Esercitazione: Progettazione e realizzazione di un «alternatore»

Laura Semini, thanks to Carlo Montangero, Ingegneria del Software

Dipartimento di Informatica, Università di Pisa

PREMESSA

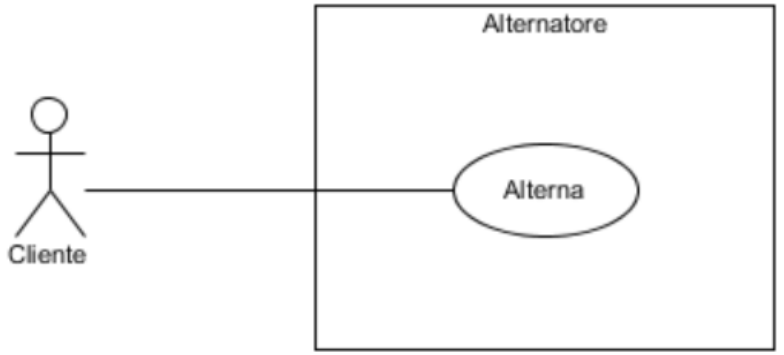
L’esempio è puramente simbolico.

Nessun progettista sano di mente definirebbe un’architettura così complessa per un requisito così banale.

Il requisito è un’astrazione. Una possibile istanza è la risoluzione di un problema di load balancing.

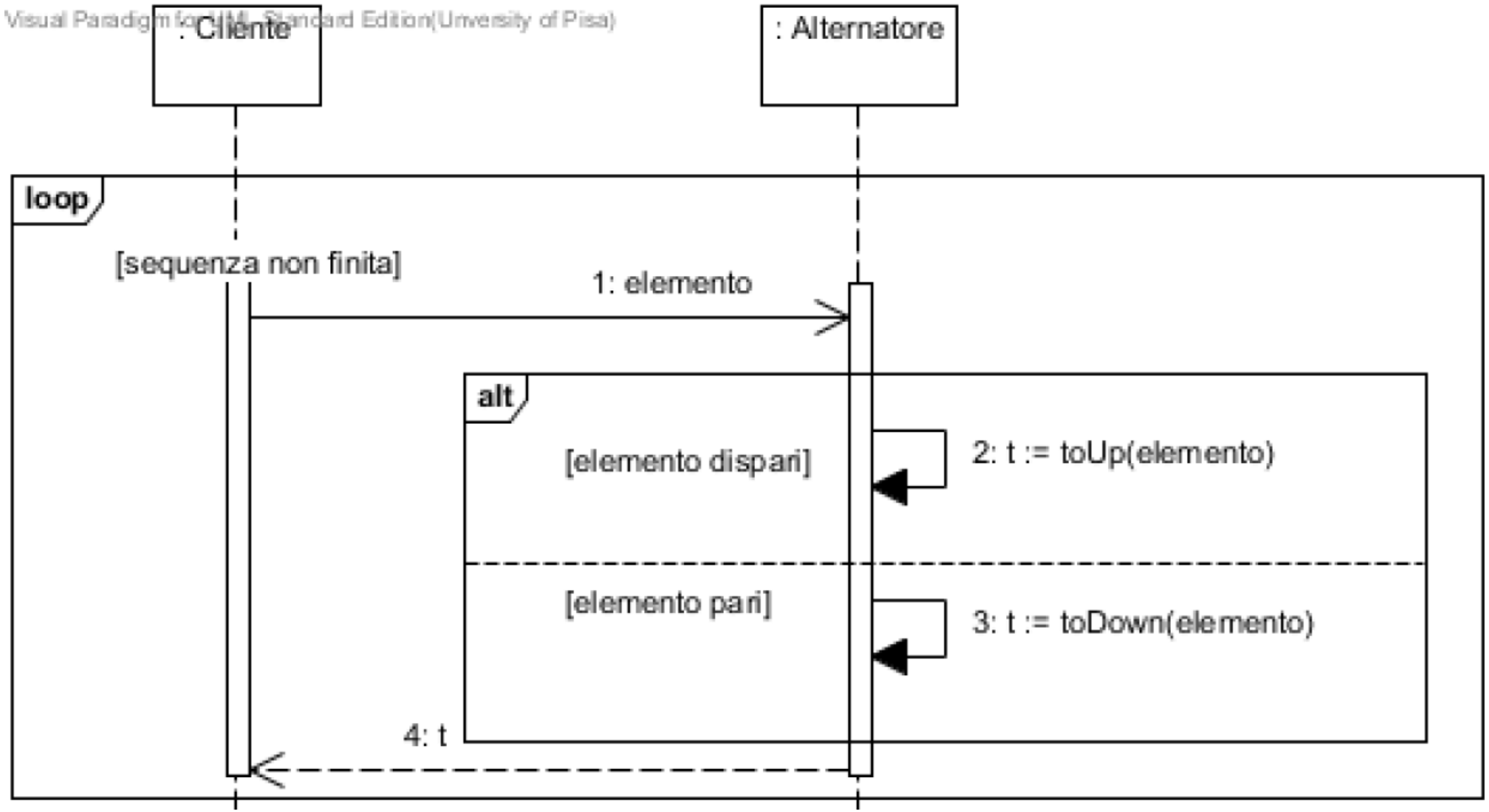
Della soluzione ci interessano solo gli aspetti di comunicazione, non la funzionalità realizzata dalle componenti, che per semplicità assumiamo la più banale possibile.

Caso d'uso: Alterna

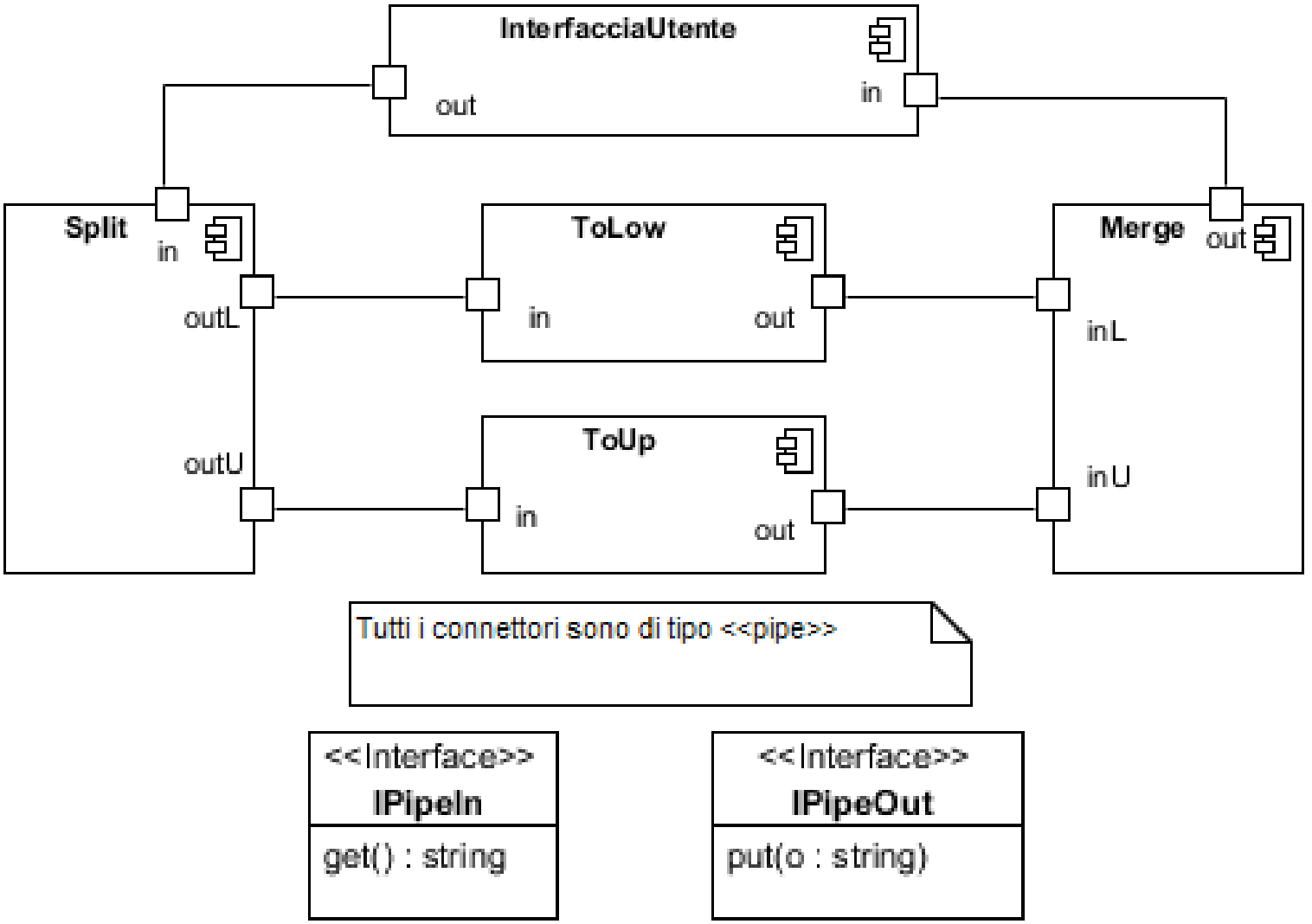


Name	Documentation
Alterna	Il sistema applica alternativamente una di due trasformazioni agli elementi di una sequenza. Per esempio, su una sequenza di caratteri, le trasformazioni potrebbero essere "a maiuscolo" e "a minuscolo". La sequenza 'abCdEF' diventa 'AbCdEf'. Convenzionalemente, nel seguito chiameremo toUp e to Down le due trasformazioni.
Alternatore	Questo sistema deve trasformare una sequenza di elementi, fornita dal Cliente, come specificato nel caso d'uso Alterna. Ai fini dell'esercizio, assumiano che le trasformazioni siano sufficientemente pesanti da giustificare un'architettura che ne richieda la dislocazione su macchine diverse.

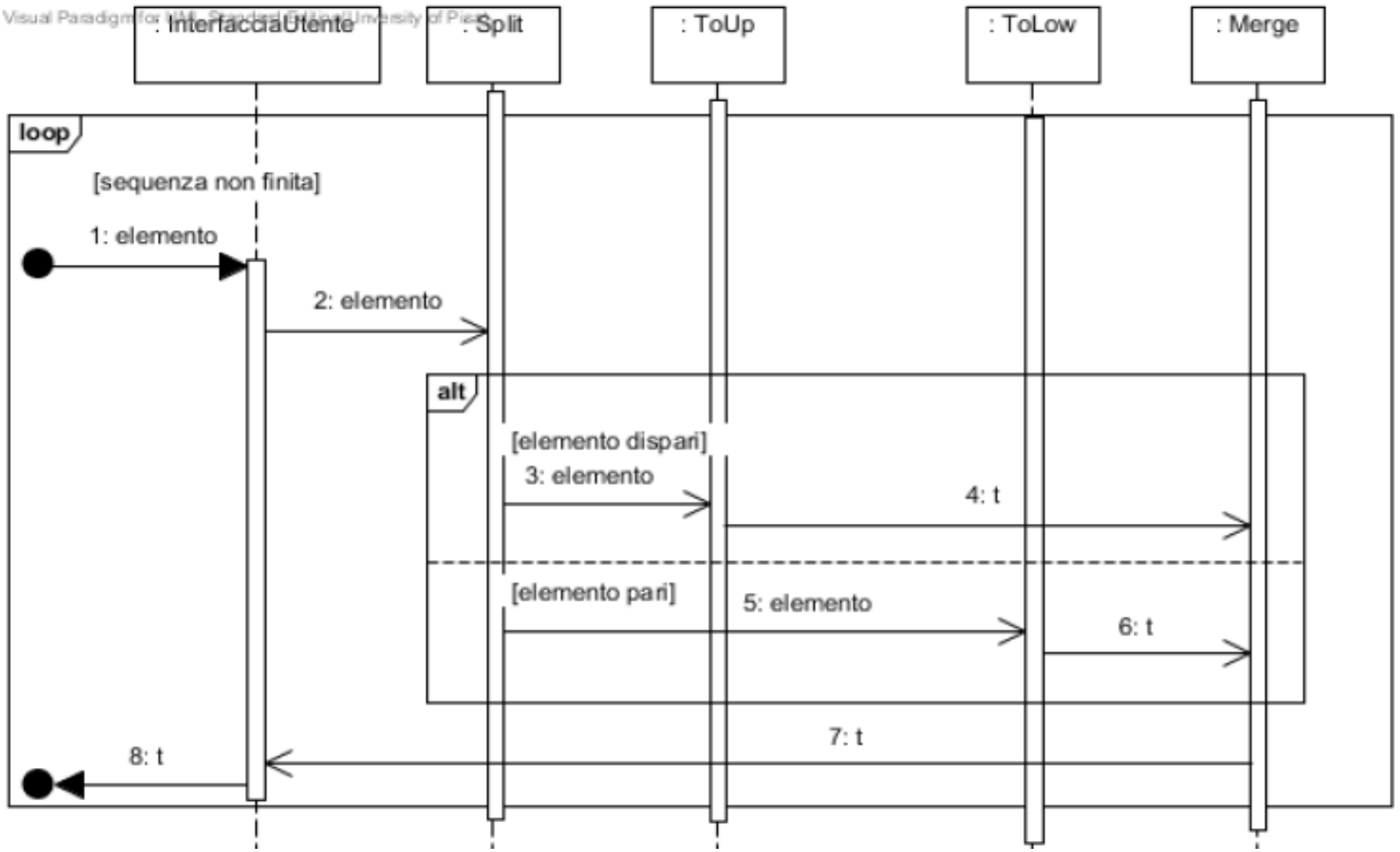
Realizzazione di Alterna



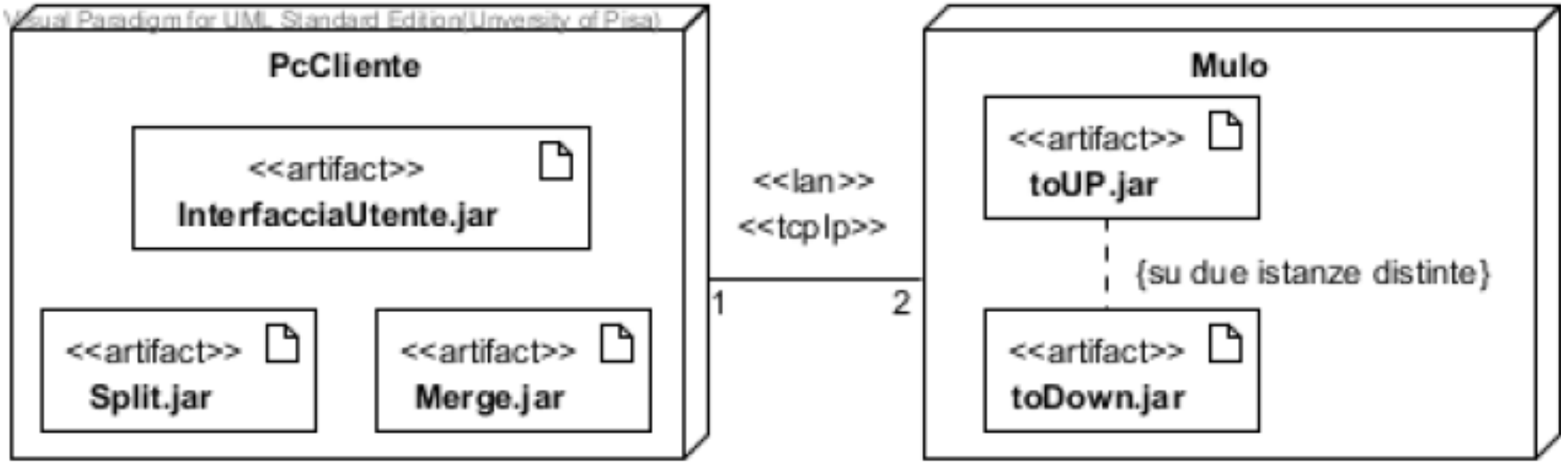
Vista C&C


InterfacciaUtente
out
in
Split
in
outL
outU
ToLow
in
out
ToUp
in
out
Merge
out
inL
inU
Tutti i connettori sono di tipo <<pipe>>
<<Interface>>
IPipeIn
get() : string
<<Interface>>
IPipeOut
put(o : string)

Realizzazione di Alterna



Esempio: dislocazione su rete locale

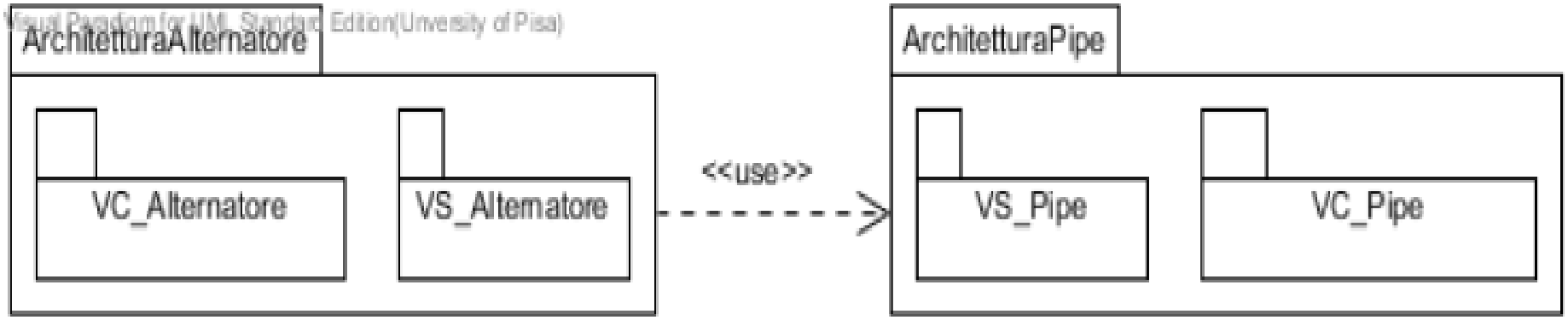


Name	Documentation
Mulo	E' il tipo dei due nodi di elaborazione su cui sono dislocati gli artefatti toUp e toDown, rispettivamente.

Documentation

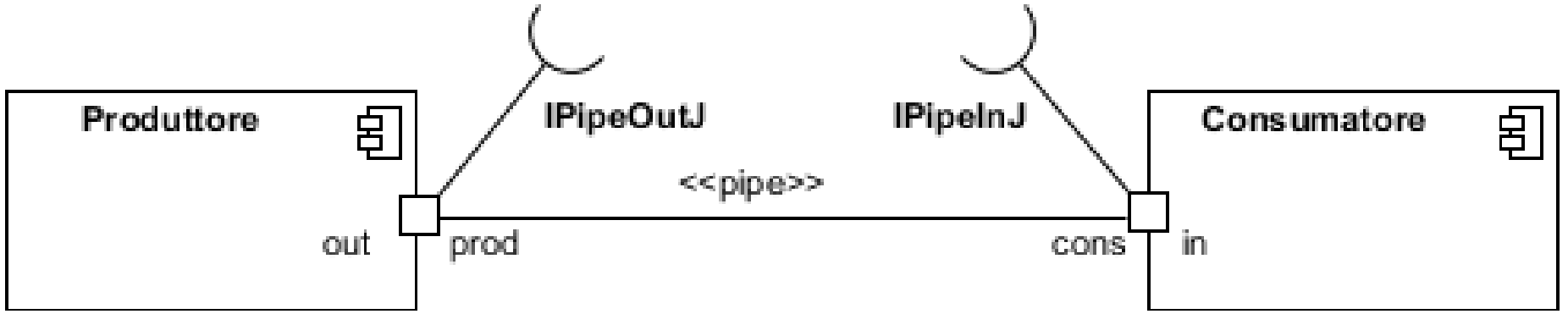
Il collegamento tra il pcCliente e i due Muli deve offrire il protocollo tcp-ip, richiesto dalla realizzazione scelta delle pipe.

Struttura



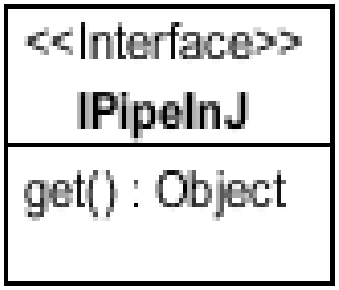
Name	Documentation
ArchitetturaAlternatore	Descrizione delle componenti che realizzano il sistema Alternatore.
ArchitetturaPipe	Descrizione del protocollo associato a un connettore <<pipe>>. Si assume che l'operazione di put sia asincrona, quella di get sincrona, ossia bloccante. Lo scopo di questo package è di fattorizzare le informazioni necessarie per connettere due componenti con una pipe (vista comportamentale) e di condividerne la realizzazione (vista strutturale e codice associato).

Pipe : contesto generico

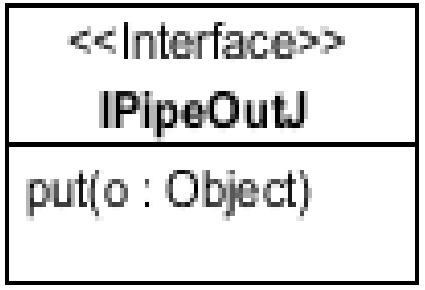


Name	Documentation
IPipeOut	Il porto out del produttore richiede alla pipe l'operazione put, per fornire il prossimo elemento. L'operazione non è bloccante (si assume capacità infinita del buffer). Data la genericità di questi elementi, il tipo degli oggetti trasferiti è Object, con il vincolo che sia stato sovrascritto il metodo toString.
IPipeIn	Il porto in del consumatore richiede l'operazione get, offerta dalla pipe, per ottenere il prossimo elemento da consumare. L'operazione è bloccante, se la pipe è vuota. Data la genericità di questi elementi, il tipo degli oggetti trasferiti è Object.

Pipe: interfacce generiche




```
package javapipe;

public interface IPipeInJ {
    public Object get();
}
```




```
package javapipe;

public interface IPipeOutJ {
    public void put(Object element);
}
```

Realizzazione pipe

Utilizziamo

- *socket* da java.net
- *stream* da java.io

Lo stream realizza

- la coda
- il blocco in assenza di dati in input

Usiamo *Object* Stream

- serializza gli oggetti di tipo *Serializable*
  - e.g., String

Realizzazione IPipeOut

```
public class Prod implements IPipeOutJ {

    private ObjectOutputStream out;

    public Prod(String consAddress, int consPort)  {
        try {
            Socket consSocket = new Socket(consAddress, consPort);
            OutputStream outS = consSocket.getOutputStream();
            ObjectOutputStream out = new ObjectOutputStream(outS);
        } catch (IOException e) {
            /** minima (pessima) gestione delle eccezioni */
            System.err.println("put failed.");
            System.exit(1);
        }
}
```

Realizzazione *put*

```
public void put(Object anObject) {
    try {
        out.writeObject(anObject);
    } catch (InvalidClassException e) {
        /** minima (pessima) gestione delle eccezioni */
        System.err.println("put: problemi di serializzazione");
        System.exit(1);
    }catch (IOException e) {
        System.err.println("put failed.");
        System.exit(1);
    }
}
```

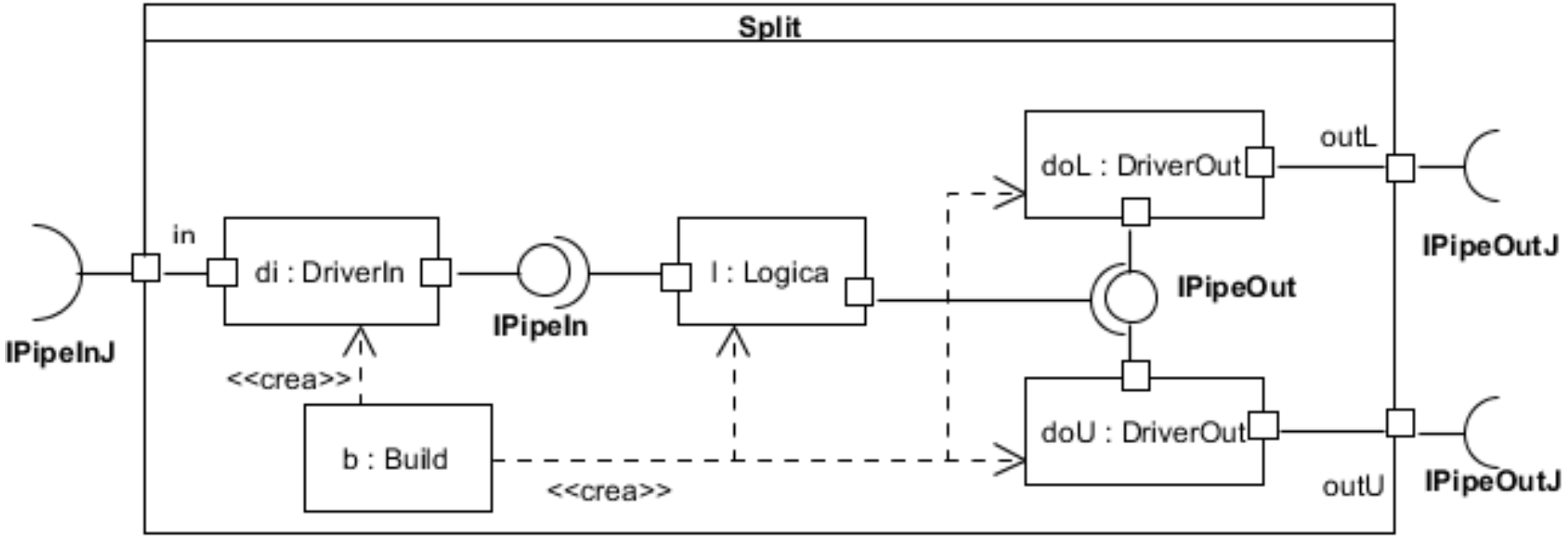
Realizzazione IPipeIn

```
public class Cons implements IPipeInJ {
    private ObjectInputStream in;

    public Cons(Integer prodPort)  {
        try {
            ServerSocket prodSocket = new ServerSocket(prodPort);
            Socket consSocket = prodSocket.accept();
            in = new ObjectInputStream(consSocket.getInputStream());
        } catch (IOException e) {
            /** minima (pessima) gestione delle eccezioni */
            System.err.println("Cons build failed");
            System.exit(1);
        }
    }
```

Realizzazione *get*

```
public Object get() {
    Object res = null;
    try {
        res = in.readObject();
    } catch (ClassNotFoundException e) {
        /** minima (pessima) gestione delle eccezioni */
        System.err.println("get: problemi di serializzazione");
        System.exit(1);
    } catch (IOException e) {
        System.err.println("get failed.");
        System.exit(1);
    }
    return res;
}
```

Split in dettaglio



DriverIn

```
/**import javapipe.Cons; */

public class DriverIn implements IDriverIn {

    javapipe.Cons in;

    public DriverIn(String pp) {
        Integer prodPort = Integer.decode(pp);
        in = new javapipe.Cons(prodPort);
    }

    public String get() {
        return (String) in.get();
    }
}
```

DriverOut

```
public class DriverOut implements IDriverOut {
    javapipe.Prod out;

    /** @param address IP address of the consuming component
       * @param port port number of the above
       * @return a DriverOut connected to the out port
       * identified by the previous arguments
       */
    public DriverOut(String consAddress, String consPort) {
        out = new javapipe.Prod(consAddress, Integer.decode(consPort));
    }

    public void put(String s) {
        out.put(s);
    }
}
```

Logica: struttura

```
package alternatore.split;

import alternatore.drivers.driverIn.IDriverIn;
import alternatore.drivers.driverOut.IDriverOut;

/** Realizza il comportamento di Split, usando tre driver dei porti.
 */
class Logica {

    private IDriverIn in;
    private IDriverOut outL;
    private IDriverOut outU;

    Logica(IDriverIn in, IDriverOut outL, IDriverOut outU) {
        this.in = in;
        this.outL = outL;
        this.outU = outU;
    }
```

Logica: comportamento

```
/** minimo comportamento per la Logica
 */
void run() {
    while (true) {
        outL.put(in.get());
        outU.put(in.get());
    }
}
```

Costruzione di split: supporto

un oggetto ParmsDB è una tabella

- costruita dalla riga di comando che attiva il main
- <switch, valore>
- per split, i dati sui porti

get(sw) restituisce il valore associato a sw

```
private static ParmsDB db;

public static void main(String [] args) {
    ParmsDB db = new ParmsDB(args);   /** -pp ... -lca ... -lcp ... -uca ... -ucp ... */
```

Costruzione di split

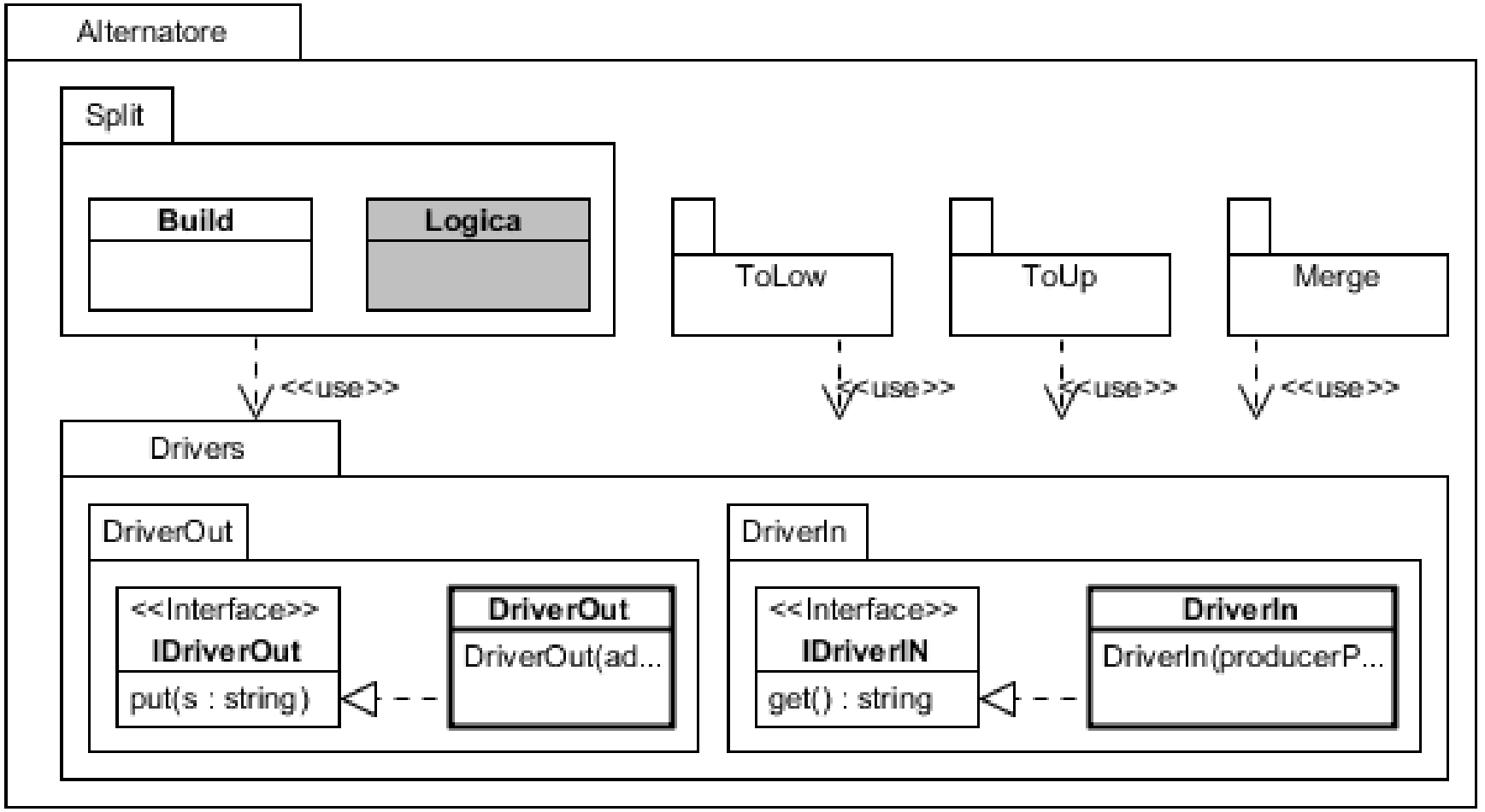
```
import util.ParmsDB;

public class Build {

    private static Logica logica;

    private static ParmsDB db;

    public static void main(String [] args) {
        ParmsDB db = new ParmsDB(args);   /** -pp ... -lca ... -lcp ... -uca ... -ucp ... */
        build();
        logica.run();
    }
```

Build

```
private static void build() {
    String ProducerPort = db.getPar("pp"); /** port to listen to producer */
    IDriverIn in = new DriverIn(ProducerPort);

    String LowConsumerAddress = db.getPar("lca"); /** low consumer address and port
    String LowConsumerPort = db.getPar("lcp");
    IDriverOut outL = new DriverOut(LowConsumerAddress, LowConsumerPort);

    String UpConsumerAddress = db.getPar("uca"); /** up consumer address and port */
    String UpConsumerPort = db.getPar("ucp");
    IDriverOut outU = new DriverOut(UpConsumerAddress, UpConsumerPort);

    logica = new Logica(in,outL,outU);
}
```

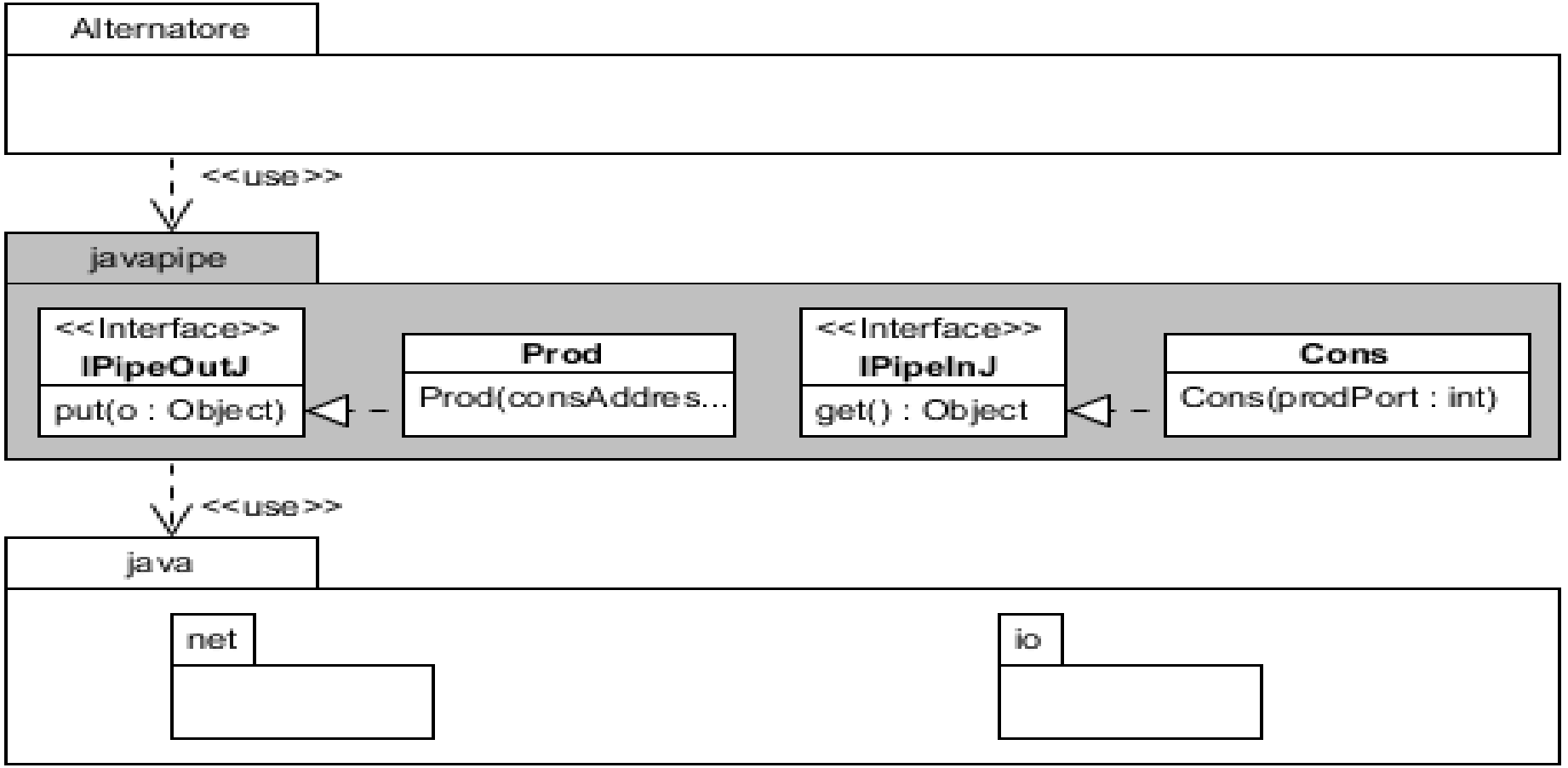
Vista strutturale

- grigio: modifiche al comportamento
- bordo spesso: diversa implementazione pipe



Vista globale

grigio: responsabilità aziendale



Dislocazione

Uno script di costruzione globale

- lancia le componenti sulle macchine scelte