VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Ročníkový projekt

2005/2006

Štěpán Moník, Bc.

Poděkování

Chci poděkovat svému vedoucímu Ing., Ph. D. Radkovi Burgetovi za přátelský přístup a pomoc při rešení obtížných problémů.

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením Ing., Ph. D. Radka Burgeta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Štěpán Moník, Bc.

Abstrakt

Projekt se zabývá návrhem a tvorbou grafického uživatelského rozhraní webových aplikací. Smyslem je převést klasickou aplikaci napsanou v jazyce Java do podoby webového formuláře pomocí technologie JSP.

Klíčová slova

Java, JSP, Java servlety, grafické komponenty, webové formuláře, HTML, CSS, JavaScript

Abstract

This project is engaged in concept and creation of web application’s graphics user interface. The reason is to convert a classic application written in Java language to the web form by force of JSP technology.

Key Words

Java, JSP, Java servlets, graphics components, web forms, HTML, CSS, JavaScript

Obsah

1 Úvod

V poslední době je zřetelná tendence převádět klasické aplikace, využívající grafické komponenty operačního systému Windows či v Linuxu prostředí Xwindow, do podoby webových formulářů. Java je moderní objektově orientovaný prodramovací jazyk, pomocí něhož se dají vyvíjet jak klasické GUI aplikace pro libovolné operační systémy, tak aplikace webové. Cílem mého projektu je navrhnout a realizovat Javové třídy pro generování webových formulářů, které se budou používat stejně jako třídy pro vytvátváření běžných aplikací.

V kapitole č. 2 popíši, jak v Javě fungují vestavěné třídy pro tvorbu grafického uživatelského rozhraní. V kapitole č. 3 se zabývám technologiemi, které Java poskytuje pro webové programování na straně serveru. A konečně kapitola č. 4 pojednává o samotné realizaci výše popsaného problému.

2 GUI v Javě

V současnosti existují dva přístupy k tvorbě grafického uživatelského rozhraní v Javě. Buď ho můžete založit na třídách (komponentách) z balíčku *java.awt* nebo použít balíček *javax.swing*, který tzv. Abstract Window Toolkit (AWT) rozšiřuje. Nicméně například obsluhu událostí či správce rozvržení si *swing* bere z *awt*. Při realizaci svého projektu jsem vycházel ze starších AWT komponent (grafické rozhraní se pomocí nich implementuje jednodušeji a intuitivněji).

2.1 Zobrazení jednoduchého okna v Javě

Zobrazení jednoduchého okna pomocí třídy *Frame* z balíčku *java.awt* vypadá následovně:

```
public class SimpleForm extends java.awt.Frame {
    public SimpleForm() {
        add(new java.awt.Button("OK"));
    }
    public static void main(String[] args) {
        SimpleForm sf = new SimpleForm();
        sf.pack();
        sf.setVisible(true);
    }
}
```

A to samé pomocí třídy *JFrame* z *javax.swing*:

```
public class SimpleForm extends javax.swing.JFrame {
    public SimpleForm() {
        add(new javax.swing.JButton("OK"));
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                SimpleForm sf = new SimpleForm();
                sf.pack();
                sf.setVisible(true);
            }
        });
    }
}
```

Metoda *javax.swing.SwingUtilities.invokeLater()* by se teoreticky volat nemusela, avšak doporučuje se to, aby se zamezilo případnému konfliktu, kdyby chtěl formulář při svém zobrazování přistupovat k prostředkům, které zrovna využívá jiné vlákno (tzv. thread-safety problem). Další problém se swingem spočívá v tom, že v případě tzv. top-level kontejnerů (což je v našem případě *JFrame*), se komponenty nepřidávají přímo do kontejneru, nýbrž do jeho 'content pane', takze až do J2SE v1.5 se muselo psát:

```
this.getContentPane().add(new javax.swing.JButton("OK"));
```

2.2 Správci rozvržení

V prostředí Java lze třídu *java.awt.Container* a její potomky používat k zobrazení skupin komponent. Instanci komponenty *JPanel* (případně *Panel*) můžete použít k zobrazení sady souvisejících tlačítek. Můžete rovněž přidat komponenty do podokna obsahu instance třídy *JFrame* (či *Frame*). **Správci rozvržení (layout managers)** jsou třídy používané k řízení velikosti a umístění jednotlivých komponent přidávaných do kontejneru. Ve většině případů jsou správci rozvržení odpovědni rovněž za zjištění rozměrů kontejneru pomocí jeho metod *getMinimumSize()*, *getPreferredSize()* a *getMaximumSize()*. Správci rozvržení jsou velmi důležitým prvkem programování v Javě, protože zjednodušují nejen rozmístění komponent, ale i určení jejich rozměrů, a umožňují tvorbu flexibilních uživatelských rozhraní.

Java poskytuje mnoho různých správců rozvržení. Každý z nich má své výhody i nevýhody. Někteří se používají snadno, ale jejich možnosti jsou omezené. Jiní se zase používají hůře, zato jsou velmi flexibilní. Pokud žádný ze správců rozvržení integrovaných do prostředí Java nevyhovuje vašim potřebám, můžete si snadno vytvořit správce vlastní.

Chcete-li správce rozvržení přidružit ke kontejneru, musíte vytvořit instanci správce a předat ji metodě *setLayout()* poskytované třídou *Container*. Následující příklad je názornou ukázkou toho, jak vytvořit instanci správce typu *BorderLayout* a přiřadit ji k instanci komponenty *JPanel*:

```
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());
```

K přidání komponenty do kontejneru se používá přetížená metoda *add()* definovaná ve třídě *Container*. Kontejner se následně stane **rodičovským kontejnerem (parent container)** komponenty. Komponenta přidaná do kontejneru se označuje jako **dceřiná komponenta (child component)**. Přestože třída *Container* definuje mnoho různých implementací metody *add()*, nejčastěji se používají tyto:

- add(Component comp)
- add(Component comp, Object constraint)

V obou případech je odkaz na dceřinou komponentu odeslán instanci třídy *Container*. Druhá implementace obsahuje rovněž argument **constraint (omezující pravidlo)**. Tento argument poskytuje informace, které umožní správci rozvržení *vymezit* prostor určený pro zobrazení komponenty. To, jaký typ potomka třídy *Object* je v argumentu *constraints* použit, závisí na typu použitého správce rozvržení. Používáte-li instanci správce *GridBagLayout*, musí být argument *constraints* instancí typu *java.awt.GridBagConstraints*. Jiní správci vyžadují textovou hodnotu (typu *String*).

Někteří správci rozvržení omezující pravidla vůbec nepodporují, ale k určení pozice jednotlivých komponent používají pořadí, v němž jsou komponenty do kontejneru přidávány.

2.3 Někteří správci rozvržení integrovaní do prostředí Java

Záměrně se zde nebudu zabývat všemi integrovanými správci rozvržení, protože jsou vyčerpávajícím způsobem popsáni např. na domovských stránkách Javy (http://java.sun.com/). Zmíním se tu pouze o těch, kteří jsou již implementováni v mém projektu.

2.3.1 FlowLayout

Instance správce rozvržení *FlowLayout* uspořádá komponenty v řadách zleva doprava a shora dolů na základě pořadí, v němž byly do kontejneru přidány. Komponentám přitom umožní zabrat tolik prostoru, kolik potřebují. Tento správce rozvržení je užitečný zejména v případech, kdy chcete vytvořit kolekci sousedících komponent, které lze zobrazit v jejich implicitních rozměrech.

Příklad:

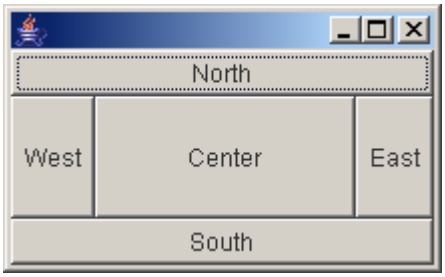
```
public class SimpleForm extends JFrame {

    public SimpleForm() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout(FlowLayout.LEFT));
        add(new JButton("OK"));
        add(new JButton("Cancel"));
        add(new JButton("Long Button"));
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                SimpleForm sf = new SimpleForm();
                sf.pack();
                sf.setVisible(true);
            }
        });
    }
}
```

Obrázek 1: Uspořádání komponent pomocí správce *FlowLayout*

Při tvorbě nové instance *FlowLayout* můžete použít více konstruktorů, které umožňují, vedle nastavení způsobu zarovnávání komponent v kontejneru (LEFT, CENTER, RIGHT), také nastavení velikostí vertikálních a horizontálních mezer mezi komponentami.

2.3.2 BorderLayout

Instance třídy *BorderLayout* dělí kontejner na pět oblastí, do nichž pak lze komponenty přidávat. Pět oblastí odpovídá hornímu, levému, spodnímu a pravému okraji kontejneru plus oblasti uvnitř kontejneru (viz obrázek).



Obrázek 2: Uspořádání komponent pomocí správce *BorderLayout*

Výše zobrazeného okna docílíme následujícím kódem:

```
//..
setLayout(new BorderLayout());
add(new Button("North"), BorderLayout.NORTH);
add(new Button("South"), BorderLayout.SOUTH);
add(new Button("East"), BorderLayout.EAST);
add(new Button("West"), BorderLayout.WEST);
add(new Button("Center"), BorderLayout.CENTER);
//..
```

2.3.3 GridBagLayout

Správce rozvržení *GridBagLayout* je nejflexibilnějším správcem integrovaným do prostředí Java. Jeho hlavní nevýhodou je ovšem jeho složitost a občas i málo intuitivní použití. Na druhou stranu je to jediný správce, který je dostatečně flexibilní k tomu, aby uspořádal komponenty uvnitř kontejneru přesně tak, jak si přejete. I to je jeden z důvodů, proč je tak často používán.

GridBagLayout rozděluje dostupnou zobrazovací plochu kontejneru do mřížky buňek. Komponenty se potom do jednotlivých buňek umisťují. Zároveň ovšem mohou přetékat i do buňek sousedních. Vedle toho se pro každou vkládanou komponentu dají nastavit další

omezující pravidla. To se uskutečňuje pomocí instance třídy *GridBagConstraints*, jejíž použití nyní popíši podrobněji.

Třída *GridBagConstraints* nemá (mimo zděděných) žádné metody, veškerá nastavení se provádějí přímo přiřazením hodnoty k proměnné. Hodnoty jednotlivých datových složek jsou většinou typu *int*. Výjimku tvoří složka *insets*, která je odkazem na třídu *java.awt.Insets*, a složky *weightx* a *weighty*, které jsou typu *double*.

gridx, gridy: Tyto omezující pravidla určují, podle kterého sloupce (řádku) bude komponenta zarovnána. První sloupec (řádek) odpovídá hodnotě 0. Datovým složkám *gridx* a *gridy* můžeme ovšem přiřadit konstantu *GridBagConstraints.RELATIVE*, která určuje, že komponenta může být uvnitř kontejneru umístěna relativně vůči nějaké jiné komponentě. Použijete-li např. konstantu RELATIVE pro datovou složku *gridx* a absolutní hodnotu pro datovou složku *gridy*, bude komponenta umístěna na konec řádku určeného hodnotou datové složky *gridy*.

fill: Velikost komponenty je implicitně nastavena na upřednostňovanou nebo minimální hodnotu, bez ohledu na to, jaká je velikost k ní přiřazené buňky. Omezující pravidlo *fill* určuje, že komponenta by měla být roztažena na celou dostupnou šířku či výšku nebo šířku a výšku (konstanty HORIZONTAL, VERTICAL, BOTH a NONE).

gridwidth, gridheight: Tyto pravidla vymezí počet sloupců (řádků), na nichž je komponenta zobrazena. Implicitně obsahuje hodnotu 1. Kromě určení explicitního počtu sloupců (řádků), na které má být komponenta roztažena, můžeme použít ještě konstantu REMAINDER. Tato konstanta použitá v datové složce *gridwidth* určuje, že by zobrazovací oblast komponenty měla začínat sloupcem určeným hodnotou *gridx* a končit na posledním dostupném sloupci vpravo. Můžeme použít také konstantu RELATIVE, která v tomto případě způsobí roztažení komponenty na všechny zbývající sloupce s *výjimkou* toho posledního.

anchor: Touto datovou složkou určujeme bod, ke kterému chceme ukotvit komponentu ve vybrané zobrazovací oblasti (buňce), je-li komponenta menší než přidělená zobrazovací oblast. Datová složka *anchor* může obsahovat jednu z následujících devíti hodnot: CENTER, NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST nebo NORTHWEST. Implicitní hodnota je CENTER.

insets: Tato vlastnost (omezující pravidlo) je odkazem na instanci třídy *Insets*. Umožňuje definovat určitou výplň kolem komponenty, tj. počet pixelů, které by měly být rezervovány kolem čtyř okrajů (*top*, *left*, *bottom* a *right*) zobrazovací oblasti komponenty. Tato vlastnost se obvykle používá k určení velikosti mezer mezi sousedícími komponentami.

ipadx, ipady: Tyto hodnoty jsou přidávány k upřednostňované nebo minimální velikosti komponenty. Slouží k určení šířky komponenty a předpona „i“ odkazuje na skutečnost, že hodnota výplně je přidána k „interní“ (upřednostňované nebo minimální) šířce (či výšce) komponenty, nikoli ke skutečné (zobrazené) šířce. Pokud má komponenta nastavenu upřednostňovanou šířku například na 40 pixelů a vy prostřednictvím datové složky *ipadx* určíte hodnotu výplně na 10 pixelů, bude se šířka (v případě zobrazení v upřednostňovaných rozměrech) zobrazené komponenty rovnat 50 pixelům.

weightx, weighty: Tyto hodnoty se používají k úpravě šířky sloupců či výšky řádek. Je-li např. šířka kontejneru větší nebo menší než šířka potřebná k zobrazení komponent v jejich upřednostňovaných nebo minimálních rozměrech. Mají-li všechny komponenty v mřížce definovánu vlastnost *weightx* jako 0.0 (implicitní nastavení), je všechen dodatečný prostor rozdělen mezi pravou a levou výplň kontejneru. Teorie ohledně používání datových složek *weightx* a *weighty* je poměrně rozsáhlá a přesahuje rámec této kapitoly, proto případné zájemce odkazuji na http://java.sun.com/.

Popisu *GridBagLayoutu* jsem záměrně věnoval více prostoru, protože jeho implementaci do webových aplikací považuji za stěžejní část svého projektu. Příklad použití tohoto správce naleznete v kapitole 4.3.

3 Webové programování na straně serveru

Dnes existuje spousta technologií pro generování dynamických webových stránek. Z počátku se používala pouze technologie CGI. Ta definuje ověřené mechanismy pro integraci **externích bránových programů**, tedy programů, které vytvářejí bránu mezi informacemi jinými než HTML a webovými servery. Webový server a CGI program komunikují na úrovni procesu operačního systému. V operačním systému je proces základní spustitelnou jednotkou programů. Procesy mohou spouštět jiné procesy a během toho předávat informace. Webový server tedy může spustit CGI procesy a předávat informace o požadavku.

CGI je sice mocný nástroj, ale v současné době se již moc nepoužívá. Při zpracování požadavků klientů dochází ke zpouštění programů, které nadměrně zatěžují prostředky serveru včetně CPU a ztěžují generování stránek s odezvou. V tom nejhorším možném případě by každý požadavek klienta vyžadoval spuštění jednoho nebo více procesů. Protože je v moderním internetu CGI program zpravidla implementován jazykem *Perl* nebo jiným skriptovacím jazykem, zvyšuje se navíc neefektivnost také tím, že skriptovací jazyky obvykle ke zpracování systémových požadavků spouštějí další procesy. Uvedené vlastnosti CGI procesů omezují použitelnost webových aplikací, které jsou na této technologii založeny.

Další nevýhodou je to, že CGI programy nepřetrvávají během více požadavků a musejí tedy o klientovi uchovávat stavové informace v systému souborů nebo v databázi. Obecně potřebujeme těsnější vztah mezi našimi doplňky serveru a webovým serverem, aby bylo možné zmenšit dobu odezvy a zlepšit správu stavu klienta. V současné době existuje hodně technologií, které toto zajišťují. Mezi nejznámější a nejoblíbenější patří PHP, ASP, ASP.NET a především Java servlety a s nimi související stránky JSP.

3.1 Java servlety

Java servlety představují rozsáhlou, na platformě nezávislou technologii pro rozšíření funkcí webových serverů. Ryzí webové nebo aplikační Java servery mohou servlet kontejner implementovat jako vlákno uvnitř hlavního procesu. Oblíbenější webové servery jako *Apache* nebo *IIS* vyžadují odlišné technologie v závislosti na jejich vlastních rozhraních. Architektura servletů zahrnuje mimo jiné také zásuvný modul webového serveru, který přesměruje požadavky servletu do odděleného procesu Javy, jež je implementací servlet kontejneru.

Takovéto použití zásuvného modulu webového serveru představuje kompromis mezi pevným a volným provázáním, protože modul webového serveru poskytuje pevně provázaný most, který předává informace mezi serverem a volně provázaným servletovým procesorem.

3.2 JSP

JSP doplňuje architekturu Java servletů, protože poskytuje JSP kontejner, který zajišťuje správu JSP stránek a jejich překládání na servlety.

3.3 Aplikační servery Java 2 Enterprise Edition (J2EE)

Specifikace J2EE zahrnuje Java servlety i JSP stránky. Aplikační servery J2EE zpravidla poskytují platformu Javy pro webové a další služby, například *Enterprise JavaBeans* a *Java Messaging Service*.

3.4 JSP kontejner

J2EE definuje několik kontejnerů včetně JSP kontejneru, servlet kontejneru a kontejneru *Enterprise JavaBeans*. Kontejner je v žargonu objektově orientovaného programování třída nebo komponenta, která uspořádává ostatní třídynebo komponenty. Specifikace JSP tento původní význam rozšiřuje. Kontejnery J2EE poskytují úplné aplikační prostředí, ve kterém řídí životní cyklus komponent a poskytují jim různé služby. Navíc také ovlivňují vzájemné vazby mezi komponentami a větším aplikačním prostředím.

Kontejnery J2EE nemohou pracovat správně, pokud vývojář nenapíše softwarové komponenty tak, aby se řídily programovacími pravidly, které kontejner definuje. Vývojář musí tato pravidla respektovat, protože kontejner je v různých fázích komponenty předpokládá. Specifikace JSP zvýrazňuje význam těchto pravidel tím, že je nazývá **dohodami**. Kontejnery splnění dohod zajišťují tím, že po aplikacích požadují, aby měly implementována přesná Java rozhraní.

Každý kontejner J2EE poskytuje služby těm komponentám, za které má zodpovědnost. JSP kontejner překládá JSP stránky do kódu Java servletů a výsledek poté překládá a načítá do servlet kontejnerů. Dále také koordinuje vzájemný vztah mezi servlet

kontejnerem a přeloženými JSP stránkami. Servlet kontejner poskytuje aplikační prostředí pro Java servlety.

3.4.1 Fáze JSP

Požaduje-li prohlížeč poprvé určitou JSP stránku, stane se následující:

1. Interpretuej se JSP stránka.
2. Vygeneruje se Java servlet
3. Servlet se pomocí standardního překladače, který je dodán s JSP kontejnerem, převede do bajtového kódu Java.
4. Servlet je načten do *virtuálního stroje Java* (JVM) servlet kontejneru.
5. U servletu je vyvolána **služební** metoda.

Požaduje-li prohlížeč následně stejnou JSP stránku a nebyla-li stránka od posledního volání změněna, musí JSP kontejner provést pouze krok 5. Pokud se stránka změnila, je nutné před odesláním odpovědi znovu zopakovat všech pět kroků. Zpracováním kroků 1 až 4 lze vysvětlit, proč mají JSP stránkypomalejší odezvu při prvním zobrazení v prohlížeči.

4 Transformace GUI na webové rozhraní

Vzhledem k povaze zadání jsem se rozhodl, že bude nejlepší, použít technologii Java servletů. Základní třída *webawt.Frame* (ekvivalent třídy z balíčku *java.awt*) je tedy potomkem třídy *javax.servlet.http.HttpServlet* a vývojové prostředí *NetBeans* ji implementuje (předgeneruje) následujícím způsobem:

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Frame extends HttpServlet {
    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        /* TODO output your page here
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet Frame</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet Frame at " + request.getContextPath () + "</h1>");
        out.println("</body>");
        out.println("</html>");
         */
        out.close();
    }
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
}
```

Obsluha žádostí GET a POST je zde sloučena do jediné metody *processRequest()*. V té tedy bývá umístěna stěžejní část kódu servletu. Zapoznámkovaný kód demonstruje její jednoduché použití.

Rozšířením třídy *javax.servlet.http.HttpServlet* o metody a vlastnosti, které zabezpečují přidávání a následné rozmístění komponent (resp. vygenerování patřičného HTML kódu), jsem tedy vytvořil ekvivalent třídy *java.awt.Frame*. Vedle toho bylo potřeba vytvořit třídy správců rozvržení a samotných komponent.

4.1 Správci rozvržení

V balíčku *java.awt* existují dvě rozhraní, která mohou správci rozvržení implementovat. Jsou to *LayoutManager* a *LayoutManager2*. Já mám rozhraní pouze jedno, jmenuje se jednoduše *Layout*. Dvě klíčové metody tohoto rozhraní jsou *add()* a *paintComponents()*. Pomocí *add()* přidává třída *Frame* do správce jednotlivé komponenty, pokud použije překrytou verzi metody, může určit i tzv. omezení (constraints), kterými se má správce při zobrazování komponenty řídit, metoda *paintComponents()* potom musí zajistit vygenerování patřičného HTML kódu a správně do něj komponenty umístit. Správci rozvržení, které se mi zatím podařilo implementovat jsou: *FlowLayout*, *BorderLayout* a *GridBagLayout*.

4.2 Komponenty

Rodičovská třída všech komponent se jmenuje *Component*, stejně jako v balíčku *java.awt*. Následující schéma porovnává předky tlačítek z *java.awt* a *webawt*.

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Button

java.lang.Object
  └ webawt.Component
      └ webawt.Button
```

Třída *Component* má spoustu metod, kterými se nastavují vlastnosti komponenty. Také obsahuje jednu abstraktní metodu *paint()*, kterou musejí potomci implementovat a vygenerovat v ní patřičný HTML kód.

Veřejné metody třídy *webawt.Component*:

```
public GridBagConstraints getConstraints()
public void setConstraints(GridBagConstraints gbc)
public void setText(String text)
public String getText()
public void setSize(int width, int height)
public int getWidth()
public int getHeight()
public void setFont(java.awt.Font newfont)
public java.awt.Font getFont()
public void setForeground(java.awt.Color color)
public java.awt.Color getForeground()
public void setBackground(java.awt.Color color)
public java.awt.Color getBackground()
public abstract String paint(String style, boolean usepreferredwidth,
boolean usepreferredheight)
```

Z výše uvedeného vyplývá, že rozšiřování balíčku *webawt* o další komponenty je velice snadné. Spočívá v podstatě pouze z implementace patřičného HTML kódu do metody *paint()*. Ta má tři parametry. V prvním správce rozvržení předává metodě řetězec obsahující libovolný počet CSS stylů, které si přeje, aby komponenta implementovala, další dva parametry určují, zda má komponenta použít své preferované rozměry (pokud ne, jsou v parametru style obsaženy patřičné CSS styly, které rozměr/rozměry nastaví).

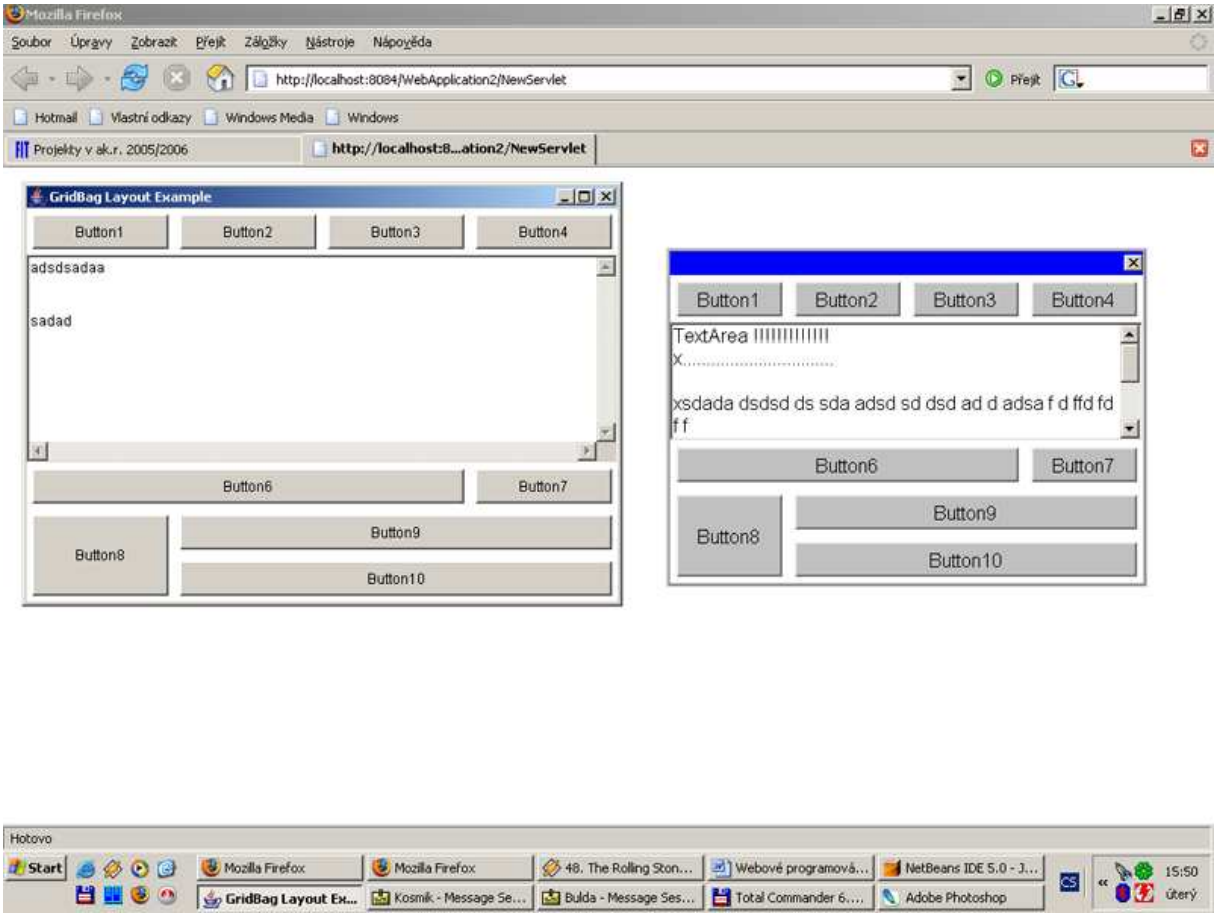
4.3 Stejný kód Javy zobrazí v prohlížeči to samé, co v okně

Výsledek samozřejmě ještě není ve všech případech stoprocentní, ale například u velmi komplikovaného *GridBagLayoutu* už zbývá jen dodělat JavaScript pro měnění velikosti okna. Následuje ukázka použití správce rozložení *GridBagLayout*:

```
protected void makebutton(String name,
                              GridBagLayout gridbag,
                              GridBagConstraints c) {
         Component button;
         if (name=="TextArea")
            button = new TextArea(name);
         else
            button = new Button(name);
         gridbag.setConstraints(button, c);
         add(button);
}

public NewServlet() {
         GridBagLayout gridbag = new GridBagLayout();
         GridBagConstraints c = new GridBagConstraints();
         setPosition(300,100); //Umistuje okno v prohlizeci, neni v awt.
         setLayout(gridbag);
         c.fill = GridBagConstraints.BOTH;
         c.insets = new Insets(5,5,5,5);
         makebutton("Button1", gridbag, c);
         makebutton("Button2", gridbag, c);
         makebutton("Button3", gridbag, c);
         c.gridwidth = GridBagConstraints.REMAINDER; //end row
         makebutton("Button4", gridbag, c);

         c.insets = new Insets(0,0,0,0);
         makebutton("TextArea", gridbag, c); //another row

         c.insets = new Insets(5,5,5,5);
         c.gridwidth = GridBagConstraints.RELATIVE; //next-to-last in row
         makebutton("Button6", gridbag, c);

         c.gridwidth = GridBagConstraints.REMAINDER; //end row
         makebutton("Button7", gridbag, c);
```

```
        c.gridwidth = 1;                //reset to the default
        c.gridheight = 2;
        makebutton("Button8", gridbag, c);

        c.gridwidth = GridBagConstraints.REMAINDER; //end row
        c.gridheight = 1;               //reset to the default
        makebutton("Button9", gridbag, c);
        makebutton("Button10", gridbag, c);
}

protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        paint(request,response); //Nutno volat pro vykresleni formulare
}
```

Výše uvedený kód zobrazí, v závislosti na tom, zda ho použijete v konstruktoru formuláře *java.awt.Form* nebo *webawt.Form*, buď klasické okno s komponentami nebo webový formulář, jak je vidět na obrázku č.3.



Obrázek 3: Porovnání skutečného okna s komponentami rozloženými pomocí správce rozvržení *GridBagLayout* s formulářem z *webawt*, ve kterém je použit stejný správce.

5 Plány do budoucna

Vzhledem k velkému rozsahu projektu mě čeká ještě hodně práce, než se mi podaří dosáhnout stejné funkčnosti, jakou mají integrované Javové třídy pro tvorbu grafického uživatelského rozhraní. V nejbližší budoucnosti plánuji napsat JavaScript, který bude uživateli umožňovat dynamicky měnit velikost okna zobrazeného v prohlížeči. Dále bude potřeba ještě trochu doladit některé správce rozvržení a přidat kromě formuláře (okna) nějaký další kontejner na komponenty (panel). A v neposlední řadě je tu náročný úkol, zařídit, aby všechno správně fungovalo i v Internet Exploreru.

6 Závěr

Na základě požadavků jsem navrhl způsob transformace klasického GUI na webové rozhraní pro technologii JSP a v podobě balíčku *webawt* jsem implementoval ekvivalenty některých tříd z *java.awt*. Projekt je použitelný především při převodu klasických GUI aplikací do webové podoby. Dá se snadno rozšiřovat, proto v něm lze pokračovat např. v rámci diplomové práce. Téma jsem si vybral, protože mám rád moderní objektově orientované programování, jehož symbolem se pro mě stal jazyk Java. Dále je na projektu zajímavé spojení problematiky programování webových scriptů na straně serveru s klasickými GUI aplikacemi.

7 Literatura

[1] Brett Spell: Java Programujeme profesionálně
Computer Press 2002
ISBN 80-7226-667-5

[2] Gary Bollinger, Bharathi Natarajan:
JSP – Java Server Pages
Podrobný průvodce začínajícího tvůrce
Grada Publishing a.s. 2003
ISBN 80-247-0340-8

[3] Pavel Herout: Java – grafické uživatelské prostředí a čeština
Kopp 2001
ISBN 80-7232-150-1

[4] Pavel Herout: Java - bohatství knihoven
Kopp 2003
ISBN 80-7232-209-5

[5] Domovské stránky jazyka Java (The Source for Java Developers)
http://java.sun.com

8 Přílohy

CD-ROM se zdrojovými kódy a přeloženými *.class* soubory.