

Progettazione e realizzazione di un alternatore

Versione 1.0



Carlo Montangelo, Laura Semini

Corso di Ingegneria del Software

Dipartimento di Informatica - Università di Pisa

Gennaio 2011

Table of Contents

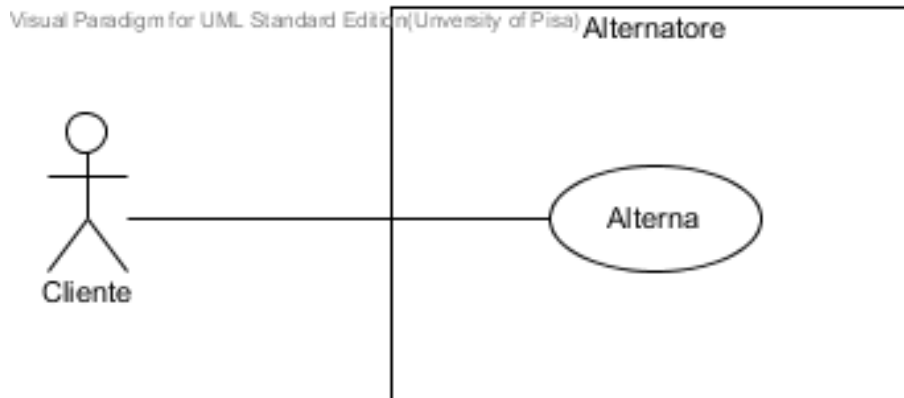
CasoD'usoAlterna	3
Summary	3
RealizzazioneCasoDUsoAlterna-Narrativa	4
Summary	4
VistaComporamentaleAlternatore	5
Summary	5
RealizzazioneCasoDUsoAlterna-Componenti	6
Summary	6
DislocazioneSuReteLocale	7
Summary	7
Alternatore.....	8
Summary	8
VistaComportamentaleDiUnaGenericaPipe.....	9
Summary	9
InterfacceDellaPipe	10
Summary	10
ComportamentoPipe	11
Summary	11
VistaStrutturaleAlternatoreEPipe	12
Summary	12
Split	14
Summary	14
RealizzazioneCasoDUsoAlterna-Split.....	15
Summary	15
VistaStrutturaleAlternatore	16
Summary	16
Appendice: codice	17

Avvertenza



Questa versione è messa a disposizione in vista della vicina verifica. Tra una versione più curata e completa, e una tempestiva, gli autori hanno preferito la seconda, pur consapevoli di esporsi a critiche. In particolare il codice allegato non è stato testato, ed è presentato solo come indicazione di come i concetti astratti usati nei modelli UML trovino una corrispondenza concreta in quelli di un linguaggio di programmazione, in questo caso Java.

Use Case Diagram

Caso D'uso Alterna



Summary

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'. Convenzionalmente, 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, assumiamo che le trasformazioni siano sufficientemente pesanti da giustificare un'architettura che ne richieda la dislocazione su macchine diverse.

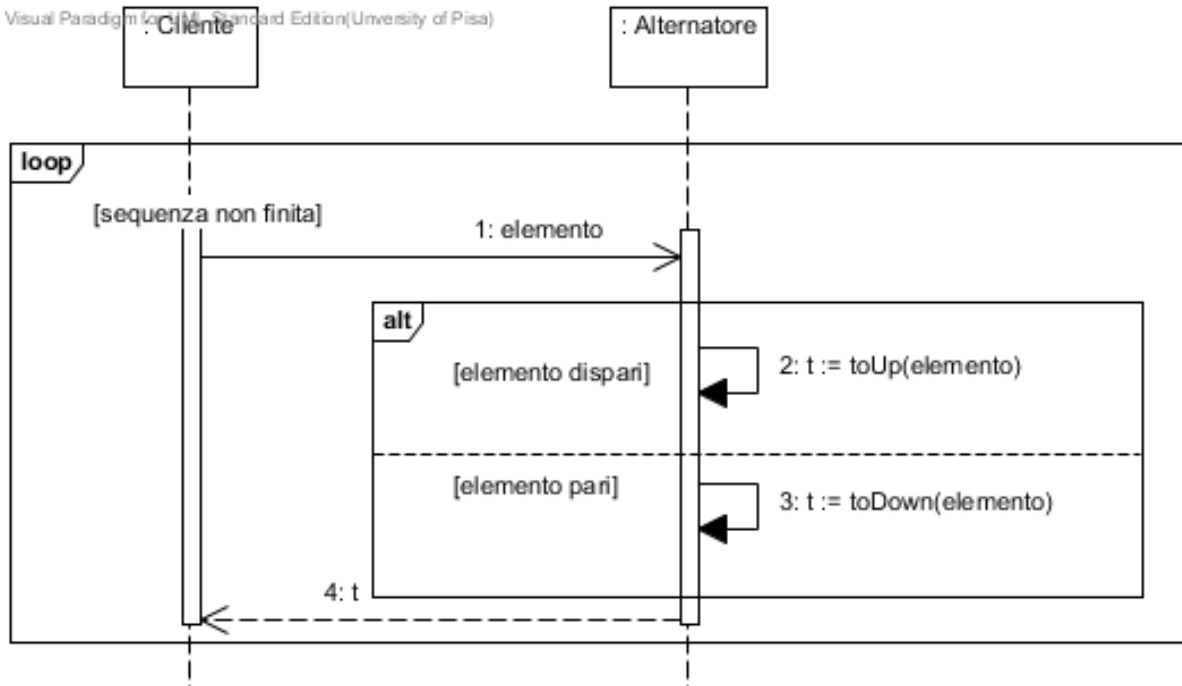
Documentation

Questo documento è un esempio di documentazione di un progetto software secondo gli standard del corso di IS. S'inizia quindi dai casi d'uso, qui uno solo.

Sequence Diagram

Realizzazione Caso D'Uso Alternativa

Visual Paradigm for UML Standard Edition (University of Pisa)



Summary



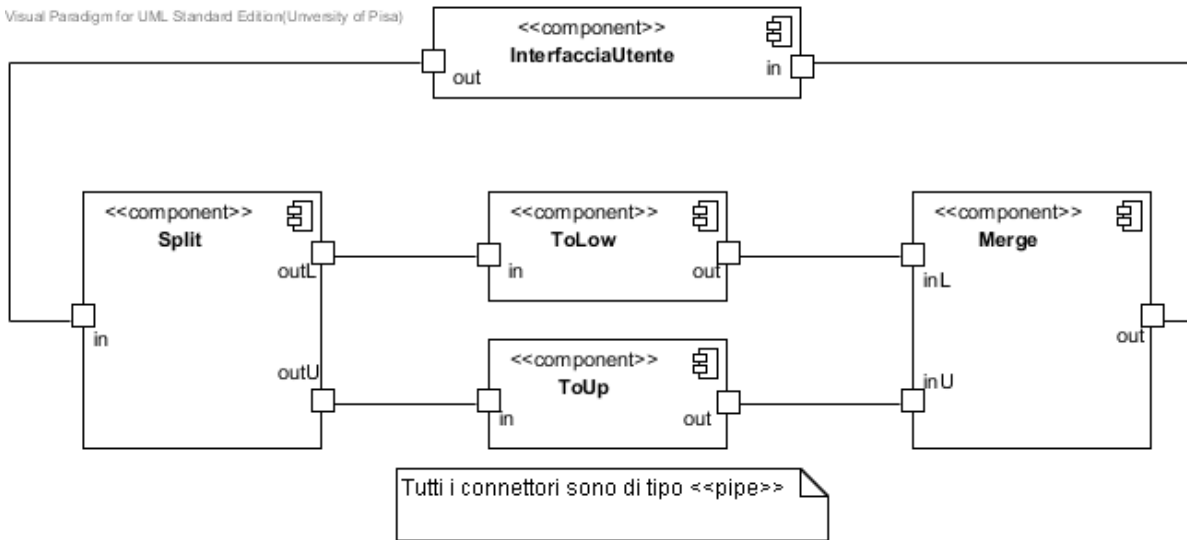
Documentation

Il diagramma mostra il comportamento atteso dal sistema.







Component Diagram

VistaComporamentaleAlternatore

Visual Paradigm for UML Standard Edition(University of Pisa)



Summary

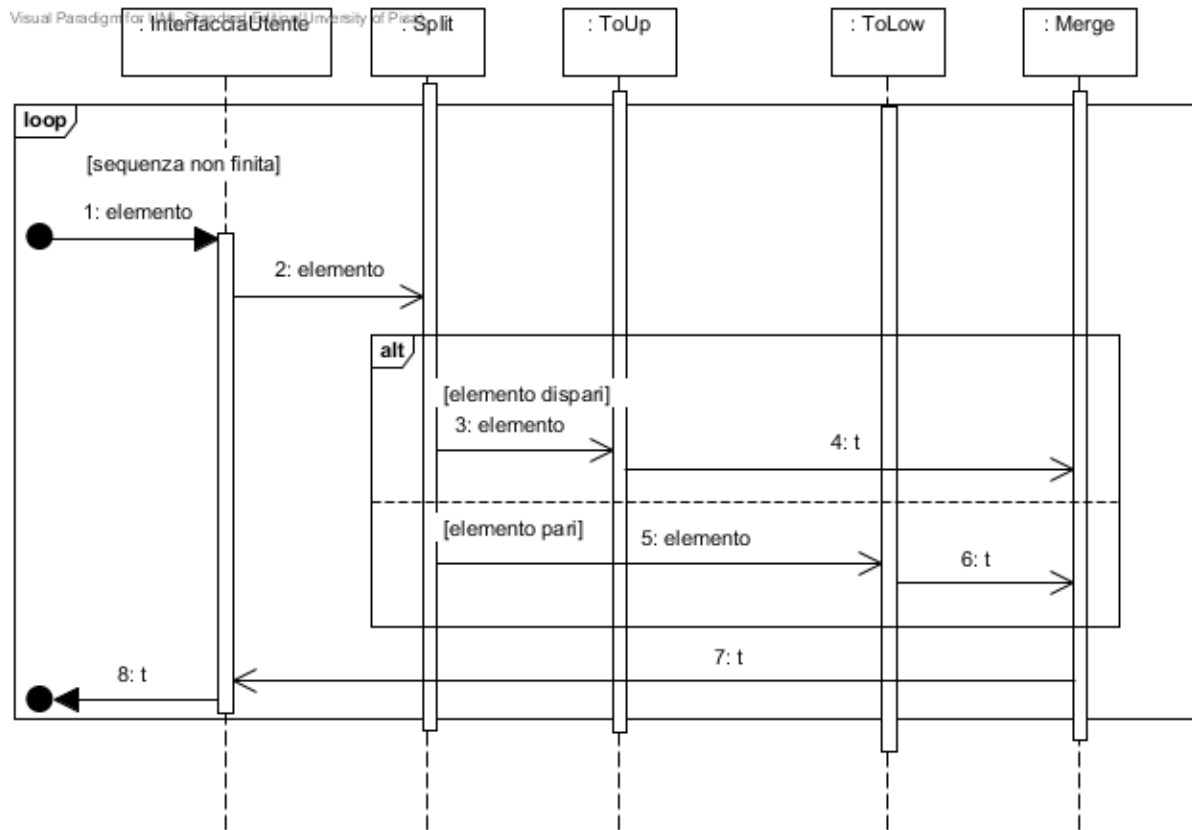
Name	Documentation
 InterfacciaUtente	Questa componente riceve dal Cliente l'elemento da alternare e si comporta come un produttore per la componente Split
 Split	Questa è la componente che opera lo smistamento.
 ToLow	Questa è la componente che opera la trasformazione toLow.
 Merge	Questa è la componente che opera la fusione.
 ToUp	Questa è la componente che opera la trasformazione toUp.
 N/A	Tutti i connettori sono di tipo <<pipe>>

Documentation

In rispetto del requisito non funzionale, si introducono due componenti distinte per le trasformazioni. Per ulteriore flessibilità, anche per lo smistamento e la fusione degli elementi sono state introdotte due componenti distinte.

Sequence Diagram

Realizzazione Caso D'uso Alterna-Componenti



Summary

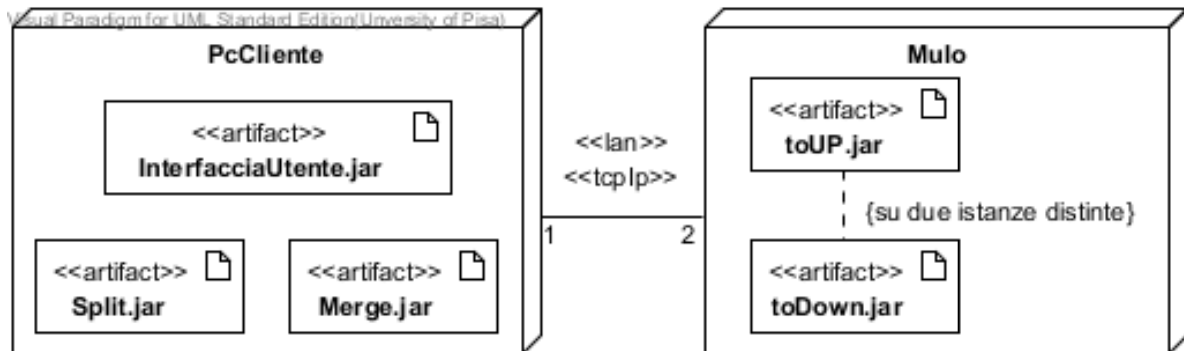
Name	Documentation
------	---------------

Documentation


Viene mostrato il flusso di dati tra le componenti per realizzare il caso d'uso Alterna.

Deployment Diagram

Dislocazione Su Rete Locale



Summary

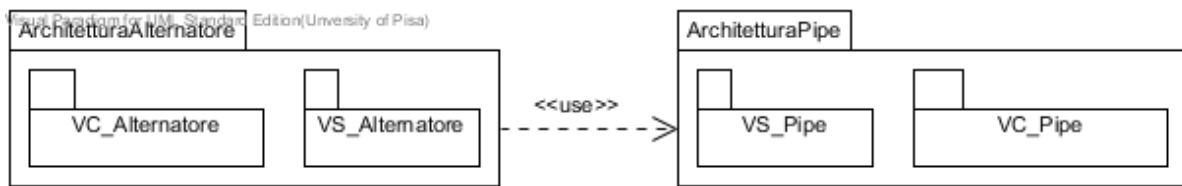
Name	Documentation
 Mulo	E' il tipo dei due nodi di elaborazione su cui sono dislocati gli artefatti toUp e toDown, rispettivamente.

Documentation







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

Package Diagram

Alternatore



Summary

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).
 VC_Alternatore	Questa vista mostra la struttura C&C del sistema Alternatore. Si tratta di un caso particolare di un pattern ben noto della computazione parallela, il Farm.
 VS_Alternatore	Questo package contiene le viste strutturali dell'architettura del sistema.
 VS_Pipe	Questo package contiene la vista strutturale della realizzazione Java del connettore <<pipe>> con il supporto di socket alla comunicazione.
 VC_Pipe	Questo package contiene la vista comportamentale di un connettore <<pipe>> tra un generico produttore e un generico consumatore.

Documentation

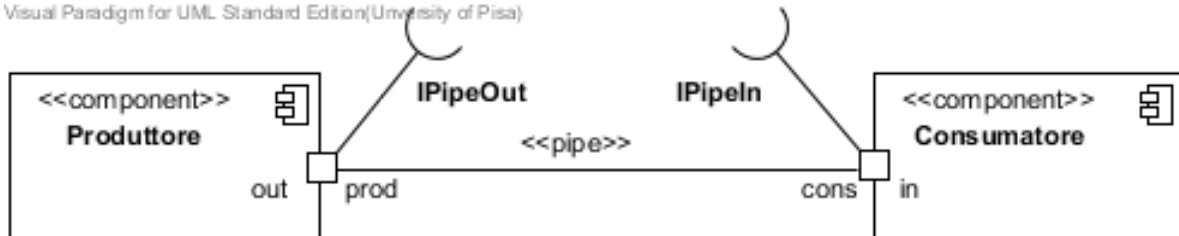
Il progetto architettonico del sistema mostrato in questo diagramma documenta la decisione fatta di utilizzare una realizzazione già disponibile di <<pipe>> per connettere le componenti individuate nella vista C&C (in VC_Alternatore) per rispettare il requisito di distribuire l'elaborazione delle trasformazioni richieste su nodi distinti.

I prossimi quattro diagrammi descrivono la realizzazione del connettore.



Component Diagram

Vista Comportamentale Di Una Generica Pipe

Visual Paradigm for UML Standard Edition (University of Pisa)



Summary

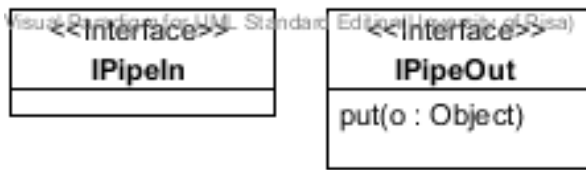
Name	Documentation
 IPipeOut	<p>Il porto out del produttore richiede alla pipe l'operazione put, per fornire il prossimo elemento.</p> <p>L'operazione non è bloccante (si assume capacità infinita del buffer).</p> <p>Data la genericità di questi elementi, il tipo degli oggetti trasferiti è Object, con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

Documentation



Questa vista mostra il generico contesto di una <<pipe>>: una componente produce, l'altra consuma. Il connettore bufferizza i prodotti in attesa della richiesta da parte del consumatore, che, a sua volta, resta bloccato in mancanza di prodotti da utilizzare.

Class Diagram

InterfacceDellaPipe

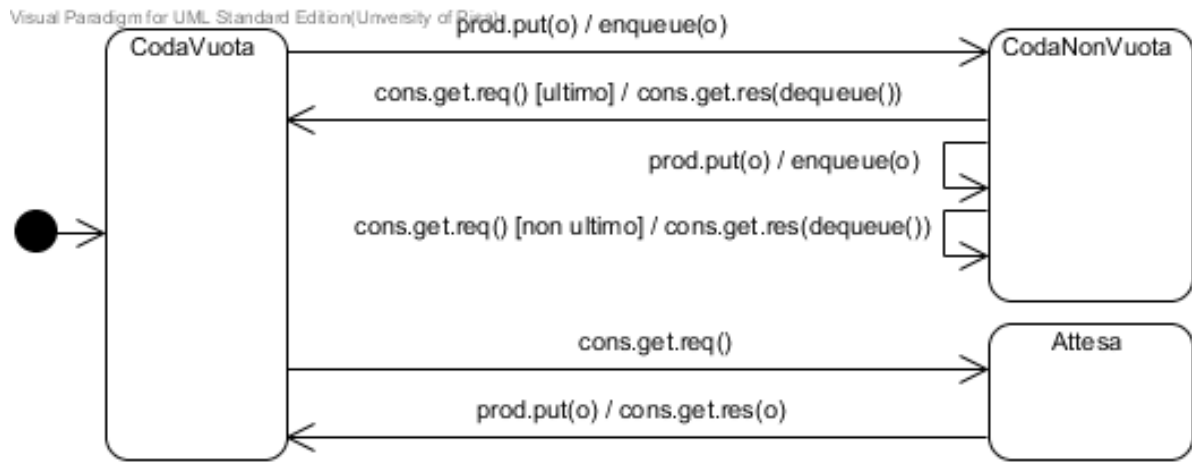


Summary

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

State Machine Diagram

ComportamentoPipe



Summary

Name	Documentation
------	---------------

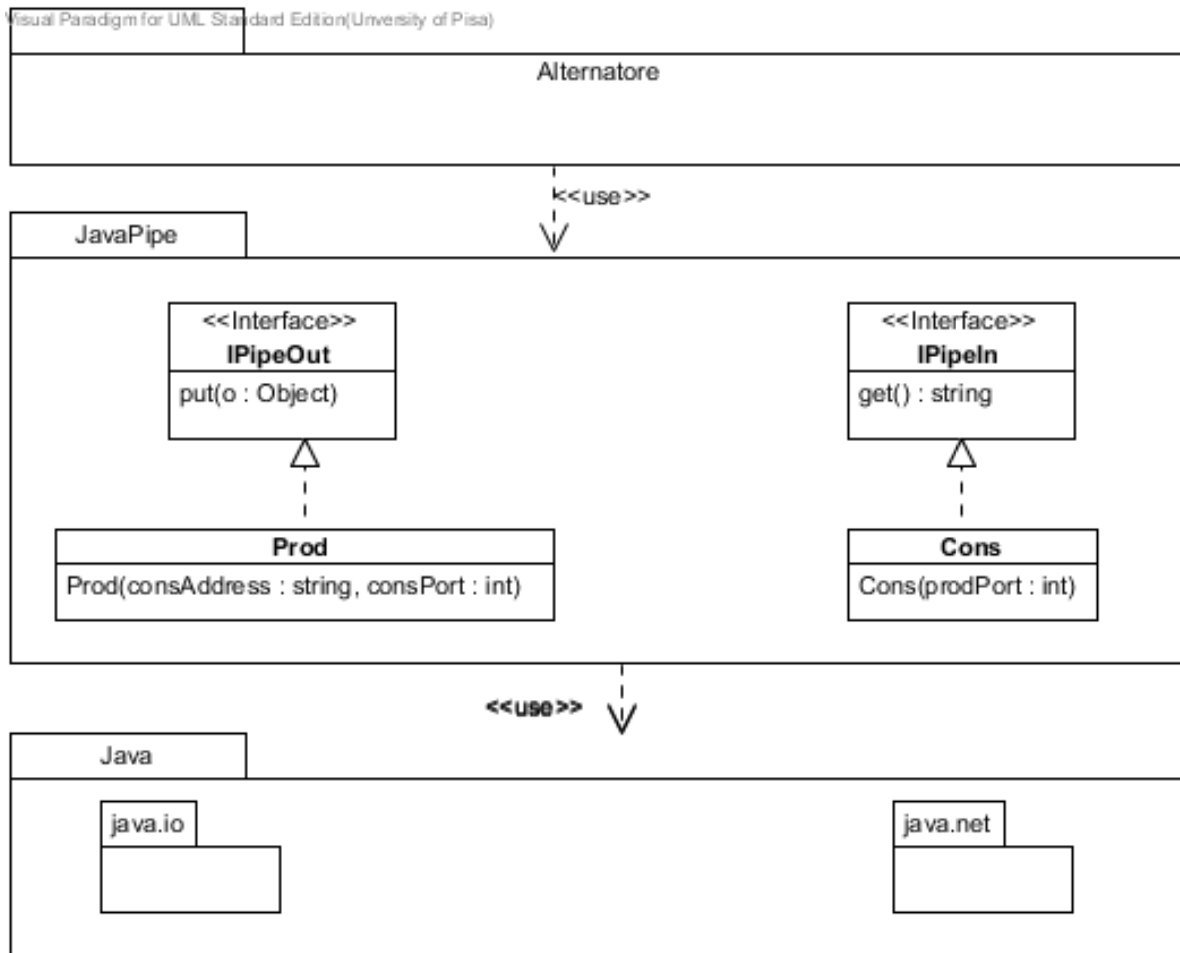
Documentation

Il diagramma mostra l'evoluzione della pipe, nell'ipotesi di capacità infinita.




Convenzioni: l'evento di invocazione dell'operazione op con argomento o è espresso da op.req(o); l'azione di restituzione del risultato r da op.res(r).



Class Diagram

VistaStrutturaleAlternatoreEPipe



Summary

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.
 Prod	Realizza il lato produttore della pipe, costruendo una socket (lato client) e implementando l'operazione put usando il package java.io per fare la enqueue necessaria.

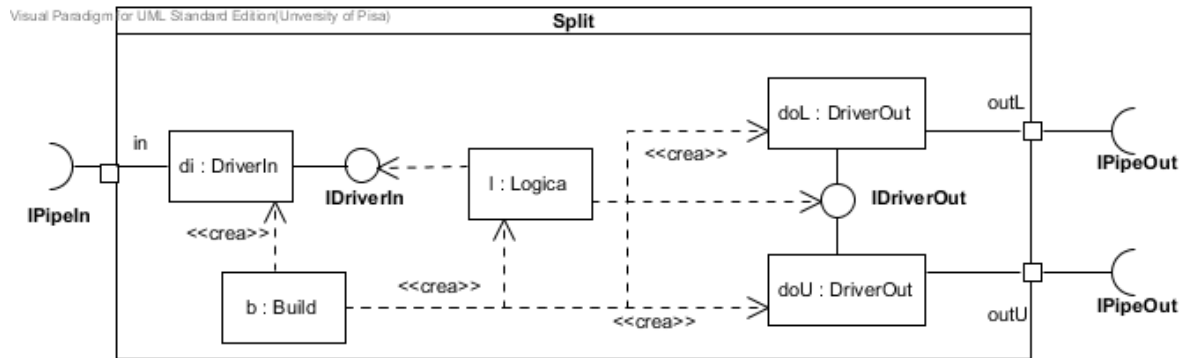
 Cons	Realizza il lato consumatore della pipe, costruendo una socket (lato server) e implementando l'operazione get usando il package java.io per fare la dequeue.
 java.net	Si utilizzano in particolare le classi: ServerSocket, per la creazione di socket lato server; Socket, per la creazione di socket lato client

Documentation

I consumatori vanno attivati prima dei produttori.

Composite Structure Diagram

Split



Summary

Name	Documentation
------	---------------

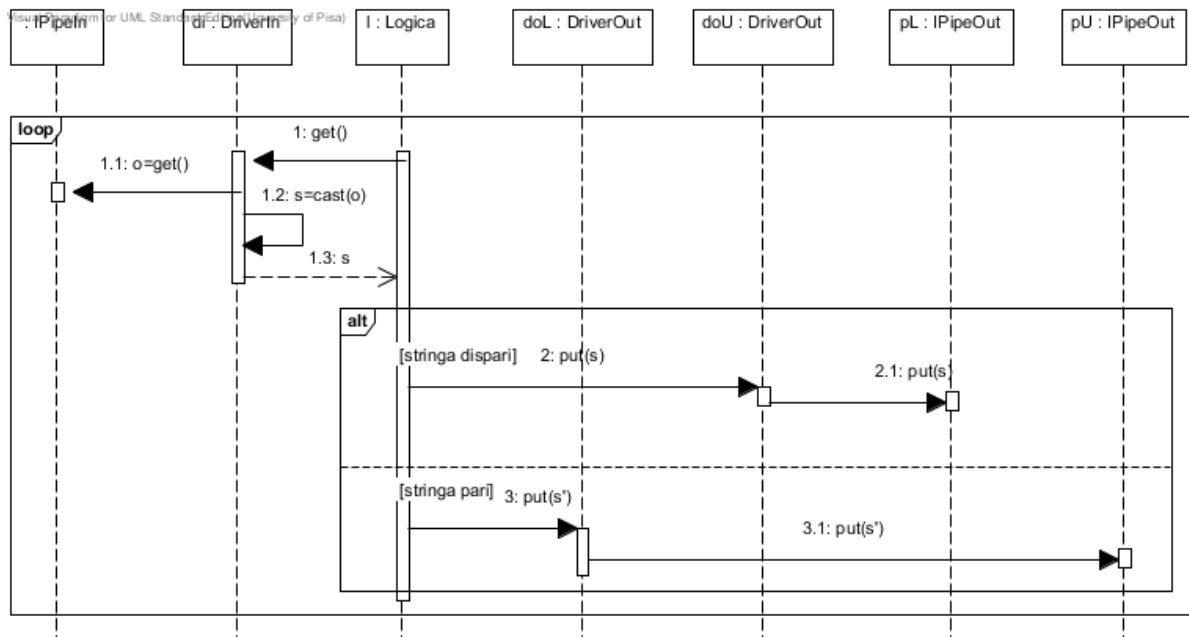
Documentation

La componente Split è realizzata:

dalla Logica, che realizza la funzionalità richiesta alla componente (smistare gli elementi da alternare) dai Driver, che realizzano eventuali conversioni di tipo per interfacciare la Logica con le pipe. da una Build, che costruisce la componente e invoca il metodo run() della Logica.

Sequence Diagram

Realizzazione Caso D'uso Alterna-Split



Summary

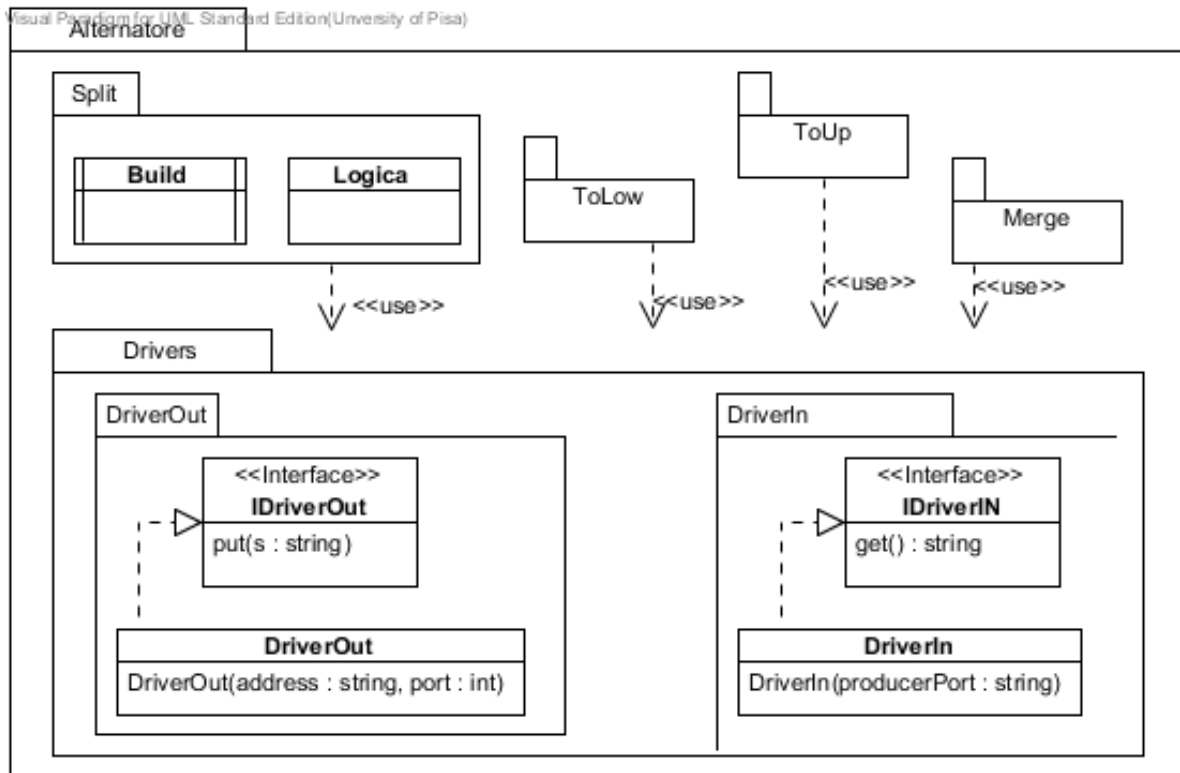
Name	Documentation
------	---------------

Documentation


Viene illustrato il contributo della componente Split alla realizzazione del caso d'uso Alterna.

Class Diagram

VistaStrutturaleAlternatore



Summary

Name	Documentation
 DriverIn	L'implementazione del metodo get in questo caso restituisce semplicemente la stringa restituita dalla get di IPipeIn. In generale, un driver che invoca la get di IPipeIn deve convertire la stringa (che è la serializzazione dell'oggetto spedito dal produttore) in un oggetto del tipo richiesto dalla componente.

Documentation

Tutte le componenti usano la stessa realizzazione dei Driver.

Appendice: codice

```
package split;

import drivers.driverIn.DriverIn;
import drivers.driverIn.IDriverIn;
import drivers.driverOut.DriverOut;
import drivers.driverOut.IDriverOut;
import util.ParmsDB;

public class Build {

    private static Logica logica;

    public static void main(String [] args) throws Exception {
        build(args);
        logica.run();
    }
    /**
     * Costruisce il distributore Split.
     *
     * @param args of the switches on the command line
     * @throws Exception
     */
    private static void build(String[] args) throws Exception {
        ParmsDB db = new ParmsDB(args);
        IDriverIn in = new DriverIn(db.getPar("pp")); // port to
listen to producer
        IDriverOut outL = new DriverOut(db.getPar("cla"),
db.getPar("clp")); // low consumer address and port
        IDriverOut outU = new DriverOut(db.getPar("cua"),
db.getPar("cup")); // up consumer address and port
        logica = new Logica(in,outL,outU);
    }
}
```

```
package split;

import java.io.IOException;

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

/**
 *
 * Realizza il comportamento di split, usando i tre driver dei porti.
 *
 * @author Carlo Montangero
 */
class Logica {

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

    /**
     * @param in driver del porto in
     * @param outL driver del porto outL
     * @param outU driver del porto outL
     */
    Logica(IDriverIn in, IDriverOut outL, IDriverOut outU) {
```

```

        this.in = in;
        this.outL = outL;
        this.outU = outU;
    }

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

```

```

package drivers.driverIn;

import java.io.IOException;

public interface IDriverIn {
    public String get() throws IOException;
}

```

```

package drivers.driverIn;

import java.io.IOException;
import java.pipe.Cons;

public class DriverIn implements IDriverIn {

    Cons in;

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

    /** (non-Javadoc)
     * @see drivers.driverIn.IDriverIn#get()
     */
    public String get() throws IOException {
        String input = in.get();
        /**
         * Il comando che segue è inutile in questo caso particolare.
         * In generale qui si deve ricostruire (a partire da input)
         l'oggetto del tipo da restituire.
         * La soluzione più generale passa attraverso l'uso
         dell'interfaccia Serializable.
         */
        String res = input;
        return res;
    }
}

```

```

package drivers.driverOut;

/**
 * Interfaccia verso il driver del porto di out.
 *
 * @author Carlo Montangero
 */
public interface IDriverOut {
    /**
     * Specializza IPipeOut al tipo trattato in Alternatore.
     * @see java.pipe.javaPipe.prod.IPipeOut
     *
     * @param c carattere inviato all'esterno della componente
     */
    void put(String c);
}

```

```

package drivers.driverOut;

import java.io.IOException;
import java.net.UnknownHostException;
import java.pipe.Prod;

/**
 * Drives an output port of the component.
 *
 * @author Carlo Montangero
 */
public class DriverOut implements IDriverOut {
    Prod out;

    /**
     * @param address IP address of the consuming component
     * @param port port number of the above
     * @return a DriverOut connected to the in port identified by the
previous arguments
     * @throws Exception
     * @throws UnknownHostException
     * @throws IOException
     */
    public DriverOut(String address, String port)
throws Exception, UnknownHostException, IOException {
        out = new Prod(address, Integer.decode(port));
    }
    public void put(String s) {
        out.put(s);
    }
}

```

```

package util;

/**
 * TODO implement
 *
 * da raffinare, prima di realizzarla
 *
 * @author Carlo Montangero
 */
public class ParmSDB {
    /**

```

```

    * Builds a map from command line switches to the associated
    arguments (default value if switch missing)
    *
    * @param args command line arguments
    *
    */
    public ParamsDB(String [] args) {
        return;
    }

    /**
    * @param commandLineSwitch
    *
    * @return the command line argument associated with
    commandLineSwitch
    */
    public String getPar(String commandLineSwitch) {
        return null;
    }
}

```

```

package java.pipe;
import java.io.IOException;
public interface IPipeIn {
    public Object get() throws IOException;
}

```

```

package java.pipe;
import java.net.*;
import java.pipe.IPipeIn;
import java.io.*;

public class Cons implements IPipeIn {
    private BufferedReader in;
    public Cons(Integer serverPort) throws IOException {
        serverSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(serverPort);
        } catch (IOException e) {
            System.err.println("Could not listen on port: " +
serverPort);
            System.exit(1);
        }
        Socket clientSocket = null;
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException e) {
            System.err.println("Accept failed.");
            System.exit(1);
        }
        InputStreamReader isr = new
InputStreamReader(clientSocket.getInputStream());
        in = new BufferedReader(isr);
    }

    public String get() throws IOException {
        return in.readLine();
    }
}

```

```
}
```

```
package java.pipe;
```

```
/**  
 * Interfaccia lato prod della java.pipe.javaPipe.  
 *  
 * @author Carlo Montangero  
 *  
 */  
public interface IPipeOut {  
    /**  
     * @param element elemento immesso nella java.pipe.javaPipe.  
     *  
     * vincolo: il metodo toString deve essere opportunamente definito  
per permettere  
     * la ricostruzione dell'oggetto da parte del consumatore.  
     * Una soluzione più generale passa attraverso l'uso di  
Serializable.  
     *  
     */  
    public void put(Object element);  
}
```

```
package java.pipe;
```

```
import java.io.IOException;  
import java.io.PrintWriter;  
import java.net.Socket;  
import java.net.UnknownHostException;  
import java.pipe.IPipeOut;  
  
public class Prod implements IPipeOut {  
    private PrintWriter out;  
  
    public Prod(String consAddress, int consPort) throws  
UnknownHostException, IOException {  
        Socket socket = new Socket(consAddress, consPort);  
        out = new PrintWriter(socket.getOutputStream(), true);  
    }  
    public void put(Object a0) {  
        out.println(a0.toString());  
    }  
}
```