KDE4 namizje – plazma in javascript plazmoidi

Milan Pulko, Borko Bošković, Janez Brest

Inštitut za računalništvo
Fakulteta za elektrotehniko, računalništvo in informatiko
Univerza v Mariboru
E-pošta: milan@mladika.si

Plasma – KDE4 desktop and javascript plasmoids

Plasma is the name for desktop shell of KDE4. Instead of a background image with icons, the desktop is now host to desktop widgets also known as plasmoids.
Plasmoids may be currently written using various programming languages (C++, JavaScript, Python, Ruby). In this paper we present how to develop plasmoids in JavaScript.

1 Uvod

Namizja operacijskih sistemov kot so, Windows Vista, Windows 7 in Mac X OS, so med drugimi novostmi vpeljala tudi tako imenovane Desktop Gadgete oziroma Dashboard Widgete. To so majhni programčki, ki so stalno prisotni na namizju in opravljajo različne naloge, kot so na primer prikaz vremenske napovedi, ura, pregled novic, beležka ipd.

Podobne gradnike so vpeljali tudi v prosto dostopno namizno okolje KDE4. To po novem nosi ime Plazma, gradniki pa se imenujejo plazmoidi. Za razliko od okolij Windows in Mac kjer ti predstavljajo zgolj pomožne programčke, pa je okolje plazma v celoti sestavljeno iz njih.

2 Plazma in plazmoidi

2.1 Plazma

Cilj tvorcev plazme je bil ustvariti čimbolj prilagodljivo grafično namizje. Tako je nastala ideja, da je celotno grafično namizje sestavljeno iz množice preprostih gradnikov. Vsak gradnik predstavlja majhno aplikacijo (programček), ki opravlja različno funkcijo. Ta je lahko prikaz ikon za hitri zagon, prikaz menija Start, prikaz ozadja namizja, prikaz stanja akumulatorja pri prenosniku, prikaz ikon itn.

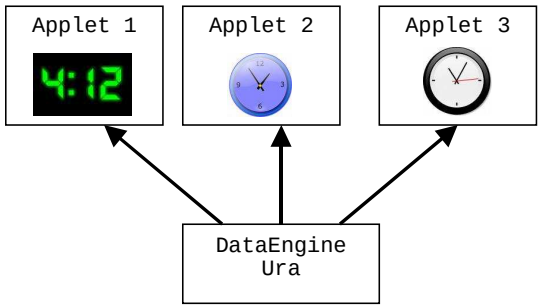
Tak pristop omogoča, da z menjavo različnih gradnikov enostavno spreminjamo namizje. Ne le njegov videz ampak tudi uporabo in vsebino. Z različnimi gradniki je mogoče sestaviti preprosta grafična namizja za manj zmogljive naprave (netbooki, mobilni telefoni, ...) ali bolj bogata namizja za zmogljivejše osebne računalnike.

Plazma je uvedla tudi podporo za vektorsko grafiko SVG, ki omogoča od resolucije neodvisen prikaz grafičnih elementov ter gladko zumiranje in skaliranje.

2.2 Plazmoidi

Posamezne plazma aplikacije imenujemo plazmoidi. Ti so običajno ločeni na podatkovne komponente (*DataEngine*) in komponente za prikaz (*Applet*). Takšen način omogoča, da določene podatke prikažemo na različne načine.

Vzemimo za primer aplikacijo za prikaz časa. Podatke o času generira podatkovna komponenta, za prikaz pa skrbijo različni prikazovalniki.



Slika 1. Primer plazmoida za prikaz časa – podatkovna komponenta in različni prikazovalniki

KDE4 plazmoidi temeljijo na knjižnici Qt4. Razvijamo pa jih lahko v programskih jezikih C++, javascript, python in ruby.

Čeprav se pri razvoju plazmoidov najpogosteje uporablja jezik C++, pa je v tem članku prikazan razvoj z uporabo skriptnega jezika javascript.

2.3 Javascript plazmoidi

Javascript plazmoidi se izvajajo s pomočjo Qt-jevega sistema Qtscript. Ta vsebuje ECMA skriptni interpreter iz odprtokodnega projekta WebKit, vendar z določenimi spremembami. Tako ne vsebuje nekaterih elementov, ki jih poznamo iz spletnih brskalnikov (npr. DOM), po drugi strani pa vsebuje Qt-jev mehanizem signalov in rež.

Vsak plazmoid sestavljajo javascript koda in meta podatki. Dodatno pa lahko še slike, katalogi sporočil (internacionalizacija), nastavitvene datoteke itn. Vsi našteti elementi se morajo nahajati v ustreznih imenikih.

primer/
- metadata.desktop
- **contents/**
  - glavni.js
  - ...
  - **images/**
    - ozadje.svg
    - ...
  - **config/**
    - main.xml
    - ...
  - **locale/**
    - sl/LC_MESSAGES/primer.mo
    - it/LC_MESSAGES/primer.mo
    - ...
  - **ui/**
    - config.ui

Slika 2. Imeniška struktura plazmoida z imenom primer

Poglejmo sedaj primer preprostega plazmoida, ki ob kliku na gumb izpiše "Pozdravljen svet".

Slika 3. Plazmoid "Pozdravljen svet"

Najprej ustvarimo imenik z imenom *pozdravljen* in podimenik *contents*. V glavnem imeniku ustvarimo datoteko z imenom *metadata.desktop* vse ostale datoteke pa shranjujemo v podimenik *contents*.

Datoteka *metadata.desktop* vsebuje podatke, ki jih Plazma potrebuje za zagon plazmoida (*ime skriptne datoteke, začetna velikost, ...*), vrsto plazmoida (*Applet, DataEngine, ...*) ter podatke o avtorju.

```
[Desktop Entry]
Encoding=UTF-8
Name=Pozdravljen svet
Comment=Primer preprostega plazmoida
Type=Service
X-KDE-ServiceTypes=Plasma/Applet
X-Plasma-API=javascript
X-Plasma-MainScript=glavni.js
X-Plasma-DefaultSize=200,150
X-KDE-PluginInfo-Author=Milan
X-KDE-PluginInfo-Name=pozdravljen
X-KDE-PluginInfo-Version=0.1
X-KDE-PluginInfo-Category=Examples
X-KDE-PluginInfo-License=GPL
X-KDE-PluginInfo-EnabledByDefault=true
```

Program 1a. Meta datoteka *metadata.desktop*

Naslednji korak je programska koda. To shranimo v datoteko z imenom, ki smo ga določili v *metadata.desktop* v podimenik *contents*.

```
lay = new LinearLayout(plasmoid);
lay.orientation = QtVertical;

label1 = new Label();
label1.text = "Klikni na gumb";
label1.styleSheet = "font: bold 14px; color: blue;";
label1.alignment = QtAlignLeft | QtAlignVCenter;

gumb1 = new PushButton;
gumb1.text = "OK";
gumb1.clicked.connect(Klik);

lay.addItem(label1);
lay.addItem(gumb1);

function Klik()
{
    label1.text = "Pozdravljen svet";
}
```

Program 1b. Skriptna datoteka *glavni.js*

V programu definiramo grafične elemente izpis *Label*, gumb *PushButton* ter razporeditev *LinearLayout*, ki poskrbi za samodejno razporeditev vseh grafičnih elementov.

S *clicked.connect* povežemo signal, ki se sproži ob kliku na gumb z režo *Klik* (Qt mehanizem signalov in rež).

Pravilnost delovanja plazmoida preizkusimo z ukazom:
plasmoidviewer /pot/do/pozdravljen

Na koncu plazmoid pripravimo še za distribucijo:
- vsebino celotnega imenika *pozdravljen* stisnemo v datoteko *pozdravljen.zip*
- dobljeno datoteko preimenujemo v *pozdravljen.plasmoid*

Plazmoid namestimo z ukazom:
plasmapkg -i pozdravljen.plasmoid

odstranimo pa s:
plasmapkg -r pozdravljen.plasmoid

Plazma aplikacije običajno poleg komponent za prikaz (*Applet*) vsebujejo še podatkovne komponente (*DataEngine*). Naloga teh je zajem in obdelava podatkov, ter pošiljanje rezultatov prikazovalnikom.

Plazma vsebuje mnogo že narejenih podatkovnih komponent, ki oskrbujejo prikazovalnike z raznimi podatki. Tako nam v mnogih primerih ni potrebno razvijati lastnih podatkovnih komponent, ampak ustvarimo le svoj prikazovalnik.

Vsaka podatkovna komponenta lahko posreduje podatke več prikazovalnikom, vsak prikazovalnik pa

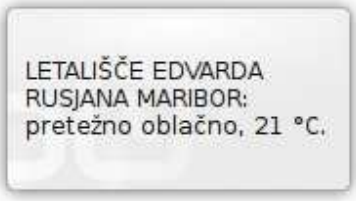
lahko sprejema podatke od več različnih podatkovnih komponent.

Seznam vseh podatkovnih komponent, ki so na voljo ter njihovo strukturo si lahko ogledamo s pomočjo orodja *plasmaengineexplorer*.

V javascriptu ustvarimo povezavo med določeno podatkovno komponento in prikazovalnikom z ukazom *connectSource*, pri čemer določimo ime podatkovne komponente ter interval osveževanja podatkov. Podatkovna komponenta nato v določenih intervalih samodejno pošilja podatke prikazovalniku. Ob vsakem sprejemu podatkov se v prikazovalniku izvrši funkcija *plasmoid.dataUpdated*.

V naslednjem primeru je prikazan plazmoid "Vreme", ki prikazuje trenutne vremenske razmere na mariborskem letališču. Plazmoid sprejema podatke iz podatkovne komponente z imenom RSS, ki je standardni del Plazme. Ta pa dobiva podatke iz spletne strani *http://www.meteo.si* v zapisu RSS.

Podatki o vremenu se avtomatsko osvežijo in posredujejo prikazovalniku na vsakih 10 sekund.



Slika 4. Plazmoid "Vreme"

```
plasmoid.drawStandardBackground = true;

lay = new LinearLayout(plasmoid);
lay.orientation = QtVertical;

label1 = new Label();
label1.text = "---";

lay.addItem(label1);

plasmoid.dataUpdated = function(source, data)
{
    s1 = data["items"];
    s2 = s1["0"];
    label1.text = s2["title"];
}

eng = dataEngine("rss");

eng.connectSource("http://www.meteo.si/uploads/probase/www/observ/surface/text/sl/observation_MARIBOR_SLIVNICA_latest.rss",
plasmoid, 10000);
```

Program 2. Skript za plazmoid "Vreme"

Na začetku programa določimo obliko ozadja *drawStandardBackground* ter definiramo grafična elementa za izpis *Label* in razporeditev *LinearLayout*, ki samodejno razporedi vse grafične elemente.

Funkcija *plasmoid.dataUpdated* se izvrši vsakokrat, ko podatkovna komponenta RSS pošlje podatke. Slednja bere podatke iz spletne strani v RSS v formatu xml, ter jih nato pošlje prikazovalniku kot podatek tipa *QVariant*.

Ustrezno podatkovno komponento izberemo z ukazom *dataEngine*, povezavo do nje pa ustvarimo s *connectSource*. Pri tem tudi določimo osvežitveni interval, ki je v našem primeru 10000 ms. Funkcijo *plasmoid.dataUpdated* moramo definirati pred uporabo ukazov *dataEngine* in *connectSource*.

Meta datoteka *metadata.desktop* je podobna kot v prvem primeru.

2.4 Podpora internacionalizaciji

Internacionalizacija (*internationalization – "i", 18 vmesnik črk in "n" na koncu*) je postopek, ki omogoča prevajanje aplikacije v druge naravne jezike. To dosežemo na način, da sporočila v aplikaciji ne zapišemo neposredno, ampak uporabimo posebne identifikatorje. Vsak identifikator ima prirejeno določeno sporočilo. Iz vseh identifikatorjev in njim prirejenih sporočil zgradimo poseben katalog. Aplikacija nato na mestu identifikatorja uporabi ustrezno sporočilo iz tega kataloga. S takim pristopom je doseženo, da prevajalci prevajajo samo kataloge, same programske kode oziroma programiranja pa jim ni potrebno poznati.

Vse nize, ki jih naša aplikacija uporablja in za katere želimo, da so prevedljivi v druge jezike, zapišemo (v angleščini) kot argumente metode *i18n*:
label1.text = i18n("Hello world");

V primeru daljših sporočil, ki vsebujejo tudi izpise spremenljivk, lahko uporabimo naslednjo obliko:
label1.text = i18n("Question %1 of %2", st_vprasanja, vseh_vprasanj);

Pri prevodih posameznih sporočil se pogosto pojavi težava, da prevajalec ne more natančno ugotoviti, kateri prevod bi bil najbolj smiseln. Na primer angleško besedo "Next" lahko prevedemo kot "Naprej", "Naslednji", "Naslednja" itn. V takih primerih uporabimo namesto metode *i18n* metodo *i18nc*. Ta kot prvi parameter sprejme dodatni opis sporočila, ki ga prevajamo in s tem olajša delo prevajalcem:
gumb1.text = i18nc("Next window in Wizard", "Next");

Pri prevajanju se srečamo tudi s problemom množine. V teh primerih uporabimo metodo *i18np* (oziroma *i18ncp)*:
msg = i18np("Delete %1 File", "Delete %1 Files", st_datotek);

V postopku prevajanja najprej izločimo sporočila iz izvornih programskih datotek. To storimo z orodjem *xgettext*. Na ta način dobimo besedilno datoteko s končnico *po*. V to nato s poljubnim urejevalnikom besedil ali z orodjem *Localize* vnesemo prevod.

Na koncu, zaradi hitrejšega dostopa aplikacije do prevedenih sporočil, tekstovno *.po* datoteko z orodjem *msgfmt* prevedemo v binarno obliko (datoteka *.mo*).

Datoteke s prevodi shranjujemo v podimeniku *contents* (slika 2):
locale/xx/LC_MESSAGES/ime_programa.mo
pri čemer *xx* predstavlja dvoznakovno kodo jezika ISO-639 (npr. en – angleščina, sl – slovenščina, it – italijanščina itn.).

2.5 Lastnosti uporabe javascripta

Razvoj plazmoidov v javascriptu je hitrejši in lažji od razvoja v C++. Napake v programski kodi niso tako usodne kot tiste v C++, ki lahko pripeljejo celo do rušenja celotnega namizja (vsi plazmoidi namreč tečejo v enem procesu). Mnogo enostavnejša pa je tudi distribucija in namestitev, saj odpade vmesni korak prevajanja.

V prid javascripta govori tudi dejstvo, da so programčki na sistemih Windows oz. Mac prav tako pisani v istem jeziku ter je zato prevajanje iz enega sistema v drugi za spoznanje lažje, čeprav določene razlike med skriptami obstajajo.

Slaba stran javascripta se kaže predvsem v tem, da zaradi varnostnih razlogov nimamo dostopa do vseh KDE komponent. Poleg tega je tudi hitrost izvajanja manjša. Primerjava hitrosti med javascriptom in C++ je predstavljena v tabeli 1.

Tabela 1. Primerjava hitrosti med javascriptom in C++

Test	**JS [ms]**	**C++ [ms]**
shrani nastavitve *1000-krat*	20368	164
izračunaj +, -, *, / *1000000-krat*	93	6
izračunaj Math.sin *1000000-krat*	387	192
if stavek *1000000-krat*	28	2
for stavek *1000000-krat*	22	0 (optimiz.)
združi dva niza *1000000-krat*	1053	595
sortiranje *1000000-krat*	14544	819
beri nastavitve *1000-krat*	102	148
Povprečje	*4574*	*240*

Testni PC: Via Nano 1,6 GHz, 1 GB Ram

3 Sklep

V članku smo na kratko predstavili razvoj KDE4 plazmoidov z uporabo javascripta.

Trenutno je večina KDE4 plazmoidov napisanih v programskem jeziku C++. S tem člankom pa smo želeli prikazati, da je javascript lahko v nekaterih primerih boljša izbira.

Literatura

[1] Creating Plasmoids, http://www.linux-magazine.com/w3/issue/114/036-040_plasma.pdf

[2] Re-invent Your Desktop with Plasma!, http://www.linuxjournal.com/article/10638

[3] Plasma/JavaScript/GettingStarted, http://techbase.kde.org/Development/Tutorials/Plasma/JavaScript/GettingStarted

[4] Plasma/JavaScript/DataEngine, http://techbase.kde.org/Development/Tutorials/Plasma/JavaScript/DataEngine

[5] JavaScript/API, http://techbase.kde.org/Development/Tutorials/Plasma/JavaScript/API

[6] Plasma/FAQ, http://userbase.kde.org/Plasma/FAQ

[7] i18n, http://techbase.kde.org/Development/Tutorials/Localization/i18n

[8] JavaScript/SystemMonitor, http://techbase.kde.org/Development/Tutorials/Plasma/JavaScript/SystemMonitor

[9] JavaScript/CheatSheet, http://techbase.kde.org/Development/Tutorials/Plasma/JavaScript/CheatSheet