GridBagLayout

`GridBagLayout` è un `LayoutManager` complesso ma meno complicato di quanto possa, a prima vista, sembrare.

Elementi.

Gli elementi necessari all'uso di `GridBagLayout` sono:

un `Container`
un `GridBagLayout`
un `GridBagConstraints`
un insieme di `Component`

Dato un insieme di `Component` ed un `Container`, un ordine d'uso potrebbe essere:

1. creare un `GridBagLayout`
2. creare un `GridBagConstraints`
3. impostare i campi del `GridBagConstraints` necessari alla distribuzione dello spazio disponibile ad un `Component`
4. registrare nel `GridBagLayout` l'associazione tra il `Component` e il `GridBagConstraints`
5. ripetere a partire dal punto 3 fino all'esaurimento dei `Component`
6. impostare il `GridBagLayout` per il `Container`
7. aggiungere tutti i `Component` al `Container` (ad esempio con `container.add(unComponent)`).

Quando il contenitore sarà proiettato sullo schermo, i diversi componenti in esso inseriti riceveranno una porzione dello spazio totale del contenitore determinata dai valori del `GridBagConstraints` associato e dalle tre dimensioni del componente (minima, massima e preferita).

La griglia ideale.

Il `LayoutManager` `GridBagLayout` definisce esattamente una griglia costituita di un certo numero di righe e un certo numero di colonne. Le righe possono avere altezze tra loro diverse. Le colonne possono avere larghezze tra loro diverse. L'altezza di una riga è determinata dall'altezza della più alta delle celle contenute in quella riga. La larghezza di una colonna è determinata dalla larghezza della più larga delle celle contenute in quella colonna. La complessità di `GridBagLayout` sta in ciò che la dimensione di una cella è stabilita in base ad un insieme di variabili tra cui alcuni dei valori di `GridBagConstraints`, alcuni dei valori del `Component` inserito in quella cella e, indirettamente, alcuni dei valori del `Container` gestito. La soluzione della complessità è nel metodo.

GridBagConstraints in pratica, prima parte.

La creazione di una qualsiasi interfaccia grafica utente passa inevitabilmente per una bozza, reale o ideale, di ciò che si vuole presentare sullo schermo. Supponiamo che l'immagine seguente rappresenti lo schema della costruenda interfaccia.

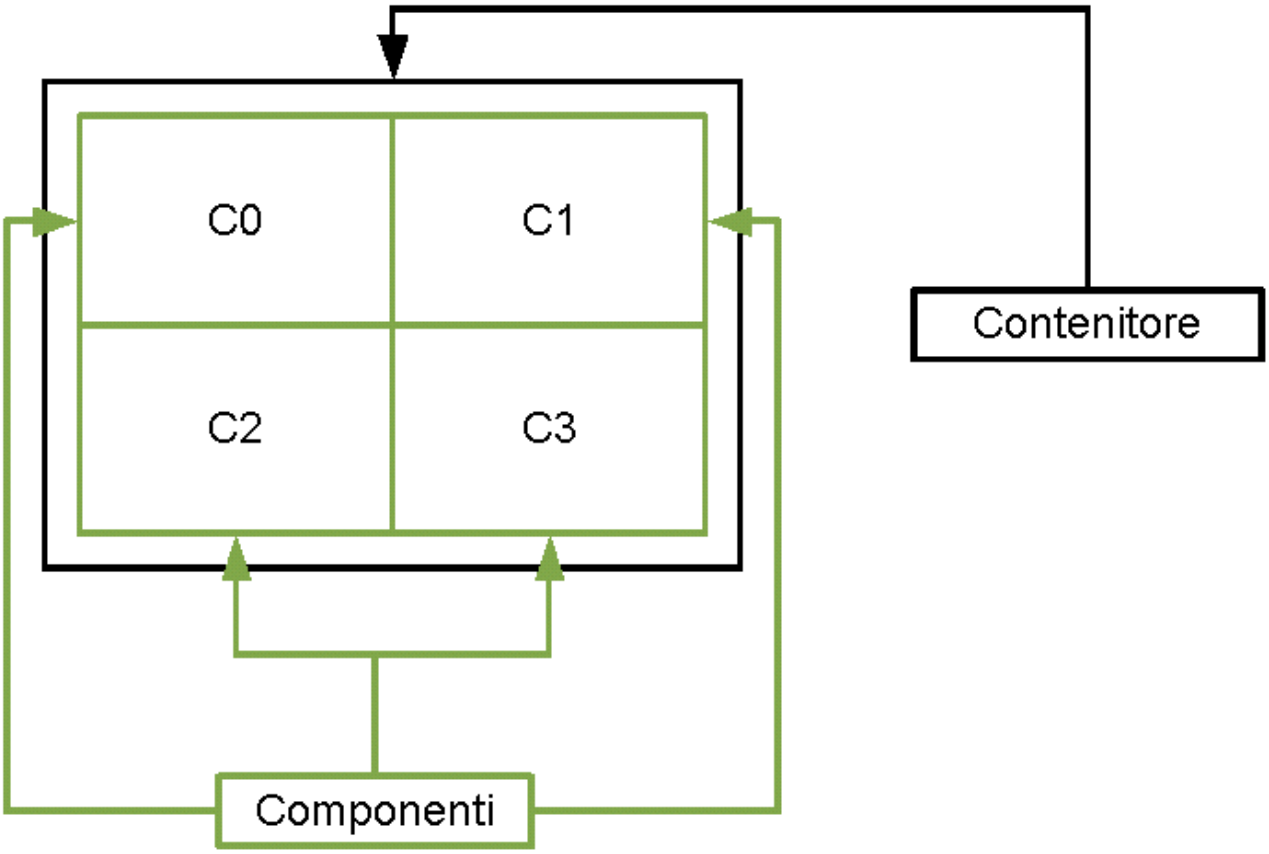


Figura 1 – Bozza di una GUI

Per tradurre questa bozza attraverso un `GridBagLayout` evidenziamo la struttura della griglia.

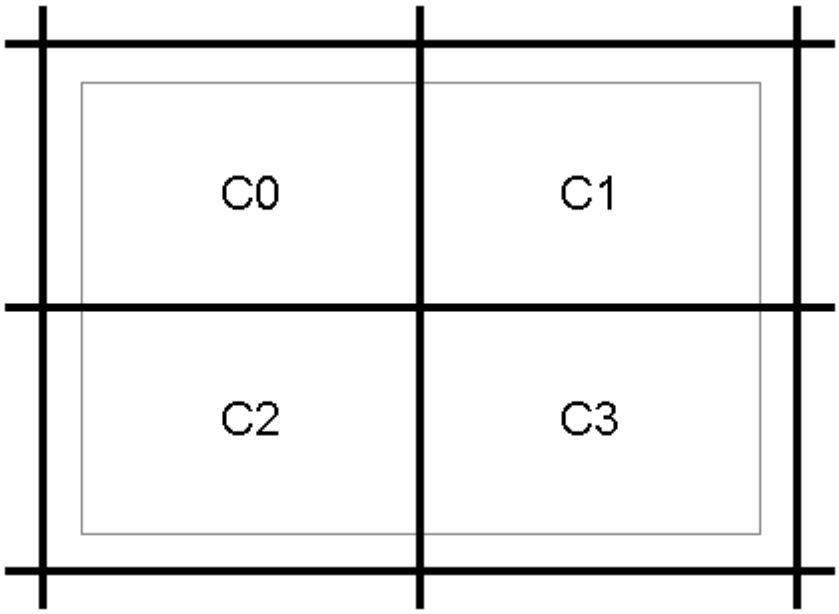


Figura 2 – La griglia evidenziata

Dopo aver messo in risalto la griglia, indicizziamo le righe e le colonne, partendo da zero, dall'alto verso il basso e da sinistra verso destra.

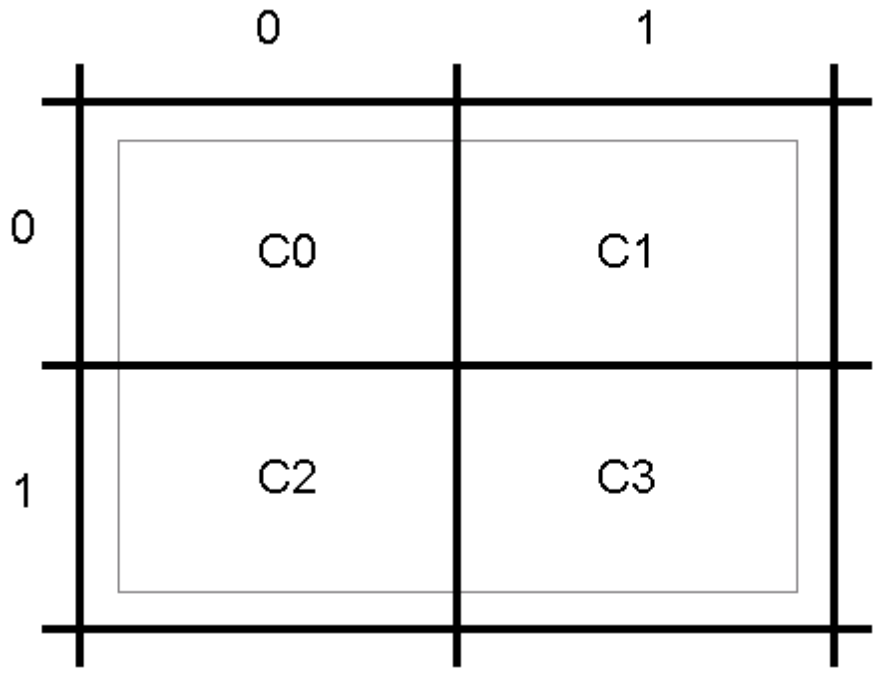


Figura 3 – Indicizzazione delle righe e delle colonne.

È facile constatare come ogni componente si trovi in una sola cella della griglia ed è altrettanto semplice determinare gli indici di riga e colonna di quella cella. Il componente con etichetta `C0` occupa la cella di riga zero e colonna zero. Il componente con etichetta `C1` occupa la cella di riga zero e colonna uno. Il componente con etichetta `C2` occupa la cella di riga uno e colonna zero. Il componente con etichetta `C3` occupa la cella di riga uno e colonna uno.
Dato un certo componente e determinata la cella in cui tale componente risiede, rispetto alla griglia ideale con cui è possibile rappresentare la distribuzione degli spazi in un contenitore, gli indici di riga e di colonna di quella cella sono direttamente rappresentati attraverso gli attributi `gridx` e `gridy` di un `GridBagConstraints`: `gridx` contiene l'indice della colonna, `gridy` contiene l'indice della riga.

Intermezzo: una base per le applicazioni d'esempio.

Definiamo in principio uno schema per semplificare la sperimentazione di diverse applicazioni di `GridBagLayout`. Lo schema si basa su un'interfaccia Java, `GeneratoreGUI` e un “motore” che usa quell'interfaccia.
L'interfaccia è presentata nel codice seguente.

```
package gbsample;

public interface GeneratoreGUI {

	/** Crea e aggiunge dei componenti a contenitore */
	void creaInterfaccia(java.awt.Container contenitore);
}
```

Il motore dell'applicazione, basato sull'interfaccia `GeneratoreGUI`, è il seguente.

```
package gbsample;

import java.awt.*;
import javax.swing.*;

public class Main implements Runnable {
	public static void main(String[] args) {
```

```
		GeneratoreGUI generatore = null;
		/* Crea un'istanza di generatore a partire da una
		classe il cui nome pienamente qualificato è contenuto
		nel primo paramentro di avvio del programma */
		try {
			Class<?> classeGeneratore = Class.forName(args[0]);
			generatore = (GeneratoreGUI)classeGeneratore.newInstance();
		} catch(ArrayIndexOutOfBoundsException ex) {
			System.err.println("Manca il nome della classe "+
				"che genera il layout");
			System.out.println("Uso: java Main nomeClasse");
			System.out.println("nomeClasse è il nome pienamente " +
				"qualificato di una classe che implementa " +
				"GeneratoreGUI e che possiede un costruttore "+
				"senza argomenti.");
		} catch(Exception ex) {
			System.err.println(
				"Errore durante la produzione"+
				" del generatore di layout");
			System.err.println(ex.getMessage());
		}
		/* Nel caso in cui non si siano verificate eccezioni
		generatore sarà diverso da null */
		if(generatore != null) {
			SwingUtilities.invokeLater(new Main(generatore));
		}
	}

	/* Campo terminale per ragioni di concorrenza */
	private final GeneratoreGUI GENERATORE_GUI;

	private Main(GeneratoreGUI g) {
		GENERATORE_GUI = g;
	}

	/** Crea una finestra, imposta layout e contenuto del
	suo pannello del contenuto usando il GeneratoreGUI
	determinato in costruzione, proietta la finestra sullo
	schermo */
	public void run() {
		Container contenitore = new JPanel();
		GENERATORE_GUI.creaInterfaccia(contenitore);
		JFrame finestra = new JFrame("GridBagLayout - Sample");
		finestra.setContentPane(contenitore);
		finestra.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
		finestra.pack();
		finestra.setVisible(true);
	}
}
```

Molto semplicemente, si tratta di poter lanciare un'applicazione Java determinando al momento dell'avvio quale oggetto sia responsabile della costruzione e distribuzione dei componenti nella finestra creata da `Main`. Avendo a disposizione una classe Java di nome `Prova` concretizzante l'interfaccia `GeneratoreGUI`, l'uso dell'applicazione sarebbe:

```
java gbsample.Main Prova
```

L'effetto risultate è che il pannello del contenuto della finestra proposta da `Main` contiene ciò che è stabilito nel metodo `creaInterfaccia` di `Prova` nel modo lì specificato.

GridBagConstraints in pratica, seconda parte.

Creiamo un file sorgente `Esempio001.java` in cui è definita una classe `Esempio001` che concretizza l'interfaccia `gbsample.GeneratoreGUI`.

```
import java.awt.*;
import javax.swing.*;

public class Esempio001 implements gbsample.GeneratoreGUI {
```

```
	public void creaInterfaccia(Container contenitore) {
		//da definire...
	}
}
```

Dato il `Container contenitore`, riproduciamo in esso la distribuzione dei componenti esemplificata nella figura 1, sulla base delle informazioni rappresentate in figura 3, riportata per comodità nella figura seguente.

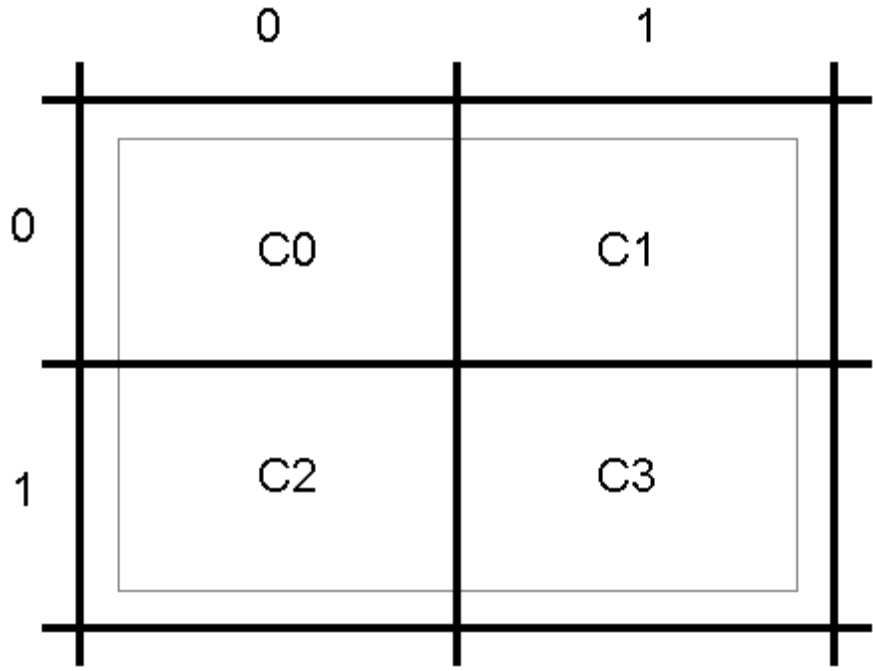


Figura 4 – Lo schema della griglia.

Il primo passo è creare un `GridBagLayout`.

```
GridBagLayout layout = new GridBagLayout();
```

Il secondo è creare un `GridBagConstraints`:

```
GridBagConstraints lim = new GridBagConstraints(); //lim sta per limits o limiti
```

Il terzo è impostare `layout` come `LayoutManager` di contenitore:

```
contenitore.setLayout(layout);
```

Usiamo dei pulsanti per componenti poiché i pulsanti hanno un bordo ben definito che aiuta ad evidenziare la corrispondenza tra quanto voluto e quanto ottenuto. Creiamo il primo componente.

```
Component c0 = new JButton("C0");
```

Basandoci sul nostro schema, sappiamo che il componente `c0` occupa la cella di riga zero e colonna zero. Nota la corrispondenza in `GridBagConstraints` tra il campo `gridx` e l'indice di colonna e tra il campo `gridy` e l'indice di riga, risulta:

```
lim.gridx = 0;
lim.gridy = 0;
```

Così facendo abbiamo semplicemente impostato il valore di due campi in `lim`. Affinchè questi campi producano l'effetto desiderato, cioè disporre il componente c0 nella cella individuata in un certo contenitore, servono altri due enunciati. Il primo crea nel `GridBagLayout layout` un'associazione tra un `GridBagConstraints` e un `Component`:

```
layout.setConstraints(c0, lim);
```

Il secondo aggiunge il componente al contenitore:

```
contenitore.ad(c0);
```

Da notare che l'associazione `GridBagConstraints` – `Component` del primo enunciato opera clonando i valori del `GridBagContraints` usato. Ciò significa che eventuali ulteriori modifiche dei campi di `lim` non avranno effetto sul componente associato.
La cella in cui si trova il secondo componente, `C1`, si trova nella riga di indice zero e nella colonna di indice uno. Creiamo il componente, impostiamo i valori `gridx` e `gridy` del `GridBagConstraints`, associamo il `GridBagConstraints` al componente nel `GridBagLayout` e aggiungiamo il componente al contenitore:

```
Component c1 = new JButton(“C1”);
lim.gridx = 1;
lim.gridy = 0;
layout.setConstraints(c1, lim);
contenitore.add(c1);
```

Le linee precedenti hanno lo stesso significato delle seguenti:

```
GridBagConstraints c1Limits = new GridBagConstraints();
c1Limits.gridx = 1;
c1Limits.gridy = 0;
layout.setConstraints(c1, c1Limits);
contenitore.add(c1);
```

Usare un unico `GridBagConstraints`, come faremo noi con l'unico `GridBagConstraints lim`, per tutti i componenti ha un vantaggio e uno svantaggio. Il vantaggio sta nella possibilità di riciclare uno stesso valore per ogni componente inserito senza dover ripetere la stessa impostazione per ogni componente. Lo svantaggio sta nella necessità di dover azzerare tale impostazione “predefinita” ogni volta che si stabiliscano i limiti di una cella per cui l'impostazione predefinita non valga. Vedremo più avanti le implicazioni pratiche di questo rilievo.
Impostiamo gli indici delle celle dei restanti componenti, ottenendo al termine una classe `Esempio001` così fatta:

```
import java.awt.*;
import javax.swing.*;

public class Esempio001 implements gbsample.GeneratoreGUI {

	public void creaInterfaccia(Container contenitore) {
		/* Impostazioni iniziali */
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		contenitore.setLayout(layout);

		/* Crea il componente che occupa la cella C0 */
		Component c0 = new JButton("C0");
		lim.gridx = 0; //Colonna 0
		lim.gridy = 0; //Riga 0
		layout.setConstraints(c0, lim); //Associazione
		contenitore.add(c0); //Inserimento

		/* Crea il componente che occupa la cella C1 */
```

```
                Component c1 = new JButton("C1");
                lim.gridx = 1; //Colonna 1
                lim.gridy = 0; //Riga 0
                layout.setConstraints(c1, lim); //Associazione
                contenitore.add(c1); //Inserimento

                /* Crea il componente che occupa la cella C2 */
                Component c2 = new JButton("C2");
                lim.gridx = 0; //Colonna 0
                lim.gridy = 1; //Riga 1
                layout.setConstraints(c2, lim); //Associazione
                contenitore.add(c2); //Inserimento

                /* Crea il componente che occupa la cella C3 */
                Component c3 = new JButton("C3");
                lim.gridx = 1; //Colonna 0
                lim.gridy = 1; //Riga 1
                layout.setConstraints(c3, lim); //Associazione
                contenitore.add(c3); //Inserimento
        }
}
```

Previa compilazione di `Esempio001.java`, l'esecuzione del motore per gli esercizi:

```
java gbsample.Main Esempio001
```

produce una finestra simile a quella riportata nella figura che segue.

Figura 5 – L'interfaccia prodotta da `Esempio001`

Ridimensionando la finestra dell'applicazione è facile notare una proprietà della distribuzione dei componenti da parte del `GridBagLayout` così come è stato da noi usato.

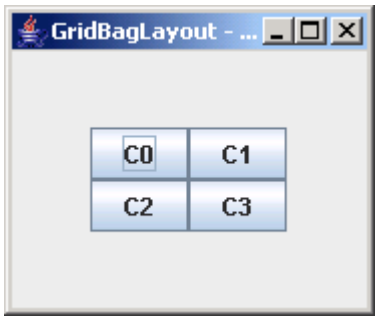


Figura 6 – Incremento delle dimensioni della finestra di Esempio001

Usando solo gli attributi `gridx` e `gridy` cioè determinando unicamente la "posizione" dei componenti nella griglia, i componenti collassano al centro del contenitore. L'effetto è prodotto dai valori predefiniti di ogni `GridBagConstraints` e, quindi, anche del nostro `lim`. Il primo partecipante a questo effetto è la ripartizione dello spazio in eccesso. Lo spazio in eccesso è quella porzione dello spazio disponibile nel contenitore che eccede la quantità di spazio necessaria e sufficiente a proiettare il contenuto di ogni cella secondo le sue dimensioni preferite.

Figura 7 – Spazio in eccesso nel pannello del contenuto della finestra di `Esempio001`

Per sfruttare lo spazio in eccesso la meccanica di un `GridBagLayout` si basa in prima battuta su due valori double rappresentati dai campi `weightx` e `weighty` di un `GridBagConstraints`. Meno vagamente, i pesi `weightx` e `weighty` contribuiscono a determinare la ripartizione dello spazio in eccesso tra le celle in cui si trovano i componenti.
Osserviamo il funzionamento dei pesi attribuendo ad ogni cella uno stesso peso orizzontale, `weigthx`, e verticale, `weighty`, in un nuovo file sorgente.

```
import java.awt.*;
import javax.swing.*;

/** Distribuzione dello spazio in eccesso alle righe e
alle colonne della griglia ideale */
public class Esempio002 implements gbsample.GeneratoreGUI {

	public void creaInterfaccia(Container contenitore) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		contenitore.setLayout(layout);

		Component c0 = new JButton("C0");
		lim.gridx = 0;
		lim.gridy = 0;
		lim.weightx = 1;
		lim.weighty = 1;
		layout.setConstraints(c0, lim);
		contenitore.add(c0);

		Component c1 = new JButton("C1");
		lim.gridx = 1;
		lim.gridy = 0;
		lim.weightx = 1;
		lim.weighty = 1;
		layout.setConstraints(c1, lim);
		contenitore.add(c1);

		Component c2 = new JButton("C2");
		lim.gridx = 0;
		lim.gridy = 1;
		lim.weightx = 1;
		lim.weighty = 1;
		layout.setConstraints(c2, lim);
		contenitore.add(c2);

		Component c3 = new JButton("C3");
		lim.gridx = 1;
		lim.gridy = 1;
		lim.weightx = 1;
		lim.weighty = 1;
		layout.setConstraints(c3, lim);
		contenitore.add(c3);
	}
}
```

L'effetto di questa attribuzione dei pesi è raffigurato nell'immagine che segue.

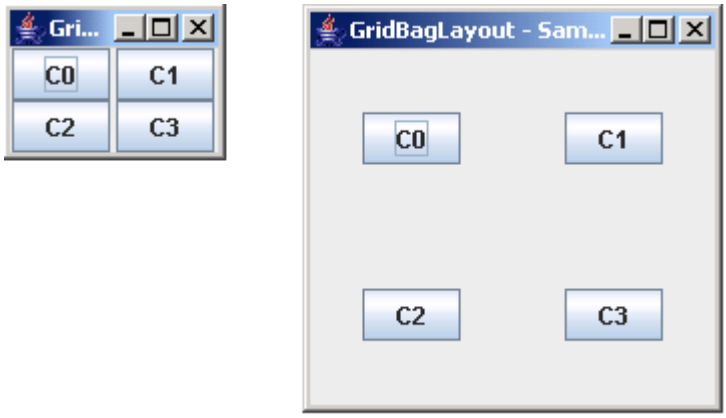


Figura 9 – Ripartizione dello spazio in eccesso tra le celle con `weightx` e `weighty`

Per comprendere ciò che è successo è sufficiente evidenziare con un bordo le quattro celle in cui si trovano i pulsanti di etichetta `C0`, `C1`, `C2`, `C3`, come è stato fatto nella figura che segue.

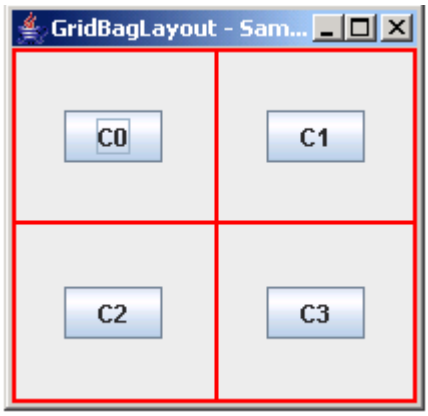


Figura 9 – Contorni delle quattro celle del `GridBagLayout` in `Esempio002`

La superficie del contenitore è maggiore della superficie necessaria e sufficiente a proiettare il contenuto delle celle secondo la sua dimensione preferita. Ogni cella dichiara un peso verticale e orizzontale identico. Ogni cella riceve un'identica porzione dello spazio in eccedenza[1]. Si nota chiaramente, nella figura 9, che la dimensione della cella e la dimensione del suo contenuto non siano due cose diverse. In particolare quando la cella abbia una dimensione maggiore del contenuto il contenuto ha sua disposizione una regione di spazio entro cui collocarsi. Ci sono due parametri che consentono di regolare lo sfruttamento dello spazio di una cella da parte del contenuto. Il primo è rappresentato dal campo `fill` di `GridBagConstraints`. Il campo fill può assumere uno dei valori terminali:

```
GridBagConstraints.NONE
GridBagConstraints.BOTH
GridBagConstraints.HORIZONTAL
GridBagConstraints.VERTICAL
```

Quando il campo `fill` di `GridBagConstraints` vale `GridBagConstraints.NONE`, il componente non sfrutta lo spazio in più offerto dalla sua cella. Quando il campo `fill` di `GridBagConstraints` vale `GridBagConstraints.BOTH`, il componente si estende fino a diventare grande quanto la sua cella, sia in verticale che in orizzontale. Quando il campo `fill` di `GridBagConstraints` vale

1 L'interpretazione è approssimativa. Più avanti vedremo in dettaglio cosa esattamente significhi ripartizione dei pesi.

GridBagConstraints.HORIZONTAL, il componente si estende orizzontalmente fino ad occupare tutto lo spazio orizzontale offerto dalla sua cella mentre assume un'estensione verticale pari alla sua altezza preferita. Quando il campo fill di GridBagConstraints vale GridBagConstraints.VERTICAL, il componente si estende verticalmente fino ad occupare tutto lo spazio verticale offerto dalla sua cella mentre assume un'estensione orizzontale pari alla sua larghezza preferita.
Possiamo vedere in azione questi quattro valori costanti in un nuovo file sorgente, Esempio003.java.

```
import java.awt.*;
import javax.swing.*;

/** Sfruttamento dello spazio delle celle da parte del loro
contenuto attraverso il valore del campo fill */
public class Esempio003 implements gbsample.GeneratoreGUI {

	public void creaInterfaccia(Container contenitore) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		contenitore.setLayout(layout);

		Component c0 = new JButton("C0");
		lim.gridx = 0;
		lim.gridy = 0;
		lim.weightx = 1;
		lim.weighty = 1;
		lim. fill = GridBagConstraints.NONE;
		layout.setConstraints(c0, lim);
		contenitore.add(c0);

		Component c1 = new JButton("C1");
		lim.gridx = 1;
		lim.gridy = 0;
		lim.weightx = 1;
		lim.weighty = 1;
		lim.fill = GridBagConstraints.BOTH;
		layout.setConstraints(c1, lim);
		contenitore.add(c1);

		Component c2 = new JButton("C2");
		lim.gridx = 0;
		lim.gridy = 1;
		lim.weightx = 1;
		lim.weighty = 1;
		lim.fill = GridBagConstraints.HORIZONTAL;
		layout.setConstraints(c2, lim);
		contenitore.add(c2);

		Component c3 = new JButton("C3");
		lim.gridx = 1;
		lim.gridy = 1;
		lim.weightx = 1;
		lim.weighty = 1;
		lim.fill = GridBagConstraints.VERTICAL;
		layout.setConstraints(c3, lim);
		contenitore.add(c3);
	}
}
```

La figura seguente mostra il risultato ottenuto dall'esecuzione di Esempio003 attraverso gbsample.Main.

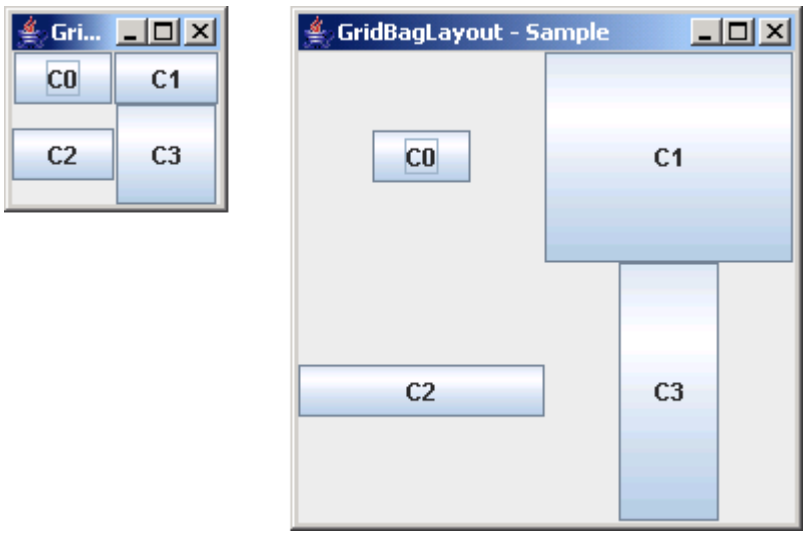


Figura 11 – Sfruttamento dello spazio nelle celle da parte del loro contenuto

Il pulsante di etichetta `C0` è registrato in associazione ad un `GridBagConstraints` il cui attributo `fill` ha il valore `GridBagConstraints.NONE`. Concordemente a quanto su detto, il pulsante di etichetta `C0` non sfrutta, né orizzontalmente né verticalmente, lo spazio della cella che ecceda le proprie dimensioni preferite. Il pulsante di etichetta `C1` è registrato in associazione ad un `GridBagConstraints` il cui attributo `fill` ha il valore `GridBagConstraints.BOTH` ciò che comporta lo sfruttamento da parte del componente dello spazio della propria cella sia in verticale che in orizzontale. Il pulsante di etichetta `C2` è registrato in associazione ad un `GridBagConstraints` il cui attributo `fill` ha il valore `GridBagConstraints.HORIZONTAL` dunque si estende ad occupare tutto lo spazio orizzontale di cui dispone la sua cella e si limita ad occupare una porzione di spazio verticale pari alla propria altezza preferita. Il pulsante di etichetta `C3` è registrato in associazione ad un `GridBagConstraints` il cui attributo `fill` ha il valore `GridBagConstraints.VERTICAL` dal che consegue un'estensione verticale del componente ad occupare tutto lo spazio verticale della cella ed un'estensione orizzontale pari alla larghezza preferita del componente.
Il secondo parametro che regola il posizionamento di un componente nel riquadro delineato da una cella è rappresentato dall'attributo `anchor` di `GridBagConstraints`. Un componente che non occupi tutta la superficie della cella a cui appartiene può assumere una posizione relativa ai lati della cella. Il valore dell'attributo `anchor` può essere uno tra:

```
GridBagConstraints.CENTER
GridBagConstraints.NORTH
GridBagConstraints.NORTHEAST
GridBagConstraints.EAST
GridBagConstraints.SOUTHEAST
GridBagConstraints.SOUTH
GridBagConstraints.SOUTHWEST
GridBagConstraints.WEST
GridBagConstraints.NORTHWEST
```

oppure uno tra:

```
GridBagConstraints.PAGE_START
GridBagConstraints.PAGE_END
GridBagConstraints.LINE_START
GridBagConstraints.LINE_END
GridBagConstraints.FIRST_LINE_START
GridBagConstraints.FIRST_LINE_END
GridBagConstraints.LAST_LINE_START
GridBagConstraints.LAST_LINE_END
```

Il primo gruppo di attributi è detto assoluto, il secondo è detto relativo. In verità tutti gli attributi

determinano un posizionamento relativo alla regione di spazio determinata dalla cella in cui si trova il componente. Il secondo gruppo è detto relativo in riferimento alla località di esecuzione.
Prendendo come riferimento i componenti di figura 11, supponiamo di voler spostare il pulsante di etichetta `C0` in aderenza al lato superiore della sua cella. Lo spostamento è realizzato nel codice `Esempio004.java`.

```
import java.awt.*;
import javax.swing.*;

/** Posizionamento di un componente all'interno della
sua cella con l'attributo anchor */
public class Esempio004 implements gbsample.GeneratoreGUI {

	public void creaInterfaccia(Container contenitore) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		contenitore.setLayout(layout);

		Component c0 = new JButton("C0");
		lim.gridx = 0;
		lim.gridy = 0;
		lim.weightx = 1;
		lim.weighty = 1;
		lim. fill = GridBagConstraints.NONE;
		lim.anchor = GridBagConstraints.NORTH;
		layout.setConstraints(c0, lim);
		contenitore.add(c0);

		Component c1 = new JButton("C1");
		lim.gridx = 1;
		lim.gridy = 0;
		lim.weightx = 1;
		lim.weighty = 1;
		lim.fill = GridBagConstraints.BOTH;
		lim.anchor = GridBagConstraints.CENTER;
		layout.setConstraints(c1, lim);
		contenitore.add(c1);

		Component c2 = new JButton("C2");
		lim.gridx = 0;
		lim.gridy = 1;
		lim.weightx = 1;
		lim.weighty = 1;
		lim.fill = GridBagConstraints.HORIZONTAL;
		lim.anchor = GridBagConstraints.CENTER;
		layout.setConstraints(c2, lim);
		contenitore.add(c2);

		Component c3 = new JButton("C3");
		lim.gridx = 1;
		lim.gridy = 1;
		lim.weightx = 1;
		lim.weighty = 1;
		lim.fill = GridBagConstraints.VERTICAL;
		lim.anchor = GridBagConstraints.CENTER;
		layout.setConstraints(c3, lim);
		contenitore.add(c3);
	}
}
```

Il risultato ottenuto è riprodotto nella figura che segue.

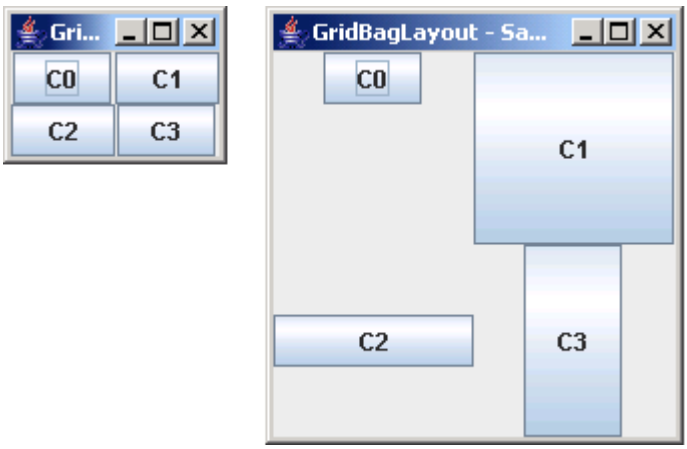


Figura 12 – Collocazione di un componente con l'attributo `anchor`.

La figura successiva mostra la relazione tra la posizione di un componente ed il valore dell'attributo anchor del GridBagConstraints a cui è associato.

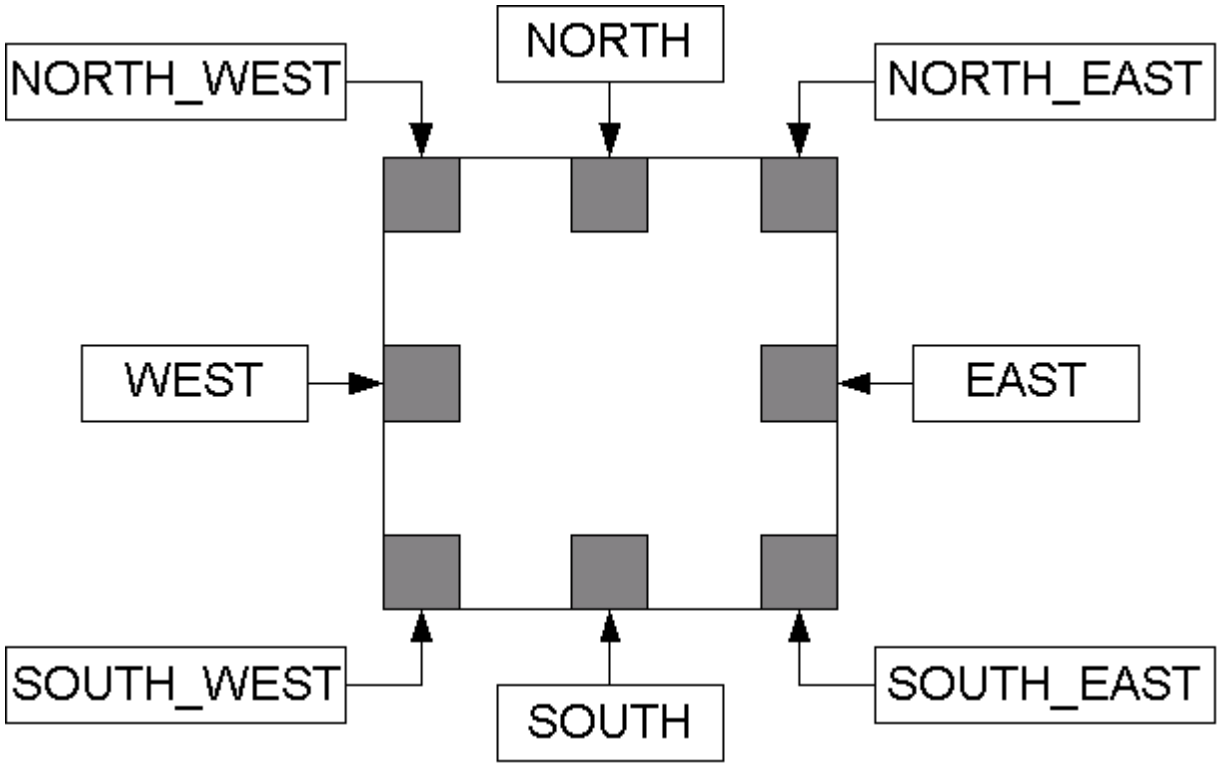


Figura 13 – Rapporto tra la posizione di un componente nella sua cella e l'attributo `anchor`

Oltre a posizionare e ridimensionare i contenuti, `GridBagLayout` usa, attraverso `GridBagConstraints`, tre variabili per l'impostazione di margini interni ed estensioni. I margini sono impostati usando il campo `insets` di `GridBagConstraints`. Il campo `insets` è di tipo `java.awt.Insets`. Un `java.awt.Insets` è una tupla di quattro valori interi, usati per rappresentare i valori di un margine superiore (`insets.top`), inferiore (`insets.bottom`), destro (`insets.right`) e sinistro (`insets.left`). Possiamo vedere i margini all'opera sul pulsante di etichetta "`C1`", precedentemente rappresentato in figura 11, nel codice che segue.

```
import java.awt.*;
import javax.swing.*;

/** Applicazione di un margine ad un componente */
public class Esempio005 implements gbsample.GeneratoreGUI {
```

```
        public void creaInterfaccia(Container contenitore) {
                GridBagLayout layout = new GridBagLayout();
                GridBagConstraints lim = new GridBagConstraints();
                contenitore.setLayout(layout);

                Component c0 = new JButton("C0");
                lim.gridx = 0;
                lim.gridy = 0;
                lim.weightx = 1;
                lim.weighty = 1;
                lim. fill = GridBagConstraints.NONE;
                lim.anchor = GridBagConstraints.NORTH;
                lim.insets.top = 0;
                lim.insets.bottom = 0;
                lim.insets.left = 0;
                lim.insets.right = 0;
                layout.setConstraints(c0, lim);
                contenitore.add(c0);

                Component c1 = new JButton("C1");
                lim.gridx = 1;
                lim.gridy = 0;
                lim.weightx = 1;
                lim.weighty = 1;
                lim.fill = GridBagConstraints.BOTH;
                lim.anchor = GridBagConstraints.CENTER;
                lim.insets.top = 16;
                lim.insets.bottom = 16;
                lim.insets.left = 16;
                lim.insets.right = 16;
                layout.setConstraints(c1, lim);
                contenitore.add(c1);

                Component c2 = new JButton("C2");
                lim.gridx = 0;
                lim.gridy = 1;
                lim.weightx = 1;
                lim.weighty = 1;
                lim.fill = GridBagConstraints.HORIZONTAL;
                lim.anchor = GridBagConstraints.CENTER;
                lim.insets.top = 0;
                lim.insets.bottom = 0;
                lim.insets.left = 0;
                lim.insets.right = 0;
                layout.setConstraints(c2, lim);
                contenitore.add(c2);

                Component c3 = new JButton("C3");
                lim.gridx = 1;
                lim.gridy = 1;
                lim.weightx = 1;
                lim.weighty = 1;
                lim.fill = GridBagConstraints.VERTICAL;
                lim.anchor = GridBagConstraints.CENTER;
                lim.insets.top = 0;
                lim.insets.bottom = 0;
                lim.insets.left = 0;
                lim.insets.right = 0;
                layout.setConstraints(c3, lim);
                contenitore.add(c3);
        }
}
```

L'effetto risultante è riprodotto nella figura che segue.

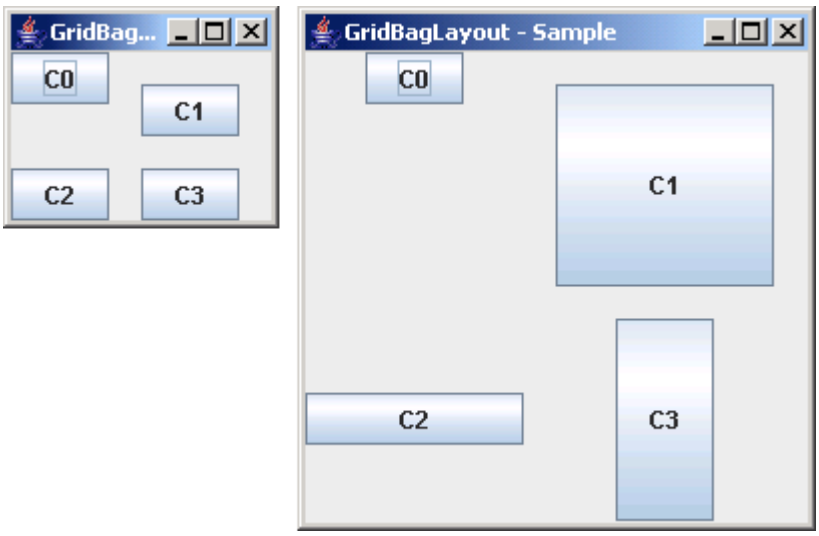


Figura 14 – Un componente con un margine.

Il margine impostato con il valore `insets` di un `GridBagLayout` si traduce in una distanza applicata tra il bordo del contenuto ed il lato della cella che lo contiene. Il valore `insets.top` contiene la distanza tra il lato superiore del componente ed il lato superiore della cella, il valore `insets.bottom` contiene la distanza tra il lato inferiore del componente ed il lato inferiore della cella, il valore `insets.right` contiene la distanza tra il lato destro del componente ed il lato destro della cella, il valore `insets.left` contiene la distanza tra il lato sinistro del componente ed il lato sinistro della cella.
Le due variabile per impostare l'estensione della dimensione di un componente sono `ipadx` e `ipady`. L'attributo `ipadx` di `GridBagConstraints` determina un allungamento del componente sull'asse orizzontale, l'attributo `ipady` di `GridBagConstraints` determina un allungamento del componente sull'asse verticale. La quantità di queste estensioni è rappresentata dal valore assoluto degli attributi `ipadx` e `ipady`, in pixel. Il valore di `ipadx` e `ipady` può essere un intero negativo ma l'effetto risultate è l'attribuzione ad un componente di una dimensione inferiore a quella minima necessaria alla sua corretta proiezione ciò che potrebbe generare dei problemi di visibilità del contenuto.
Il codice seguente applica un'estensione di 50 pixel lungo entrambi gli assi al pulsante di etichetta "`C0`".

```
import java.awt.*;
import javax.swing.*;

/** Estensione alle dimensioni di un componente */
public class Esempio006 implements gbsample.GeneratoreGUI {

	public void creaInterfaccia(Container contenitore) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		contenitore.setLayout(layout);

		Component c0 = new JButton("C0");
		lim.gridx = 0;
		lim.gridy = 0;
		lim.weightx = 1;
		lim.weighty = 1;
		lim.ipadx = 50;
		lim.ipady = 50;
		lim. fill = GridBagConstraints.NONE;
		lim.anchor = GridBagConstraints.NORTH;
		layout.setConstraints(c0, lim);
		contenitore.add(c0);

		Component c1 = new JButton("C1");
		lim.gridx = 1;
		lim.gridy = 0;
```

```
                lim.weightx = 1;
                lim.weighty = 1;
                lim.ipadx = 0;
                lim.ipady = 0;
                lim.fill = GridBagConstraints.BOTH;
                lim.anchor = GridBagConstraints.CENTER;
                layout.setConstraints(c1, lim);
                contenitore.add(c1);

                Component c2 = new JButton("C2");
                lim.gridx = 0;
                lim.gridy = 1;
                lim.weightx = 1;
                lim.weighty = 1;
                lim.ipadx = 0;
                lim.ipady = 0;
                lim.fill = GridBagConstraints.HORIZONTAL;
                lim.anchor = GridBagConstraints.CENTER;
                layout.setConstraints(c2, lim);
                contenitore.add(c2);

                Component c3 = new JButton("C3");
                lim.gridx = 1;
                lim.gridy = 1;
                lim.weightx = 1;
                lim.weighty = 1;
                lim.ipadx = 0;
                lim.ipady = 0;
                lim.fill = GridBagConstraints.VERTICAL;
                lim.anchor = GridBagConstraints.CENTER;
                layout.setConstraints(c3, lim);
                contenitore.add(c3);
        }
}
```

L'effetto prodotto è osservabile nella figura seguente.

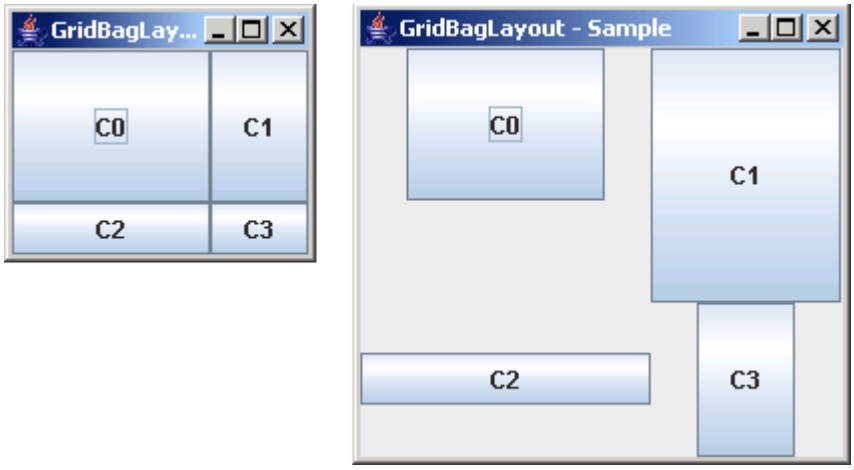


Figura 15 – Applicazione di un'estensione ad un componente.

Gli ultimi due attributi di `GridBagConstraints` che mancano all'appello sono `gridwidth` e `gridheight`. La cella della griglia ideale in cui si trova un componente può occupare più di una riga e più di una colonna. Righe e colonne occupate da un'unica cella devono essere consecutive. Il valore di `gridwidth` indica il numero di colonne occupate dalla cella del componente. Il valore di `gridheight` indica il numero di righe occupate dalla cella del componente.
Allo scopo di esemplificare l'uso dei valori `gridwidth` e `gridheight` definiamo una nuova bozza di interfaccia grafica, riprodotta nell'immagine che segue.

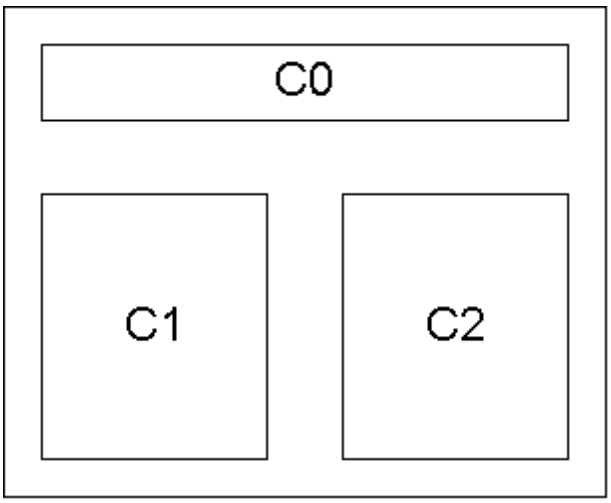


Figura 16 – Lo schema di un'interfaccia grafica utente.

A differenza dello schema precedente, questo non evidenzia direttamente una divisione degli spazi in righe e colonne ma se consideriamo la possibilità di far occupare ad un componente più di una riga o più di una colonna una rappresentazione dello schema di figura 16 attraverso un `GridBagLayout` potrebbe basarsi sulla struttura della figura che segue.

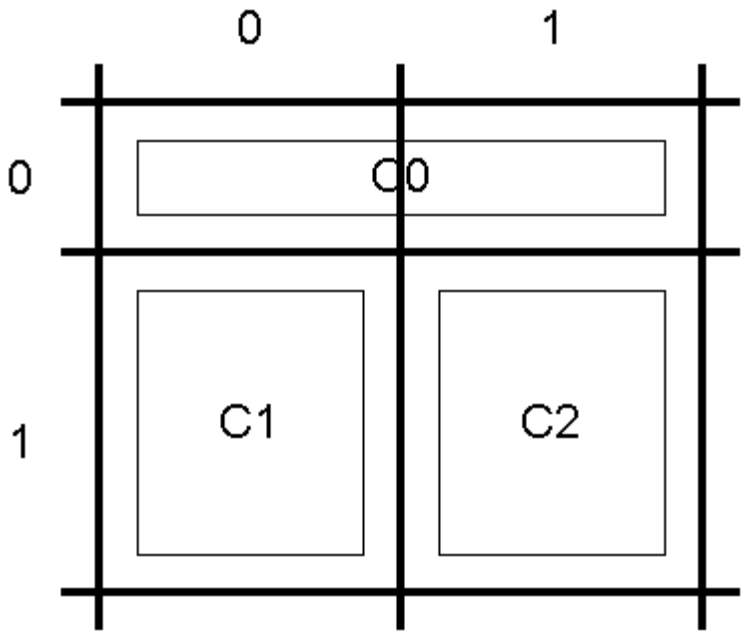


Figura 16 – Una griglia per l'interfaccia.

Basandoci sullo schema di figura 16 possiamo dire che il componente C1 si trova nella cella di riga uno e colonna zero e che il componente C2 si trova nella cella di riga uno e colonna uno. Il componente C0 occupa due celle il che ci porta all'esatta definizione del significato degli attributi `gridx` e `gridy` di `GridBagConstraints`: `gridx` contiene l'indice della prima colonna occupata da un componente nella griglia ideale e `gridy` contiene l'indice della prima riga occupata da un componente nella griglia ideale. Possiamo esprimere il concetto di prima riga e prima colonna in senso assoluto o in senso relativo alla località di esecuzione. In senso assoluto per prima riga e prima colonna si intendono quella riga e quella colonna che determinano la cella in cui è contenuto l'angolo superiore sinistro del componente inserito. In senso relativo, la prima riga e la prima colonna dipendono dalla direzionalità della scrittura nella località di esecuzione. Per costruire

un'interfaccia con `GridBagLayout` tenendo conto della direzionalità della scrittura nella località di esecuzione si possono usare i valori `GridBagConstraints.REMAINDER` e `GridBagConstraints.RELATIVE`. La figura che segue evidenzia il significato assoluto del concetto di prima riga e prima colonna identificando l'angolo di ogni componente che determina il valore degli attributi `gridx` e `gridy`.

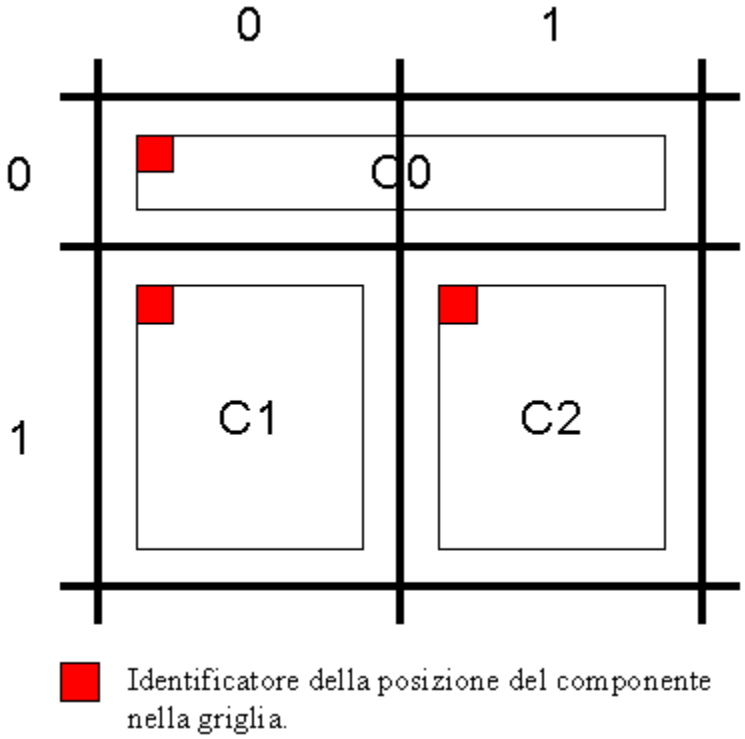


Figura 18 – Identificazione dei valori `gridx` e `gridy` per un componente.

Nella griglia ideale, il componente C0 è collocato all'interno di una regione di spazio che inizia nella cella di riga zero e colonna zero (`gridx = 0`, `gridy = 0`) e si espande per due colonne e una riga (`gridwidth = 2`, `gridheight = 1`). Il componente C1 è collocato all'interno di una regione di spazio che inizia nella cella di riga uno e colonna zero (`gridx = 0`, `gridy = 1`) e si espande per una colonna e una riga (`gridwidth = 1`, `gridheight = 1`). Il componente C2 è collocato all'interno di una regione di spazio che inizia nella cella di riga uno e colonna uno (`gridx = 1`, `gridy = 1`) e si espande per una colonna e una riga (`gridwidth = 1`, `gridheight = 1`).
Per quanto detto appare evidente, quindi, che il significato di `gridwidth` e `gridheight` è riconducibile a quello di "numero di colonne e numero di righe occupate dal componente nella griglia ideale". Il codice che segue è un'applicazione dei rilievi precedenti.

```
import java.awt.*;
import javax.swing.*;

/** Determinazione completa della regione di spazio
occupata da ciascuno componente */
public class Esempio007 implements gbsample.GeneratoreGUI {

	public void creaInterfaccia(Container contenitore) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		contenitore.setLayout(layout);

		Component c0 = new JButton("C0");
		lim.gridx = 0;
		lim.gridy = 0;
		lim.gridwidth = 2;
```

```
                lim.gridheight = 1;
                layout.setConstraints(c0, lim);
                contenitore.add(c0);

                Component c1 = new JButton("C1");
                lim.gridx = 0;
                lim.gridy = 1;
                lim.gridwidth = 1;
                lim.gridheight = 1;
                layout.setConstraints(c1, lim);
                contenitore.add(c1);

                Component c2 = new JButton("C2");
                lim.gridx = 1;
                lim.gridy = 1;
                lim.gridwidth = 1;
                lim.gridheight = 1;
                layout.setConstraints(c2, lim);
                contenitore.add(c2);

        }
}
```

La figura successiva propone il risultato dell'esecuzione di `Esempio007`.

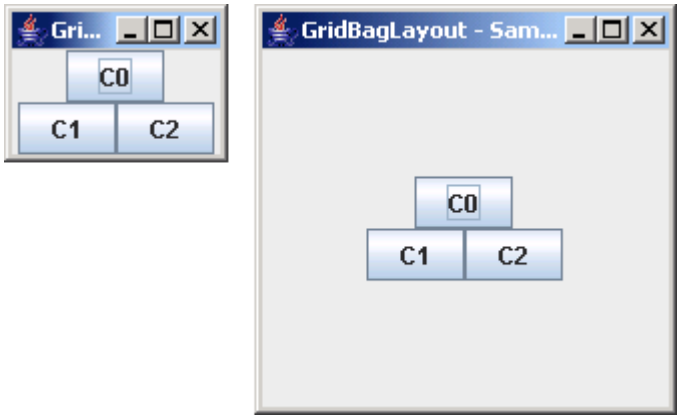


Figura 19 – Applicazione di `gridwidth`.

I pesi e la distribuzione proporzionale dello spazio in eccesso.

Sappiamo che la non corrispondenza tra il prodotto mostrato nella figura 19 e lo schema della figura 16 è dovuto alla mancata attribuzione alle celle dello spazio eccedente le necessità di proiezione dei componenti. Per distribuire lo spazio in eccesso occorre attribuire alle celle un opportuno "peso" verticale e orizzontale. La cella del componente c0 deve ricevere lo spazio orizzontale eccedente le sue necessità mentre sull'asse verticale deve limitarsi all'altezza del suo contenuto. Imponendo per il `GridBagConstraints` associato al componente `c0` i pesi `weightx = 1` e `weighty = 0` avremmo la trasformazione raffigurata nell'immagine che segue.

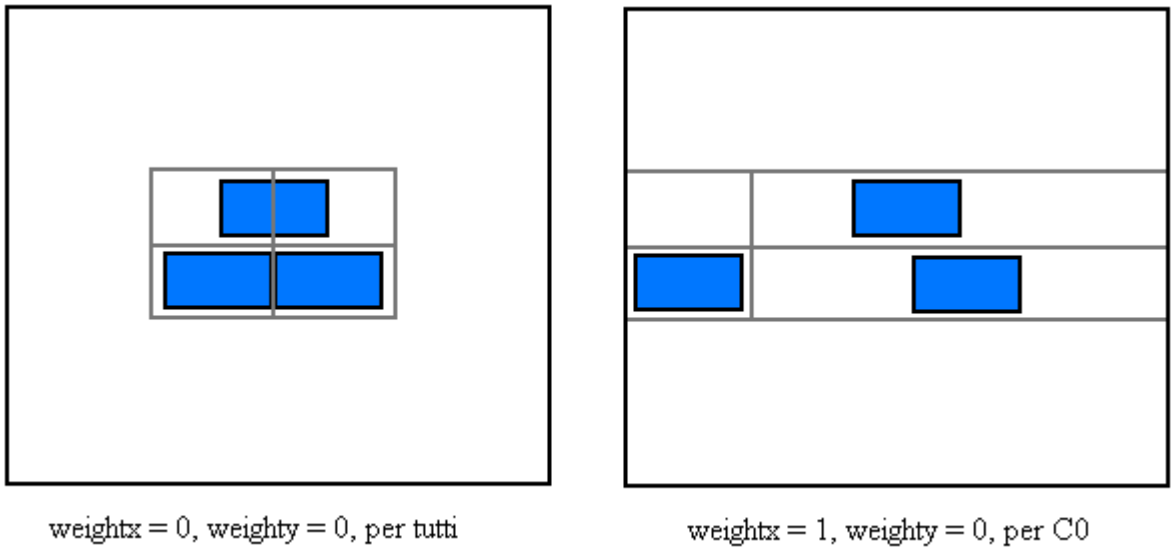


Figura 20 – Effetto dell'applicazione di un peso `weightx = 1`, `weighty = 0` alla cella di C0

Una sola delle celle occupate da un singolo componente subisce l'effetto dell'applicazione di un peso. Tale cella è sempre l'ultima del gruppo in cui è collocato il componente, in basso a destra nel caso di posizionamento assoluto. Si nota anche come l'attribuzione dello spazio alle celle sia un effetto risultate dall'attribuzione di spazio alla colonna ed alla riga a cui la cella appartiene. In altri termini, la cella del componente c2 si allarga per effetto dell'attribuzione dello spazio in eccesso alla seconda cella del componente c0: la cella più larga di una colonna determina anche la larghezza di tutta la colonna e, dunque, la larghezza di tutte le celle appartenenti a quella colonna.
Al fine di ottenere una ripartizione dello spazio uniforme tra le due colonne della nostra griglia ideale tutto quello che dobbiamo fare è attribuire alle singole celle che, insieme, determinano la larghezza di quelle colonne uno stesso peso orizzontale. Applicando un peso identico alle celle dei componenti c1 e c2 e lasciando a zero il peso orizzontale della cella del componente c0 otteniamo il risultato della figura che segue.

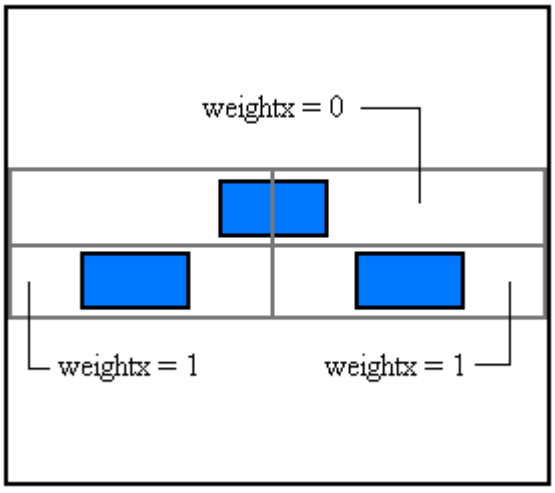


Figura 21 – Effetto dei pesi orizzontali sulle colonne della griglia ideale.

Il contributo delle dimensioni dei componenti.

Vale la pena spiegare cosa si celi dietro i pesi, cioè quale sia il contributo reale dei valori `weightx` e `weighty` alla "forma" della griglia. I pesi realmente considerati sono solo i pesi maggiori tra quelli

associati alle celle di una stessa riga e alle celle di una stessa colonna. Ogni riga ha un peso verticale il cui valore è il maggiore tra i valori `weighty` di ogni cella appartenente a quella riga. Ogni colonna ha un peso orizzontale il cui valore è il maggiore tra i valori `weightx` di ogni cella appartenente a quella colonna. Ogni colonna ha una larghezza “naturale”. La larghezza naturale di una colonna è la larghezza del più largo dei componenti contenuti in una delle sue celle. Allo stesso modo, ogni riga ha un'altezza “naturale”. L'altezza naturale di una riga è l'altezza del più alto dei componenti contenuti in una delle sue celle. L'altezza e la larghezza di un componente che partecipa alla determinazione dell'altezza e larghezza naturale di una riga o di una colonna è la sua altezza preferita o minima e la sua larghezza preferita o minima. Se una colonna contiene due componenti, il primo ha una dimensione preferita (100, 100), il secondo ha una dimensione preferita (75, 75), la colonna sarà larga 100. Ma se lo spazio disponibile non sia sufficiente a proiettare il primo componente secondo la sua dimensione preferita allora la dimensione considerata per esso sarà la sua minima. Supponendo che la dimensione minima del primo componente sia (0, 0), la larghezza naturale della colonna sarà di 75 pixel. Occorre tenere ben presente che il passaggio alla dimensione minima qui esemplificato avviene su entrambi gli assi a prescindere dall'asse in cui lo spazio disponibile sia insufficiente. Sempre nel nostro esempio, se lo spazio verticale disponibile nel contenitore fosse insufficiente a proiettare il primo componente secondo la sua altezza preferita ma più che sufficiente a proiettare il primo componente secondo la sua larghezza preferita comunque ai fini del computo della larghezza della colonna varrebbe la dimensione minima del primo componente: il componente assume per intero la sua dimensione minima a prescindere dal fatto che lo spazio manchi solo in larghezza o solo in altezza.

Ogni riga ha un sua altezza e ogni colonna ha una sua larghezza. Ogni riga ha un proprio peso verticale ed ogni colonna ha un proprio peso orizzontale. Il peso verticale di una riga ha un valore pari al maggiore tra i pesi verticali delle celle appartenenti a quella riga. Il peso orizzontale di una colonna ha un valore pari al maggiore tra i pesi orizzontali delle celle appartenenti a quella colonna. Per quanto riguarda la meccanica dell'attribuzione dello spazio in eccesso, si prende la somma delle altezze dei componenti la si divide per la somma dei pesi verticali di tutte le righe e si moltiplica per il peso della singola riga. Risulta un certo numero che, aggiunto all'altezza del più alto dei componenti su quella riga, determinerà l'altezza totale della riga. Lo stesso vale per la larghezza delle colonne.

Riprendiamo in esame lo schema di figura 17. Quali devono essere i pesi delle celle affinché un `GridLayout` possa mimare quella distribuzione di spazi? Se intendiamo attribuire alla riga zero un'altezza pari all'altezza preferita o minima del componente C0 e distribuire il restante spazio verticale ai componenti C1 e C2, sapendo che il peso determina la distribuzione dello spazio in eccesso e che il peso verticale rilevante è il maggiore tra i pesi verticali attribuiti alle celle C1 e C2 possiamo concludere che:

il `GridBagConstraints` associato al componente C0 avrà un peso verticale `weighty = 0`;

il `GridBagConstraints` associato al componente C1 avrà un qualsiasi peso verticale maggiore di zero;

il `GridBagConstraints` associato al componente C2 avrà un qualsiasi peso verticale maggiore o uguale a zero.

Se intendiamo attribuire alla colonna zero metà dello spazio in eccesso disponibile ed alla colonna uno l'altra metà:

il `GridBagConstraints` associato al componente C1 avrà un qualsiasi peso orizzontale maggiore di zero;

il `GridBagConstraints` associato al componente C2 avrà un peso orizzontale uguale al peso orizzontale del componente C1

il `GridBagConstraints` associato al componente C0 avrà un qualsiasi peso orizzontale maggiore o uguale a zero e minore o uguale al peso orizzontale di C2;

Se desideriamo che le due colonne mantengano la stessa proporzione, cioè che lo spazio disponibile – e non solo quello eccedente – sia equamente ripartito tra le colonne zero e uno:

le dimensioni preferite dei componenti C1 e C2 devono coincidere;

Se vogliamo che la proporzione su citata permanga a prescindere da una riduzione delle dimensioni del contenitore:

le dimensioni minime dei componenti C1 e C2 devono coincidere.

Dopo aver distribuito lo spazio tra le celle faremo espandere il loro contenuto ad occupare tutta la loro superficie impostando il valore `GridBagConstraints.BOTH` per l'attributo `fill`. Il codice che segue è l'applicazione di quanto esposto.

```
import java.awt.*;
import javax.swing.*;

/** Determinazione completa della regione di spazio
occupata da ciascuno componente */
public class Esempio008 implements gbsample.GeneratoreGUI {
	public void creaInterfaccia(Container container) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		container.setLayout(layout);

		lim.insets.top = 10;
		lim.insets.bottom = 10;
		lim.insets.left = 10;
		lim.insets.right = 10;
		lim.fill = GridBagConstraints.BOTH;

		Component c0 = new JButton("C0");
		lim.gridx = 0;
		lim.gridy = 0;
		lim.gridwidth = 2;
		lim.gridheight = 1;
```

```
            lim.weightx = 0;
            lim.weighty = 0;
            container.add(c0, lim);

            Component c1 = new JButton("C1");
            c1.setPreferredSize(new Dimension(150, 200));
            c1.setMinimumSize(new Dimension(150, 200));
            lim.gridx = 0;
            lim.gridy = 1;
            lim.gridwidth = 1;
            lim.gridheight = 1;
            lim.weightx = 1;
            lim.weighty = 1;
            container.add(c1, lim);

            Component c2 = new JButton("C2");
            c2.setPreferredSize(new Dimension(150, 200));
            c2.setMinimumSize(new Dimension(150, 200));
            lim.gridx = 1;
            lim.gridy = 1;
            lim.gridwidth = 1;
            lim.gridheight = 1;
            lim.weightx = 1;
            lim.weighty = 0;
            container.add(c2, lim);
        }
}
```

La disposizione dei componenti prodotta dal codice di Esempio008.java è la seguente.

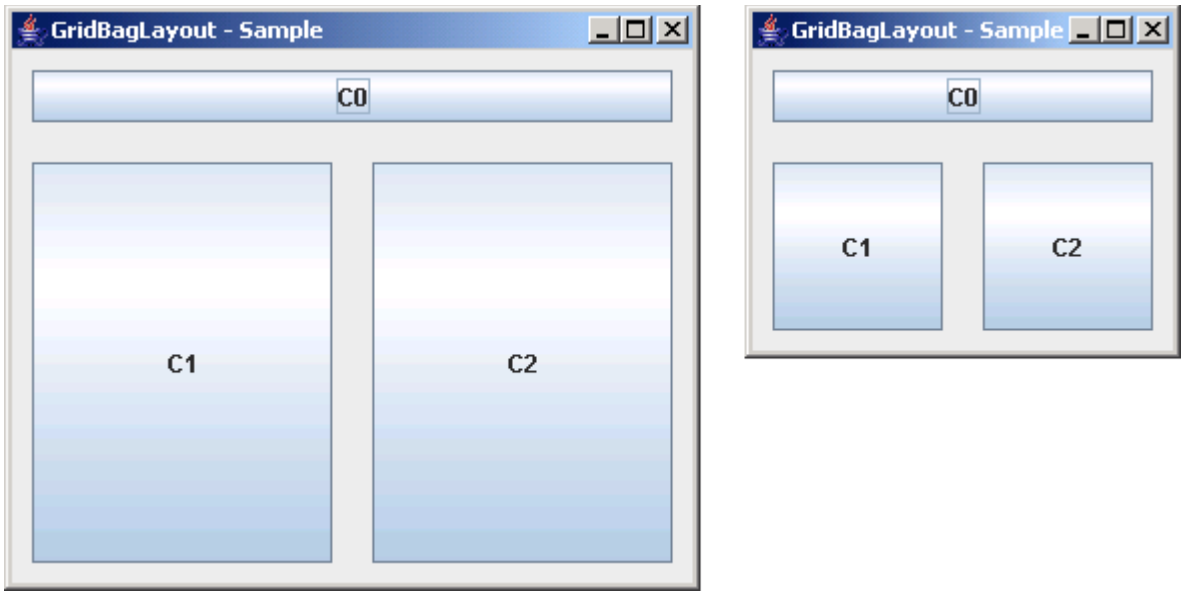


Figura 22 – Prodotto di `Esempio008.java`

Volendo è possibile sperimentare gli sbilanciamenti indotti da variazioni nelle dimensioni minime e preferite con il seguente codice.

```
import java.awt.*;
import javax.swing.*;

/** Determinazione completa della regione di spazio
occupata da ciascuno componente */
public class Esempio009 implements gbsample.GeneratoreGUI {
	private static final class Token extends JPanel {
		private Dimension minSize, prefSize;

		/* Crea un componente opaco impostando direttamente le sue
		dimensioni minime e preferite
		@param minW larghezza minima
		@param minH altezza minima
		@param prefW larghezza preferita
		@param prefH altezza preferita */
		private Token(int minW, int minH, int prefW, int prefH) {
			minSize = new Dimension(minW, minH);
			prefSize = new Dimension(prefW, prefH);
			setBorder(BorderFactory.createLineBorder(Color.BLACK, 1));
			setOpaque(true);
			setBackground(Color.BLUE.darker().darker());
		}

		public Dimension getPreferredSize() {
			return prefSize;
		}

		public Dimension getMinimumSize() {
			return minSize;
		}
	};

	public void creaInterfaccia(Container container) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		container.setLayout(layout);

		lim.insets.top = 10;
		lim.insets.bottom = 10;
		lim.insets.left = 10;
		lim.insets.right = 10;
		lim.fill = GridBagConstraints.BOTH;
```

```
            Component c0 = new Token(150, 25, 150, 25);
            lim.gridx = 0;
            lim.gridy = 0;
            lim.gridwidth = 2;
            lim.gridheight = 1;
            lim.weightx = 0;
            lim.weighty = 0;
            container.add(c0, lim);

            Component c1 = new Token(75, 100, 150, 200);
            lim.gridx = 0;
            lim.gridy = 1;
            lim.gridwidth = 1;
            lim.gridheight = 1;
            lim.weightx = 1;
            lim.weighty = 1;
            container.add(c1, lim);

            Component c2 = new Token(75, 100, 150, 200);
            lim.gridx = 1;
            lim.gridy = 1;
            lim.gridwidth = 1;
            lim.gridheight = 1;
            lim.weightx = 1;
            lim.weighty = 0;
            container.add(c2, lim);
        }
}
```

Un ampio form.

La figura seguente mostra lo schema del contenuto di una finestra di inserimento dati.

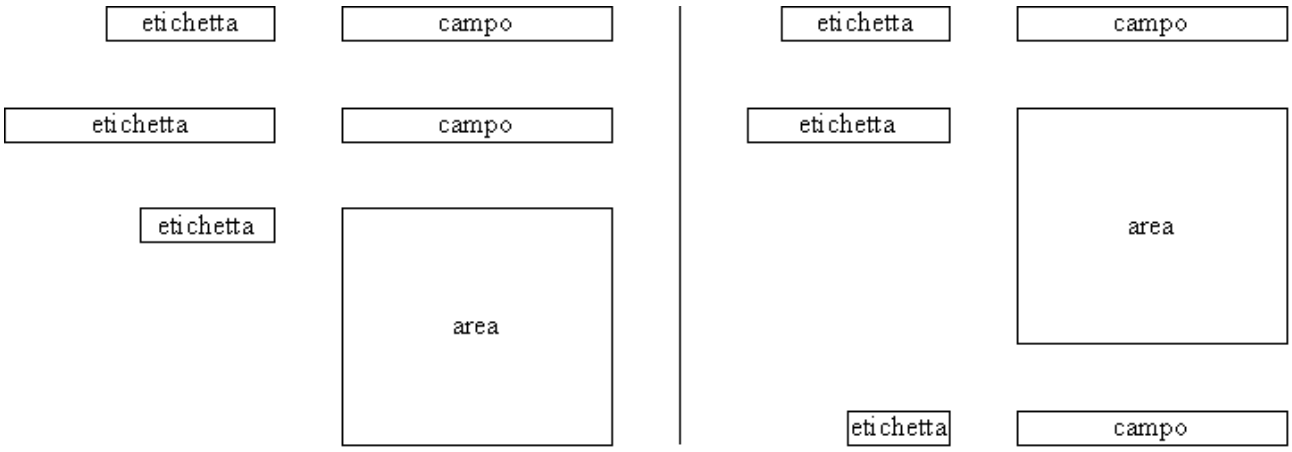


Figura 23 – Schema del contenuto di una finestra.

Un possibile metro che guidi la produzione di una griglia ideale consiste nel creare tante righe e tante colonne quante siano strettamente necessarie affinché per ogni riga e per ogni colonna esista almeno un componente che risieda su un'unica cella. La figura che segue mostra una possibile griglia.

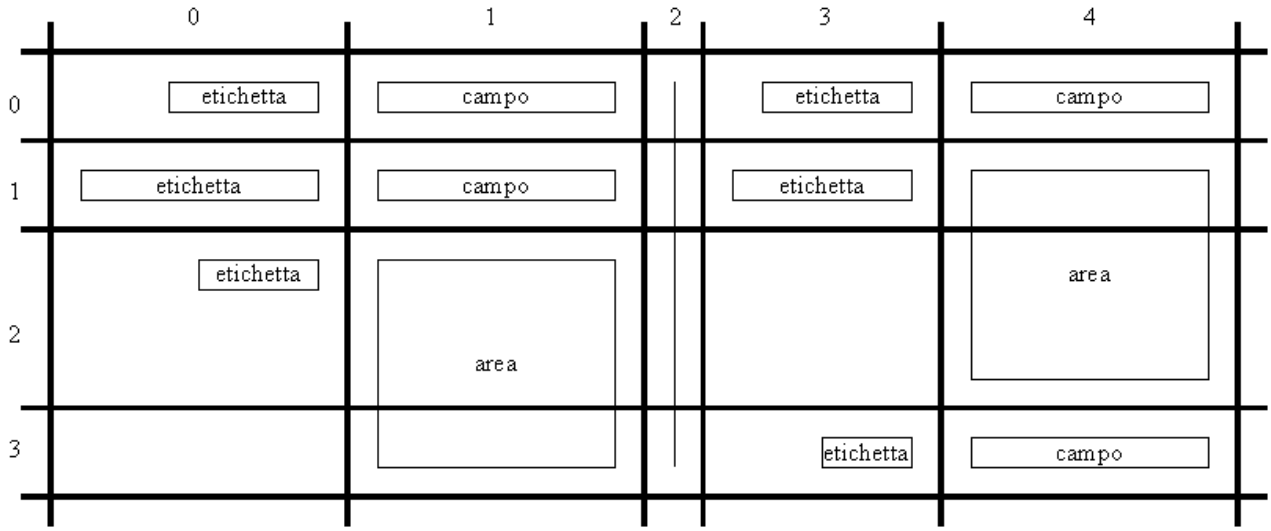


Figura 24 – Una griglia applicata alla bozza dell'interfaccia.

Tramite questa seconda bozza possiamo facilmente individuare i valori `gridx` e `gridy` che collocano un componente in una cella. Per semplificare l'associazione tra un componente ed i suoi valori di layout applichiamo un identificatore univoco ad ogni componente. La figura che segue mostra l'associazione tra i componenti ed i loro identificatori.

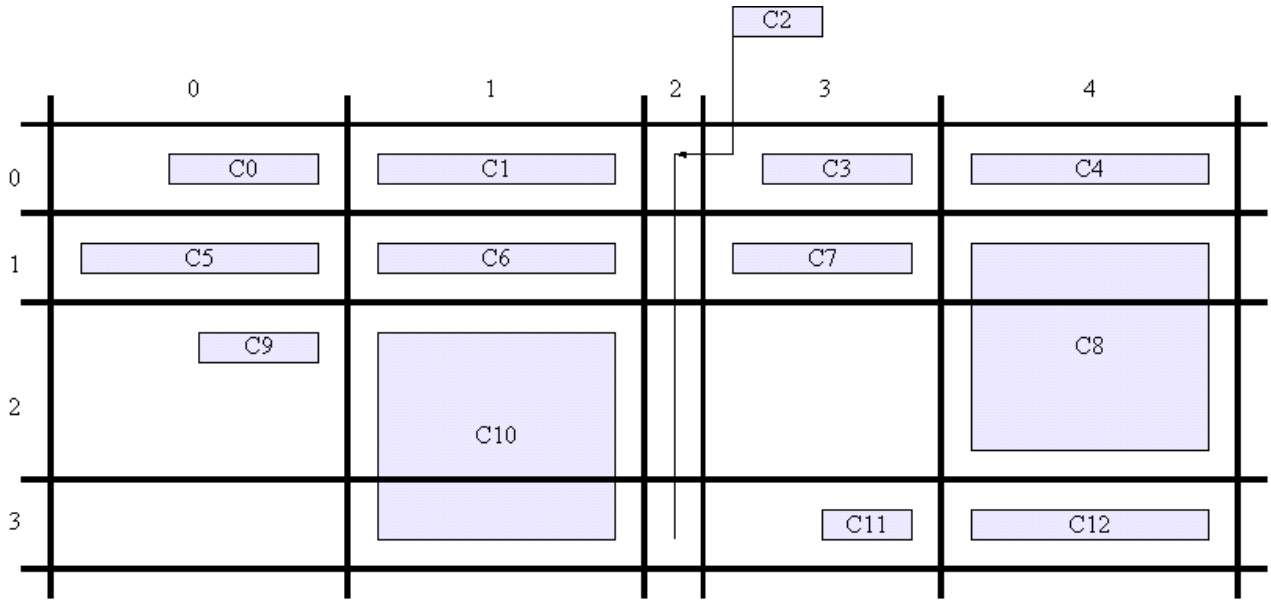


Figura 25 – Assegnazione di un identificatore ai componenti.

Ricordando che, in `GridBagConstraints`, `gridx` rappresenta l'indice della colonna e `gridy` l'indice della riga, risultano evidenti i valori `gridx` e `gridy` da associare ad ogni componente:

```
C0 (gridx = 0, gridy = 0)
C1 (gridx = 1, gridy = 0)
C2 (gridx = 2, gridy = 0)
C3 (gridx = 3, gridy = 0)
C4 (gridx = 4, gridy = 0)
C5 (gridx = 0, gridy = 1)
C6 (gridx = 1, gridy = 1)
C7 (gridx = 3, gridy = 1)
C8 (gridx = 4, gridy = 1)
C9 (gridx = 0, gridy = 2)
C10 (gridx = 1, gridy = 2)
C11 (gridx = 3, gridy = 3)
C12 (gridx = 4, gridy = 3)
```

Con lo stesso schema appare anche il numero di righe – `gridheight` – ed il numero di colonne – `gridwidth` – occupate da ciascun componente. Le informazioni associabili ad ogni componente diventano quindi:

```
C0 (gridx = 0, gridy = 0, gridwidth = 1, gridheight = 1)
C1 (gridx = 1, gridy = 0, gridwidth = 1, gridheight = 1)
C2 (gridx = 2, gridy = 0, gridwidth = 1, gridheight = 4) //occupa 4 righe
C3 (gridx = 3, gridy = 0, gridwidth = 1, gridheight = 1)
C4 (gridx = 4, gridy = 0, gridwidth = 1, gridheight = 1)
C5 (gridx = 0, gridy = 1, gridwidth = 1, gridheight = 1)
C6 (gridx = 1, gridy = 1, gridwidth = 1, gridheight = 1)
C7 (gridx = 3, gridy = 1, gridwidth = 1, gridheight = 1)
C8 (gridx = 4, gridy = 1, gridwidth = 1, gridheight = 2) //occupa 2 righe
```

```
C9 (gridx = 0, gridy = 2, gridwidth = 1, gridheight = 1)
C10 (gridx = 1, gridy = 2, gridwidth = 1, gridheight = 2) //occupa due righe
C11 (gridx = 3, gridy = 3, gridwidth = 1, gridheight = 1)
C12 (gridx = 4, gridy = 3, gridwidth = 1, gridheight = 1)
```

Possiamo anche dire qualcosa circa il riempimento, cioè il comportamento del contenuto rispetto alla superficie della sua cella, circa l'allineamento del contenuto nelle celle e riguardo ai margini. Tutti i componenti appaiono distanziati gli uni dagli altri cosa che possiamo facilmente riprodurre usando un solo `insets` per tutti. Ad esempio:

```
lim.insets = new Insets(10, 10, 10, 10); //lim è un GridBagConstraints
```

Il componente C0 non occupa tutta la sua cella ed è appoggiato sul lato destro della sua cella.

```
C0 (fill = NONE, anchor = EAST)
```

Il componente C1, contando i margini, occupa tutta la sua cella ciò che rende superfluo l'allineamento:

```
C1 (fill = BOTH)
```

Supponendo che C2 sia un componente di tipo `JSeparator`, cioè la rappresentazione di una linea, esso occupa tutta la sua cella, costituita dall'unione delle celle sulla colonna due.

```
C2 (fill = BOTH)
```

C3 si comporta come C1: non occupa tutta la sua cella ed è allineato a destra:

```
C3 (fill = NONE, anchor = EAST)
```

C4 si comporta come C1:

```
C4 (fill = BOTH)
```

Dallo schema sembrerebbe che C5 occupi tutta la sua cella ma, tornando alla bozza iniziale, possiamo osservare che tutto quello che era etichetta non occupa per intero la cella assegnata ed è allineato a destra. Dunque:

```
C5 (fill = NONE, anchor = EAST)
```

I campi e le aree, al contrario delle etichette, si appropriano di tutta la superficie disponibile.

```
C6 (fill = BOTH)
```

Proseguendo:

```
C7 (fill = NONE, anchor = EAST) //etichetta
C8 (fill = BOTH) //area di testo
```

C9 appartiene ad una riga che ha un'altezza idealmente maggiore dell'etichetta che C9 rappresenta. Dallo schema ricaviamo che il suo allineamento non possa essere semplicemente a destra ma debba essere in alto a destra.

```
C9 (fill = NONE, anchor = NORTHEAST) //etichetta
C10 (fill = BOTH) //area di testo
C11 (fill = NONE, anchor = EAST) //etichetta
C12 (fill = BOTH) //campo di testo
```

Possiamo fermarci qui? La risposta è: dipende. Abbiamo visto come le dimensioni preferite e minime dei componenti contribuiscano a determinare le proporzioni dell'interfaccia. Per contare su un risultato consistente dobbiamo quindi conoscere il tipo di componenti con cui lavoriamo perché taluni controlli Swing hanno un comportamento tipico per quanto attiene al valore delle loro dimensioni minime e preferite. Proviamo ad esaminare la distribuzione dello spazio che abbiamo ottenuto finora, in termini di altezza delle righe e larghezza delle colonne. Supponiamo che, inizialmente, vi sia spazio sufficiente a proiettare ogni componente secondo la sua dimensione preferita. Ogni colonna è larga tanto quanto è largo il più grande dei componenti contenuti nelle sue celle[2]. La colonna zero assumerà una larghezza pari a quella della più larga tra le etichette (C0, C5, C9). La colonna uno assumerà una larghezza pari a quella del più largo tra i campi di testo e l'area di testo (C1, C6, C10). La colonna due assumerà la larghezza dell'unico componente presente. La colonna assume la larghezza della più larga tra le etichette (C3, C7, C11). Infine, la colonna quattro assumerà la larghezza del più largo tra i campi di testo e l'area di testo (C4, C12, C8).

Possiamo desumere dalla bozza dell'interfaccia che le colonne uno e quattro debbano avere la stessa larghezza. Ciò significa che il più largo tra i componenti della colonna uno ed il più largo tra i componenti della colonna quattro dovranno avere pari dimensione orizzontale.

Supponendo che tutti i campi di testo abbiano le stesse impostazioni d'aspetto – font, bordi, margini – e che lo stesso valga per le aree di testo, tra loro, sappiamo che la dimensione preferita di questi componenti è determinata dal numero di colonne loro attribuito attraverso il costruttore costruzione e, per le aree di testo, dal numero di colonne e righe[3].

Rilievi analoghi valgono per l'altezza delle righe della griglia ideale in cui vogliamo collocare i componenti usando un `GridBagLayout`.

Se un `JTextField` così generato:

2 Al valore della larghezza devono essere sommati i valori dei margini – `insets` – e delle espansioni – `ipadx` e `ipady`. Tali valori sono tuttavia costanti per un componente e, idealmente, accorpabili al concetto di "più grande dei componenti".

3 Qui righe e colonne sono intese in senso editoriale, non in riferimento alle righe e colonne della griglia ideale di un `GridBagLayout`.

```
JTextField a = new JTextField(10);
```

ha la stessa larghezza di un altro `JTextField` parimenti inizializzato:

```
JTextField b = new JTextField(10);
```

e se una `JTextArea` così prodotta:

```
JTextArea a = new JTextArea(10, 10);
```

ha la stessa altezza e la stessa larghezza di un'altra area di testo ugualmente inizializzata:

```
JTextArea b = new JTextArea(10, 10);
```

allora la distribuzione iniziale degli spazi dell'interfaccia proposta in figura 23 può essere riprodotta con il seguente codice.

```
import java.awt.*;
import javax.swing.*;
import static java.awt.GridBagConstraints.*;

public class Esempio010 implements gbsample.GeneratoreGUI {
	public void creaInterfaccia(Container container) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		container.setLayout(layout);

		lim.insets.top = 10;
		lim.insets.bottom = 10;
		lim.insets.left = 10;
		lim.insets.right = 10;

		Component c0 = new JLabel("Nome");
		set(lim, 0, 0, 1, 1, NONE, EAST);
		container.add(c0, lim);

		Component c1 = new JTextField(20);
		set(lim, 1, 0, 1, 1, BOTH, CENTER);
		container.add(c1, lim);

		Component c2 = new JSeparator(SwingConstants.VERTICAL);
		set(lim, 2, 0, 1, 4, BOTH, CENTER);
		container.add(c2, lim);
```

```
        Component c3 = new JLabel("Occupazione");
        set(lim, 3, 0, 1, 1, NONE, EAST);
        container.add(c3, lim);

        Component c4 = new JTextField(20);
        set(lim, 4, 0, 1, 1, BOTH, CENTER);
        container.add(c4, lim);

        Component c5 = new JLabel("Cognome");
        set(lim, 0, 1, 1, 1, NONE, EAST);
        container.add(c5, lim);

        Component c6 = new JTextField(20);
        set(lim, 1, 1, 1, 1, BOTH, CENTER);
        container.add(c6, lim);

        Component c7 = new JLabel("Referenze");
        set(lim, 3, 1, 1, 1, NONE, EAST);
        container.add(c7, lim);

        Component c8 = new JScrollPane(new JTextArea(10, 10));
        set(lim, 4, 1, 1, 2, BOTH, CENTER);
        container.add(c8, lim);

        Component c9 = new JLabel("Note personali");
        set(lim, 0, 2, 1, 1, NONE, NORTHEAST);
        container.add(c9, lim);

        Component c10 = new JScrollPane(new JTextArea(10, 10));
        set(lim, 1, 2, 1, 2, BOTH, CENTER);
        container.add(c10, lim);

        Component c11 = new JLabel("Data");
        set(lim, 3, 3, 1, 1, NONE, EAST);
        container.add(c11, lim);

        Component c12 = new JTextField(20);
        set(lim, 4, 3, 1, 1, BOTH, CENTER);
        container.add(c12, lim);
}

private void set(GridBagConstraints lim, int gridx, int gridy,
        int gridwidth, int gridheight, int fill, int anchor)
{
        lim.gridx = gridx;
        lim.gridy = gridy;
```

```
            lim.gridwidth = gridwidth;
            lim.gridheight = gridheight;
            lim.anchor = anchor;
            lim.fill = fill;
        }
}
```

La figura successiva mostra il prodotto dell'esecuzione di `Esempio010`.

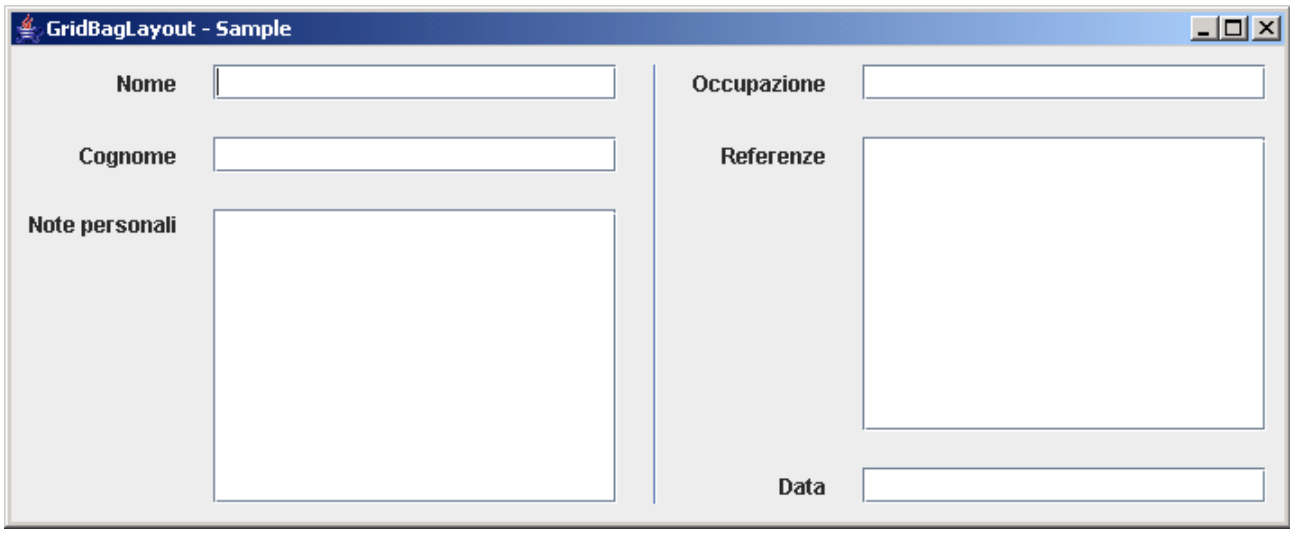


Figura 26 – Interfaccia realizzata con `GridBagLayout`.

Il termine distribuzione iniziale non è stato usato a caso. La figura successiva mostra l'effetto di una riduzione delle dimensioni del contenitore.

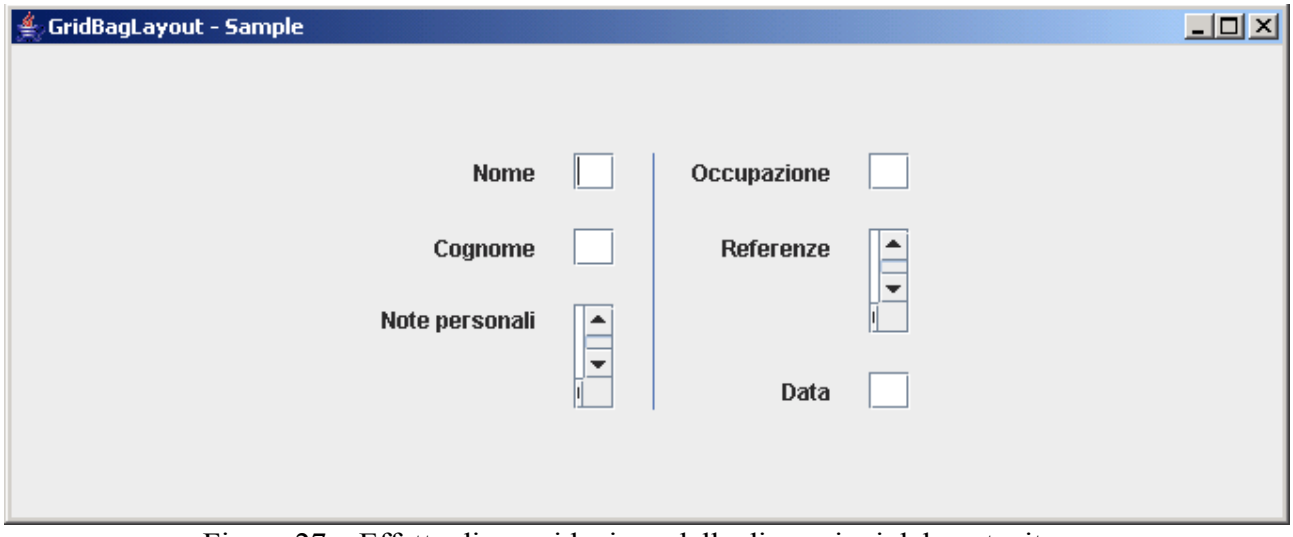


Figura 27 – Effetto di una riduzione delle dimensioni del contenitore.

Sappiamo perfettamente cosa è capitato. Inizialmente l'interfaccia aveva a disposizione uno spazio esattamente sufficiente alla proiezione dei componenti secondo le loro dimensioni preferite[4]. Quando lo spazio diventa insufficiente, `GridBagLayout` passa a considerare non più le dimensioni

4 Come si può facilmente notare esaminando il codice sorgente di `gbsample.Main`, la finestra contenitrice subisce un'invocazione `pack()` prima della proiezione. Il metodo `pack()` determina l'assunzione da parte della finestra di dimensioni tali da visualizzare ogni componente presente nel pannello del contenuto secondo le sue dimensioni preferite.

preferite ma le dimensioni minime di ciò che proietta. I componenti presentati nella figura 27 sono appunto gli stessi componenti di figura 26 ma secondo le rispettive dimensioni minime. Si evince chiaramente come le dimensioni minime di un componente di testo siano di gran lunga inferiori alle sue dimensioni preferite e come le dimensioni minime di un'etichetta `JLabel` siano pari alle sue dimensioni preferite.

Adattamento delle dimensioni.

Enumeriamo i principi dell'adattabilità di un `GridBagLayout`.

Ogni colonna ha una larghezza attuale pari alla larghezza attuale del più largo dei componenti appartenenti a quella colonna più una frazione dello spazio in eccesso determinata dal valore del maggiore dei pesi orizzontali dichiarati per le celle appartenenti a quella colonna.

Ogni riga ha un'altezza attuale pari all'altezza attuale del più alto dei componenti appartenenti a quella riga più una frazione dello spazio in eccesso determinata dal valore del maggiore dei pesi verticali dichiarati per le celle appartenenti a quella riga.

Una colonna non ha mai una larghezza attuale minore della larghezza attuale del più largo dei componenti appartenenti a quella colonna.

Una riga non ha mai un'altezza attuale minore dell'altezza attuale del più alto dei componenti appartenenti a quella riga.

Un componente per cui l'attributo `fill` sia `NONE` non subisce mai modifiche alle sue dimensioni derivanti da una mutazione delle dimensioni della cella in cui è contenuto.

Un componente per cui l'attributo `fill` sia `HORIZONTAL` occupa una regione di spazio di larghezza pari alla larghezza della colonna in cui è inserita la sua cella e non subisce mutazioni della sua altezza dovute a mutazione dell'altezza della riga in cui è inserita la sua cella.

Un componente per cui l'attributo `fill` sia `VERTICAL` occupa una regione di spazio di altezza pari all'altezza della riga in cui è inserita la sua cella e non subisce mutazioni della sua larghezza dovute a mutazione della larghezza della colonna in cui è inserita la sua cella.

Un componente per cui l'attributo `fill` sia `BOTH` occupa una regione di spazio di larghezza pari alla larghezza della colonna e di altezza pari all'altezza della riga a cui appartiene la cella in cui è inserito.

Una colonna in cui il maggiore dei pesi orizzontali tra quelli attribuiti ad ogni cella ad essa appartenente valga zero non riceve alcuna porzione delle spazio orizzontale in eccedenza.

Una riga in cui il maggiore dei pesi verticali tra quelli attribuiti ad ogni cella ad essa appartenente valga zero non riceve alcuna porzione dello spazio verticale in eccedenza.

Ai termini larghezza e altezza del componente va aggiunta una costante il cui valore dipende dai margini e dalle estensioni definite nel suo `GridBagConstraints` associato. Questa costante non cambia i termini dell'adattamento.

Adattamento progressivo.

Il primo passo per creare un adattamento progressivo dell'interfaccia consiste nel fissare le dimensioni dei componenti. Un componente Swing può determinare, e in genere determina, la propria dimensione preferita nel momento in cui è connesso ad un albero di visualizzazione. Ciò dipende dal fatto che, prima della connessione, il componente non è in grado di stabilire, ad esempio, la dimensione del carattere usato per rappresentare un certo testo. Per imporre ad un componente Swing l'equivalenza tra la sua dimensione preferita e la sua dimensione minima

occorre passare per un `AncestorListener`. Il codice seguente mostra l'applicazione di `AncestorListener` all'interfaccia di figura 26.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import static java.awt.GridBagConstraints.*;

/** Determinazione completa della regione di spazio
occupata da ciascuno componente */
public class Esempio011 implements gbsample.GeneratoreGUI {
       public void creaInterfaccia(Container container) {
              GridBagLayout layout = new GridBagLayout();
              GridBagConstraints lim = new GridBagConstraints();
              container.setLayout(layout);

              lim.insets.top = 10;
              lim.insets.bottom = 10;
              lim.insets.left = 10;
              lim.insets.right = 10;

              Component c0 = new JLabel("Nome");
              set(lim, 0, 0, 1, 1, NONE, EAST);
              container.add(c0, lim);

              Component c1 = new JTextField(20);
              set(lim, 1, 0, 1, 1, BOTH, CENTER);
              container.add(c1, lim);

              Component c2 = new JSeparator(SwingConstants.VERTICAL);
              set(lim, 2, 0, 1, 4, BOTH, CENTER);
              container.add(c2, lim);

              Component c3 = new JLabel("Occupazione");
              set(lim, 3, 0, 1, 1, NONE, EAST);
              container.add(c3, lim);

              Component c4 = new JTextField(20);
              set(lim, 4, 0, 1, 1, BOTH, CENTER);
              container.add(c4, lim);

              Component c5 = new JLabel("Cognome");
              set(lim, 0, 1, 1, 1, NONE, EAST);
              container.add(c5, lim);

              Component c6 = new JTextField(20);
```

```
        set(lim, 1, 1, 1, 1, BOTH, CENTER);
        container.add(c6, lim);

        Component c7 = new JLabel("Referenze");
        set(lim, 3, 1, 1, 1, NONE, EAST);
        container.add(c7, lim);

        Component c8 = new JScrollPane(new JTextArea(10, 10));
        set(lim, 4, 1, 1, 2, BOTH, CENTER);
        container.add(c8, lim);

        Component c9 = new JLabel("Note personali");
        set(lim, 0, 2, 1, 1, NONE, NORTHEAST);
        container.add(c9, lim);

        Component c10 = new JScrollPane(new JTextArea(10, 10));
        set(lim, 1, 2, 1, 2, BOTH, CENTER);
        container.add(c10, lim);

        Component c11 = new JLabel("Data");
        set(lim, 3, 3, 1, 1, NONE, EAST);
        container.add(c11, lim);

        Component c12 = new JTextField(20);
        set(lim, 4, 3, 1, 1, BOTH, CENTER);
        container.add(c12, lim);

        if(container instanceof JComponent) {
                DimensionFreezer freezer = new DimensionFreezer(
                        c0, c1, c2, c3, c4, c5, c6, c7, c8, c9,
                        c10, c11, c12);
                JComponent jcontainer = (JComponent)container;
                jcontainer.addAncestorListener(freezer);
        }
}

private class DimensionFreezer implements AncestorListener {
        private final Component[] components;
        private DimensionFreezer(Component... comps) {
                components = comps;
        }

        public void ancestorAdded(AncestorEvent event) {
                for(Component c : components) {
                        c.setPreferredSize(c.getSize());
                        c.setMinimumSize(c.getSize());
                }
```

```
            }

            public void ancestorMoved(AncestorEvent event) {}
            public void ancestorRemoved(AncestorEvent event) {}
        };

        private void set(GridBagConstraints lim, int gridx, int gridy,
            int gridwidth, int gridheight, int fill, int anchor)
        {
            lim.gridx = gridx;
            lim.gridy = gridy;
            lim.gridwidth = gridwidth;
            lim.gridheight = gridheight;
            lim.anchor = anchor;
            lim.fill = fill;
        }
}
```

Poiché i componenti hanno ora una dimensione minima uguale alla preferita, non si verifica il collasso riportato in figura 27. L'interfaccia diventa rigida perchè, per ogni riga e per ogni colonna, esiste almeno un componente che non si adatta alle dimensioni della sua cella[5]. Se vogliamo che l'interfaccia diventi progressivamente adattabile dobbiamo in primo luogo stabilire come vogliamo che si adatti. Poiché la colonna zero e la colonna tre contengono delle etichette possiamo decidere di lasciare che tali colonne si attestino sulla larghezza minima necessaria e sufficiente a proiettare queste etichette. Per garantirlo è sufficiente che nessuno dei componenti possa subire il ridimensionamento della colonna a cui appartiene. Possiamo fare lo stesso per la colonna due, che contiene il separatore. L'immagine che segue rappresenta schematicamente questa applicazione.

5 La rigidità di quest'interfaccia non deve essere confusa con un posizionamento assoluto dei componenti. Dato il comportamento dei componenti Swing e dei LayoutManager di Swing per ogni piattaforma è garatito che ogni componente avrà lo spazio necessario e sufficiente alla sua corretta proiezione – questo essendo il significato di dimensione preferita.

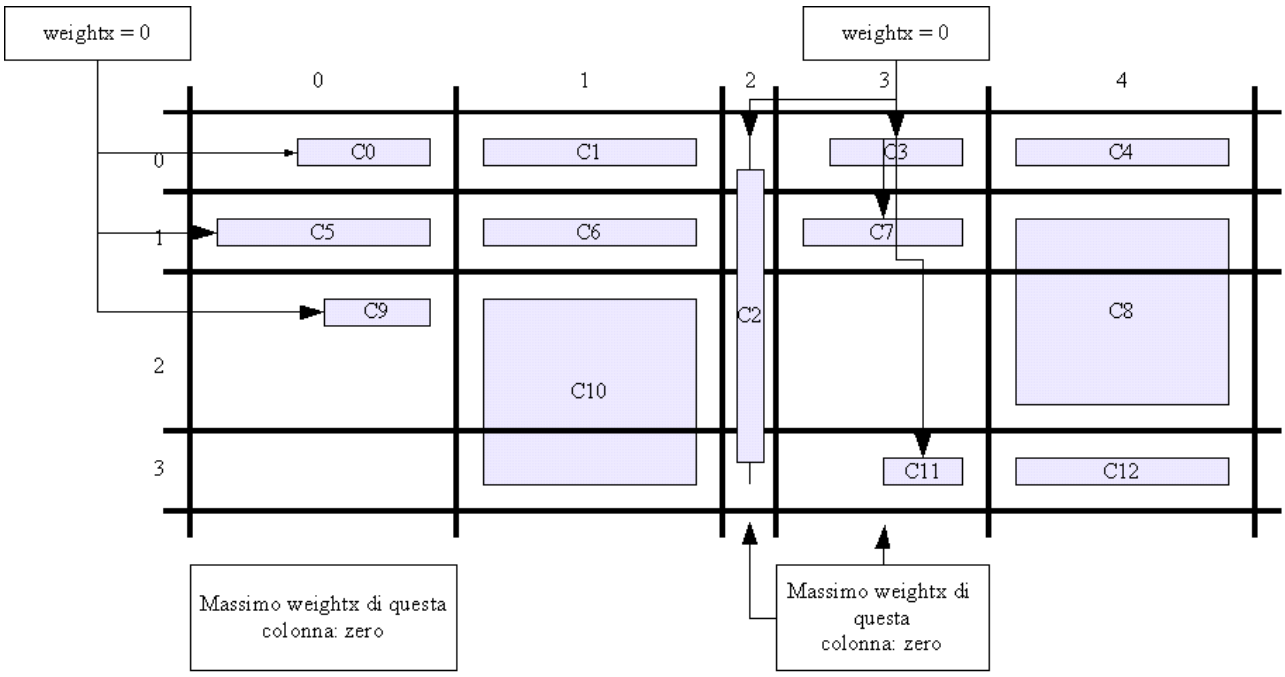


Figura 28 – Pesi per le colonne non ridimensionabili.

Ora dotiamo la colonna uno della capacità di essere ridimensionata. Poiché nel caso di presenza di più pesi orizzontali nella stessa colonna è comunque considerato solo il maggiore, possiamo limitarci ad attribuire un peso weightx = 1 alla cella che contiene il componente C1, lasciano zero come valore dei pesi orizzontali delle altre celle. Dunque:

```
C1 (weightx = 1)
C6 (weightx = 0)
C10 (weightx = 0)
```

Di seguito troviamo il codice con le proprietà discusse in evidenza.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import static java.awt.GridBagConstraints.*;

public class Esempio013 implements gbsample.GeneratoreGUI {
	public void creaInterfaccia(Container container) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		container.setLayout(layout);

		lim.insets.top = 10;
		lim.insets.bottom = 10;
		lim.insets.left = 10;
		lim.insets.right = 10;
```

```
Component c0 = new JLabel("Nome");
set(lim, 0, 0, 1, 1, NONE, EAST);
lim.weightx = 0;
container.add(c0, lim);

Component c1 = new JTextField(20);
set(lim, 1, 0, 1, 1, BOTH, CENTER);
lim.weightx = 1;
container.add(c1, lim);

Component c2 = new JSeparator(SwingConstants.VERTICAL);
set(lim, 2, 0, 1, 4, BOTH, CENTER);
lim.weightx = 0;
container.add(c2, lim);

Component c3 = new JLabel("Occupazione");
set(lim, 3, 0, 1, 1, NONE, EAST);
lim.weightx = 0;
container.add(c3, lim);

Component c4 = new JTextField(20);
set(lim, 4, 0, 1, 1, BOTH, CENTER);
lim.weightx = 0;
container.add(c4, lim);

Component c5 = new JLabel("Cognome");
set(lim, 0, 1, 1, 1, NONE, EAST);
lim.weightx = 0;
container.add(c5, lim);

Component c6 = new JTextField(20);
set(lim, 1, 1, 1, 1, BOTH, CENTER);
lim.weightx = 0;
container.add(c6, lim);

Component c7 = new JLabel("Referenze");
set(lim, 3, 1, 1, 1, NONE, EAST);
lim.weightx = 0;
container.add(c7, lim);

Component c8 = new JScrollPane(new JTextArea(10, 10));
set(lim, 4, 1, 1, 2, BOTH, CENTER);
lim.weightx = 0;
container.add(c8, lim);

Component c9 = new JLabel("Note personali");
```

```
        set(lim, 0, 2, 1, 1, NONE, NORTHEAST);
        lim.weightx = 0;
        container.add(c9, lim);

        Component c10 = new JScrollPane(new JTextArea(10, 10));
        set(lim, 1, 2, 1, 2, BOTH, CENTER);
        lim.weightx = 0;
        container.add(c10, lim);

        Component c11 = new JLabel("Data");
        set(lim, 3, 3, 1, 1, NONE, EAST);
        lim.weightx = 0;
        container.add(c11, lim);

        Component c12 = new JTextField(20);
        set(lim, 4, 3, 1, 1, BOTH, CENTER);
        lim.weightx = 0;
        container.add(c12, lim);

        if(container instanceof JComponent) {
                DimensionFreezer freezer = new DimensionFreezer(
                        c0, c1, c2, c3, c4, c5, c6, c7, c8, c9,
                        c10, c11, c12);
                JComponent jcontainer = (JComponent)container;
                jcontainer.addAncestorListener(freezer);
        }
}

private class DimensionFreezer implements AncestorListener {
        private final Component[] components;
        private DimensionFreezer(Component... comps) {
                components = comps;
        }

        public void ancestorAdded(AncestorEvent event) {
                for(Component c : components) {
                        c.setPreferredSize(c.getSize());
                        c.setMinimumSize(c.getSize());
                }
        }

        public void ancestorMoved(AncestorEvent event) {}
        public void ancestorRemoved(AncestorEvent event) {}
};

private void set(GridBagConstraints lim, int gridx, int gridy,
        int gridwidth, int gridheight, int fill, int anchor)
```

```
	{
		lim.gridx = gridx;
		lim.gridy = gridy;
		lim.gridwidth = gridwidth;
		lim.gridheight = gridheight;
		lim.anchor = anchor;
		lim.fill = fill;
	}
}
```

La figura successiva mostra cosa accade quando la larghezza del contenitore diminuisce.

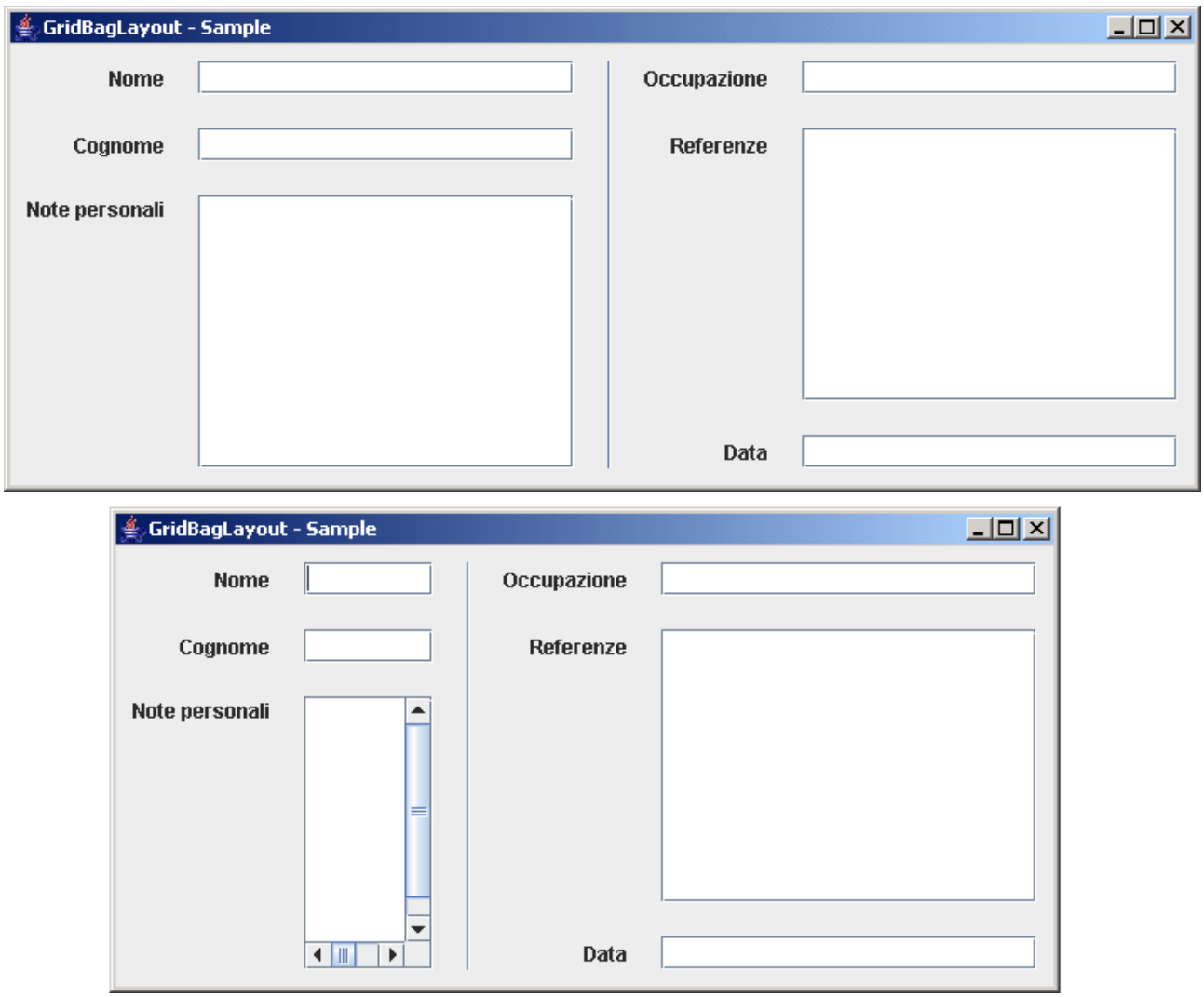


Figura 29 – Cedimento della colonna uno alla riduzione della larghezza del contenitore.

Possiamo fare lo stesso per la colonna quattro. Anche qui, il peso orizzontale della colonna, cioè il peso orizzontale maggiore tra quelli delle sue celle, deve essere maggiore di zero. Se imponiamo anche alla seconda colonna di partecipare alla distribuzione dello spazio in eccesso avremo due contendenti al ridimensionamento. Per far si che essi si spartiscano in modo equo lo spazio orizzontale, dovremo assicurarci che il peso della colonna quattro sia uguale al peso della colonna uno ciò che determinerà l'attribuzione di un identico numero di pixel[6] alle due colonne. Dunque

6 Salvo ovviamente il caso in cui il numero totale sia dispari.

avremo:

```
C4 (weightx = 1)
C8 (weightx = 0)
C12 (weightx = 0)
```

Il codice che segue propone queste modifiche in evidenza.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import static java.awt.GridBagConstraints.*;

public class Esempio014 implements gbsample.GeneratoreGUI {
       public void creaInterfaccia(Container container) {
              GridBagLayout layout = new GridBagLayout();
              GridBagConstraints lim = new GridBagConstraints();
              container.setLayout(layout);

              lim.insets.top = 10;
              lim.insets.bottom = 10;
              lim.insets.left = 10;
              lim.insets.right = 10;

              Component c0 = new JLabel("Nome");
              set(lim, 0, 0, 1, 1, NONE, EAST);
              lim.weightx = 0;
              container.add(c0, lim);

              Component c1 = new JTextField(20);
              set(lim, 1, 0, 1, 1, BOTH, CENTER);
              lim.weightx = 1;
              container.add(c1, lim);

              Component c2 = new JSeparator(SwingConstants.VERTICAL);
              set(lim, 2, 0, 1, 4, BOTH, CENTER);
              lim.weightx = 0;
              container.add(c2, lim);

              Component c3 = new JLabel("Occupazione");
              set(lim, 3, 0, 1, 1, NONE, EAST);
              lim.weightx = 0;
              container.add(c3, lim);

              Component c4 = new JTextField(20);
```

```
set(lim, 4, 0, 1, 1, BOTH, CENTER);
lim.weightx = 1;
container.add(c4, lim);

Component c5 = new JLabel("Cognome");
set(lim, 0, 1, 1, 1, NONE, EAST);
lim.weightx = 0;
container.add(c5, lim);

Component c6 = new JTextField(20);
set(lim, 1, 1, 1, 1, BOTH, CENTER);
lim.weightx = 0;
container.add(c6, lim);

Component c7 = new JLabel("Referenze");
set(lim, 3, 1, 1, 1, NONE, EAST);
lim.weightx = 0;
container.add(c7, lim);

Component c8 = new JScrollPane(new JTextArea(10, 10));
set(lim, 4, 1, 1, 2, BOTH, CENTER);
lim.weightx = 0;
container.add(c8, lim);

Component c9 = new JLabel("Note personali");
set(lim, 0, 2, 1, 1, NONE, NORTHEAST);
lim.weightx = 0;
container.add(c9, lim);

Component c10 = new JScrollPane(new JTextArea(10, 10));
set(lim, 1, 2, 1, 2, BOTH, CENTER);
lim.weightx = 0;
container.add(c10, lim);

Component c11 = new JLabel("Data");
set(lim, 3, 3, 1, 1, NONE, EAST);
lim.weightx = 0;
container.add(c11, lim);

Component c12 = new JTextField(20);
set(lim, 4, 3, 1, 1, BOTH, CENTER);
lim.weightx = 0;
container.add(c12, lim);

if(container instanceof JComponent) {
      DimensionFreezer freezer = new DimensionFreezer(
            c0, c1, c2, c3, c4, c5, c6, c7, c8, c9,
```

```
                        c10, c11, c12);
                JComponent jcontainer = (JComponent)container;
                jcontainer.addAncestorListener(freezer);
            }
        }

        private class DimensionFreezer implements AncestorListener {
            private final Component[] components;
            private DimensionFreezer(Component... comps) {
                components = comps;
            }

            public void ancestorAdded(AncestorEvent event) {
                for(Component c : components) {
                    c.setPreferredSize(c.getSize());
                    c.setMinimumSize(c.getSize());
                }
            }

            public void ancestorMoved(AncestorEvent event) {}
            public void ancestorRemoved(AncestorEvent event) {}
        };

        private void set(GridBagConstraints lim, int gridx, int gridy,
            int gridwidth, int gridheight, int fill, int anchor)
        {
            lim.gridx = gridx;
            lim.gridy = gridy;
            lim.gridwidth = gridwidth;
            lim.gridheight = gridheight;
            lim.anchor = anchor;
            lim.fill = fill;
        }
}
```

L'immagine successiva è il risultato di `Esempio014`.

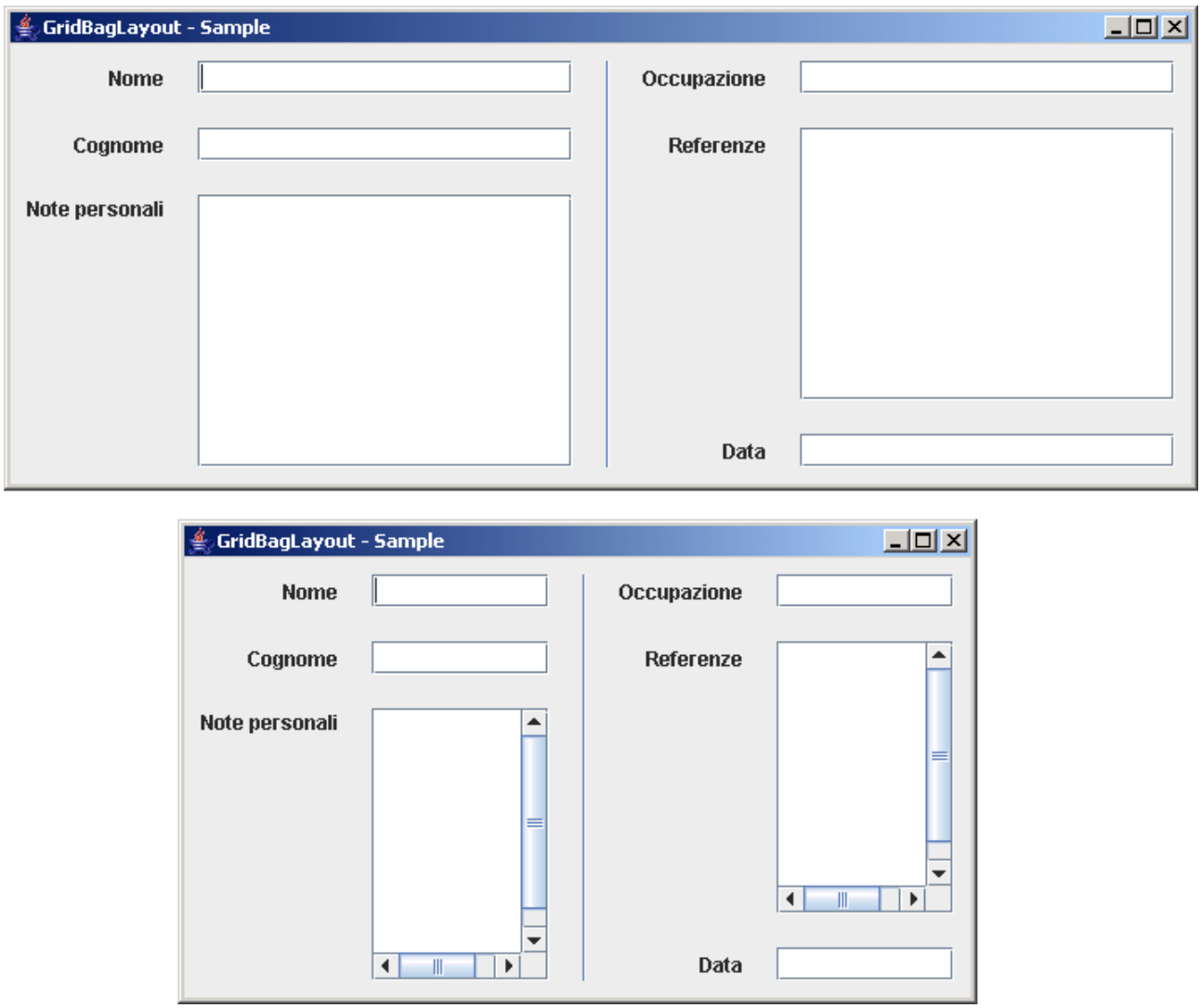


Figura 30 – Cedimento delle colonne uno e quattro.

La nostra interfaccia è ancora rigida sull'asse verticale. Per intervenire su questo versante valgono le stesse considerazioni fatte per l'asse orizzontale. Potremmo rendere tutte le righe ridimensionabili ma esiste un candidato ideale: la riga due. Sulle righe zero e uno ci sono due campi di testo. I campi di testo hanno normalmente una dimensione verticale di poco superiore a quella strettamente necessaria a proiettare adeguatamente il testo contenuto. Facendo collassare queste righe perderemmo il contenuto prima che lo spazio diventi realmente insufficiente. Prima o poi lo perderemmo comunque: c'è un limite al numero di pixel minimo sufficiente a proiettare qualcosa di significativo. Potendo, è meglio far sì che questo accada il più tardi possibile. Prendendo spazio dalla riga due riduciamo – o aumentiamo – progressivamente le dimensioni delle due aree di testo. Prendiamo la cella del componente C9 e imponiamogli un peso verticale pari a uno.

```
C9 (weighty = 1)
```

Il codice che segue evidenzia quest'imposizione.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
```

```
import static java.awt.GridBagConstraints.*;

public class EsempioO15 implements gbsample.GeneratoreGUI {
	public void creaInterfaccia(Container container) {
		GridBagLayout layout = new GridBagLayout();
		GridBagConstraints lim = new GridBagConstraints();
		container.setLayout(layout);

		lim.insets.top = 10;
		lim.insets.bottom = 10;
		lim.insets.left = 10;
		lim.insets.right = 10;

		Component c0 = new JLabel("Nome");
		set(lim, 0, 0, 1, 1, NONE, EAST);
		lim.weightx = 0;
		container.add(c0, lim);

		Component c1 = new JTextField(20);
		set(lim, 1, 0, 1, 1, BOTH, CENTER);
		lim.weightx = 1;
		container.add(c1, lim);

		Component c2 = new JSeparator(SwingConstants.VERTICAL);
		set(lim, 2, 0, 1, 4, BOTH, CENTER);
		lim.weightx = 0;
		container.add(c2, lim);

		Component c3 = new JLabel("Occupazione");
		set(lim, 3, 0, 1, 1, NONE, EAST);
		lim.weightx = 0;
		container.add(c3, lim);

		Component c4 = new JTextField(20);
		set(lim, 4, 0, 1, 1, BOTH, CENTER);
		lim.weightx = 1;
		container.add(c4, lim);

		Component c5 = new JLabel("Cognome");
		set(lim, 0, 1, 1, 1, NONE, EAST);
		lim.weightx = 0;
		container.add(c5, lim);

		Component c6 = new JTextField(20);
		set(lim, 1, 1, 1, 1, BOTH, CENTER);
		lim.weightx = 0;
		container.add(c6, lim);
```

```
        Component c7 = new JLabel("Referenze");
        set(lim, 3, 1, 1, 1, NONE, EAST);
        lim.weightx = 0;
        container.add(c7, lim);

        Component c8 = new JScrollPane(new JTextArea(10, 10));
        set(lim, 4, 1, 1, 2, BOTH, CENTER);
        container.add(c8, lim);

        Component c9 = new JLabel("Note personali");
        set(lim, 0, 2, 1, 1, NONE, NORTHEAST);
        lim.weightx = 0;
        lim.weighty = 1;
        container.add(c9, lim);

        Component c10 = new JScrollPane(new JTextArea(10, 10));
        set(lim, 1, 2, 1, 2, BOTH, CENTER);
        lim.weightx = 0;
        lim.weighty = 0;
        container.add(c10, lim);

        Component c11 = new JLabel("Data");
        set(lim, 3, 3, 1, 1, NONE, EAST);
        lim.weightx = 0;
        lim.weighty = 0;
        container.add(c11, lim);

        Component c12 = new JTextField(20);
        set(lim, 4, 3, 1, 1, BOTH, CENTER);
        lim.weightx = 0;
        container.add(c12, lim);

        if(container instanceof JComponent) {
               DimensionFreezer freezer = new DimensionFreezer(
                      c0, c1, c2, c3, c4, c5, c6, c7, c8, c9,
                      c10, c11, c12);
               JComponent jcontainer = (JComponent)container;
               jcontainer.addAncestorListener(freezer);
        }
}

private class DimensionFreezer implements AncestorListener {
        private final Component[] components;
        private DimensionFreezer(Component... comps) {
               components = comps;
        }
```

```
            public void ancestorAdded(AncestorEvent event) {
                  for(Component c : components) {
                        c.setPreferredSize(c.getSize());
                        c.setMinimumSize(c.getSize());
                  }
            }

            public void ancestorMoved(AncestorEvent event) {}
            public void ancestorRemoved(AncestorEvent event) {}
      };

      private void set(GridBagConstraints lim, int gridx, int gridy,
            int gridwidth, int gridheight, int fill, int anchor)
      {
            lim.gridx = gridx;
            lim.gridy = gridy;
            lim.gridwidth = gridwidth;
            lim.gridheight = gridheight;
            lim.anchor = anchor;
            lim.fill = fill;
      }
}
```

La figura successiva mostra il comportamento dell'interfaccia al variare delle dimensioni del contenitore.

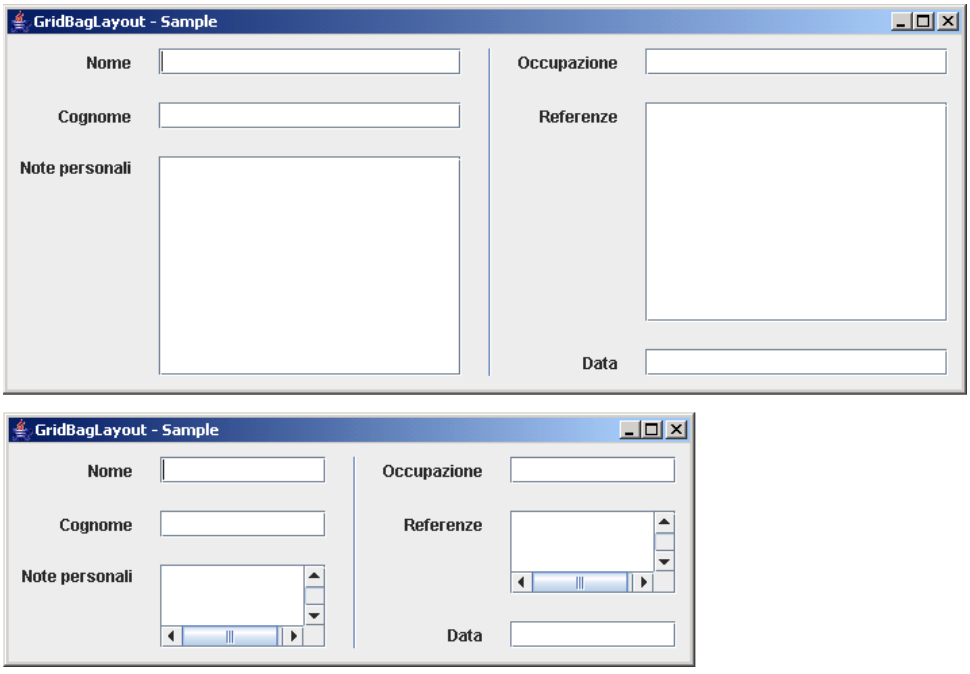


Figura 31 – Cedimento verticale e orizzontale.

Riducendo lo spazio verticale oltre un certo limite si noterà la sparizione dell'etichetta C9 – “Note

personali", riprodotta nella figura che segue.

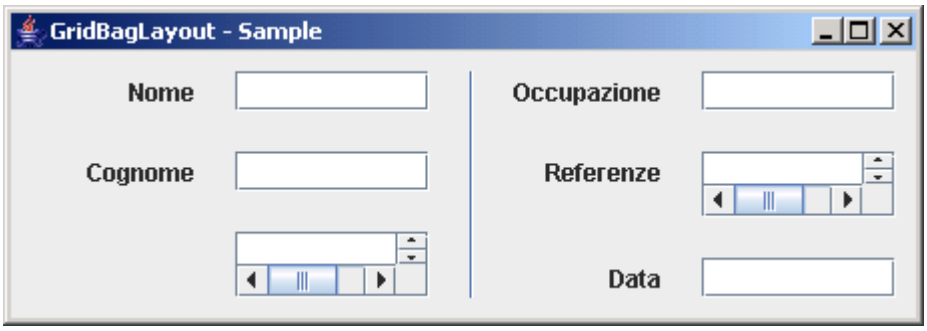


Figura 32 – La sparizione dell'etichetta C9 "Note personali"

Oltre un certo limite il problema non può essere prevenuto passando semplicemente per il LayoutManager tuttavia è possibile intervenire per evitare che il componente sparisca all'improvviso. La sparizione è dovuta al fatto che il componente esiste su una riga collassabile – la riga due che ha un peso orizzontale diverso da zero. Per com'è strutturata la griglia ideale non ci sono molte altre soluzioni, almeno non altre che preservino l'intendo di voler ridurre lo spazio a disposizione delle due aree di testo. Alterando quella struttura si può porvi rimedio. Preliminarmente rileviamo che per poter attribuire una qualche proprietà ad un riga o ad una colonna della griglia ideale deve esistere almeno un componente la cui collocazione nella griglia inizi in quella riga o in quella colonna. Ciò che faremo ora sarà inserire un componente trasparente nella griglia in una riga intermedia tra la riga due e la riga tre. Attribuiremo poi a questo un peso verticale maggiore di zero e lasceremo che i pesi verticali delle altre righe siano zero. In questo modo a collassare sarà la riga contenente il componente fantasma. L'inserimento richiederà una manipolazione degli indici delle celle. La figura seguente riporta la struttura della griglia ideale, aggiornata a contenere la riga con il componente collassabile.

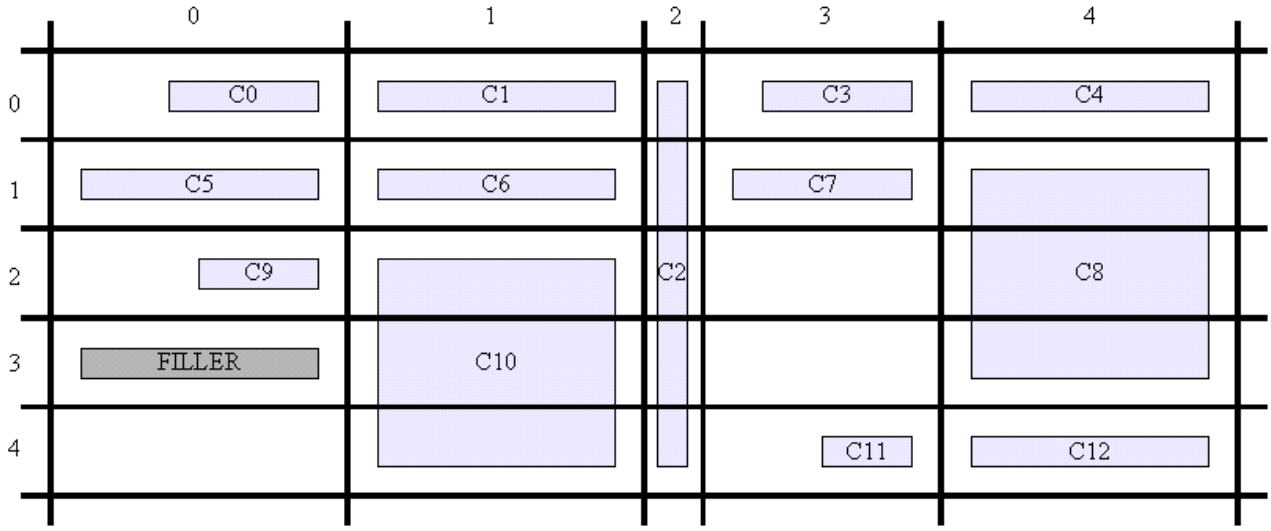


Figura 33 – Nuova griglia con riga liberamente collassabile.

Non è difficile ricostruire i parametri necessari alla nuova collocazione dei componenti.

ID	gridx	gridy	gridwidth	gridheight	weightx	weighty	fill	anchor
C0	0	0	1	1	0	0	NONE	EAST
C1	1	0	1	1	1	0	BOTH	CENTER
C2	2	0	1	5	0	0	BOTH	CENTER
C3	3	0	1	1	0	0	NONE	EAST
C4	4	0	1	1	1	0	BOTH	CENTER
C5	0	1	1	1	0	0	NONE	EAST
C6	1	1	1	1	0	0	BOTH	CENTER
C7	3	1	1	1	0	0	NONE	EAST
C8	4	1	1	3	0	0	BOTH	CENTER
C9	0	2	1	1	0	0	NONE	EAST
C10	1	2	1	3	0	0	BOTH	CENTER
FILLER	0	3	1	1	0	1	NONE	CENTER
C11	3	4	1	1	0	0	NONE	EAST
C12	4	4	1	1	0	0	BOTH	CENTER

Di seguito troviamo il codice sorgente che applica queste nuove impostazioni.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import static java.awt.GridBagConstraints.*;

public class Esempio016 implements gbsample.GeneratoreGUI {
	public void creaInterfaccia(Container container) {
		BagBuilder bag = new BagBuilder(container);
		bag.limits().insets = new Insets(10, 10, 10, 10);

		Component c0 = new JLabel("Nome");
		bag.add(c0, 0, 0, 1, 1, 0, 0, NONE, EAST);

		Component c1 = new JTextField(20);
		bag.add(c1, 1, 0, 1, 1, 1, 0, BOTH, CENTER);

		Component c2 = new JSeparator(SwingConstants.VERTICAL);
		bag.add(c2, 2, 0, 1, 5, 0, 0, BOTH, CENTER);

		Component c3 = new JLabel("Occupazione");
		bag.add(c3, 3, 0, 1, 1, 0, 0, NONE, EAST);

		Component c4 = new JTextField(20);
```

```
        bag.add(c4, 4, 0, 1, 1, 1, 0, BOTH, CENTER);

        Component c5 = new JLabel("Cognome");
        bag.add(c5, 0, 1, 1, 1, 0, 0, NONE, EAST);

        Component c6 = new JTextField(20);
        bag.add(c6, 1, 1, 1, 1, 0, 0, BOTH, CENTER);

        Component c7 = new JLabel("Referenze");
        bag.add(c7, 3, 1, 1, 1, 0, 0, NONE, EAST);

        Component c8 = new JScrollPane(new JTextArea(10, 10));
        bag.add(c8, 4, 1, 1, 3, 0, 0, BOTH, CENTER);

        Component c9 = new JLabel("Note personali");
        bag.add(c9, 0, 2, 1, 1, 0, 0, NONE, EAST);

        Component c10 = new JScrollPane(new JTextArea(10, 10));
        bag.add(c10, 1, 2, 1, 3, 0, 0, BOTH, CENTER);

        bag.limits().insets = new Insets(0, 0, 0, 0);
        JLabel filler = new JLabel();
        bag.add(filler, 0, 3, 1, 1, 0, 1, NONE, CENTER);

        bag.limits().insets = new Insets(10, 10, 10, 10);
        Component c11 = new JLabel("Data");
        bag.add(c11, 3, 4, 1, 1, 0, 0, NONE, EAST);

        Component c12 = new JTextField(20);
        bag.add(c12, 4, 4, 1, 1, 0, 0, BOTH, CENTER);

        if(container instanceof JComponent) {
              DimensionFreezer freezer = new DimensionFreezer(
                    c0, c1, c2, c3, c4, c5, c6, c7, c8, c9,
                    c10, c11, c12);
              JComponent jcontainer = (JComponent)container;
              jcontainer.addAncestorListener(freezer);
        }
}

private class DimensionFreezer implements AncestorListener {
        private final Component[] components;
        private DimensionFreezer(Component... comps) {
              components = comps;
        }

        public void ancestorAdded(AncestorEvent event) {
```

```
                for(Component c : components) {
                        c.setPreferredSize(c.getSize());
                        c.setMinimumSize(c.getSize());
                }
        }

        public void ancestorMoved(AncestorEvent event) {}
        public void ancestorRemoved(AncestorEvent event) {}
    };

    /** Semplifica la vita... */
    private class BagBuilder {
        private Container container;
        private GridBagLayout layout = new GridBagLayout();
        private GridBagConstraints lim = new GridBagConstraints();

        public BagBuilder(Container container) {
            this.container = container;
            this.container.setLayout(layout);
        }

        public GridBagConstraints limits() {
            return lim;
        }

        public void add(Component c, int gridx, int gridy,
            int gridwidth, int gridheight, int weightx,
            int weighty, int fill, int anchor)
        {
            lim.gridx = gridx;
            lim.gridy = gridy;
            lim.gridwidth = gridwidth;
            lim.gridheight = gridheight;
            lim.weightx = weightx;
            lim.weighty = weighty;
            lim.fill = fill;
            lim.anchor = anchor;
            container.add(c, lim);
        }
    };
}
```

Il risultato ottenuto, riportato nella figura successiva, è il mancato collasso dell'etichetta, a parità di dimensioni dell'interfaccia.

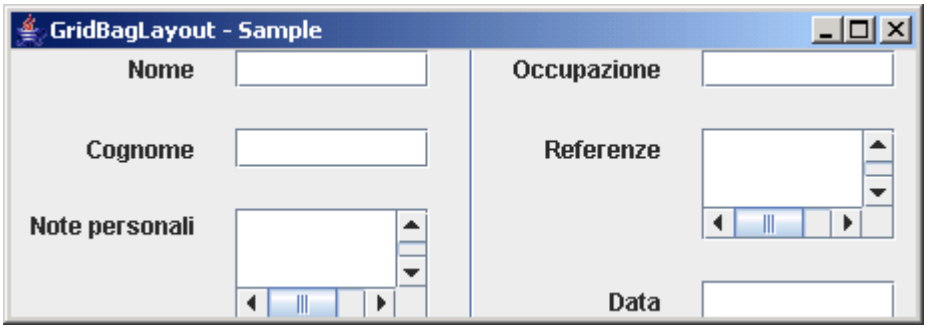


Figura 34 – Applicazione di un componente nascosto.

Spazio in eccesso: definizione.

Ora che comprendiamo l'influenza dei pesi sulla distribuzione dello spazio in eccesso è opportuno dare una definizione esatta del significato di "eccesso". Lo spazio in eccesso, distribuito dai pesi alle colonne ed alle righe, è determinato da:

spazio orizzontale in eccesso: larghezza del contenitore meno la somma delle larghezze delle colonne che abbiano un peso orizzontale uguale a zero;

spazio verticale in eccesso: altezza del contenitore meno la somma delle altezze delle righe che abbiano un peso verticale uguale a zero[7].

Nell'immagine di figura 34 si nota bene l'applicazione di questa definizione. Appena le dimensioni del contenitore diventano tali da azzerare l'equazione

spazio verticale in eccesso = altezza del contenitore meno la somma delle altezze delle righe con peso verticale (`weighty`) uguale a zero

l'interfaccia "cessa di adattarsi": non c'è più spazio verticale da distribuire dunque l'interfaccia si comporterà, all'ulteriore diminuzione dell'altezza, come se fosse rigida[8].

Dunque le righe e le colonne che abbiano un peso, verticale o orizzontale, diverso da zero non solo partecipano alla distribuzione dello spazio eccedente le necessità dell'interfaccia ma contribuiscono anche a determinare la quantità di spazio eccedente. Nel caso in cui ogni riga abbia un peso diverso da zero, lo spazio verticale eccedente sarà sempre pari all'intera altezza del contenitore ed questa quantità sarà distribuita tra le righe in base alle proporzioni stabilite dai loro pesi. Lo stesso vale, in senso orizzontale, per le colonne.

Griglie non evidenti.

Continuiamo il discorso prendendo in esame una bozza di interfaccia utente in cui la disposizione dei componenti renda meno immediata la definizione di una griglia ideale. La bozza è proposta

7 Definizione coerente con quanto riportato nel contratto di GridBagConstraints, secondo cui i pesi, orizzontali e verticali, operano solo quando ci sia dello spazio in eccesso e non quando la superficie disponibile sia semplicemente diversa, in più o in meno, da quella sufficiente alla proiezione del contenuto secondo le sue dimensioni preferite.

8 Quando la superficie del contenitore che usi un GridBagLayout diventi minore delle necessità di proiezione del contenuto secondo le sue dimensioni preferite

nella figura che segue.

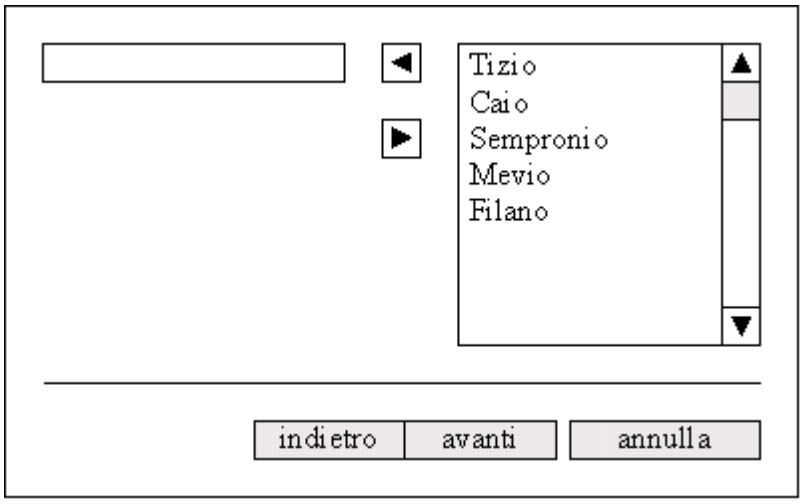


Figura 36 – Bozza di interfaccia utente.

La bozza in discussione pone un problema di struttura derivante dalla collocazione ideale dei pulsanti con etichette “indietro” e “avanti”. Il nostro punto di partenza nella definizione della griglia ideale è l'assunto che per ogni riga esista almeno un componente che abbia origine e termine in una cella di quella riga alta non più di una riga e che per ogni colonna esista almeno un componente che abbia origine e termine in una cella appartenente a quella colonna larga non più di una colonna. La figura seguente mostra un primo tentativo di stabilire una griglia.

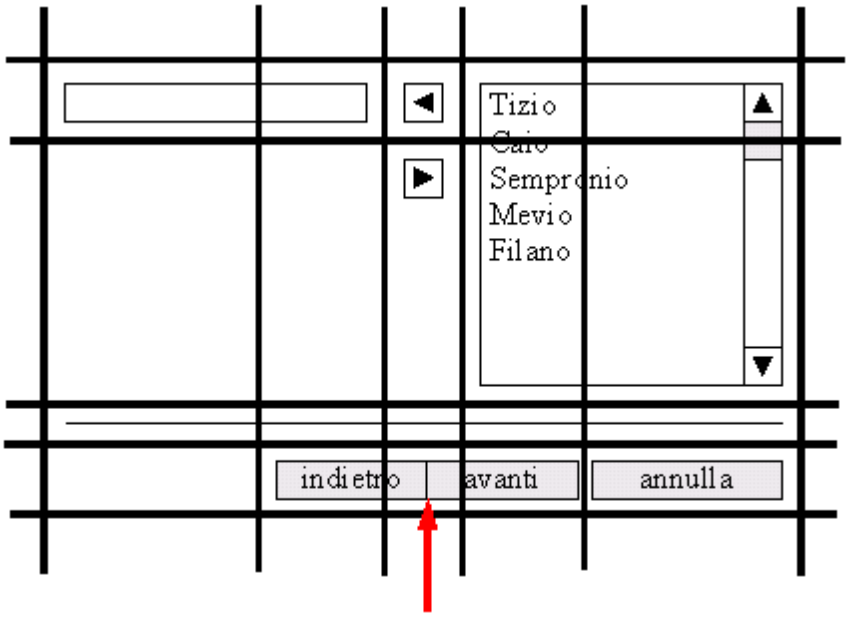


Figura 37 – Problemi di definizione di una griglia ideale.

Due componenti non possono sovrapporsi, cioè non possono occupare una stessa cella direttamente, in quanto cella di origine del componente o indirettamente, in quanto cella occupata per estensione del componente su più righe o più colonne. È una questione di corretta proiezione sulla base delle informazioni disponibili al momento della creazione dell'interfaccia. In particolare, per la maggior parte dei componenti non è dato conoscere quali siano i valori della loro larghezza e della loro altezza prima che quei componenti siano visualizzabili sullo schermo dell'utente. Non potendo

stabilire quando sia largo il pulsante “indietro” la sua estensione nella cella di origine del pulsante “avanti” potrebbe produrre una sovrapposizione della proiezione dei due pulsanti, il che è graficamente ed ergonomicamente inaccettabile.

La soluzione sta nella violazione del nostro principio guida, riprodotta nell'immagine che segue.

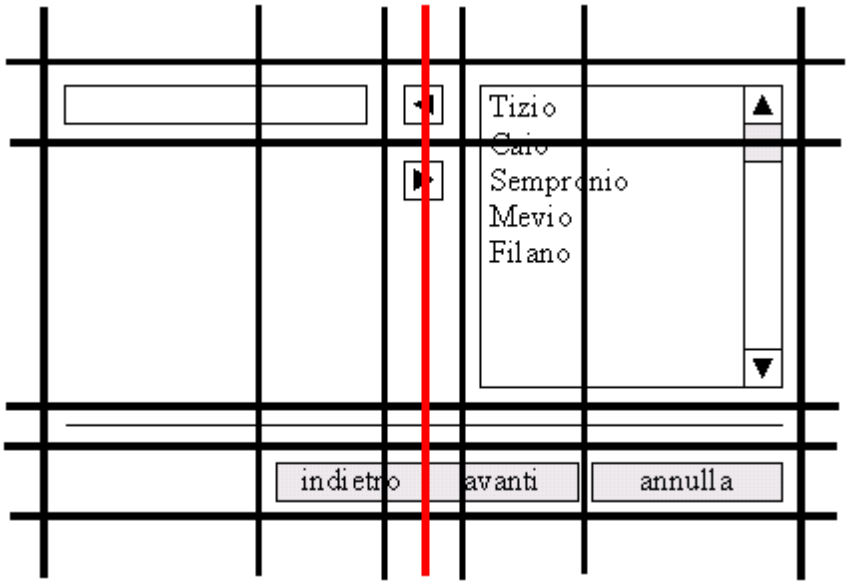


Figura 38 – Una griglia per la bozza di interfaccia utente.

La linea verticale rossa genera due colonne, ai lati della linea, per le quali non esiste un componente che abbia origine e termine in una cella di quella colonna. Il *leit–motif* delle nostre griglie ideali è solo un principio guida non un asserto che non possa essere altro che vero.

La figura seguente mostra l'indicizzazione dei componenti, delle righe e delle colonne che useremo per riprodurre l'interfaccia.

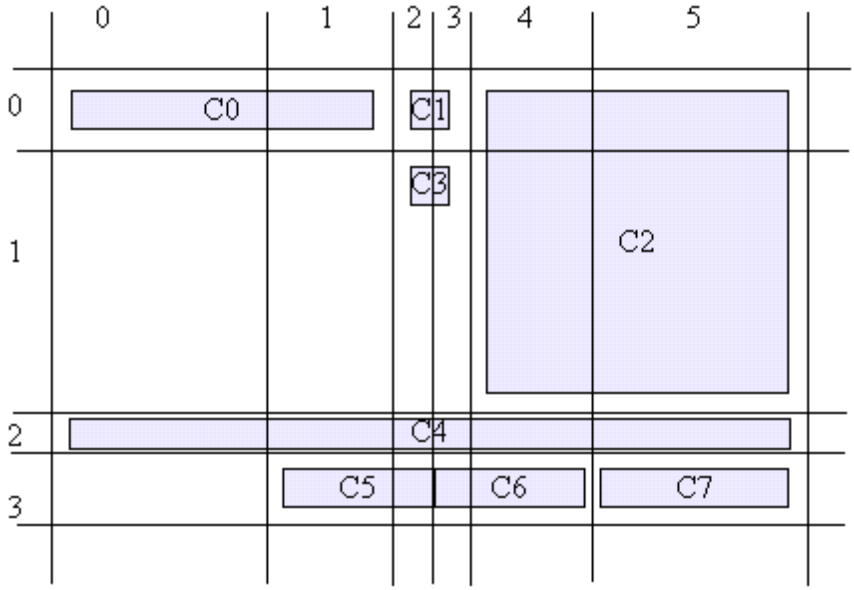


Figura 39 – Schema della griglia immaginaria.

Aggiungiamo alla griglia proposta le considerazioni inerenti all'adattabilità dell'interfaccia. Per farlo abbiamo bisogno di immaginare il risultato che vogliamo ottenere al variare delle dimensioni del contenitore. La figura successiva tenda di rendere l'idea del comportamento dell'interfaccia a fronte

di una variazione della larghezza del contenitore.

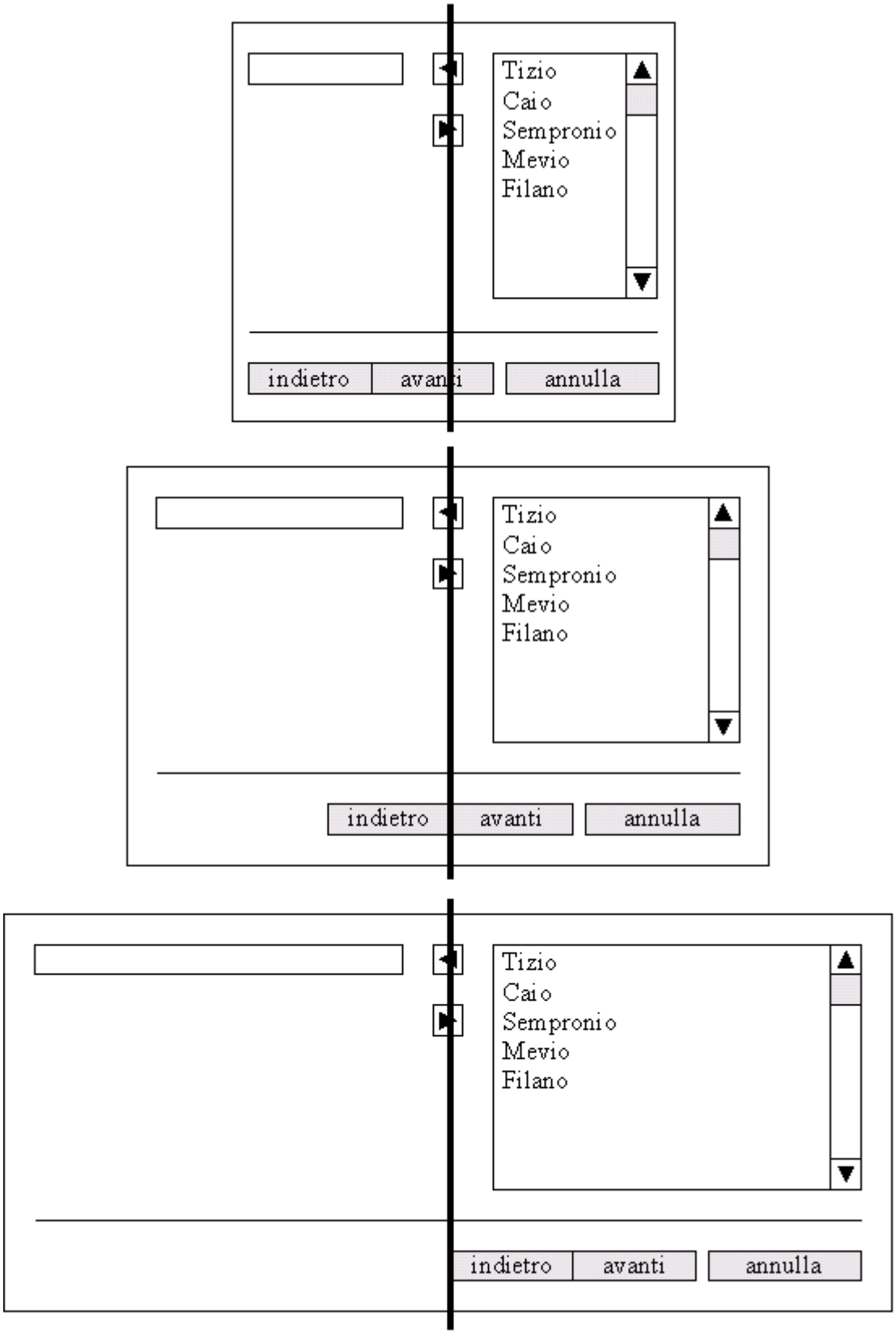


Figura 40 – Effetto di un aumento o di una diminuzione dello spazio orizzontale disponibile.

I due pulsanti al centro ed i tre in basso non subiscono una variazione di dimensione. Si riposizionano per effetto della variazione di larghezza subita dai componenti esterni – il campo di

testo e la lista. Similmente, possiamo pensare che la variazione dello spazio sull'asse verticale abbia l'effetto proposto nella figura successiva.

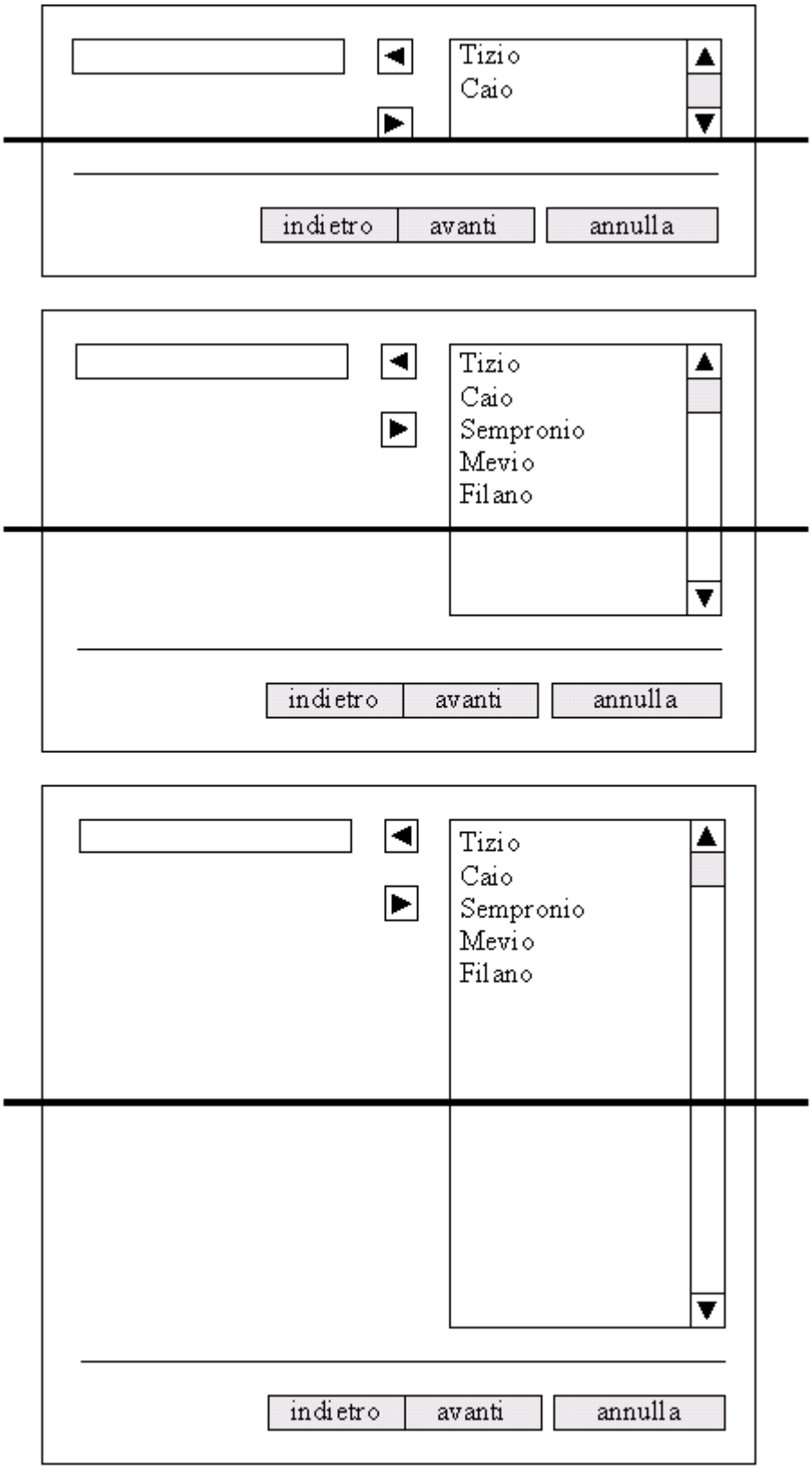


Figura 41 – Risposta dell'interfaccia alla variazione dello spazio verticale.

Ora si prenda in esame la struttura proposta nella figura 39. Quali colonne possono variare la propria larghezza affinchè si riproduca la situazione esposta in figura 40? La figura seguente mostra

lo schema della griglia a cui dobbiamo applicare l'ideale ridimensionamento.

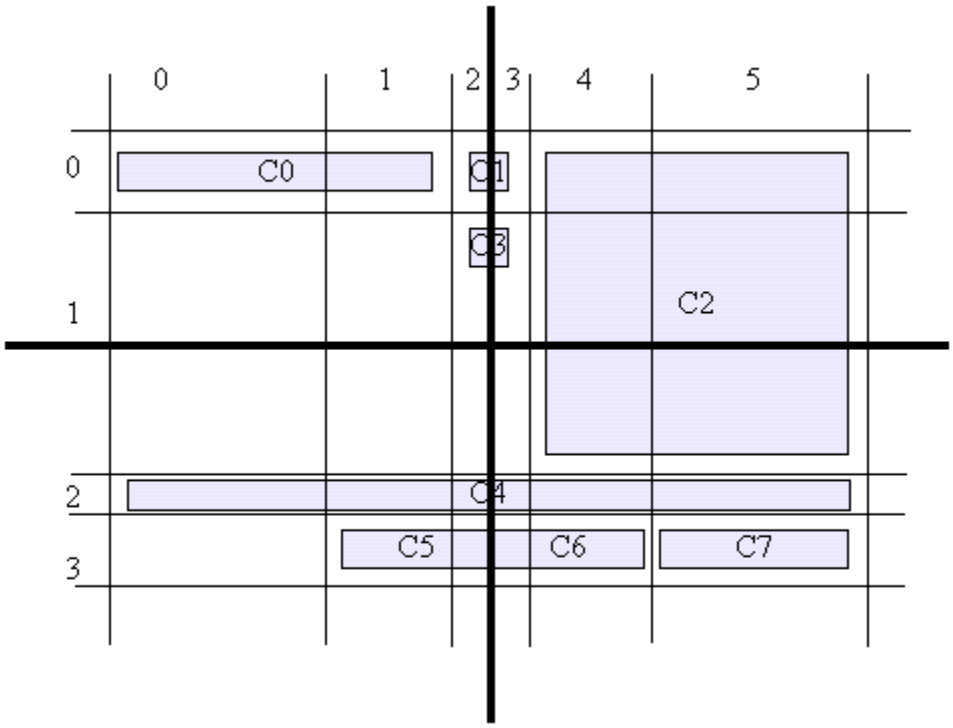


Figura 42 – La griglia da distorcere.

Sappiamo che alcuni componente hanno una dimensione che non deve variare. Ciò significa che le righe e le colonne a cui appartengono devono avere dei pesi pari a zero. In riferimento allo schema di figura 42, le righe sono la zero, perchè contiene il campo di testo e il primo pulsante di trasferimento, la uno perchè contiene il secondo pulsante di trasferimento, la due perchè contiene il separatore e la tre perchè contiene i tre pulsanti di controllo. Dunque questo schema non ci consente di applicare quel particolare tipo di ridimensionamento verticale che abbiamo immaginato.

In linea generale ci sono due problemi riconducibili al ridimensionamento di una riga o di una colonna. Il primo è che questo può comportare un'alterazione delle distanze tra i componenti se le righe e le colonne che rispondono allo "*stretching*" non sono le prime o le ultime. Il secondo è che l'assenza di spazio in eccesso può causare la sparizione di quei componenti che abbiano come cella di origine una cella della riga o della colonna adattabile.

Se noi usassimo la riga uno per ottenere una variazione dell'interfaccia in risposta all'aumento o alla diminuzione dell'altezza del contenitore, otterremo ad un certo punto la sparizione del pulsante C3, fenomeno che abbiamo già avuto modo di osservare nella figura 32, relativa alla precedente bozza di interfaccia.

Come abbiamo fatto allora, così faremo ora: inserendo un componente trasparente possiamo creare una riga invisibile che tuttavia partecipi al ridimensionamento. L'inserimento è proposto nella figura che segue.

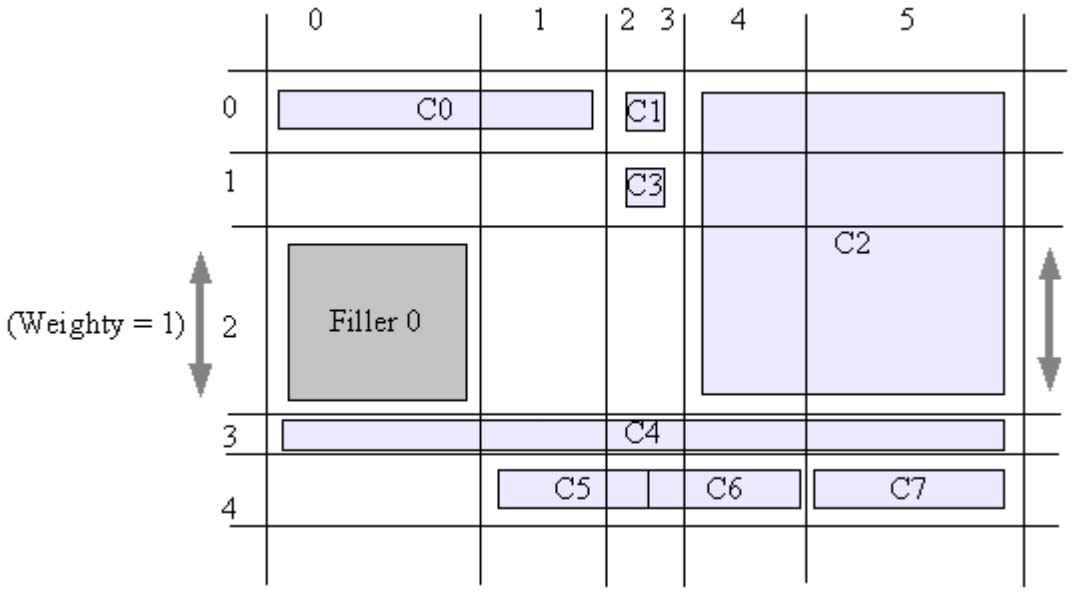
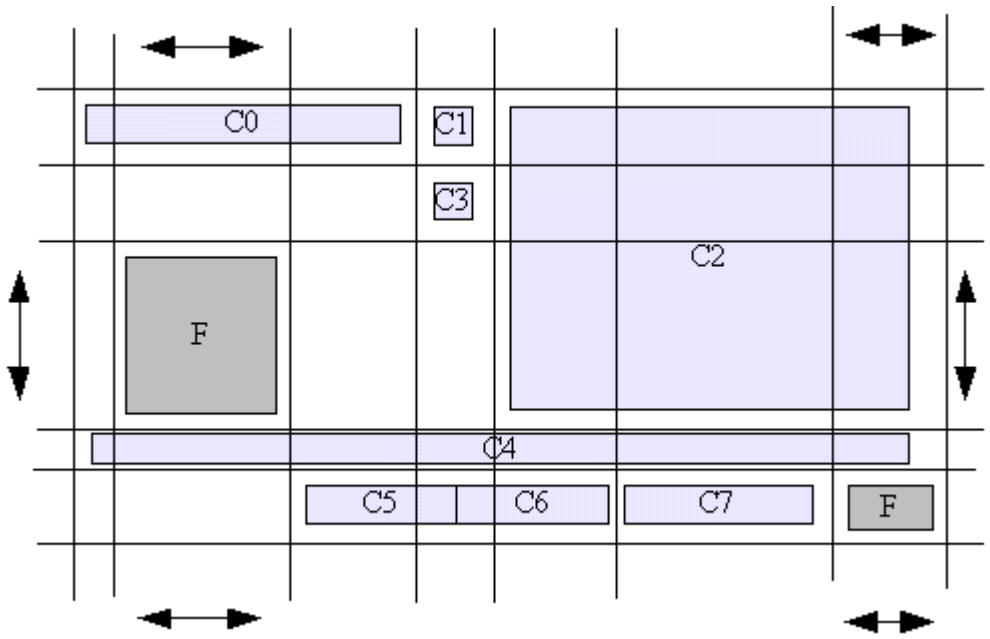


Figura 43 – Produzione di una riga invisibile.

Ovviamente se questo andasse bene anche in questo caso l'interfaccia che staremmo realizzando non sarebbe che un doppione propedeutico di quella precedente. Stavolta non va bene. Consideriamo infatti il ridimensionamento orizzontale. La colonna zero non può essere ridimensionata. Se lo facessimo il componente C0 ad un certo punto sparirebbe perchè la colonna zero contiene la cella d'origine di C0[9]. La colonna uno contiene la cella di origine di C5. La colonna due oltre a contenere i pulsanti C1 e C3 determina anche il margine effettivo tra questi pulsanti e il campo di testo, a sinistra, e la lista, a destra. Nelle nostre prove di allungamento quel margine era pensato costante. La colonna quattro contiene l'origine di C2 e la colonna cinque l'origine di C7.

Rendere questa bozza ridimensionabile evitando che dei componenti possano sparire, richiede più di una manipolazione. Una possibilità è rappresentata nella figura che segue.



9 Cosa che invece avevamo lasciato verificarsi nel caso dell'interfaccia di figura 29.

Figura 44 – Inserimento di componenti trasparenti per ridimensionare l'interfaccia.

Al fine di valutare l'efficacia della proposta, senza dover mettere mano al codice, è sufficiente immaginare cosa accadrebbe se le colonne e le righe segnalate dalla frecce bidirezionali venissero espanse o ridotte. Per quanto riguarda la mutazione dello spazio verticale lo schema sembra corrispondere pienamente alle necessità. Tutti i componenti dei quali devono essere salvaguardate dimensioni e visibilità risultano correttamente gestiti. Sull'asse orizzontale, il lato sinistro della struttura appare idoneo a conseguire il risultato proposto nella figura 40. La porzione destra, al contrario, è del tutto insoddisfacente per via del mancato allineamento dei pulsanti al lato destro della lista. Preservare questo allineamento senza mutare la larghezza del pulsante C7 richiederebbe la capacità di spostare un componente da una colonna ad un'altra. `GridBagLayout` non dispone di uno strumento idoneo a questo scopo[10].

Nella maggior parte dei casi un `GridBagLayout` è in grado di disporre un insieme di componenti esattamente nell'esatto modo desiderato, ciò che chiamiamo collocazione statica dei componenti. Il codice che segue mostra le impostazioni di layout che soddisfano staticamente la bozza proposta.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import static java.awt.GridBagConstraints.*;

public class Esempio017 implements gbsample.GeneratoreGUI {
	public void creaInterfaccia(Container container) {
		Object[] listData = {
			"Tizio", "Caio", "Sempronio", "Mevio", "Filano",
		};

		BagBuilder bag = new BagBuilder(container);

		Component c0 = new JTextField(10);
		Component c1 = createIconButton("left.bmp");
		Component c2 = new JScrollPane(new JList(listData));
		Component c3 = createIconButton("right.bmp");
		Component c4 = new JSeparator(SwingConstants.HORIZONTAL);
		Component c5 = new JButton("indietro");
		Component c6 = new JButton("avanti");
		Component c7 = new JButton("annulla");

		bag.limits().insets = new Insets(10, 10, 10, 10);
		bag.add(c0, 0, 0, 2, 1, 0, 0, BOTH, CENTER);
		bag.add(c1, 2, 0, 2, 1, 0, 0, NONE, CENTER);
		bag.add(c2, 4, 0, 2, 2, 0, 0, BOTH, CENTER);
		bag.add(c3, 2, 1, 2, 1, 0, 0, NONE, NORTH);
		bag.add(c4, 0, 2, 6, 1, 0, 0, BOTH, CENTER);
```

10 Per realizzare l'effetto citato occorrerebbe creare dinamicamente la griglia virtuale, disponendo i componenti in colonne diverse secondo le dimensioni del contenitore. Si tratta di un fine pienamente raggiungibile ma che riteniamo estraneo al comportamento per cui `GridBagLayout` è stato pensato.

```
        bag.limits().insets.top = 0;
        bag.limits().insets.left = 0;
        bag.limits().insets.right = 0;
        bag.add(c5, 1, 3, 2, 1, 0, 0, NONE, EAST);
        bag.add(c6, 3, 3, 2, 1, 0, 0, NONE, EAST);

        bag.limits().insets.left = 4;
        bag.limits().insets.right = 10;
        bag.add(c7, 5, 3, 1, 1, 0, 0, NONE, CENTER);

        if(container instanceof JComponent) {
                DimensionFreezer freezer = new DimensionFreezer(
                        c0, c1, c2, c3, c4, c5, c6, c7);
                JComponent jcontainer = (JComponent)container;
                jcontainer.addAncestorListener(freezer);
        }
}

private JButton createIconButton(String imageResourceName) {
        JButton button = new JButton();
        button.setMargin(new Insets(0, 0, 0, 0));
        try {
                java.net.URL location = getClass().getResource(imageResourceName);
                if(location == null) {
                        throw new java.io.IOException("Invalid URL");
                }
                java.awt.image.BufferedImage image =
                        javax.imageio.ImageIO.read(location);
                button.setIcon(new ImageIcon(image));
        } catch(java.io.IOException ex) {
                System.err.println("Cannot load image " + imageResourceName);
        }
        return button;
}

private class DimensionFreezer implements AncestorListener {
        private final Component[] components;
        private DimensionFreezer(Component... comps) {
                components = comps;
        }

        public void ancestorAdded(AncestorEvent event) {
                for(Component c : components) {
                        c.setPreferredSize(c.getSize());
                        c.setMinimumSize(c.getSize());
                }
```

```
            }

            public void ancestorMoved(AncestorEvent event) {}
            public void ancestorRemoved(AncestorEvent event) {}
        };

        /** Semplifica la vita... */
        private class BagBuilder {
            private Container container;
            private GridBagLayout layout = new GridBagLayout();
            private GridBagConstraints lim = new GridBagConstraints();

            public BagBuilder(Container container) {
                this.container = container;
                this.container.setLayout(layout);
            }

            public GridBagConstraints limits() {
                return lim;
            }

            public void add(Component c, int gridx, int gridy,
                int gridwidth, int gridheight, int weightx,
                int weighty, int fill, int anchor)
            {
                lim.gridx = gridx;
                lim.gridy = gridy;
                lim.gridwidth = gridwidth;
                lim.gridheight = gridheight;
                lim.weightx = weightx;
                lim.weighty = weighty;
                lim.fill = fill;
                lim.anchor = anchor;
                container.add(c, lim);
            }
        };
}
```

La figura successiva è il risultato della proiezione del codice di `Esempio017`.

Figura 45 – L'interfaccia di `Esempio017` nel `LookAndFeel Windows`.

Minore è la quantità di strutture realizzabili che oltre a disporre i componenti nel modo opportuno siano anche in grado di sostenere un'opportuna reazione alle mutazioni della superficie disponibile, ciò che chiamiamo collocazione dinamica dei componenti. Nella figura 45 ben si nota la ragione per cui l'adattabilità sia un requisito di primaria importanza. L'interfaccia risultante, infatti, appare più compressa sull'asse orizzontale di quanto è stato pensato nella bozza. La compressione apparente è il risultato del computo da parte dei componenti inseriti delle loro dimensioni preferite. Ciò che dovremmo introdurre è un semplice meccanismo che consenta al contenitore di assumere una larghezza maggiore nel caso in cui le proporzioni indotte dalle dimensioni preferite dei componenti, con le quali non si dovrebbe interferire, risultino in una variazione dell'aspetto che si discosti dall'idea originale.

Immaginiamo per un attimo che i pulsanti C5, C6 e C7 non esistano. La griglia ideale rappresentata nella figura che segue risulta idonea sia alla collocazione statica che alla proiezione dinamica del contenuto.

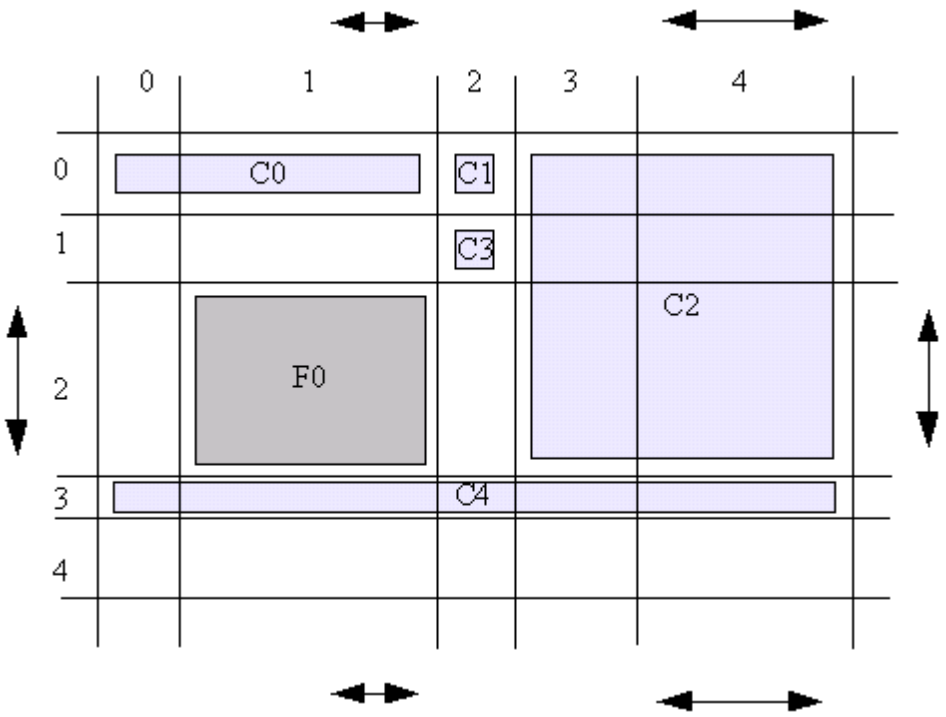


Figura 46 – L'interfaccia senza i pulsanti.

L'idoneità si desume dal fatto che ridimensionando le righe e le colonne, evidenziate dalle frecce bidirezionali, sono preservate sia la visibilità dei componenti sia il loro allineamento reciproco sia la costanza delle distanze relative. A questo è facile immaginare la soluzione. Inserendo nella quarta riga un contenitore intermedio e collocando in questo intermedio i tre pulsanti, otteniamo la capacità apparente dei pulsanti di spostarsi da una colonna all'altra al mutare delle dimensioni orizzontali del contenitore.

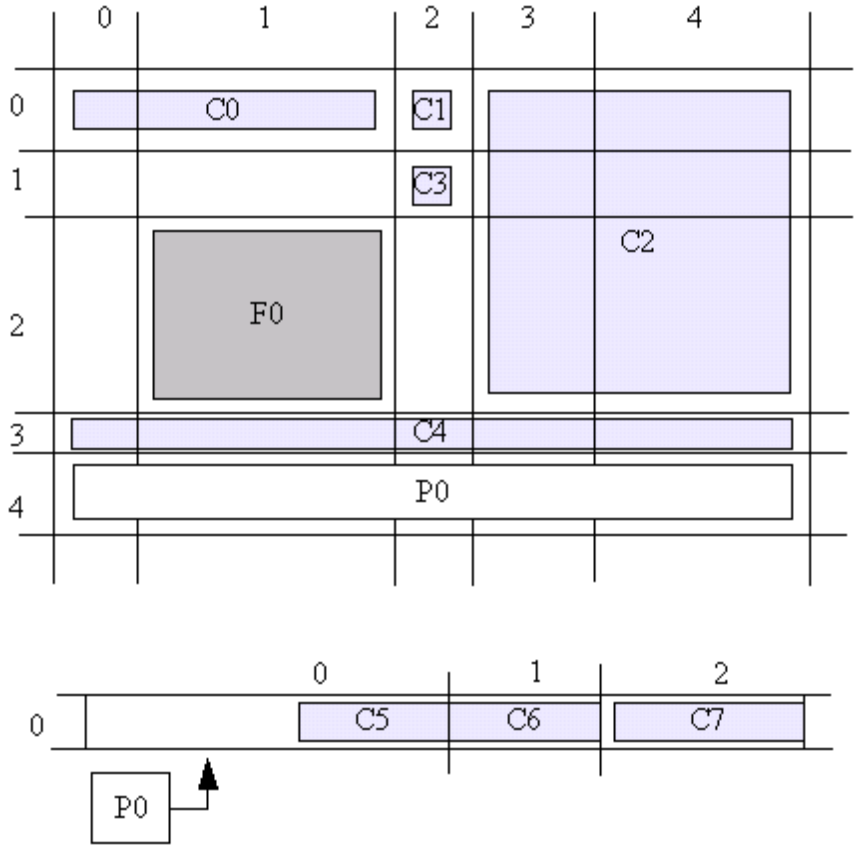


Figura 47 – Uso di un contenitore intermedio.

Applicare questa soluzione è una mera questione di numeri. La seguente tabella mostra i valori applicati ai componenti del contenitore principale.

ID	Gridx	Gridy	Gridwidth	Gridheight	Weightx	Weighty	Fill	Anchor
C0	0	0	2	1	0	0	BOTH	CENTER
C1	2	0	1	1	0	0	NONE	CENTER
C2	3	0	2	3	1	0	BOTH	CENTER
C3	2	1	1	1	0	0	NONE	NORTH
C4	0	3	5	1	0	0	BOTH	CENTER
F0	1	2	1	1	1	1	BOTH	CENTER
P0	0	4	0	5	0	0	NONE	EAST

La tabella successiva mostra i valori applicati ai componenti del contenitore intermedio P0.

ID	Gridx	Gridy	Gridwidth	Gridheight	Weightx	Weighty	Fill	Anchor
C5	0	0	1	1	0	0	NONE	EAST
C6	1	0	1	1	0	0	NONE	WEST
C7	2	0	1	1	0	0	NONE	WEST

Il codice seguente è la trasposizione dei parametri su riportati con l'aggiunta di alcuni valori per i margini.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import static java.awt.GridBagConstraints.*;

public class Esempio018 implements gbsample.GeneratoreGUI {
	public void creaInterfaccia(Container container) {
		Object[] listData = {
			"Tizio", "Caio", "Sempronio", "Mevio", "Filano",
		};

		BagBuilder bag = new BagBuilder(container);

		Component c0 = new JTextField(10);
		Component c1 = createIconButton("left.bmp");
		Component c2 = new JScrollPane(new JList(listData));
		Component c3 = createIconButton("right.bmp");
		Component c4 = new JSeparator(SwingConstants.HORIZONTAL);
		Component f0 = new JLabel();
		Container p0 = new JPanel();

		Component c5 = new JButton("indietro");
		Component c6 = new JButton("avanti");
		Component c7 = new JButton("annulla");

		bag.limits().insets = new Insets(10, 10, 10, 10);
		bag.add(c0, 0, 0, 2, 1, 0, 0, BOTH, CENTER);
		bag.add(c1, 2, 0, 1, 1, 0, 0, NONE, CENTER);
		bag.add(c2, 3, 0, 2, 3, 1, 0, BOTH, CENTER);
		bag.add(c3, 2, 1, 1, 1, 0, 0, NONE, NORTH);
		bag.add(c4, 0, 3, 5, 1, 0, 0, BOTH, CENTER);
		bag.add(f0, 1, 2, 1, 1, 1, 1, BOTH, CENTER);
		bag.limits().insets.top = 0;
		bag.add(p0, 0, 4, 0, 5, 0, 0, NONE, EAST);

		BagBuilder p0Bag = new BagBuilder(p0);
		p0Bag.add(c5, 0, 0, 1, 1, 0, 0, NONE, EAST);
		p0Bag.add(c6, 1, 0, 1, 1, 0, 0, NONE, WEST);
		p0Bag.limits().insets.left = 4;
		p0Bag.add(c7, 2, 0, 1, 1, 0, 0, NONE, WEST);

		container.setPreferredSize(new Dimension(400, 250));

		if(container instanceof JComponent) {
			DimensionFreezer freezer = new DimensionFreezer(
				c0, c1, c2, c3, c4, c5, c6, c7, p0);
			JComponent jcontainer = (JComponent)container;
			jcontainer.addAncestorListener(freezer);
		}
	}

	private JButton createIconButton(String imageResourceName) {
		JButton button = new JButton();
		button.setMargin(new Insets(0, 0, 0, 0));
		try {
			java.net.URL location = getClass().getResource(imageResourceName);
			if(location == null) {
				throw new java.io.IOException("Invalid URL");
			}
			java.awt.image.BufferedImage image =
				javax.imageio.ImageIO.read(location);
```

```
                button.setIcon(new ImageIcon(image));
            } catch(java.io.IOException ex) {
                System.err.println("Cannot load image " + imageResourceName);
            }
            return button;
        }

        private class DimensionFreezer implements AncestorListener {
            private final Component[] components;
            private DimensionFreezer(Component... comps) {
                components = comps;
            }

            public void ancestorAdded(AncestorEvent event) {
                for(Component c : components) {
                    c.setPreferredSize(c.getSize());
                    c.setMinimumSize(c.getSize());
                }
            }

            public void ancestorMoved(AncestorEvent event) {}
            public void ancestorRemoved(AncestorEvent event) {}
        };

        /** Semplifica la vita... */
        private class BagBuilder {
            private Container container;
            private GridBagLayout layout = new GridBagLayout();
            private GridBagConstraints lim = new GridBagConstraints();

            public BagBuilder(Container container) {
                this.container = container;
                this.container.setLayout(layout);
            }

            public GridBagConstraints limits() {
                return lim;
            }

            public void add(Component c, int gridx, int gridy,
                int gridwidth, int gridheight, int weightx,
                int weighty, int fill, int anchor)
            {
                lim.gridx = gridx;
                lim.gridy = gridy;
                lim.gridwidth = gridwidth;
                lim.gridheight = gridheight;
                lim.weightx = weightx;
                lim.weighty = weighty;
                lim.fill = fill;
                lim.anchor = anchor;
                container.add(c, lim);
            }
        };
}
```

La figura successiva mostra il risultato dell'applicazione di quel codice.

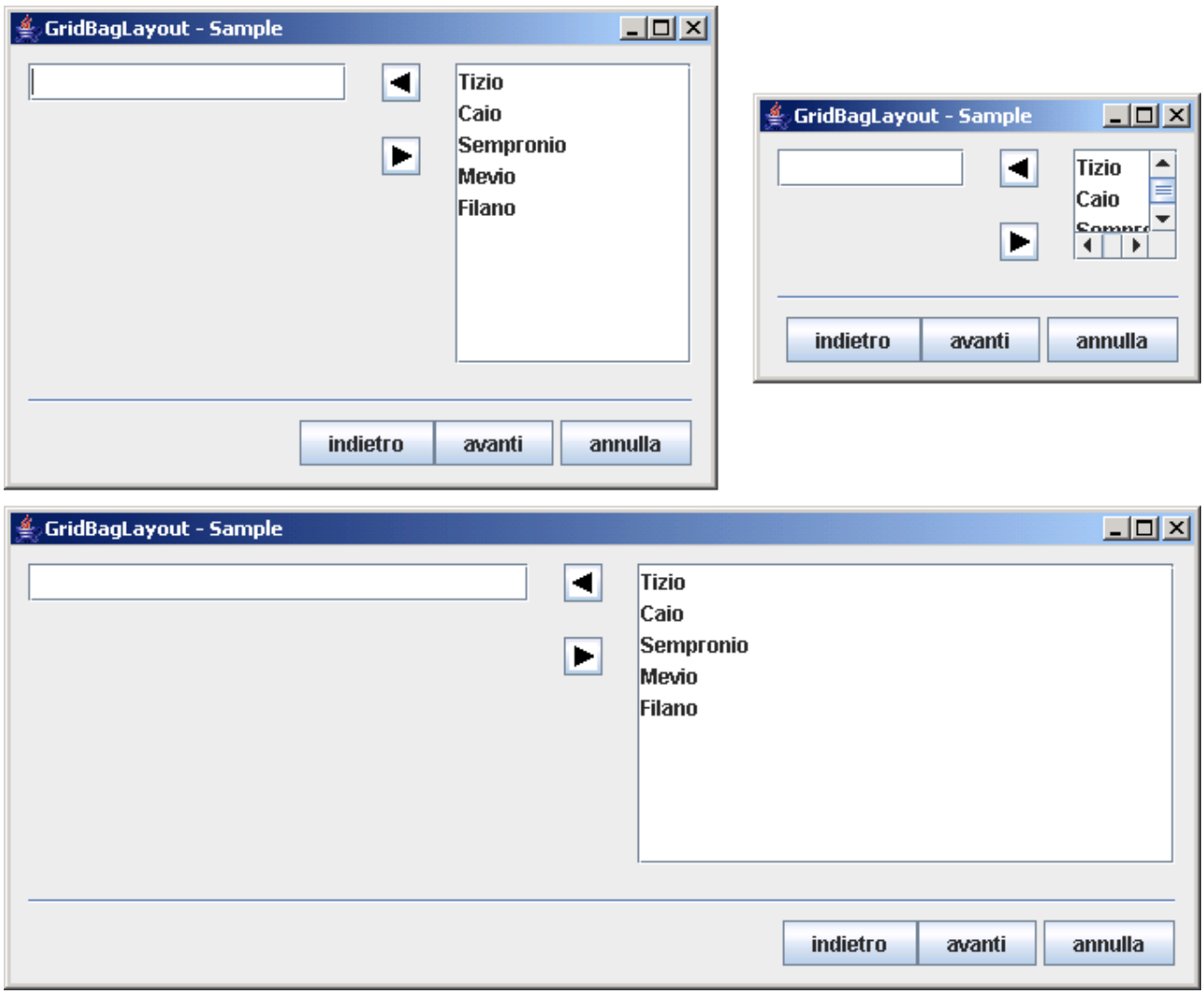


Figura 48 – L'interfaccia definitiva e la sua reazione al mutamento di dimensioni.