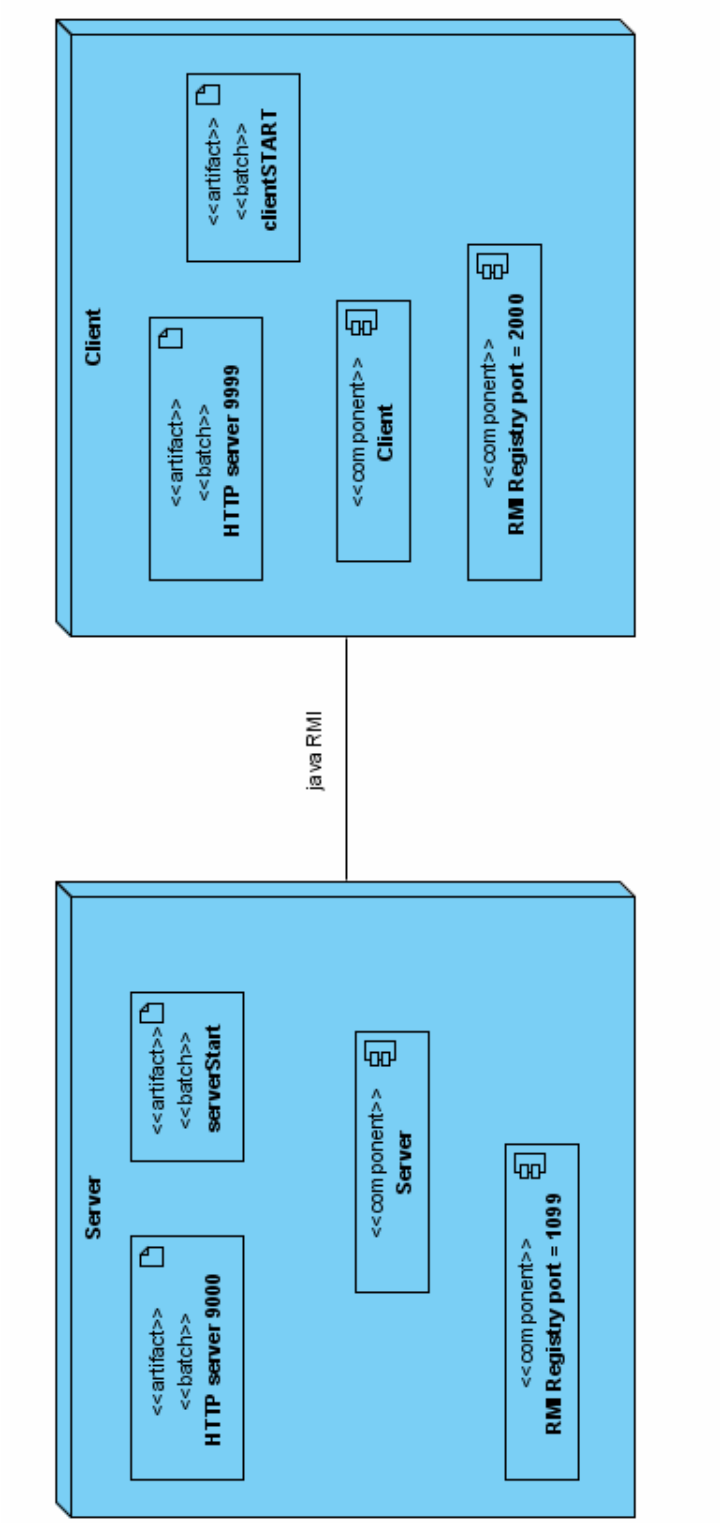
Principali casi di test

Test della business logic di client e server



Diagrammi di deployment

Diagramma di deployment



vengono usati 2 registry uno per pubblicare gli oggetti del server (sulla porta classica 1009) uno per pubblicare l’oggetto del client (porta = 2000).
Questo non sarebbe stato necessario se client e server fossero stati su macchine differenti, perché avrebbero indirizzo IP diverso.
Anche se non presente nella figura, sia Client sia Server hanno un file di policy che determina le politiche di accesso alle risorse, anche se in questo caso non sono state applicate restrizioni significative.