

Progettazione e realizzazione di un «alternatore»

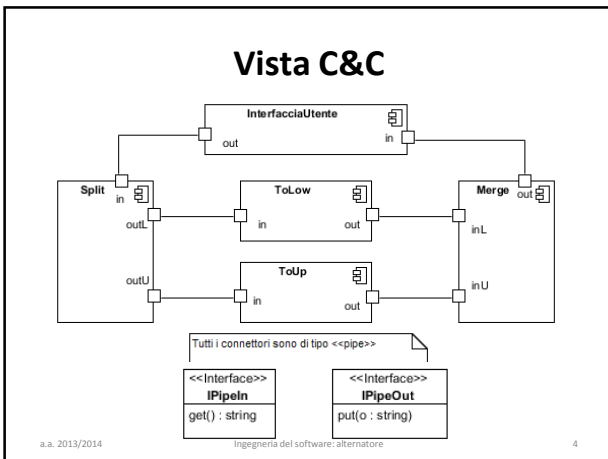
Carlo Montanero, Laura Semini
Ingegneria del Software
Dipartimento di Informatica
Università di Pisa

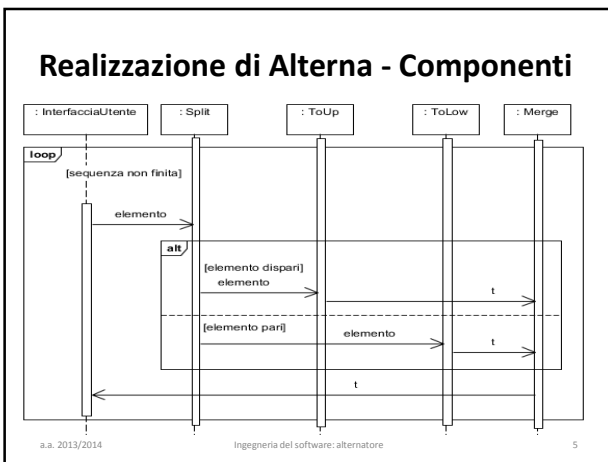
Caso d'uso: Alterna

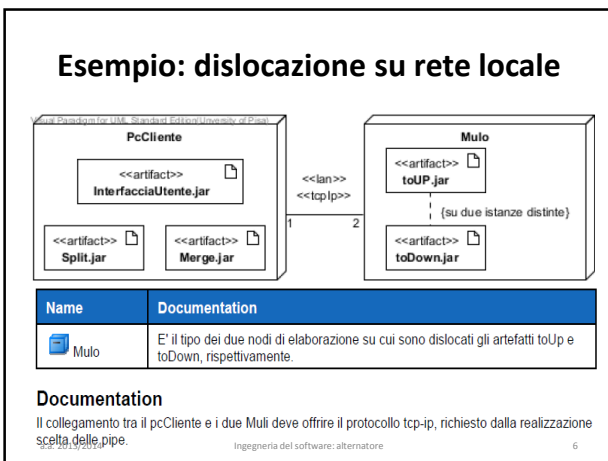
Name	Documentation
Alterna	Il sistema applica alternativamente una di due trasformazioni agli elementi di una sequenza. Per esempio, su una sequenza di caratteri, le trasformazioni potrebbero essere "a maiuscolo" e "a minuscolo". La sequenza "abCdEF" diventa "AbCdEf". Convenzionalmente, nel seguito chiameremo toUp e toDown 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.

Realizzazione di Alterna

a.a. 2013/2014 Ingegneria del software: alternatore 3







Struttura del documento di AS

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

a.a. 2013/2014 Ingegneria del software: alternatore 7

Pipe : contesto generico

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>

a.a. 2013/2014 Ingegneria del software: alternatore 8

Pipe: interfacce generiche

<<Interface>>
IPipeInJ
get(): Object

```
package javapipe;

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

<<Interface>>
IPipeOutJ
put(o : Object)

```
package javapipe;

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

Si chiamano ...J per un'idiosincrasia dell'ambiente usato ...

a.a. 2013/2014 Ingegneria del software: alternatore 9

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

a.a. 2013/2014

Ingegneria del software: alternatore

10

Realizzazione IPipeOutJ

```
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);
        }
    }
}
```

a.a. 2013/2014

Ingegneria del software: alternatore

11

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);
    }
}
```

a.a. 2013/2014

Ingegneria del software: alternatore

12

Realizzazione IPipeInJ

```
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);
        }
    }
}
```

a.a. 2013/2014

Ingegneria del software: alternatore

13

Realizzazione get

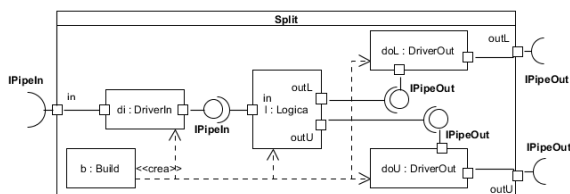
```
public Object get() {
    Object res = null;
    try {
        res = in.readObject();
    } catch (ClassNotFoundException e) {
        /** minima (pessima) gestione delle eccezioni */
        System.err.println("get: problemi di serializzazione");
        System.exit(1);
    } catch (IOException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

a.a. 2013/2014

Ingegneria del software: alternatore

14

Split in dettaglio



a.a. 2013/2014

Ingegneria del software: alternatore

15

DriverIn

```

/**import javapipe.Cons; */
public class DriverIn implements IPipeIn {
    javapipe.Cons in;

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

    public String get() {
        Object s = in.get();
        if (!(s instanceof String)) {
            System.err.println("get failed.");
            System.exit(1);
        }
        return (String) s ;
    }
}

```

a.a. 2013/2014

Ingegneria del software: alternatore

16

DriverOut

```

/** import javapipe.Prod; */
/**
 * Drives an output port of the component.
 */
public class DriverOut implements IPipeOut {
    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);
    }
}

```

a.a. 2013/2014

Ingegneria del software: alternatore

17

Logica: struttura

```

import alternatore.drivers.driverIn.IPipeIn;
import alternatore.drivers.driverOut.IPipeOut;

/** Realizza il comportamento di Split, usando tre driver dei porti.
 */
class Logica {
    private IPipeIn in;
    private IPipeOut outL;
    private IPipeOut outU;

    Logica(IPipeIn in, IPipeOut outL, IPipeOut outU) {
        this.in = in;
        this.outL = outL;
        this.outU = outU;
    }
}

```

a.a. 2013/2014

Ingegneria del software: alternatore

18

Logica: comportamento

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

a.a. 2013/2014

Ingegneria del software: alternatore

19

Costruzione di split: supporto

```
private static ParamsDB db;

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

- un oggetto ParamsDB 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

a.a. 2013/2014

Ingegneria del software: alternatore

20

Costruzione di split

```
import util.ParamsDB;

public class Build {

    private static Logica logica;
    private static ParamsDB db;

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

a.a. 2013/2014

Ingegneria del software: alternatore

21

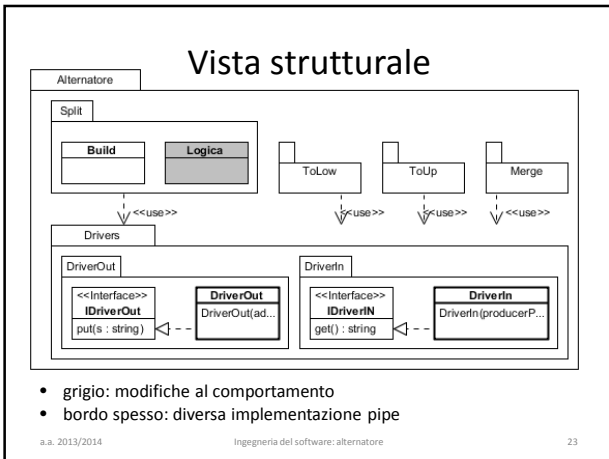
Build

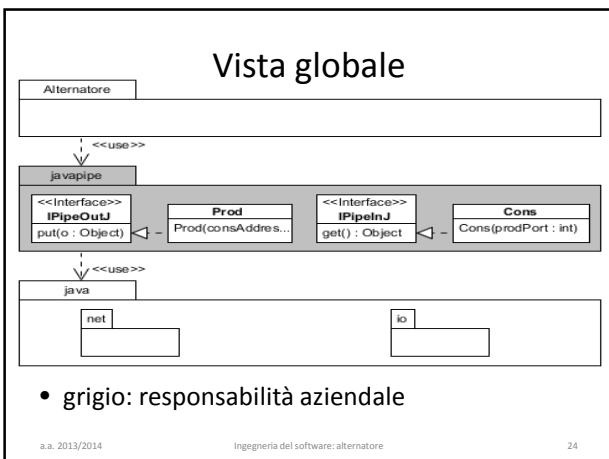
```
private static void build() {
    /** port to listen to producer */
    String ProducerPort = db.getPar("pp");
    IPipeIn in = new DriverIn(ProducerPort);
    /** port to lowConsumer */
    String LowConsumerAddress = db.getPar("lca");
    String LowConsumerPort = db.getPar("lcp");
    IPipeOut outL = new DriverOut(LowConsumerAddress, LowConsumerPort);
    /** up consumer address and port */
    String UpConsumerAddress = db.getPar("uca");
    String UpConsumerPort = db.getPar("ucp");
    IPipeOut outU = new DriverOut(UpConsumerAddress, UpConsumerPort);

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

```

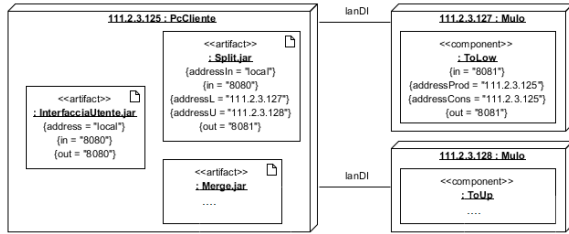
a.a. 2013/2014 Ingegneria del software: alternatore 22





Dislocazione

- Uno script di costruzione globale
– lancia le componenti sulle macchine scelte



a.a. 2013/2014

Ingegneria del software: alternatore

25
