Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita
Proxy

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it

Applicazione client/server con socket

- In un'applicazione Client/Server, tipicamente client e server devono implementare meccanismi per:
  - la connessione
  - creazione dei dati da trasmettere
  - l'invio/ricezione delle richieste
  - la ricostruzione dei valori dei dati ricevuti
- Problemi:
  - molti dettagli di implementazione della comunicazione C/S distraggono dalla realizzazione delle funzionalità dell'applicazione;
  - difficoltà di manutenzione e porting: client e server mischiano il codice applicativo a quello per la comunicazione.

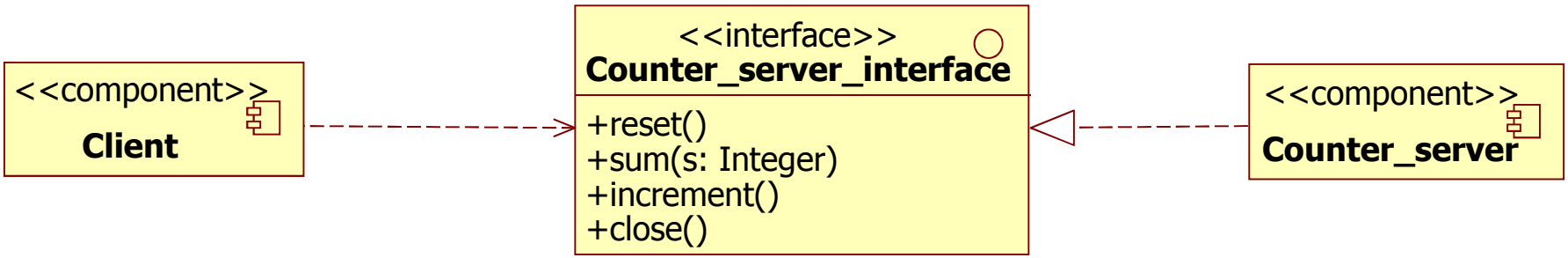
Separare logica applicativa e comunicazione

- Idealmente, il programmatore lato client dovrebbe concentrarsi sulla logica applicativa
  - richiesta dei servizi al server (sulla base della loro interfaccia)
  - elaborandone dei risultati in base alla logica applicativa
  - **separazione** della logica applicativa dai dettagli dei meccanismi di interazione con il server.
- Analogamente, il programmatore lato server dovrebbe concentrarsi sulla codifica dei servizi da offrire
  - **separazione** dei meccanismi per la comunicazione col client.

Esempio: Contatore Remoto

- Il Server gestisce un oggetto Contatore che
  - permette l'inizializzazione o reset di una variabile
  - permette l'incremento della stessa variabile di conteggio
- Il Client effettua il calcolo del tempo medio di servizio su un certo numero di incrementi.

Contatore: vista logica



Esempio: Contatore Remoto

```
public class CounterServer {
  public static final int PORT = 8888;
  public static void main(String[] args) throws IOException {
    int _counter = 0;
    ServerSocket serverSocket = new ServerSocket(PORT);
    System.out.println("Started: " + serverSocket);
    while (true) {
      System.out.println("Waiting a connection...");
      Socket socket = serverSocket.accept();
      try {
        BufferedReader istream = new BufferedReader(new
           InputStreamReader(socket.getInputStream()));
        PrintWriter ostream = new PrintWriter(new
           BufferedWriter(new OutputStreamWriter(
           socket.getOutputStream())), true);
        // ... Interpretazione dei comandi del client
      } finally {
           System.out.println("closing...");
           socket.close();
}} }}
```

Esempio: Contatore Remoto

```
// ... Interpretazione dei comandi del client
String myOper;
while ((myOper = istream.readLine())!= null) {
  if (myOper.equals("<incr>"))
    _counter++;
  else if (myOper.equals("<reset>"))
    _counter=0;
  else if (myOper.startsWith("<sum>")){
    StringTokenizer st = new StringTokenizer(myOper);
    String op = st.nextToken();
    String add = st.nextToken();
    if (op.equals("<sum>")){
      _counter += Integer.parseInt(add);
    }
  } else{
     System.out.println("operation not recognized: " + myOper);
  }
  ostream.println(_counter);
}
```

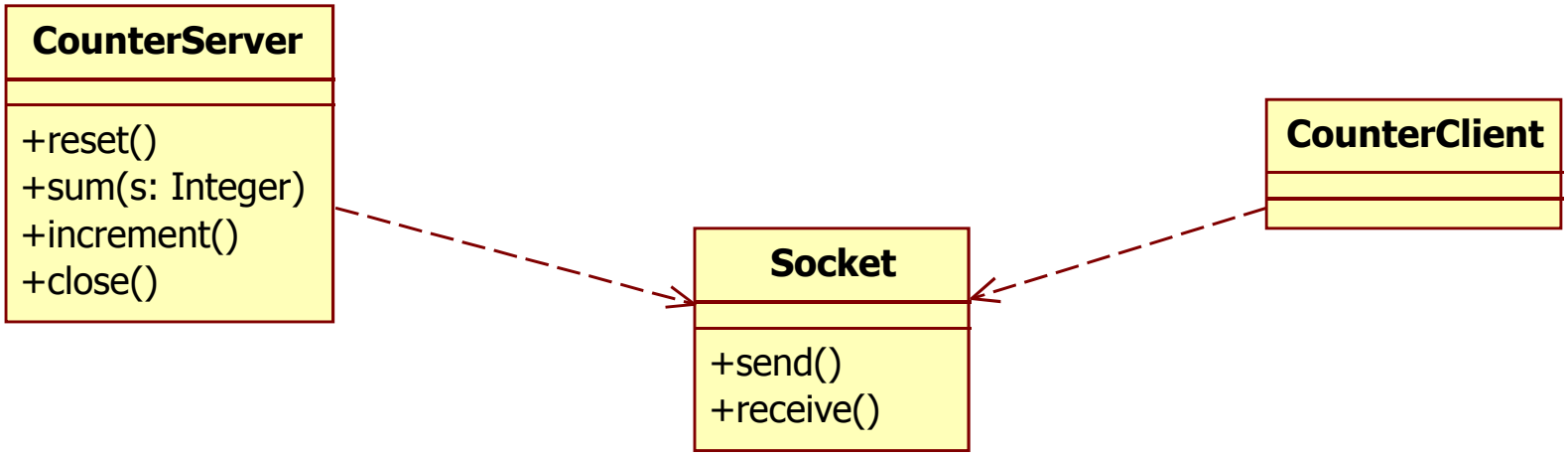
Esempio: Contatore Remoto

```
public class CounterClient {
  public static void main(String[] args) throws IOException {
    InetAddress addr = InetAddress.getByName(null);
    System.out.println("addr = " + addr);
    Socket socket = new Socket(addr, CounterServer.PORT);
    try {
      System.out.println("socket = " + socket);
      BufferedReader in = new BufferedReader(new
         InputStreamReader(socket.getInputStream()));
      PrintWriter out = new PrintWriter(new BufferedWriter(
         new OutputStreamWriter(socket.getOutputStream())),true);
      // ... preparazione ed invio dei comandi al server
    } finally {
       System.out.println("closing...");
       socket.close();
      }
   }
}
```

Esempio: Contatore Remoto

```
// ... preparazione ed invio dei comandi al server
// azzera il contatore
out.println("<reset>");
String strResult = in.readLine();
System.out.println("reset: "+strResult);
long startTime = System.currentTimeMillis();
// Effettuo 1000 richieste di incremento:
for(int i = 0; i < 1000; i++){
  out.println("<incr>");
  strResult = in.readLine();
  System.out.println("increment: "+strResult);
}
//Registro l'istante di conclusione:
long endTime = System.currentTimeMillis();
System.out.println("Elapsed time: "+(endTime-startTime)+"ms");
```

Contatore remote: vista di design

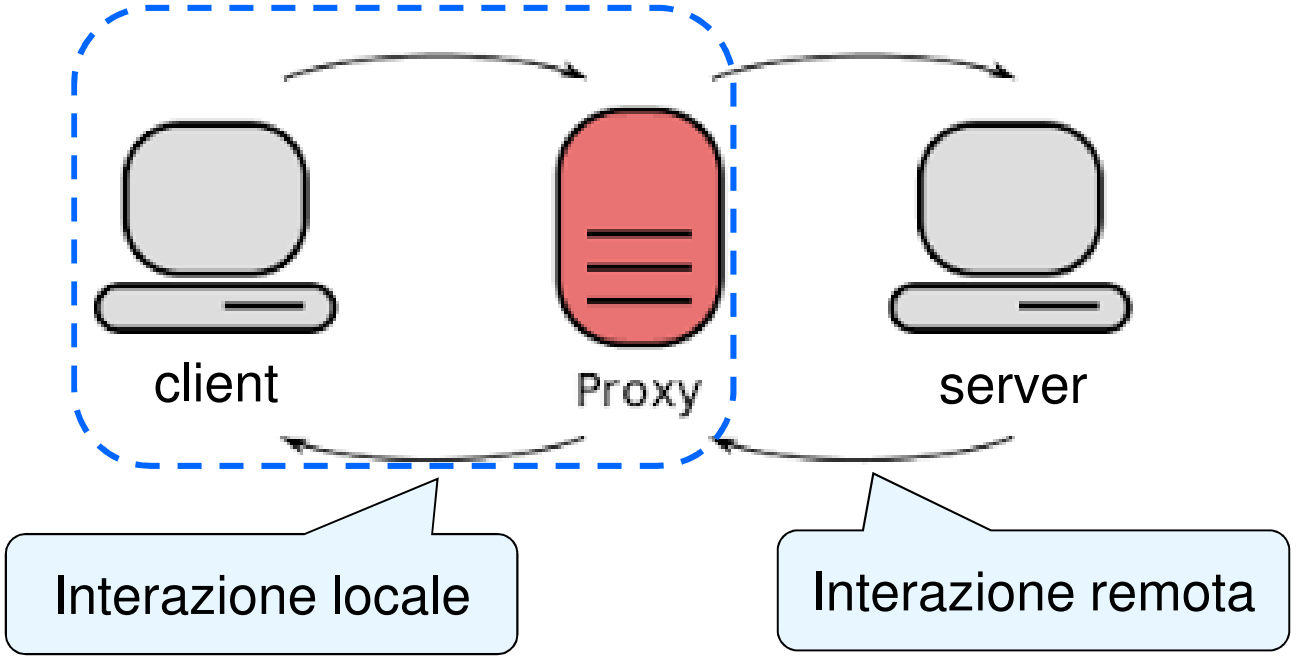


Pattern Proxy lato client

- Il pattern Proxy introduce un ulteriore componente nel modello di interazione client-server.

Pattern Proxy Remoto

- Il Proxy si presenta come una implementazione del servizio remoto, locale al client
  - interfaccia del proxy = interfaccia server reale
- I meccanismi di basso livello necessari per l'instaurazione della comunicazione, e lo scambio dei dati, sono implementati ed incapsulati all'interno del Proxy.



Esempio: Contatore Remoto con ProxyServer

```
public interface ServerInterface {
   public static final int PORT = 8888;

   public int sum(int s) throws IOException;
   public int reset() throws IOException;
   public int increment() throws IOException;
   public void close() throws IOException;
}
```

Queste sono le funzionalità offerte dal server ...
E quindi anche dal proxy server.

Esempio: Contatore Remoto con ProxyServer

```
public class ProxyServer implements ServerInterface {
  private Socket socket;
  private BufferedReader in;
  private PrintWriter out;
  // connessione:
  public ProxyServer() throws Exception {
    InetAddress addr = InetAddress.getByName(null);
    System.out.println("addr = " + addr);
    socket = new Socket(addr, InterfacciaServer.PORT);
    // crea gli stream di input/output
    System.out.println("socket = " + socket);
    in = new BufferedReader(new InputStreamReader(
           socket.getInputStream()));
    out = new PrintWriter(new BufferedWriter(
        new OutputStreamWriter(socket.getOutputStream())), true);
  }
  // metodi dell'interfaccia ServerInterface
  // ...
}
```

Il proxy server stabilisce la connessione col server ...

Esempio: Contatore Remoto con ProxyServer

```
// metodi dell'interfaccia ServerInterface
public int sum(int s) throws IOException {
  out.println("<sum> "+Integer.toString(s));
  String strResult = in.readLine();
  return Integer.parseInt(strResult);
}
public int reset() throws IOException {
  out.println("<reset>");
  String strResult = in.readLine();
  return Integer.parseInt(strResult);
}
public int increment() throws IOException {
  out.println("<incr>");
  String strResult = in.readLine();
  return Integer.parseInt(strResult);
}
public void close() throws IOException {
  System.out.println("closing...");
  out.println("<end>");
  socket.close();
}
```

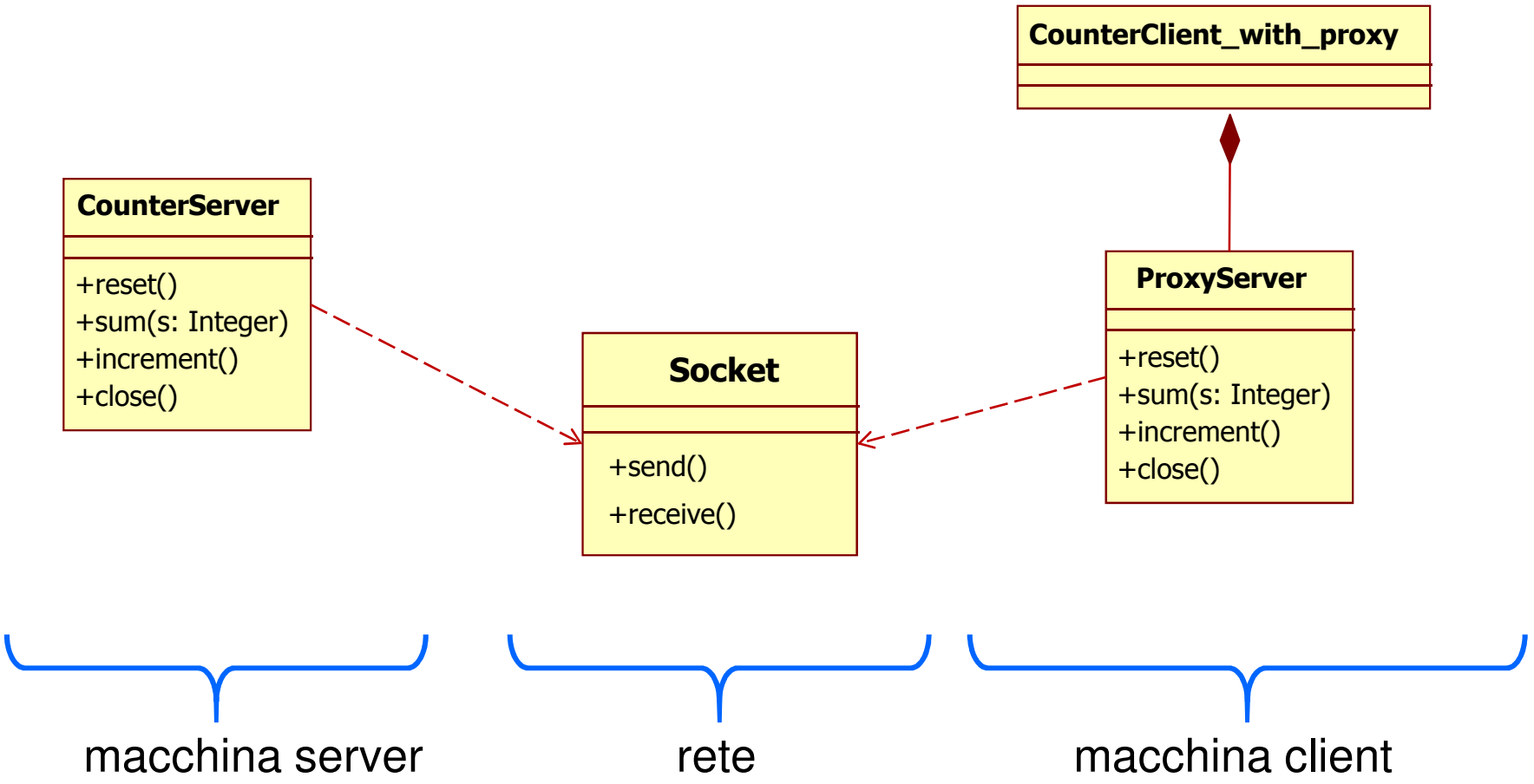
...
il proxy server inoltra al server tutte le richieste arrivate dal client

Esempio: Contatore Remoto con ProxyServer

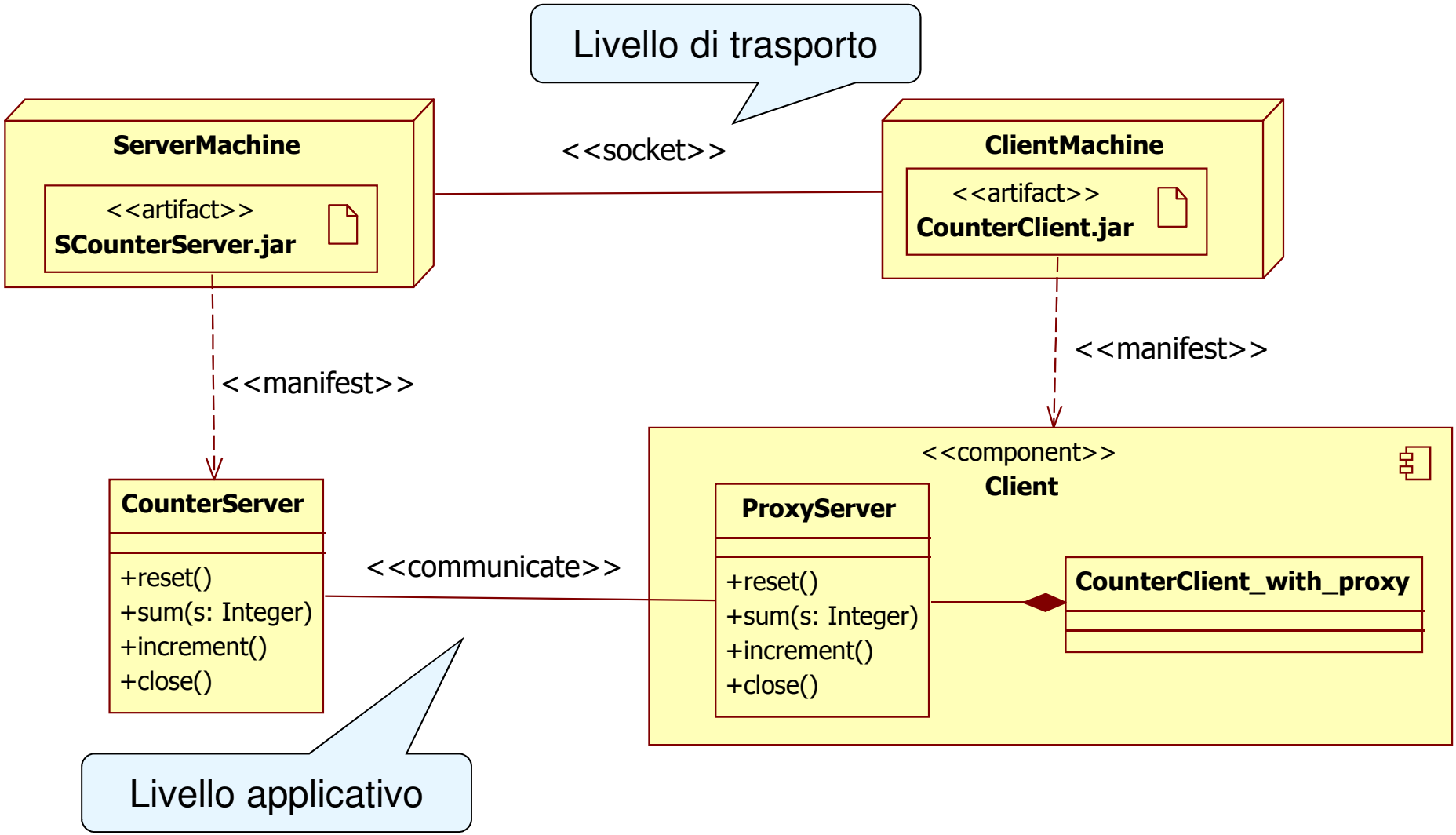
```
public class CounterClient {
  public static void main(String[] args) throws Exception {
    ServerInterface server = null;
    try {
      server = new ProxyServer();
      int init = server.reset();
      System.out.println("reset: "+init);
      long startTime = System.currentTimeMillis();
      for(int i = 0; i < 1000; i++){
        int r = server.increment();
        System.out.println("increment: "+r);
      }
      long endTime = System.currentTimeMillis();
      System.out.println("Elapsed time: "+(endTime-startTime)+
                         "ms");
    } finally {  server.close();  }
  }
}
```

Il client non deve più preoccuparsi della comunicazione: si rivolge a un proxy server locale.

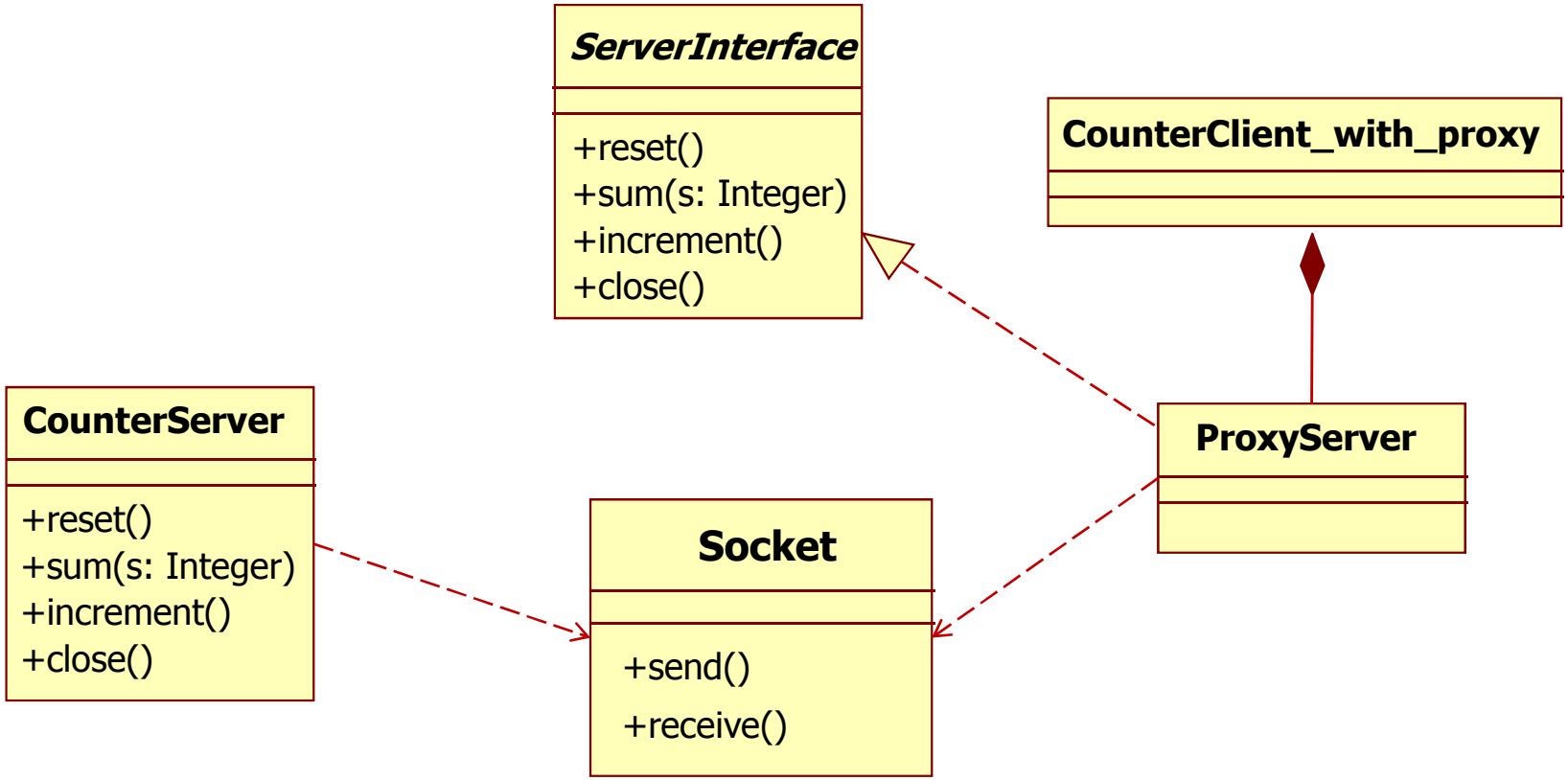
Design diagram



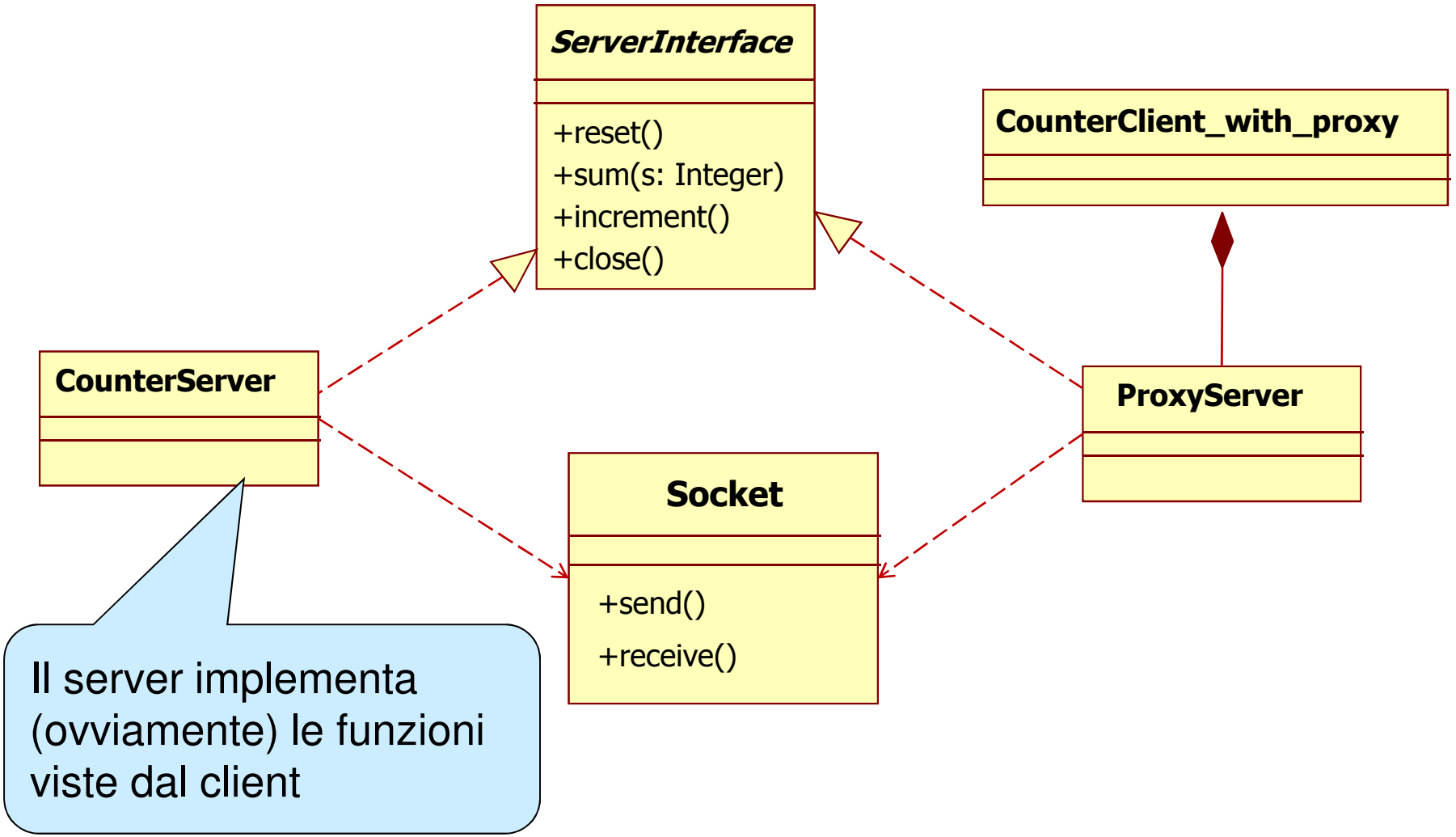
Deployment diagram



Design diagram (con interface)



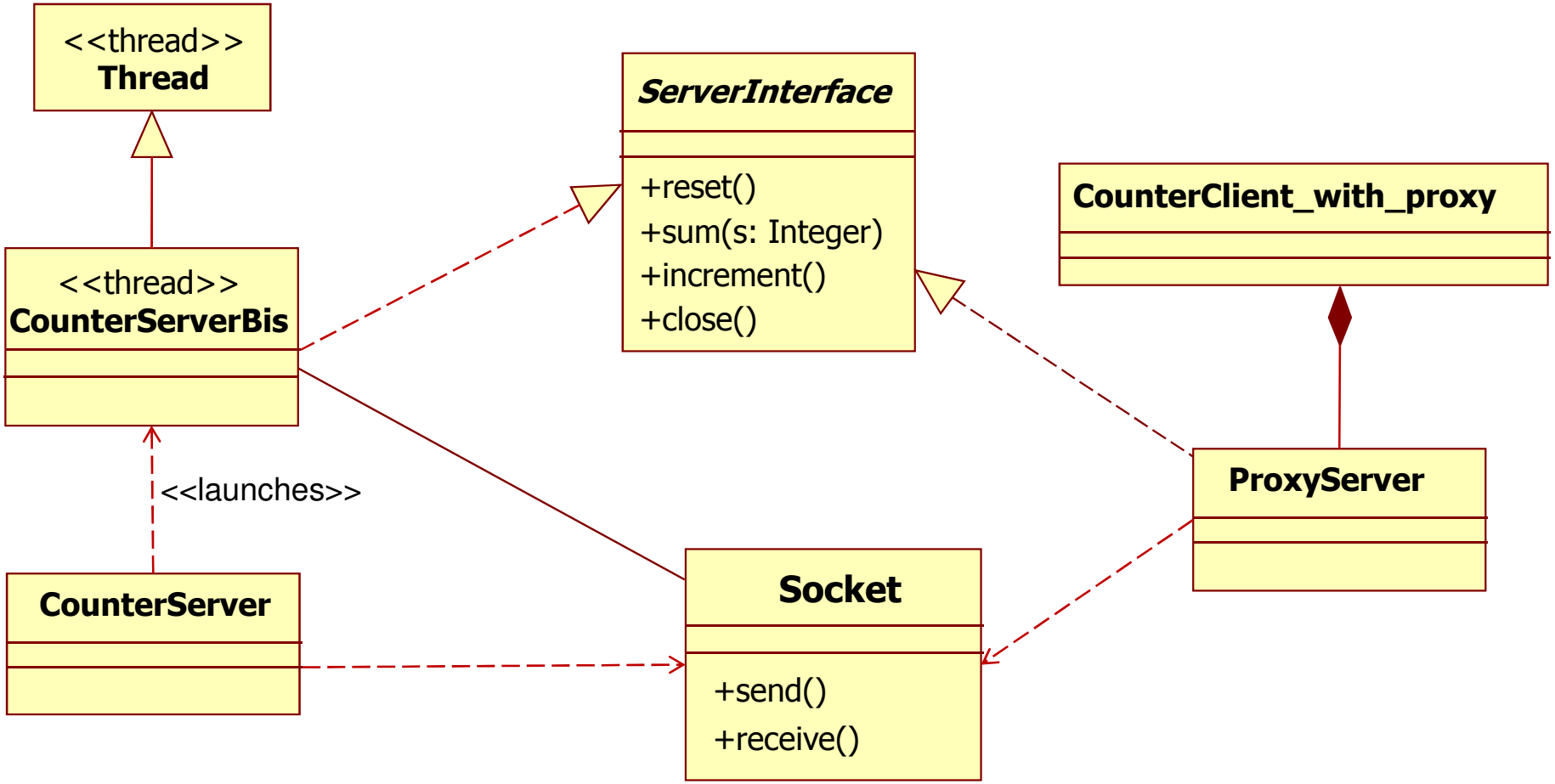
Design diagram (con interface)



Contatore Remoto: server multithread

- Reimplementiamo il server, facendo in modo che
  - Implementi la stessa interfaccia vista dal server
  - Lanci un thread per ogni connessione ricevuta

Design diagram



Esempio

- Realizziamo un contatore remote con le seguenti caratteristiche:
  - Il server genera un thread dedicato per ciascun client
  - Il client è dotato di proxy server.
  - Per testare il sistema, il main client genera tanti thread client paralleli (ciascuno col suo proxy).

Esempio: Contatore Remoto con ProxyServer

```
public interface ServerInterface {
        public static final int PORT = 8888;

        public int sum(int s) throws IOException;
        public int reset() throws IOException;
        public int increment() throws IOException;
        public void close() throws IOException;
}
```

Esempio: Contatore Remoto con ProxyServer

```
public class CounterServerBis implements ServerInterface,
                                         Runnable {
  private Socket _socket;
  private int _counter = 0;
  private BufferedReader istream;
  private PrintWriter ostream;
  public CounterServerBis(Socket socket) {
    _socket = socket;
    try {
      istream = new BufferedReader(new InputStreamReader(
                                  socket.getInputStream()));
      ostream = new PrintWriter(new BufferedWriter(new
        OutputStreamWriter(socket.getOutputStream())), true);
    } catch (IOException e) {  }
  }
```

Esempio: Contatore Remoto con ProxyServer

```
// metodi dell'interfaccia ServerInterface
public int sum(int s) throws IOException {
  _counter += s; return _counter;
}

public int reset() throws IOException {
  _counter = 0;  return _counter;
}

public int increment() throws IOException {
  _counter++;    return _counter;
}
public void close() {
  System.out.println("closing...");
  try {
    _socket.close();
  } catch (IOException e) {}
}
```

Esempio: Contatore Remoto con ProxyServer

```
public void run() {
  try {
    while (!_socket.isClosed()) {
      int result = 0;
      String myOper = istream.readLine();
      if (myOper.equals("<incr>"))
        result = increment();
      else if (myOper.equals("<reset>"))
        result = reset();
      else if (myOper.startsWith("<sum>")) {
        StringTokenizer st = new StringTokenizer(myOper);
        String op = st.nextToken();
        String add = st.nextToken();
        if (op.equals("<sum>"))
          result = sum(Integer.parseInt(add));
      } else if (myOper.startsWith("<end>")) {
          close();
      } else {
          System.out.println("operation not recognized: " + myOper);
      }
      ostream.println(result);
    }
  } catch (Exception e) {
     close();
}}
```

Esempio: Contatore Remoto con ProxyServer

```
public class CounterServer {
  public static void main(String[] args) throws IOException {
    ServerSocket serverSocket = new
                              ServerSocket(ServerInterface.PORT);
    System.out.println("Started: " + serverSocket);
    while (true) {
      System.out.println("Server: waiting a connection...");
      Socket socket = serverSocket.accept();
      System.out.println("Server: new client connected...");
      new Thread(new CounterServerBis(socket)).start();
    }
  }
}
```

Un nuovo thread per ogni client

Esempio: Contatore Remoto con ProxyServer

```
public class CounterClient {
  public static void main(String[] args) throws Exception {
    int numClients=4;
    for(int i=numClients; i>0; i--) {
      new CounterClientThread(i).start();
      System.out.println("Master client: thread "+i+" created");
      Thread.sleep(2);
    }
  }
}
```

Esempio: Contatore Remoto con ProxyServer

```
public class CounterClientThread extends Thread {
  private int id;
  public CounterClientThread(int nc){ id=nc; }
  public void run(){
    ServerInterface localServer=null;
    try {
      localServer = new ProxyServer();
      int init = localServer.reset();
      System.out.println("Client "+id+" reset: "+init);
      long startTime = System.currentTimeMillis();
      for(int i = 0; i < 100; i++){
        int r = localServer.sum(1);
        System.out.println("Client "+id+" increment: "+r);
    }
    System.out.println("Client "+id+" Elapsed time: "+
              (System.currentTimeMillis()-startTime)+ "ms");
    } catch (Exception e) { e.printStackTrace();
    } finally {
      System.out.println("Client "+id+" closing...");
      try { localServer.close();} catch (IOException e) { }
} }
```

Esempio: Contatore Remoto con ProxyServer

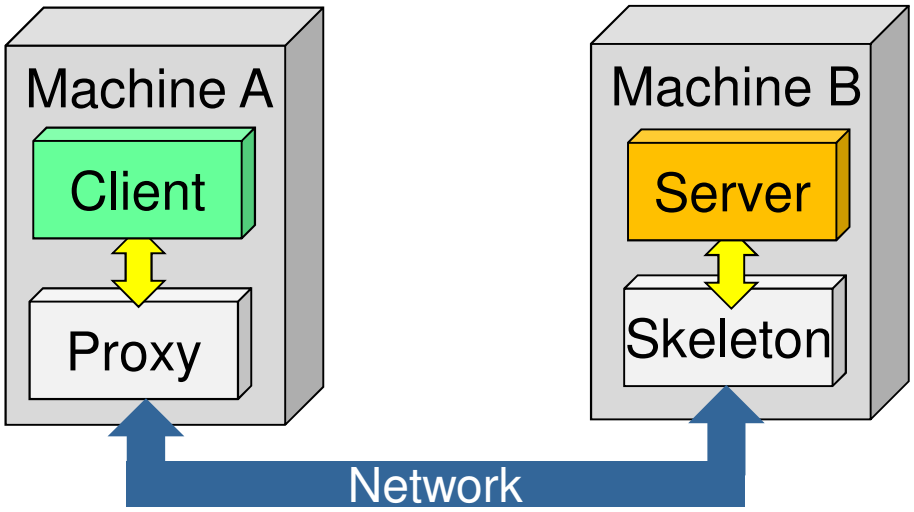
```
public class ProxyServer {
  // come prima
}
```

Pattern Proxy lato server: Skeleton

- Il Proxy lato client “solleva” il client reale dalle problematiche di comunicazione.
- Il server tuttavia ha ancora l’onere di implementare i necessari meccanismi di comunicazione assieme alla logica

Pattern Proxy lato server: Skeleton

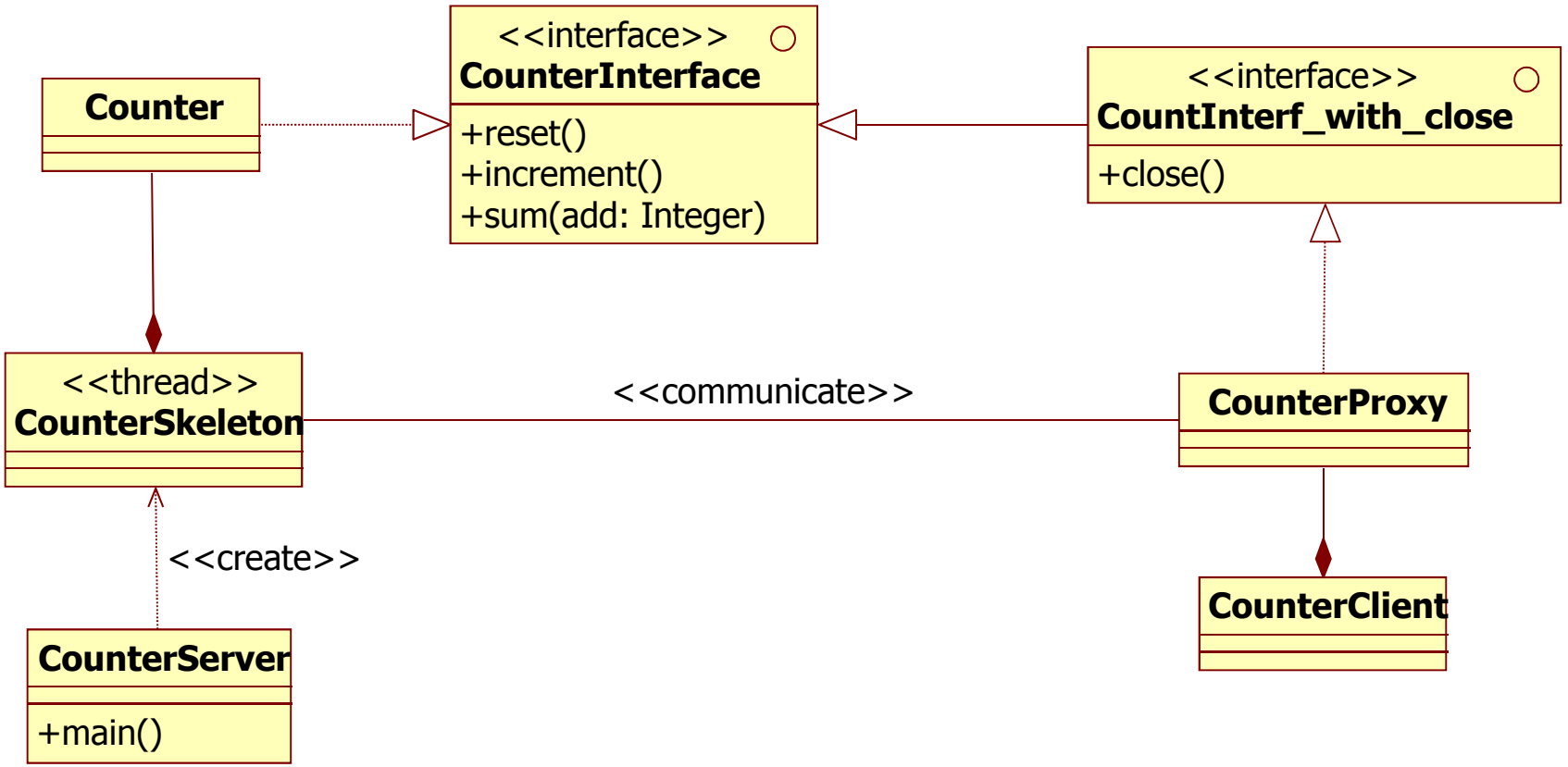
- Aggiungiamo un Proxy lato server, detto Skeleton, che si faccia carico della comunicazione con il Proxy lato client.
- Lo skeleton avrà la responsabilità di
  - ricevere le richieste di servizio
  - strutturare l'informazione fornita in ingresso
  - fare la chiamata al server reale
  - ricevere da questi eventuali risultati e rispedirli al proxy lato client



Implementare lo Skeleton

- Lo skeleton lato server può essere implementato in due modi:
  - Per delega: la classe **`Skeleton`** presenta al suo interno un riferimento al **`CounterServerBis`**
  - Per ereditarietà : la classe **`Skeleton`** implementa solo gli opportuni schemi di comunicazione, mentre il **`CounterServerBis`** fornisce l'implementazione ai metodi astratti.

Design diagram (skeleton usa delega)



Esempio: Contatore Remoto con ProxyServer e Skeleton

```
import java.io.*;

public interface CounterInterface {
       public static final int PORT = 8888;
         public int sum(int s) throws IOException;
         public int reset() throws IOException;
         public int increment() throws IOException;
}
```

È ragionevole che il numero di port faccia parte della descrizione del servizio

Non c'é il metodo **`close`**
Lo skeleton ci pensa da solo a chiudere i socket.

Esempio: Contatore Remoto con ProxyServer e Skeleton

```
public class Counter implements CounterInterface {
 private int theCounter;
   public Counter() {
     theCounter = 0;
   }
   public int sum(int s) {
     theCounter += s;
     return theCounter;
   }
   public int reset() {
     theCounter = 0;
     return theCounter;
   }
   public int increment() {
     theCounter++;
     return theCounter;
   }
}
```

Esempio: Contatore Remoto con ProxyServer e Skeleton

```
public class CounterSkeleton extends Thread {
        private Socket theSocket;
        private BufferedReader istream;
        private PrintWriter ostream;
        private Counter server = new Counter();
        public CounterSkeleton(Socket socket) {
                theSocket = socket;
        }
```

Lo skeleton gestisce le comunicazioni attraverso il socket

Esempio: Contatore Remoto con ProxyServer e Skeleton

```
 public void run() {
  try {
   istream = new BufferedReader(new
InputStreamReader(theSocket.getInputStream()));
   ostream = new PrintWriter(new BufferedWriter(new
     OutputStreamWriter(theSocket.getOutputStream())), true);
   while (!theSocket.isClosed()) {
    int result = 0;
    String myOper = istream.readLine();
    if (myOper.equals("<incr>"))
     result = server.increment();
    else if (myOper.equals("<reset>"))
     result = server.reset();
    else if (myOper.startsWith("<sum>")) {
     StringTokenizer st = new StringTokenizer(myOper);
     String op = st.nextToken(); String add = st.nextToken();
     result = server.sum(Integer.parseInt(add));
    } else if (myOper.startsWith("<end>")) {
     theSocket.close();
    } else { System.out.println("operation not recognized: "+myOper); }
    ostream.println(result);
   }
  } catch (Exception e) { }
 }
}
```

Lo skeleton legge un comando dal socket e chiama il metodo corrispondente del contatore

Esempio: Contatore Remoto con ProxyServer e Skeleton

```
import java.io.IOException;

public interface CounterInterface_with_close extends
                                                CounterInterface{
      public void close() throws IOException;
}
```

Esempio: Contatore Remoto con ProxyServer e Skeleton

```
public class CounterClient {
 void exec() {
  int numClients=4;
  for(int i=numClients; i>0; i--) {
   new CounterClientThread(i).start();
   System.out.println("Master client: thread "+i+" created");
   try { Thread.sleep(2);
   } catch (InterruptedException e) { }
  }
 }
 public static void main(String[] args) throws Exception {
  new CounterClient().exec();
 }
}
```

Esempio: Contatore Remoto con ProxyServer e Skeleton

```
public class CounterClientThread extends Thread {
 private int id;
 CounterInterface_with_close localServer=null;
 CounterClientThread(int i){ id=i; }
 public void run() {
  try {
   localServer = new CounterProxy();
   int init = localServer.reset();
   System.out.println("Client "+id+" reset: "+init);
   long startTime = System.currentTimeMillis();
   for(int i = 0; i < 100; i++){
    int r = localServer.sum(1);
    System.out.println("Client "+id+" increment: "+r);
   }
   System.out.println("Client "+id+" Elapsed time: "+
     (System.currentTimeMillis()-startTime)+ "ms");
  } catch (Exception e) { e.printStackTrace();
  } finally {
    try { localServer.close();} catch (IOException e) { }
} } }
```

Esempio: Contatore Remoto con ProxyServer e Skeleton

```
public class CounterProxy implements
CounterInterface_with_close{
 private Socket socket;
 private BufferedReader in;
 private PrintWriter out;
 public CounterProxy() throws Exception {
  InetAddress addr = InetAddress.getByName(null);
  System.out.println("addr = " + addr);
  socket = new Socket(addr, CounterInterface.PORT);
  System.out.println("socket = " + socket);
  in = new BufferedReader(new
           InputStreamReader(socket.getInputStream()));
  out = new PrintWriter(new BufferedWriter(new
           OutputStreamWriter(socket.getOutputStream())), true);
 }
```

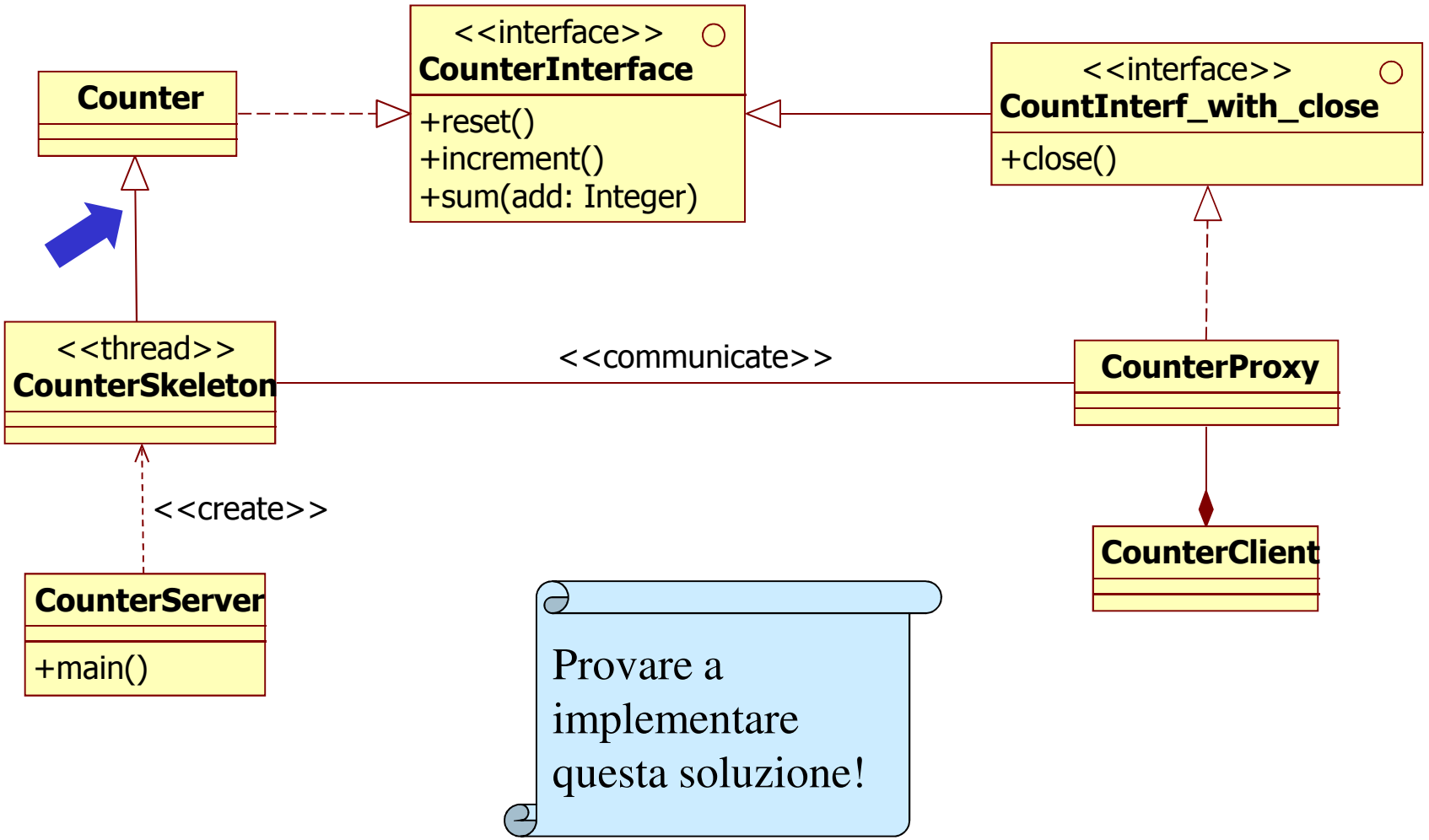
Esempio: Contatore Remoto con ProxyServer e Skeleton

```
// metodi dell'interfaccia ServerInterface
public int sum(int s) throws IOException {
 out.println("<sum> "+Integer.toString(s));
 String strResult = in.readLine();
 return Integer.parseInt(strResult);
}
public int reset() throws IOException {
 out.println("<reset>");
 String strResult = in.readLine();
 return Integer.parseInt(strResult);
}
public int increment() throws IOException {
 out.println("<incr>");
 String strResult = in.readLine();
 return Integer.parseInt(strResult);
}
public void close() throws IOException {
 System.out.println("closing...");
 out.println("<end>");
 socket.close();
}}
```

Esempio: Contatore Remoto con ProxyServer e Skeleton

```
public class CounterServer {
  void exec() {
  ServerSocket serverSocket;
  Socket clientSocket;
  try {
   serverSocket = new ServerSocket(CounterInterface.PORT);
   System.out.println("Started: " + serverSocket);
   while (true) {
     clientSocket = serverSocket.accept();
     new CounterSkeleton(clientSocket).start();
   }
  } catch (IOException e) {
   System.err.println();
  }
 }
 public static void main(String[] args) {
  new CounterServer().exec();
 }
}
```

Design diagram (skeleton usa delega)



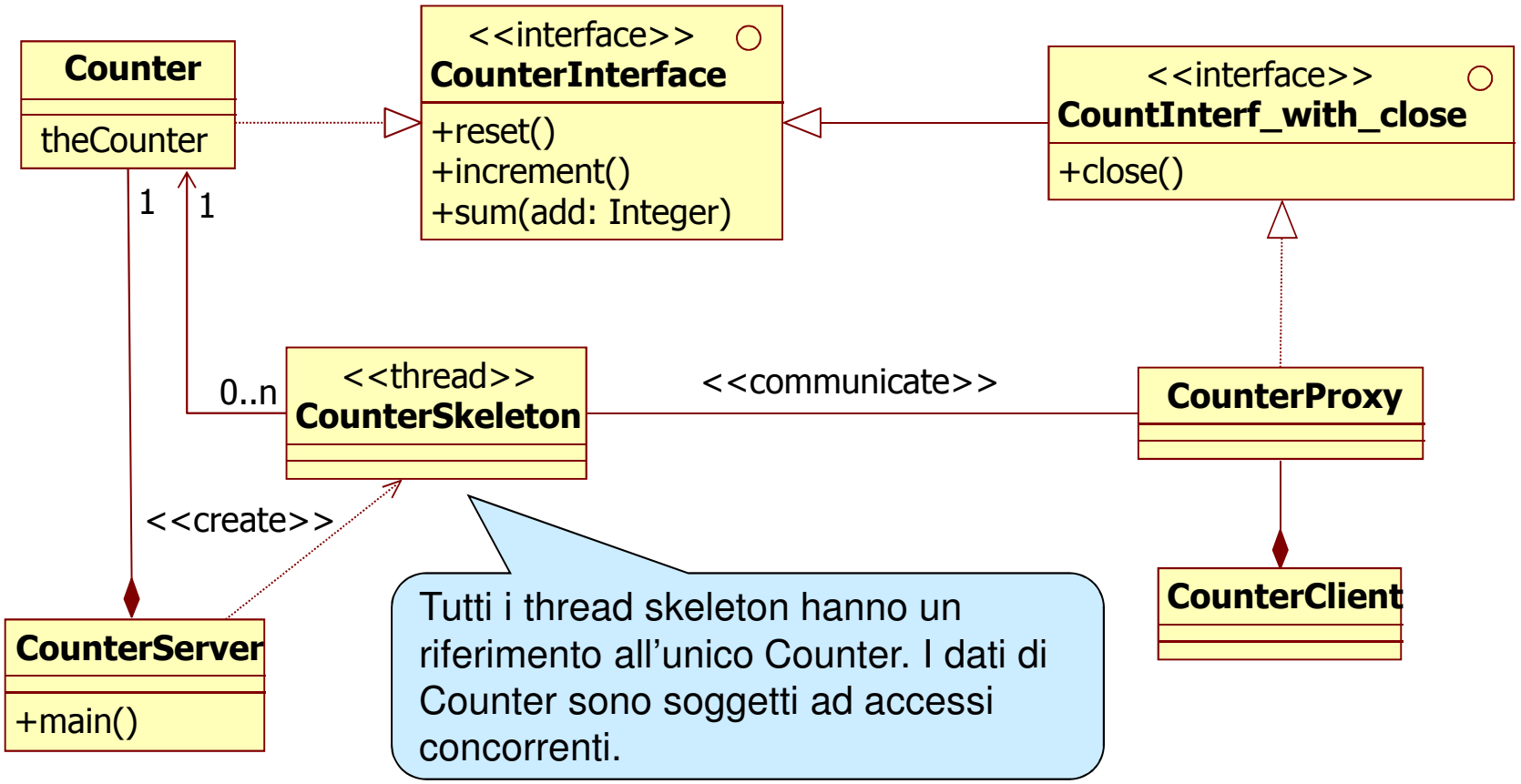
Accesso concorrente a oggetti remoti

- Nell'esempio del contatore remoto avevamo che ogni client chiedeva al server la creazione di un proprio contatore privato.
- In questo caso evidentemente non ci sono problemi di concorrenza.
- Vediamo adesso come si gestisce un server che permette l'accesso concorrente ad un *oggetto condiviso da diversi client*.
- Ci sono le solite problematiche:
  - Corse critiche, da evitare attraverso mutua esclusione
  - ...

Esempio: Contatore Remoto condiviso

- Vediamo l'esempio del contatore remoto, ma questa volta il contatore è condiviso tra tutti i client.

Design diagram (skeleton usa delega)



Contatore thread-safe

```
public class Counter implements CounterInterface {
  private int theCounter;
    public Counter() {
      theCounter = 0;
    }
    public synchronized int reset() {
      theCounter = 0;
      return theCounter;
    }
    public synchronized int increment() {
      theCounter++;
      return theCounter;
    }
    public synchronized int sum(int s) {
      theCounter += s;
      return theCounter;
    }
}
```

I metodi sono synchronized: un solo client alla volta accede al contatore condiviso.