UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Nina Krmac

IZBIRA ORODJA IN KNJIŽNIC ZA IMPLEMENTACIJO POROČIL V POSLOVNI APLIKACIJI

Diplomska naloga
na univerzitetnem študiju

Mentor: doc. dr. Marko Bajec

Ljubljana 2008

Zahvala

Zahvalila bi se staršem, ki so me vsa ta leta spodbujali in mi stali ob strani ter mi omogočili, da sem lahko širila svoje znanje na vseh področjih, za katere sem izkazala zanimanje ter me naučili, da je znanje pomembna vrednota. Poleg tega se moram zahvaliti še sodelavcem iz fakultete, ki so mi pomagali z namigi za to delo in od katerih sem se v zadnjih dveh letih ogromno naučila.

Kazalo

Povzetek ... 1
Abstract ... 3
1. Uvod ... 5
2. Glavni del ... 7
2.1. Identifikacija problema ... 9
2.2. Identifikacija kriterijev ... 10
2.2.1. Zakaj spletna aplikacija ... 10
2.2.2. Web 2.0 ... 11
2.2.3. Uporabniški vmesniki ... 11
2.2.3.1. Čas ... 12
2.2.3.2. Podatki ... 12
2.3. Razpoložljiva orodja ... 14
2.4. Spisek kriterijev ... 15
2.4.1. Struktura kriterijev ... 15
2.4.2. Merske lestvice ... 17
2.4.3. Definicija odločitvenih pravil ... 18
2.5. Opis variant ... 19
2.5.1. Splošni podatki o orodjih ... 19
2.5.1.1. Infragistics NetAdvantage for JSF ... 19
2.5.1.2. Adobe Flex ... 20
2.5.1.3. Google Web Toolkit ... 21
2.5.2. Izdelava prototipov ... 22
2.5.2.1. Opis prototipa ... 23
2.5.2.2. Izdelava prototipa v Infragistics NetAdvantage for JSF 2008 ... 26
2.5.2.3. Izdelava prototipa v Google Web Toolkitu ... 33
2.5.2.4. Izdelava prototipa v Flexu ... 38
2.6. Vrednotenje in analiza variant ... 44
2.6.1. Vrednotenje ... 44
2.6.2. Kaj-če analiza ... 51
3. Sklepne ugotovitve ... 53
Literatura ... 55
Izjava o avtorstvu ... 57

Seznam uporabljenih kratic in simbolov

AJAX- Asynchronous JavaScript and XML: asinhroni JavaScript in XML
BI – Business Intelligence: poslovno obveščanje
BIRT - Business Intelligence and Reporting Tools: orodje za poslovno obveščanje in izdelavo poročil
CSS – Cascading Style Sheet: prekrivni slogi
CSV- Comma Separated Values: format za besedilo, ki vsebuje z vejico ločene vrednosti
EJB - Enterprise Java Beans: Java zrna za poslovne aplikacije
GWT – Google Web Toolkit: ogrodje za izdelavo AJAX aplikacij
HTML - HyperText Markup Language: označevalni jezik za oblikovanje dokumentov
J2EE - Java 2 Enterprise Edition: platforma za izdelavo poslovnih aplikacij
JSF - JavaServer Faces: specifikacija za gradnjo spletnih uporabniških vmesnikov
JSNI – JavaScript Native Interface: vmesnik za delo z domorodnimi JavaScript funkcijami
JSON - JavaScript Object Notation: format za izmenjavo podatkov med sistemi
JSP - JavaServer Pages: tehnologija za dinamično generiranje spletnih strani
PDF - Portable Document Format: format večpredstavnih datotek
PHP - PHP Hypertext Preprocessor: programski jezik za izdelavo dinamično generiranih spletnih strani
RAD – Rational Application Developer: IBM-ovo razvojno okolje
RPC – Remote Procedure Call: klic oddaljenega postopka
RIA – Rich Internet Application: obogatena spletna aplikacija
RSS - Really Simple Syndication: protokol, ki vzpostavlja okolje za objavo in distribucijo spletnih vsebin v XML-formatu
SWF - Shockwave Flash file: format za multimedijsko vektorsko grafiko
TCO – Total Cost of Ownership: skupni stroški lastništva

XHTML - Extensible HyperText Markup Language: hibrid XML in HTML tehnologij

XML - Extensible Markup Language: razširljivi označevalni jezik

Povzetek

S porastom popularnosti svetovnega spleta in vse hitrejšimi povezavami se razvija tudi trg spletnih aplikacij, ki postajajo vse bolj raznovrstne in izpopolnjene. Vse pogosteje pa se dogaja tudi to, da se spletne aplikacije uporabljajo v zaprtih lokalnih omrežjih, kjer nadomeščajo klasične namizne aplikacije in to celo v poslovnem svetu.
Namen te diplomske naloge je izbira orodja in morebitnih knjižnic komponent, ki bi omogočala izdelavo vrste poslovnih poročil. S prejšnjo različico, ki je bila izdelana z odprtokodno rešitvijo Birt, smo namreč prišli do stopnje, ko ni več zadoščala našim zahtevam. Da bi se izognili težavam z licencami in različnimi orodji za poslovno inteligenco, bi radi tudi poročila izdelali v eni izmed vse bolj razvitih spletnih tehnologij in jih neopazno vključili v celotno aplikacijo.
V nalogi so predstavljeni izzivi, ki jih razvoj take aplikacije prinaša, poudarek pa je na uporabniških vmesnikih in gradnikih, ki jih sestavljajo. Predstavljeni so torej cilji, ki bi jih radi dosegli, novosti, ki jih prinaša Web 2.0 in njegov pogled na razvoj svetovnega spleta ter uporabniške zahteve. Ustavila sem se tudi pri opisu bogatih spletnih aplikacij oz. tako imenovanih RIA ter pri psihološkem vidiku uporabniške izkušnje.
Večji del te naloge predstavlja izdelava odločitvenega modela, s pomočjo katerega na koncu pridemo do odločitve, katero orodje izbrati. Uporabila sem DeXi, program za pomoč pri odločanju, ki temelji na večparametrskem modeliranju. Tak tip odločitvenega procesa zahteva vrsto kriterijev, po katerih ocenimo vse variante, nato pa s pomočjo funkcij koristnosti pridemo do končne odločitve. Za potrebe odločitvenega modela so bili izdelani trije prototipi, ki so mi bili v pomoč pri podajanju ocen. Postopek izdelave vsakega izmed njih je natančno opisan, prav tako razlogi za dodeljene ocene, nekatere podatke o orodjih pa sem pridobila kar iz njihovih spletnih strani. Na koncu je v okviru zadnjega koraka odločitvenega procesa – vrednotenje in analiza variant - opravljena tudi kaj-če analiza, ki podaja dodaten pogled na razloge za končno izbiro.

Ključne besede: uporabniški vmesniki, spletne aplikacije, večparametrsko odločanje

Abstract

With the growing popularity of the World Wide Web and the increasing usage of broadband connections web applications are also developing and are getting more diverse and perfected. There is a trend of exchanging classic desktop applications with web applications even in closed local networks for business usage.
The goal of this thesis is choosing a framework and additional libraries which would allow us to implement a series of business reports. We weren't completely satisfied with our previous solution, which was implemented using Birt, an open source reporting tool. With Birt we came to a point where our demands were greater than its capabilities. We want to avoid trouble with licensing and different business intelligence tools so the goal is choosing a web technology that would allow us to build those reports and integrate them seamlessly in our application.
This thesis includes the challenges that the development of such an application includes and there's an emphasis on user interfaces and their components. There is a specification of goals that we would like to achieve, the novelty of Web 2.0, its perspective on the development of the World Wide Web and user demands. There is also some reflection on rich internet applications and the psychological aspect of user experience.
A big part of this thesis that is dedicated to the creation of a decision model, which would help us choose a framework in which to work. To facilitate this process I used DeXi, a program for multi-attribute decision making. This type of decision process requires a number of criteria. All the options are then evaluated based on the criteria and finally we get a definitive choice based on these values and utility functions. In the process of this decision making three prototypes were developed. The actual usage of different frameworks and libraries helped in their evaluation. The implementation of those prototypes is accurately described and so are the reasons for the assigned values, while some general information was obtained directly from the tools web pages. Additionally a what-if analysis was made which brings new light on the reasons for the final decision.

Keywords: user interfaces, web applications, multi-parameter decision making

1. Uvod

V podjetju Optilab d.o.o razvijamo sisteme za detekcijo goljufij. Področje, na katerem smo začeli delovati je zavarovalništvo, natančneje zdravstveno zavarovanje. Izdelali smo produkt z imenom Admiral , ki stoji na trdni večslojni javanski platformi, dodatno pa uporablja tudi JRules, rešitev za upravljanje s poslovnimi pravili podjetja ILOG. Kot aplikacijski strežnik uporabljamo IBM-ov Websphere Application Server, podatkovna baza pa je prav tako IBM-ova DB2. Izbira Jave za osnovno platformo se je pojavila naravno, saj večina razvijalcev v ekipi obvlada ta jezik in bi bila torej izbira kakšne druge platforme nesmiselna ter bi povzročila velike časovne zamude na začetku razvoja zaradi prilagajanja neznanemu delovnemu okolju. Tudi izbira aplikacijskega strežnika in podatkovne baze je bila enostavna, saj je IBM-ovo okolje narekoval naš prvi naročnik.

Povsem druga zgodba je bil uporabniški vmesnik. Že v osnovi je uporabniški vmesnik razdeljen na dva dela: na portal in poročila. Portal je del aplikacije preko katerega zaganjamo analize, pregledujemo sprožene alarme in kritične dejavnike uspeha, poročila pa so pripomoček, ki naj bi analitikom pomagal pri odkrivanju, ali so postavljeni alarmi dejansko goljufije. Začeli smo s portalom, ki naj bi temeljil na Websphere Portal Server-ju, vendar smo že na začetku razvoja ugotovili, da ta produkt za nas ni primeren. Ker smo potrebovali hitro rešitev smo portal nato naredili v PHP-ju. Sprva naj bi bila to začasna rešitev, vendar se je izkazala za zelo učinkovito in jo zato še vedno uporabljamo. Od IBM-ovih strokovnjakov smo glede poročil dobili namig, da obstaja zelo močno zastonjsko orodje za izdelavo poročil z imenom Birt. Birt je odprtokodno orodje za poslovno inteligenco in izdelavo poročil, ki deluje v okviru razvojnega okolja Eclipse, obstaja pa tudi komercialna različica, ki jo podpira podjetje Actuate. Na začetku je Birt navdušil, saj je zelo intuitiven in omogoča hitro izdelavo preprostih poročil z izredno malo truda. S časom pa se je aplikacija začela razvijati, poročila so postajala vedno bolj raznolika, količina podatkov, ki smo jih želeli prikazati pa se je naglo večala. Velika količina podatkov je prinesla potrebo po paginaciji, ki jo Birt sicer podpira, vendar ne z uporabo AJAX-a, poleg tega je bilo Birt aplikacijo izredno težko integrirati v našo celostno grafično podobo. Zaradi raznolikosti smo začeli pogrešati dodatne gradnike in možnost njihove personalizacije. Največ težav pa je v splošnem povzročala interakcija in odzivni čas, saj je bilo za prikaz nekaterih poročil potrebno čakati tudi po deset sekund in več, kar je nesprejemljivo. Zahteve so torej več kot očitno presegle okvire orodja in zamenjava je postala nujno potrebna.

Cilj te diplomske naloge je torej izbrati novo orodje za izdelavo poročil, ki ne bo postavljalo omejitev, na katere smo naleteli pri uporabi Birta. Prva ideja je bila uporaba enega izmed znanih orodij za poslovno inteligenco, vendar smo to misel opustili iz dveh razlogov. Potrebujemo namreč samo vizualizacijski del, naš sistem sam opravi vse potrebne izračune,

zato so BI orodja za naše potrebe premočna in bi jih preplačali. Poleg tega večina velikih podjetij že uporablja svoje BI orodje, kar bi nam povzročalo težave pri prodaji produkta podjetjem, ki uporabljajo BI orodje različno od tistega, ki bi ga izbrali mi. Nakup dveh različnih sorodnih produktov bi bil namreč zanje nesmiseln.

Izbiramo torej tehnologijo oz. orodje, ki nam bo omogočalo izdelavo želenih poročil. Dejansko potrebujemo neko ogrodje, ki podpira izdelavo bogatih spletnih strani, hkrati pa si želimo, da bi zanj že obstajale komponente, ki jih potrebujemo, ali pa, da je možno vanj vključiti knjižnice, ki take komponente vsebujejo.

2. Glavni del

Tako pomembne odločitve kot je izbira vizualizacijskega orodja, se ne da sprejeti kar po občutku. Potreben je formaliziran proces, preko katerega postopoma pridemo do odločitve. Sestavila sem torej odločitveni model na podlagi večparametrskega odločanja.

Celoten potek odločanja zajema naslednje korake:

- Identifikacija problema
- Identifikacija kriterijev
- Spisek kriterijev
  - Struktura kriterijev (drevo)
  - Merske lestvice
  - Definicija odločitvenih pravil
- Opis variant
- Vrednotenje in analiza variant[4]

Pred zbiranjem informacij o orodjih in začetkom gradnje prototipov je bilo potrebno najprej sestaviti seznam vseh pomembnejših sodil, ki naj bi vplivala na našo odločitev. Za vsako od sodil je potrebno določiti tako imenovano zalogo vrednosti oz. možne ocene, ki jih orodja lahko dobijo pri posameznem sodilu. Ohlapnejši seznam sodil je sicer bil izdelan že prej, saj brez tega sploh ne bi mogli izbrati orodij, ki sem jih primerjala, predvsem pa ne bi bilo mogoče tega izbora zožiti na tri predstavnike, ki sem jih podrobneje proučila. Ta korak torej služi bolj podrobnemu opisu sodil in dodeljevanju zaloge vrednosti.
Za potrebe ocenjevanja orodij so bili izdelani trije prototipi. Seveda je bilo pred izdelavo prototipov najprej potrebno zbrati čim več informacij o orodjih, pa tudi razna navodila, dokumentacijo in primere, ki so pomagali pri čim hitrejši izdelavi prototipov. Ta korak je torej razdeljen na zbiranje informacij o orodjih in samo izdelavo prototipov.

Naslednji korak je seveda ocenjevanje orodij. Bolj grobe ocene so bile seveda podane že med izdelavo prototipov, v tem koraku pa so ocene podane bolj natančno in sistematično ter za vsako sodilo posebej in v okviru predvidene zaloge vrednosti.

Nato je potrebno izdelati odločitveni model. V tem koraku zberemo vse podatke in jih predstavimo v obliki tabelaričnih izračunov in grafičnih prikazov. Za lažjo predstavitev rezultatov in za potrebe kaj-če analize, ki spada v naslednji korak, sem za izdelavo odločitvenega modela uporabila orodje DeXi.

Na podlagi odločitvenega modela lahko izdelamo več tako imenovanih kaj-če analiz. Iz vrednosti, ki nam jih je dal odločitveni model se lahko tako lažje odločimo, katero orodje uporabiti.
Za konec je treba vse opisane korake še dokumentirati in podati končno mnenje oz. pojasnitev sprejete odločitve.

2.1. Identifikacija problema

Sam problem sem predstavila že v uvodu, zato bi moral biti naslednji korak identifikacija kriterijev. Za ta korak, pa sem se morala najprej poglobiti v naše zahteve, predvsem pa v trenutno stanje na trgu tovrstne programske opreme. Poleg tega sem proučila tudi uporabniške zahteve in sicer predvsem s psihološkega vidika. Sledi iskanje variant in seznanjenje z njimi, nato pa lahko končno preidemo na samo odločanje.

2.2. Identifikacija kriterijev

2.2.1. Zakaj spletna aplikacija

RIA oz. Rich Internet Applications so spletne aplikacije, ki imajo lastnosti in funkcionalnosti tradicionalnih aplikacij. Izraz "Rich Internet Applications" je prvič uporabila Macromedia v objavi leta 2002. Tradicionalne spletne aplikacije delujejo tako, da se procesiranje zgodi na strežniški strani, odjemalec pa se uporablja le za prikaz vsebine. Negativna plat takega pristopa je, da je potrebno za vsako interakcijo poslati strežniku zahtevo, počakati na njegov odgovor in osvežiti stran s prikazom nove vsebine. S premestitvijo dela procesiranja na stran odjemalca, se to čakanje zmanjša.
Težko je postaviti ločnico med RIA in klasičnimi spletnimi aplikacijami, za vse RIA pa je značilno, da imajo vmesni sloj med odjemalcem in strežnikom. Ta vmesni sloj deluje kot dodatek brskalnika in se navadno uporablja za prikaz uporabniškega vmesnika in za komunikacijo s strežnikom.

Čeprav je razvoj aplikacij, ki jih poganjamo v spletnem brskalniku zelo omejujoč in težji od klasičnih aplikacij, se trud marsikdaj obrestuje. V primeru uporabe spletnih aplikacij ni potrebno nameščanje sistema na posamezne računalnike, popravki in nadgradnje sistema so za uporabnike praktično nevidne, uporabniki lahko poganjajo aplikacijo praktično iz vsakega računalnika, ki je povezan v omrežje, uporabniška izkušnja pa je enaka ne glede na to, kakšen operacijski sistem uporabljamo (lahko pa se razlikuje glede na uporabljen spletni brskalnik). Potreba po resursih na odjemalcu in na strežniku je bolj izenačena in uporabljamo lahko asinhrono komunikacijo, kar omogoča prenos podatkov med klientom in strežnikom brez potrebe po čakanju.
Seveda pa imajo tovrstne aplikacije tudi pomanjkljivosti. Klasične aplikacije imajo dostop do vseh sistemskih resursov, v primeru spletnih aplikacij pa je ta dostop iz varnostnih razlogov omejen. Kljub temu, da se večji del aplikacije navadno shrani v predpomnilnik, je vedno potreben določen čas, da se aplikacija prvič naloži. V primeru počasne internetne povezave je to lahko problem tudi pri nadaljnji uporabi. Kot dodatno pomanjkljivost lahko omenim nasploh potrebo po omrežni povezavi in neskladnost prikaza od brskalnika do brskalnika.

Možnost premika kode na odjemalčevo stran da načrtovalcem in razvijalcem veliko več svobode, vendar to hkrati prinese težji razvoj, veča možnost, da se pojavljajo napake in naredi testiranje bolj kompleksno. Nekatere od teh težav lahko omilimo z uporabo ogrodij za gradnjo spletnih aplikacij, ki standardizirajo določene aspekte načrtovanja in razvoja RIA aplikacij.
Težava je v tem, da HTML ni bil mišljen za izdelavo pogledov, ki bi jih uporabljala splošna populacija, pričakovalo se je tehnične uporabnike, zato mu primanjkuje način za učinkovito predstavljanje informacij. Današnji spletni vmesniki so tako žrtve slabe zasnove in hitre rasti. Dodatna težava je tudi to, da ima uporabnik na voljo veliko nadzora nad samim okoljem (na primer sprejemanje piškotkov, prikaz slik, gumbi nazaj/naprej, personalizacija strani ipd.) [15].

2.2.2. Web 2.0

Web 2.0 je izraz, ki označuje trend v uporabi tehnologij svetovnega spleta, ki teži k povečanju kreativnosti, delitve informacij in sodelovanja med uporabniki. Kljub temu, da ime namiguje na novo verzijo svetovnega spleta, se dejansko ne nanaša na nove tehnične specifikacije, ampak spreminja način, kako razvijalci programske opreme in končni uporabniki uporabljajo splet. Po definiciji Tima O'Reillyja, je Web 2.0 poslovna revolucija v računalniški industriji, ki jo je povzročil prehod na internet kot na platformo in je poskus razumevanja pravil za uspeh na tej platformi [8]. Stephen Fry podaja definicijo, ki temelji na interaktivnosti: je ideja v glavah ljudi bolj kot realnost. Je ideja, da je recipročnost med uporabnikom in internetno stranjo najpomembnejši aspekt [16]. Po Bestu so karakteristike Web-a 2.0 naslednje: bogata uporabniška izkušnja, sodelovanje uporabnikov, dinamična vsebina, meta podatki, spletni standardi in skalabilnost [2].
Seveda pa se v podporo taki miselnosti vedno bolj razvijajo tehnologije, ki omogočajo implementacijo teh lastnosti. Spletne strani, narejene po Web 2.0 principu zato velikokrat uporabljajo naslednje tehnologije: CSS, XML ali JSON, RIA tehnike, ki pogosto temeljijo na AJAX-u, XHTML in HTML, RSS in Atom, Wiki ali forume in podobno. RIA tehnike kot so AJAX, Flash, Flex, Java in Curl so se razvile do te meje, da imajo potencial za zagotavljanje boljše uporabniške izkušnje v aplikacijah, ki tečejo v spletnih brskalnikih. Te tehnologije omogočajo, da spletna stran zahteva in posodobi del svoje vsebine, ne da bi se osvežila celotna stran hkrati.

2.2.3. Uporabniški vmesniki

Izdelati je torej potrebno popolnoma nov uporabniški vmesnik, ki pa mora zaradi prepoznavnosti ohranjati celostno grafično podobo. Zato se najprej postavlja vprašanje, kakšen naj bi moderen, kvaliteten uporabniški vmesnik sploh bil.

Ustavimo se najprej pri izgledu, ki ni zanemarljiv faktor. Uporabniki namreč enačijo uporabniški vmesnik s sistemom in po njem tudi ocenjujejo njegovo kvaliteto, odločilnega pomena je zato tudi pri prodaji, ko potencialni uporabniki sistema še ne poznajo in so prvi vtisi vse, po čemer ga lahko sodijo, v samo vsebino pa se poglabljajo kasneje. Biti mora seveda privlačen in konsistentne oblike, saj dober izgled daje vtis, da je ekipa, ki stoji za produktom, strokovna in sposobna. Vendar za ta vidik uporabniškega vmesnika v največji meri poskrbi celostna grafična podoba, zato raje nadaljujmo in se posvetimo vsebini.

Aksiom uporabniških vmesnikov pravi, da je uporabniški vmesnik dobro načrtovan, ko se program obnaša točno tako, kot je uporabnik pričakoval. Dr. Martin E.P. Seligman proučuje teorijo naučene nemočnosti, po kateri depresija v veliki meri izhaja iz občutka nemočnosti oz. nebogljenosti – občutek, da ne moremo obvladovati okolja. Bolj kot se nam zdi, da stvari, ki jih počnemo, dejansko delujejo, bolj smo zadovoljni. Obvladljivost je torej ključna lastnost sistema, kar implicira dve drugi lastnosti in sicer odzivnost (sistem mora biti hiter) ter uporabnost (sistem mora imeti bogat nabor funkcij in biti enostaven za uporabo).
ISO 9241-11 definira uporabnost kot mero, do katere lahko uporabnik uporabi produkt za doseganje določenih ciljev z uspešnostjo, učinkovitostjo in zadovoljstvom v določenem kontekstu uporabe. Po tej definiciji mora produkt, v našem primeru programska oprema z

uporabniškim vmesnikom, zagotoviti način za doseganje ciljev. Rešitev naj bi ponujala najbolj učinkovit in zadovoljiv način, bogatejša uporabniška izkušnja pa naj bi pri tem veliko pripomogla.

Ljudje, ki so navajeni klasičnih aplikacij, bodo tudi od spletnih aplikacij pričakovali, da so hitre, z bogatim naborom gradnikov ter z možnostjo izbiranja določenih opcij oz. nastavitev. Poleg tega so navajeni zdaj že praktično povsod prisotnih širokopasovnih povezav, ki omogočajo hitrejši prenos podatkov, konkurenca na spletu pa je velika in posledično se kakovost spletnih strani iz dneva v dan veča. Uporabniki torej postajajo vsak dan bolj zahtevni in da bi bil produkt lahko uspešen, se je potrebno prilagoditi njihovim zahtevam.

2.2.3.1. Čas

Uporabniki so vedno bolj nestrpni do velikih odzivnih časov, neučinkovite navigacije in lokacije želenih vsebin. V splošnem so ljudje pripravljeni počakati več za boljše vsebine, naprednejši uporabniki pa čakajo manj kot izkušeni.
Predolg odzivni čas povzroča prekinitev koncentracije, skrb in s tem se manjša produktivnost. V kratkoročnem spominu lahko hranimo podatke 15-40 sekund[1], vendar pa psihološka sedanjost traja manj kot 2-3 sekunde [6]. Manjši odzivni časi večajo produktivnost, ker izničijo omejitve kratkoročnega spomina. Po desetih sekundah se meje trenutnega opravila razblinijo.

Idealen odzivni čas, če zahtevano kontinuirano razmišljanje in si je določene informacije potrebno zapomniti preko več odgovorov, mora biti manj kot 1-2 sekundi.
Če so opravila zaključena in ni potrebno veliko koncentracije, je sprejemljiv čas 2-4 sekunde.
Če si ni potrebno zapomniti vmesnih rezultatov, se lahko čas podaljša na 4-15 sekund.
Če pa lahko uporabnik vmes počne druge stvari, je lahko odzivni čas tudi večji od 15 sekund.
V primeru, ko si ni potrebno zapomniti vmesnih rezultatov, si lahko za boljši občutek uporabnikov pomagamo z iluzijo kratke latence. Uporabnikov namreč ne moti, da čakajo, ampak jih moti dolgčas. Odzvati se je treba takoj, tudi če nimamo končnega odgovora. Dolge operacije delimo na dele ali jih izvajamo na koncu. Če ne gre drugače, je potrebno vsaj prikazati, da se procesiranje še opravlja in uporabnika seznaniti z oceno časa do rezultata. Hiter odzivni čas namreč daje občutek nadzora, tudi če rezultat še ni popoln.

2.2.3.2. Podatki

Podatke moramo predstaviti tako, da je poudarek na njih, elementi, ki niso podatki, pa so minimizirani. Nuditi je potrebno kontekst za pravilno interpretacijo, kar pomeni tudi, da je potrebno uporabnikom pustiti določeno svobodo pri urejanju prikaza podatkov. Bolje je tudi, da maksimiziramo gostoto podatkov, da jih uporabniku ni potrebno iskati preko več strani, vendar moramo vseeno dopuščati možnost, da uporabnik skrije sebi nepotrebne podatke, da postane pogled preglednejši. V tem kontekstu zato potrebujemo nekatere dodatke za čim bogatejši nabor dodatnih funkcij in sicer kontrole za izbiro datumov, drevesne strukture, drsnike, indikatorje napredovanja, zaslonske namige, ipd.

Ljudje si najbolje zapomnijo kombinacijo teksta in slik (v primerjavi s samim tekstom ali samimi slikami). V našem primeru je tekst predstavljen predvsem v obliki tabel, slike pa so grafi.
Grafi so kompleksne ideje predstavljene z jasnostjo, natančnostjo in učinkovitostjo, spodbujajo primerjavo podatkov in uporabniku nudijo največ idej v najkrajšem času in na najmanjši površini. Uporablja se jih za enostavno primerjavo podatkov, predstavitev sprememb skozi čas, statistično analizo in za prikaz razmerij. Stolpčne grafe uporabljamo takrat, ko imamo več podatkovnih virov, črtne grafe, ko je prisotna časovna komponenta, saj dajo občutek povezanosti, histograme za statistično analizo in tortne grafe za prikaz razmerij[3].
Kar se tiče podatkov, ki so prikazani v tabelarični obliki, si zanje želimo čim večjo mero prilagodljivosti prikaza. To vključuje sortiranje in filtriranje po vseh stolpcih, možnost prilagoditve razpona prikazanih podatkov (na primer datumsko), iskanje, možnost izbire prikazanih stolpcev ter označevanje določenih podatkov (na primer z barvami). Seveda si želimo tudi možnost izvoza prikazanih podatkov v kakšno drugo obliko, predvsem v marsikje zelo priljubljeno obliko za delo s preglednicami in sicer Microsoft Excel datoteko ali pa v PDF obliko, ki je primernejša za tiskanje.

2.3. Razpoložljiva orodja

Po nasvetu kolegov iz stroke smo si ogledali naslednja orodja:

- Infragistics NetAdvantage for JSF
- Click
- Adobe Flex
- Backbase
- JasperReports
- Quadbase ExpressReport
- Google Web Toolkit

Po krajši primerjavi funkcionalnosti in izgleda preko primerov, ki so jih pripadajoča podjetja objavila na svojih spletnih straneh, so Click, JasperReports, Backbase in ExpressReport odpadli.
ExpressReport ima sicer precej bogat nabor grafov, vendar je zelo skop kar se tiče ostalih gradnikov, tabele pa ne omogočajo dovolj interaktivnosti.
Zelo podobna slika je bila pri JasperReports, ki sicer za grafe uporablja zastonjsko knjižnico JFreeChart.
Click na svoji spletni strani sploh ne nudi primerov svojih gradnikov, imajo le že izdelane aplikacije, ki jih je potrebno najprej prevesti, da si jih lahko nato lokalno ogledamo. Podatke o tem orodju sem zato raje poiskala na raznih forumih. Po poročilih ostalih uporabnikov se je izkazalo, da je razvoj v tem orodju počasen, rezultati pa ne ravno primerni uporabi v komercialni aplikaciji.
Backbase je zelo močno in profesionalno izdelano orodje, vendar njegova cena letno presega 10000€, zato smo se odločili, da tudi to opcijo izpustimo.
Ostali so torej Infragistics NetAdvantage for JSF, Adobe Flex in Google Web Toolkit. Glede na primerke na njihovih spletnih straneh so si orodja precej izenačena in brez podrobnejše proučitve bi jih bilo skoraj nemogoče oceniti in na podlagi teh ocen sprejeti odločitev, katero izmed njih bomo uporabljali. Za potrebe odločitvenega procesa sem zato izdelala tri prototipe, skozi izdelavo le teh pa sem orodja vsaj delno spoznala.

2.4. Spisek kriterijev

Zgornje raziskave so pokazale predvsem, katere uporabniške kriterije je potrebno upoštevati, najbolj pa bodo prišle prav pri dejanski implementaciji končne rešitve in pri njenem preizkušanju. Glede na povedano, sem med kriterije uvrstila razne funkcionalnosti tabel kot so sortiranje, filtriranje, prilagajanje stolpcev, iskanje, ipd. Podobno kot pri tabelah naj naštejem še nekaj kriterijev, ki jih je treba upoštevati pri ocenjevanju sposobnosti prikazovanja grafov. Pomembna je raznolikost razpoložljivih grafov, možnosti kombiniranja različnih tipov grafov v enega ter možnost uporabe več podatkovnih virov na istem grafu. Tudi tu nekateri kriteriji izvirajo iz že uporabljenih funkcionalnosti ali pa iz lastnosti, ki smo jih pri prejšnjem orodju pogrešali. Vsekakor pa je izredno pomemben kriterij tudi hitrost prikaza in odzivnost aplikacije.
Poleg bolj uporabniških kriterijev, o katerih je bilo govora zgoraj, je potrebno seveda upoštevati tudi take, ki vplivajo predvsem na razvoj. Če gre za povsem novo tehnologijo, je pomembna učna krivulja, skozi celoten čas razvoja pa je pomemben tudi uporabniški vmesnik razvojnega orodja teh hitrost samega razvoja.

2.4.1. Struktura kriterijev

V grobem sem kriterije razdelila na štiri skupine:

- ekonomski,
- uporabniški,
- tehnični kriteriji ter
- dokumentacija in viri.

V skupino ekonomskih kriterijev sodi t.i. TCO (total cost of ownership), ki ga lahko razdelimo na:

- Okvirno cena nakupa določenega števila licenc orodja,
- Stroške posodabljanja ter
- Stroške in čas izobraževanja.

Uporabniški kriteriji se delijo na dve večji skupini in sicer funkcionalnost in zahtevnost.
Funkcionalnost lahko dalje razdelimo še na:

- Funkcionalnost tabel
  - Sortiranje in filtriranje
  - Prilagajanje stolpcev (širina, izbira prikaza)
  - označevanje vrstic
  - iskanje
  - dodatne funkcionalnosti
- Funkcionalnost grafov

        - raznolikost tipov grafov (stolpčni, črtni,tortni, stolpčni z deleži, meter)
        - možnost kombiniranja več tipov grafov
        - možnost uporabe več podatkovnih virov na enem grafu
        - splošen izgled
    - Dodatne komponente

Zahtevnost pa lahko naprej razdelimo na

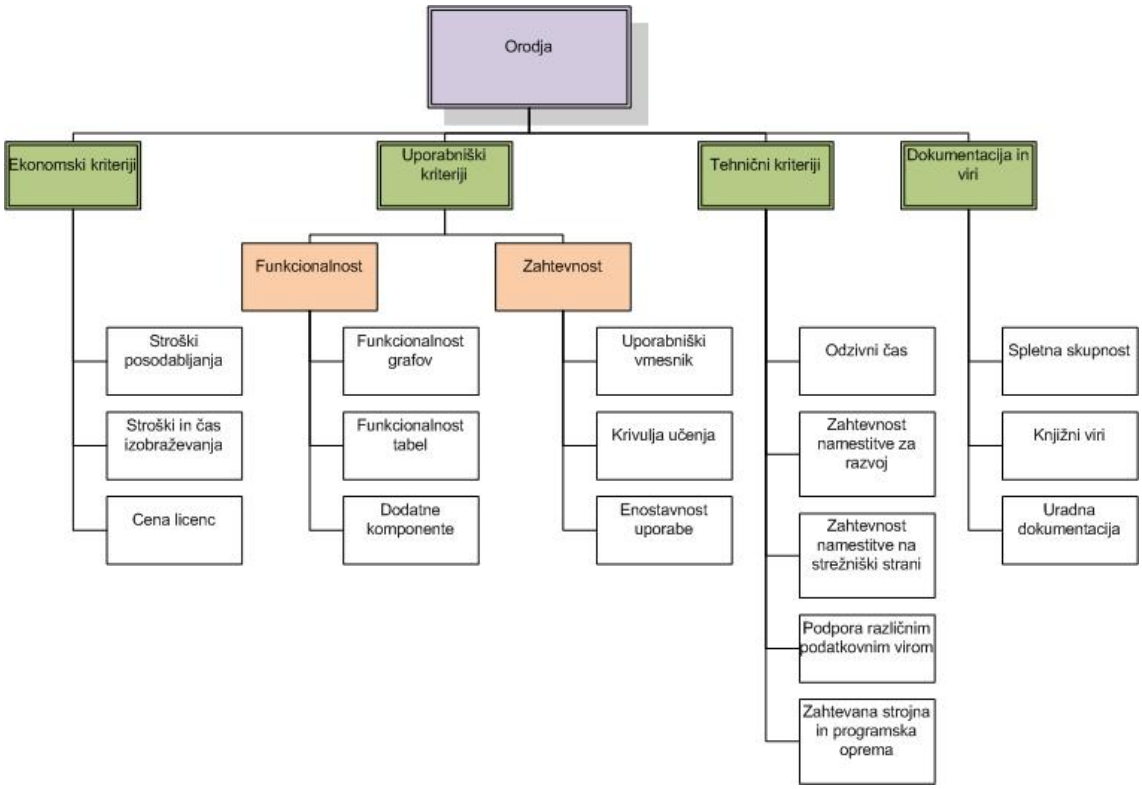
- Uporabniški vmesnik
- Enostavnost uporabe
- Krivulja učenja

Pomembna skupina so tudi tehnični kriteriji in sicer:

- Zahtevana strojna in programska oprema
- Odzivni čas oz. hitrost prikaza
- Podpora različnim podatkovnim bazam oz. različnim podatkovnim virom (EJB)
- Zahtevnost namestitve za razvoj
- Zahtevnost namestitve na strežniški strani

Dokumentacija in viri:

- Uradna dokumentacija
- Spletna skupnost
- Knjižni viri



Slika 3.4.1.1: Drevesna struktura kriterijev odločitvenega modela

2.4.2. Merske lestvice

Vsi kriteriji imajo končno zalogo vrednosti, večino lahko opišemo s kvalitativnimi lastnostmi kot so slabo, srednje in dobro. Proti korenu drevesa se število razpoložljivih ocen lahko veča. Če bi namreč že pri listih uporabila zaloge vrednosti z recimo petimi različnimi ocenami, bi tabele funkcij koristnosti postale prevelike. Proti vrhu drevesa je kriterijev, ki jih primerjamo, vedno manj, zato si lahko privoščimo natančnejše ocene.
Kot že rečeno, je večina ocen podanih opisno in sicer z ocenami zelo slabo, slabo, srednje, dobro in zelo dobro, pri čemer sta oceni zelo slabo in zelo dobro dodani šele na najvišjem nivoju. Seveda pa so tu tudi izjeme. Odzivni čas, ki spada med tehnične kriterije, lahko natančno izmerimo. Če bi vsaki izmerjeni vrednosti priredili eno oceno, bi jih bilo preveč, zato je bolje uporabiti intervale. Tako so ocene kriterija odzivni čas manj kot tri sekunde, tri do deset sekund ter več kot deset sekund. Kriterij stroški in čas izobraževanja je prav tako opisan s kvalitativnimi pridevniki, vendar sem zaradi lepše berljivosti tukaj namesto slabo, srednje in dobro uporabila ocene visoki, srednji in nizki. Ceno licenc, ki tako kot stroški in čas izobraževanja spada v ekonomske kriterije, lahko natančno določimo in izrazimo v evrih. Tudi tu sem uporabila tri intervale in sicer do 500€ letno, od 500 do 1000€ letno ter več kot 1000€ letno. Enake ocene sem uporabila tudi pri tretjem kriteriju skupine ekonomski kriteriji in sicer stroški posodabljanja, tudi te stroške je namreč možno bolj ali manj natančno predvideti in izračunati. Krivulja učenja, kriterij, ki spada med zahtevnost oz. uporabniške kriterije, ima ocene zelo strma, strma in enostavna. Ta kriterij ima tako kot stroški in čas

izobraževanje prirejene ocene zaradi boljše berljivosti. Enako velja za dodatne komponente, kriterij, ki tudi spada med uporabniške, vendar v razdelek funkcionalnost. Tu sem uporabila ocene skope, srednje in bogate.

2.4.3. Definicija odločitvenih pravil

Po tem, ko imajo vsi kriteriji določene zaloge vrednosti, po katerih jih lahko ocenjujemo, je čas, da sestavimo še odločitvena pravila, ki nam bodo dala končno oceno. V primeru večparametrskega odločanja s pomočjo programa DeXi, so odločitvena pravila podana v obliki tabele[4].

	spletna skupnost	uradna dokumentacija	knjižni viri	**dokumentacija in viri**
4	slabo	srednje	slabo	**slabo**
5	slabo	srednje	srednje	**slabo**
6	slabo	srednje	dobro	**srednje**
7	slabo	dobro	slabo	**srednje**
8	slabo	dobro	srednje	srednje
9	slabo	dobro	dobro	**srednje**
10	srednje	slabo	slabo	**slabo**
11	srednje	slabo	srednje	**slabo**
12	srednje	slabo	dobro	**srednje**
13	srednje	srednje	slabo	**srednje**
14	srednje	srednje	srednje	**srednje**
15	srednje	srednje	dobro	**srednje**
16	srednje	dobro	slabo	srednje
17	srednje	dobro	srednje	**srednje**
18	srednje	dobro	dobro	**dobro**
19	dobro	slabo	slabo	**slabo**
20	dobro	slabo	srednje	**srednje**
21	dobro	slabo	dobro	**dobro**
22	dobro	srednje	slabo	**srednje**
23	dobro	srednje	srednje	**srednje**
24	dobro	srednje	dobro	**dobro**
25	dobro	dobro	slabo	**dobro**
26	dobro	dobro	srednje	**dobro**

Slika 3.4.3.1: Primer odločitvene tabele za skupino kriterijev dokumentacija in viri

2.5. Opis variant

2.5.1. Splošni podatki o orodjih

Podatke, ki sem jih potrebovala za ocenjevanje orodij v ekonomskih in tehničnih kriterijih ter glede dokumentacije in virov, sem poiskala na spletu. Podatke za ocene v uporabniških kriterijih sem pridobila kasneje med procesom izdelave prototipov. V tem procesu so se spremenile ali dodale tudi nekatere ocene med tehničnimi kriteriji.
Podatke o cenah orodij ponujajo njihove spletne strani ali pa je preko elektronske pošte potrebno kontaktirati zadolžene za prodajo. Enako velja za podatke o zahtevani strojni in programski opremi in podpori podatkovnim virom. Na spletnih straneh posameznih orodij je mogoče dobiti tudi vso uradno dokumentacijo, knjižne vire sem poiskala preko spletne trgovine amazon.com, ocene glede spletnih skupnosti pa sem podala po krajšem brskanju po internetu, saj je že kratek pregled dal kar jasno sliko o tem, kako obširne in kredibilne so spletne skupnosti posameznih orodij.

2.5.1.1. Infragistics NetAdvantage for JSF

Trenutna verzija Infragisticsovega orodja NetAdvantage for JSF je 2008. Je knjižnica s komponentami za izdelavo uporabniških vmesnikov za J2EE aplikacije. Vsebuje grafe, koledar, menije, drevesa, tabelarične prikaze, vnosna polja in druge komponente. Fiksni stolpci omogočajo uporabniku, da zaklene določene stolpce na levi in omogoča pomikanje z drsnikom skozi podatke na desni. Vsebuje več kot dvajset tipov grafov in omogoča tudi t.i. »overlay« prikaz. Optimiziran je za spletne aplikacije. Smart-Refresh tehnologija neopazno izvaja AJAX zahteve, zato ni potrebno ponovno nalaganje strani. Zazna sposobnosti brskalnika glede AJAX zahtev. Smart-Data-Binding in Smart-Paging, tehnologiji pomagata pri upravljanju z velikimi količinami podatkov. Podpira vizualno in deklarativno programiranje komponent. Podpira t.i. facelet-e in JSR-168 standard za portale. Integriramo ga lahko v IBM Rational Application Developer, NetBeans ali Eclipse, podpira pa vse pomembnejše aplikacijske strežnike kot so JBoss, Apache Tomcat, Websphere in drugi [14].

Cena Infragistics NetAdvantage for JSF 2008 je 795$, kar po trenutnem tečaju znese 498€, cena s prioritetno podporo, ki vključuje štiriindvajseturno telefonsko podporo je 1290$ ali 808,20€. V ceni licence so všteti vsi popravki ali morebitni prehodi na novo verzijo za obdobje enega leta. Cena podaljšanja licence je odvisna od tega, kdaj licenco podaljšamo. Če to storimo do devetdeset dni pred zapadlostjo stare licence, je cena enoletnega podaljšanja 15% nižja od nove licence, če pa za podaljšanje zaprosimo do šestdeset dni pred zapadlostjo stare licence, smo deležni 10% popusta. Petletna osnovna licenca bi nas torej stala 2191,20€.

Infragistics NetAdvantage for JSF kot že rečeno podpira aplikacijske strežnike IBM WebSphere, BEA WebLogic, Sun ONE, Apache Tomcat in JBoss, aplikacije, ki ga uporabljajo, pa lahko tečejo v Microsoft Internet Explorer-ju 6.0 ali novejši, Mozilla Firefox 1.5 ali novejši, Netscape Navigator 7.0 ali novejši in Apple Safari 2.0 ali novejši.

Pri uporabi NetAdvantage for JSF 2008 torej ni posebnih strojnih omejitev, tudi programskih ne. Izbiro okolja in strojne zahteve nam tako narekujejo izključno izbira aplikacijskega strežnika in razvojnega okolja.
Kar se tiče podatkovnih virov, potrebuje NetAdvantage for JSF 2008 podatke v obliki Java objektov. Ustvariti je torej potrebno nov javanski razred, v njem pa lahko kličemo kakršenkoli podatkovni vir, ki ga podpira Java in ga nato pretvorimo v objekt, ki ga bodo komponente NetAdvantage-a klicale iz JSP strani. Glede tipov podatkovnih virov torej praktično nismo omejeni, vendar je za nekatere oblike podatkov potrebno več procesiranja kot za druge.

2.5.1.2. Adobe Flex

Je zastonjsko, odprtokodno ogrodje za gradnjo interaktivnih spletnih aplikacij, ki jih lahko konsistentno uporabljamo z vsemi večjimi brskalniki, strežniki in operacijskimi sistemi. Za opis uporabniškega vmesnika in obnašanja uporablja MXML, deklarativen jezik, ki bazira na XML-ju, za logiko odjemalca pa ActionScript 3, močan objektno orientiran programski jezik. Vključuje bogato knjižnico komponent. Vmesniki ustvarjeni s Flexom so v brskalniku prikazani s pomočjo Adobe Flash Playerja, v klasičnih aplikacijah pa z Adobe AIR-om (oba sta zastonj). Adobe Flex Builder 3 je orodje, ki temelji na Eclipse platformi in omogoča hitrejši razvoj Flex aplikacij. Omogoča lažje kodiranje, grafično sestavljanje uporabniškega vmesnika in interaktivno razhroščevanje. Obstajata standardna in profesionalna različica, slednja pa ima dodana še vizualizacijska orodja, podporo za avtomatsko funkcionalno testiranje, napredne mreže in tabele ter omogoča sledenje zmogljivosti in porabi spomina. Nudi bogato uporabniško izkušnjo, uporabljamo pa ga lahko na različnih platformah. Flash je hiter tudi pri velikih količinah podatkov, Adobe AIR pa omogoča prenos na standardne aplikacije. Vsebuje več kot sto komponent. Kombiniramo ga lahko z jeziki ColdFusion, PHP, ASP.NET in Javo [10].

Za standardno različico Adobe Flex Builderja je potrebno odšteti 249$ oz. 156€, za profesionalno različico pa 699$ oz. 437,90€. Prehod iz verzije 2 na verzijo 3 stane za standardno različico 99$ ali 62€, za profesionalno različico pa 299$ oz. 187,30€. Če predpostavimo, da nova verzija izide vsaki dve leti in se cene nadgradenj ne bodo spreminjale, ter da bomo uporabljali profesionalno različico, nas stane uporaba vedno zadnje verzije orodja 812,50€. Poleg orodja za razvoj bi ob morebitni izbiri Flexa uporabljali še Elixir, dodatno vizualizacijsko knjižnico podjetja ILOG. Cena OEM Elixirjeve licence je 799$ ali 500,60€. Licenca časovno ni omejena in je za enega razvijalca ter vključuje vse popravke in manjše prehode na nove verzije. Ob popolnoma novi različici knjižnice je potrebno plačati celotno ceno nove licence. Če predpostavimo, da nove verzije Elixirja ne bo, oziroma da nanjo ne bomo prešli, nas celotna rešitev (Flex Builder in Elixir) za obdobje petih let stane 1313,10€.
Za namestitev Flex Builderja v okolju Windows potrebujemo vsaj procesor Intel Pentium 4, Microsoft Windows XP s Service Pack 2 ali Windows Vista Home Premium, 1GB delovnega spomina (priporočeno 2GB), 500MB prostora na trdem disku (dodatnih 500MB za konfiguracijo dodatkov), Java Virtual Machine tipa Sun JRE 1.4.2, Sun JRE 1.5 (vsebovan v paketu), IBM JRE 1.5 ali Sun JRE 1.6, Eclipse 3.2.2 za konfiguracijo dodatkov (pri uporabi Windows Viste priporočajo Eclipse 3.3) in Adobe Flash Player 9. Dodatno je bil Flex Builder 3 testiran tudi za delo s platformami BEA Workshop 10.1 in IBM Rational Software Architect

7.0.0.3. Aplikacije ustvarjene s Flex Builderjem 3 lahko namestimo na kakršenkoli aplikacijski strežnik ali pa jih s pomočjo Adobe AIR prikazovalnika uporabljamo kot namizne aplikacije.

Kot podatkovni vir pri Flex aplikacijah uporabimo podatke v XML obliki. Če želimo uporabiti kak drug podatkovni vir, moramo na nek način podatke najprej pretvoriti v XML.

2.5.1.3. Google Web Toolkit

Google Web Toolkit je odprtokodno javansko ogrodje, s katerim se izognemo uporabi tehnologij, zaradi katerih je razvoj AJAX aplikacij tako zapleten in nagnjen k napakam. Z Google Web Toolkitom lahko take aplikacije razvijamo in razhroščujemo v Javi in lahko uporabljamo katerokoli razvojno orodje, ki ta jezik podpira. Našo Java kodo nato GWT prevajalnik prevede v JavaScript in HTML.
Google Web Toolkit ima dinamične komponente, ki jih lahko večkrat uporabimo in sicer tako, da ustvarimo nov t.i. »Widget«, ki ga sestavimo iz že obstoječih komponent. Widget je narejen v obliki Javanskega razreda, zato lahko njegove primerke poljubno uporabljamo. Google Web Toolkit aplikacije so razdeljene na javni, strežniški in odjemalčev del. Koda, ki se nahaja v odjemalčevem delu je na koncu prevedena v JavaScript, zato v njej ne moremo uporabiti vseh Javinih razredov. Pri tej oviri si lahko pomagamo s klicem oddaljenega postopka (RPC – remote procedure call). Postopek ali t.i. proceduro je potrebno napisati v paketu, ki vsebuje strežniški del kode. Ta koda ostane v Javi in se izvaja na strežniku, zato lahko v njej uporabljamo vso zmogljivost Jave.
Pri uporabi AJAX-a se večkrat pojavi ta težava, da izgubimo funkcionalnosti brskalnikovih gumbov naprej in nazaj. Z uporabo Google Web Toolkita lahko to funkcionalnost enostavno ohranimo [11].
Ko projekt prevedemo, GWT ustvari JavaScript kodo, med razvojem in za poganjanje v t.i. »hosted mode« različici pa se Java koda ohranja, zato lahko uporabljamo izjeme in vsa orodja za razhroščevanje, ki jih nudi izbrano razvojno okolje. Prav tako lahko uporabimo JUnit teste in sicer tako za odjemalčevo kodo kot za asinhrone klice oddaljenih postopkov.
V primeru, da bi radi zapustili svet Jave in uporabili kakšno JavaScript funkcijo, ali bi radi dodali kakšno zunanjo JavaScript komponento, lahko to storimo z uporabo JSNI – JavaScript Native Interface. To nam omogoča pisanje JavaScript kode znotraj Java kode. Take funkcije lahko kličejo druge, prave JavaScript funkcije, ki jih uporabljamo na svoji strani, velja pa seveda tudi obratno – JavaScript funkcije lahko kličejo funkcije, napisane v JSNI znotraj Java kode.
Google Web Toolkit je zastonjsko orodje, prav tako zastonj je Cypal Studio for GWT, dodatek za vsa razvojna orodja, ki temeljijo na Eclipse-u, torej tudi za IBM Rational Application Developer. Dodatno bi pa ob morebitni izbiri Google Web Toolkita uporabljali še GWT-Ext, knjižnico komponent, ki temelji na ExtJS zbirki JavaScript komponent. GWT-Ext stane 799$ ali 500,60€ in vključuje neomejeno uporabo za do pet razvijalcev ter prehode na manjše verzije. Ob prehodu na popolnoma novo verzijo je potrebno kupiti novo licenco. Če predpostavimo, da bomo v petih letih prišli do verzije 4 (trenutna je 2.0.4), in da se cene ne bodo spreminjale, nas bo to stalo 1501,80€. Poleg knjižnice s komponentami kot so tabele, gumbi, meniji in podobno, bomo potrebovali tudi knjižnico za prikaz grafov. GWT-Ext jih nekaj sicer vsebuje, vendar vsaj za zdaj ne omogočajo interaktivnosti, zato je bilo potrebno

poiskati drugo rešitev. Za najboljšo rešitev, ki uporablja JavaScript so se izkazali FusionCharts. Za OEM licence imajo FusionCharts kar precej raznoliko ponudbo, ceno namreč podajo glede na velikost podjetja, uporabljene grafe in shemo, po kateri se bo produkt plačevalo. Na voljo so tri opcije. Licenco lahko plačujemo po delih in sicer za vsako kopijo naše aplikacije, ki jo prodamo. Druga opcija je, da celotno licenco plačamo na začetku, nato pa ni omejitev glede količine prodanih aplikacij. Tretja opcija je letno plačevanje licence. Tu prav tako ni omejitev glede števila prodanih aplikacij. V vsakem primeru licenca vključuje uporabo za neomejeno število razvijalcev ter vse popravke in nadgradnje. Za naš primer so pri FusionCharts sestavili ponudbo, po kateri bi uporabili plačevanje po številu prodanih aplikacij in sicer za €. Če predpostavimo, da bomo v naslednjih petih letih imeli deset strank, nas bodo FusionCharts stali 1999€. Skupna cena take rešitve bi torej bila 2500,80€.
Google Web Toolkit(ali krajše GWT) praktično nima strojnih in programskih zahtev. Verzija Google Web Toolkita, ki jo uporabljamo, narekuje tudi določeno verzijo Jave. Prevedena JavaScript koda je napisana tako, da deluje v vseh najbolj znanih spletnih brskalnikih in sicer Microsoft Internet Explorer, Mozilla Firefox, Apple Safari in Opera ter ne glede na verzijo.
Načeloma lahko uporabljamo kakršnokoli razvojno orodje za Java platformo, saj lahko GWT prevajalnik kličemo kar iz ukazne vrstice, vendar je večina dodatkov za GWT napisana za Eclipse. Cypal Studio for GWT, ki sem ga uporabljala za razvoj prototipa v Google Web Toolkitu natančneje zahteva Eclipse 3.3 z WebTools Platform 2.0.
Ena najpomembnejših značilnosti Google Web Toolkita je ravno možnost uporabe z vsemi brskalniki, ne da bi bilo potrebno prilagajanje JavaScript kode, vendar GWT-Ext 2.0.4 priporoča uporabo Mozilla Firefox 3. Strani, ki uporabljajo gradnike iz GWT-Ext 2.0.4 sicer delujejo tudi v drugih brskalnikih, vendar gradniki te knjižnice niso dosledno testirani v vseh okoljih.
Ker večino načinov za pridobivanje podatkov v kodi na odjemalčevi strani ne moremo uporabiti, navadno v te namene uporabljamo oddaljene postopke. Le te puščajo prosto svobodo glede uporabe Jave, zato lahko uporabimo praktično kakršenkoli podatkovni vir, vrniti pa moramo Java objekt, ki implementira IsSerializable ali Serializable razred. Tudi tu torej tako kot pri NetAdvantage-u nismo omejeni glede podatkovnih virov, vendar nekatere oblike podatkov zahtevajo več procesiranja kot druge. Google Web Toolkit komponentam navadno nastavimo vsebino preko setText metod in zato lahko uporabimo kakršenkoli razred, ki vsebuje lastnosti, ki jih lahko pretvorimo v tekst. Pri GWT-Ext komponentah je to malce drugače. Zaradi posebnih funkcionalnosti na tabelah na primer, GWT-Ext vse podatke hrani v t.i. store, da z njimi med sortiranjem in podobnimi operacijami lažje manipulira. Da podatke shranimo jih je potrebno podati v obliki dvodimenzionalnega polja objektov, kar povzroča dodatno procesiranje.

2.5.2. Izdelava prototipov

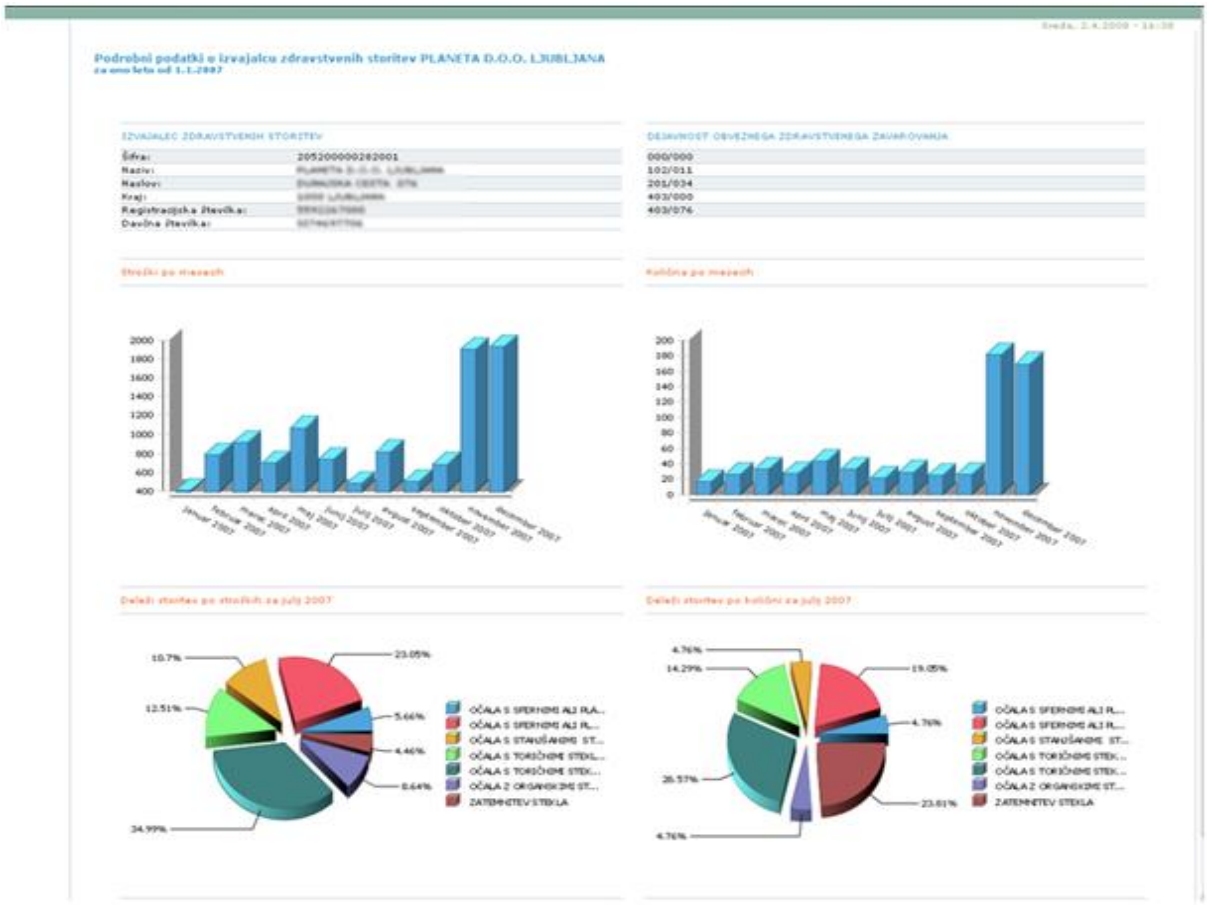
Cilj prototipov je spoznavanje orodij, saj jih je brez tega znanja praktično nemogoče oceniti. Prototip mora torej pokazati, kako bi se obnesla izdelava celotnega uporabniškega vmesnika v določenem orodju. Izdelava celotnega uporabniškega vmesnika v vseh treh izbranih orodjih bi bila izjemno dolgotrajna in nesmiselna, zato morajo biti prototipi dober model za bodoči uporabniški vmesnik in morajo predstavljati vse glavne funkcije, ki jih uporabljamo v izvirni rešitvi dodatno pa še funkcije, za katere je pričakovati, da jih bomo v prihodnosti potrebovali. Poročila so v splošnem sestavljena iz tabel in grafov, zato morajo prototipi nujno vsebovati

primerek tabelaričnega poročila in primerek poročila, ki vsebuje grafe. Prvega predstavlja seznam storitev, drugega pa poročilo, ki prikazuje podatke o izvajalcu zdravstvenih storitev, saj vsebuje štiri grafe, ki so povezani med seboj oz. med seboj omogočajo interakcijo. Ker pa želimo v bodoče poročila še dodatno obogatiti, mora biti v prototipu tudi primer gradnika, ki ga lahko sami razvijemo in prilagodimo po svojih potrebah. V našem primeru bo to drsnik, s pomočjo katerega lahko prilagajamo datumski razpon prikazanih storitev.

2.5.2.1. Opis prototipa

Vsak izmed prototipov vsebuje tri poročila, ki naj bi skupaj dala vpogled v funkcionalnost posameznih orodij. Izbrana je kombinacija obstoječih poročil, ki skupaj prikazujejo čim več različnih elementov, saj le z raznolikostjo lahko dodobra spoznamo nove tehnologije in ocenimo, ali so primerne za našo uporabo. Poleg tega bi bilo nesmiselno v ta namen izoblikovati nova poročila, saj bi za to potrebovali dodaten čas, verjetnost, da bi taka poročila tudi uporabili kot osnutke pri nadaljnjem razvoju, pa je zelo majhna. Vendar obstoječa poročila ne vsebujejo vseh elementov, ki bi jih radi preizkusili. Stara tehnologija marsikaterega želenega elementa ni omogočala, zato bomo obstoječa poročila opremili z dodatnimi funkcionalnostmi in dodatnimi elementi, s katerimi bomo preverili tudi to, kako zahtevno je v posameznih tehnologijah razviti popolnoma nove elemente, ki jih izbrana orodja oz. knjižnice same še ne vsebujejo.

Eno najpomembnejših poročil v naši aplikaciji je »Podrobni podatki o izvajalcu zdravstvenih storitev,« ki prikazuje glavne podatke o izbranem izvajalcu zdravstvenih storitev, dejavnosti osnovnega zdravstvenega zavarovanja, ki jih izvaja ter količino in ceno opravljenih storitev v zadnjem letu ter za izbran mesec. Glavni podatki in izvajane dejavnosti, ki jih krije osnovno zdravstveno zavarovanje so prikazane v dveh tabelah. Pri dejavnostih prikazujemo le šifre dejavnosti in poddejavnosti, če pa se z miško postavimo nad posamezen zapis, se za ta zapis prikažeta še opisa dejavnosti in poddejavnosti. Sledita stolpčna grafa, ki prikazujeta storitve, ki so bile opravljene v zadnjem letu. Levi graf prikazuje skupno količino teh storitev po mesecih (imamo torej dvanajst stolpcev), desni graf pa skupno ceno opravljenih storitev po mesecih (torej prav tako dvanajst stolpcev). S klikanjem po posameznih stolpcih upravljamo tretji del poročila in sicer tortne grafe. Levi tortni graf prikazuje deleže izvedenih storitev gledano po količini in sicer za mesec, ki pripada stolpcu , na katerega smo kliknili. Za desni tortni graf velja isto, le da so tu razmerja med storitvami prikazana glede na ceno, ki je bila zanje zaračunana. Ko poročilo odpremo prvič in ni izbran še noben mesec, tortni grafi niso prikazani.
Največja težava pri tem poročilu je bila njegova počasnost. Že nalaganje osnovne verzije je trajalo preveč za normalno uporabo, ob vsakem kliku za spremembo mesečnega pogleda pa je bilo potrebno naložiti celotno stran na novo, namesto da bi osvežili le spreminjajoče se tortne grafe. Poleg tega so bile funkcionalnosti grafov v Birtu zelo omejene. V novih verzijah sicer poudarjajo razvoj ravno na tem področju, vendar mi potrebujemo takojšnjo rešitev. V primeru tega poročila smo pogrešali možnost, da bi pri tortnih grafih na vsaki rezini prikazali le vrednost (količino oz. ceno storitev), ob premiku miške nad rezino pa bi se pokazala še šifra in naziv pripadajoče storitve.



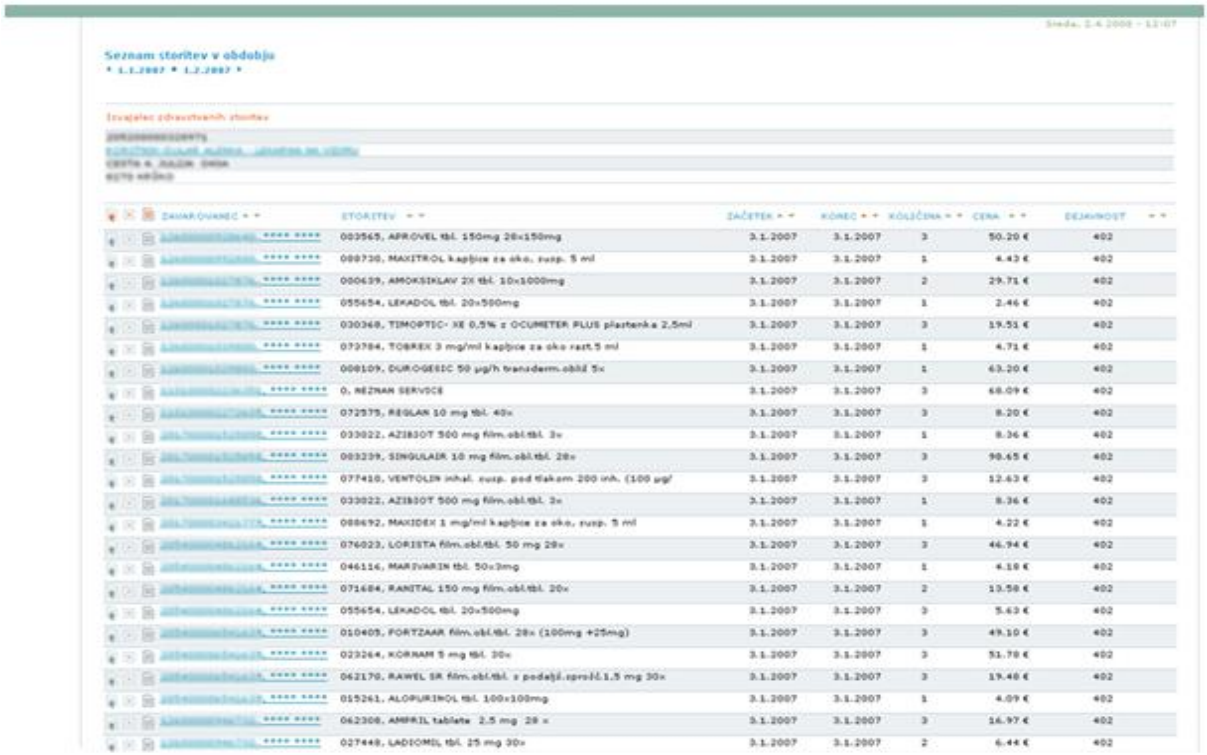
Slika 3.5.2.1.1: Primer poročila »Podrobni podatki o izvajalcu zdravstvenih storitev«

Naslednje poročilo, ki nam je prav tako povzročalo ogromno težav na področju grafov je »Odstopanje cene zdravila.« Na tem poročilu prikazujemo osnovne podatke o izvajalcu zdravstvenih storitev, ki je sprožil alarm, tu imamo tudi povezave do poročila »Podrobni podatki o izvajalcu zdravstvenih storitev,« preko katerega dobimo še dodatne podatke o tem izvajalcu in je opisano zgoraj. Sledi spisek vseh izvajalcev zdravstvenih storitev, ki so na izbrani dan prodali določeno zdravilo ter ceno, po kateri je bilo zaračunano. Posamezna vrstica, ki vsebuje izvajalca zdravstvenih storitev in ceno je obarvana z eno izmed štirih barv in sicer zeleno, modro, oranžno ali rdečo. Rdeča prikazuje cene, ki so višje od povprečja, ki mu prištejemo standardno deviacijo. Cene med povprečjem in povprečjem s prišteto standardno deviacijo so obarvane oranžno, tiste med povprečjem in povprečjem z odšteto standardno deviacijo pa z modro. Cene, ki so manjše od povprečja, ki mu odštejemo standardno deviacijo, so obarvane zeleno. Težava se je pojavila pri grafu, ki naj bi stal levo od te tabele in prikazoval razmerja med cenami še grafično in s tem omogočal boljšo predstavo o razmerju med njimi. Želeli bi namreč graf, ki je vedno enako visok in pri katerem maksimalna cena predstavlja 100% minimalna pa 0%, ostale številke pa so prikazane v razmerju do teh mejnih vrednosti. Kljub naravi grafa, ki prikazuje vrednosti v nekem razmerju, bi na oznakah za posamezno področje radi ohranili absolutne vrednosti v evrih. Poleg tega si želimo, da bi položaj cene zdravila izvajalca zdravstvenih storitev, ki je sprožil alarm bil dodatno označen s črno črto. Imamo torej kombinacijo stolpčnega grafa z razmerji in črtnega grafa.

Slika 3.5.2.1.2: Primer poročila »Odstopanje cene zdravila«

Zadnje poročilo, ki je bilo vključeno v prototip, je prikaz spiska storitev. To je tipično tabelarično poročilo, s katerim želimo prikazati čim več različnih funkcij, ki jih lahko uporabimo na tabeli. Tabele so namreč najpogostejši in najpomembnejši način za prikaz podatkov. Ker je količina podatkov v našem primeru pogosto zelo velika, potrebujemo čim več načinov, s katerimi bi uporabnikom olajšali pregledovanje. Poznamo dve različici danega poročila, vendar smo zaradi analognosti implementacije izdelali le eno. Izberemo lahko, da bomo v zgornjem delu prikazali podatke o izvajalcu zdravstvenih storitev in v spodnjem spisek storitev, ki jih je ta izvajalec opravil v določenem obdobju. V drugem primeru v zgornjem delu prikažemo podatke o zavarovancu, v spodnjem pa spisek storitev, ki jih je le-ta koristil v določenem obdobju. Ker je količina prikazanih storitev pri izvajalcih zdravstvenih storitev večja, smo za implementacijo izbrali prvo različico, saj s tem preverjamo tudi to, kako se poročilo obnaša pri zelo velikih tabelah. Imamo torej poročilo, ki ima v zgornjem delu manjšo tabelo z osnovnimi podatki o izbranem izvajalcu zdravstvenih storitev, v spodnjem delu pa tabelo s podatki o storitvah, ki jih je ta izvajalec opravil v določenem obdobju. Omogočeno mora biti vse, kar posamezno orodje nudi. V aplikaciji narejeni v Birtu smo tako imeli sortiranje, brisanje storitev določenega tipa, barvanje storitev določenega tipa in paginacijo. Na spisku storitev so povezave na postavko računa, ki pripada opravljeni storitvi ter na zavarovanca, ki je storitev koristil. Poleg tega je v naslovu še dodatna funkcionalnost, ki omogoča spreminjanje datumskega razpona, znotraj katerega so prikazane storitve in sicer po en teden naprej ali nazaj, omogočeno pa je tudi vračanje v osnovno stanje. Težava je bila v tem, da je bilo potrebno pri vsaki izmed naštetih operacij stran nalagati

znova, pri čemer se je vsakič izvedla celotna poizvedba, skupaj s prikazom pa je to vzelo ogromno časa.



Slika 3.5.2.1.3: Primer poročila »Seznam storitev«

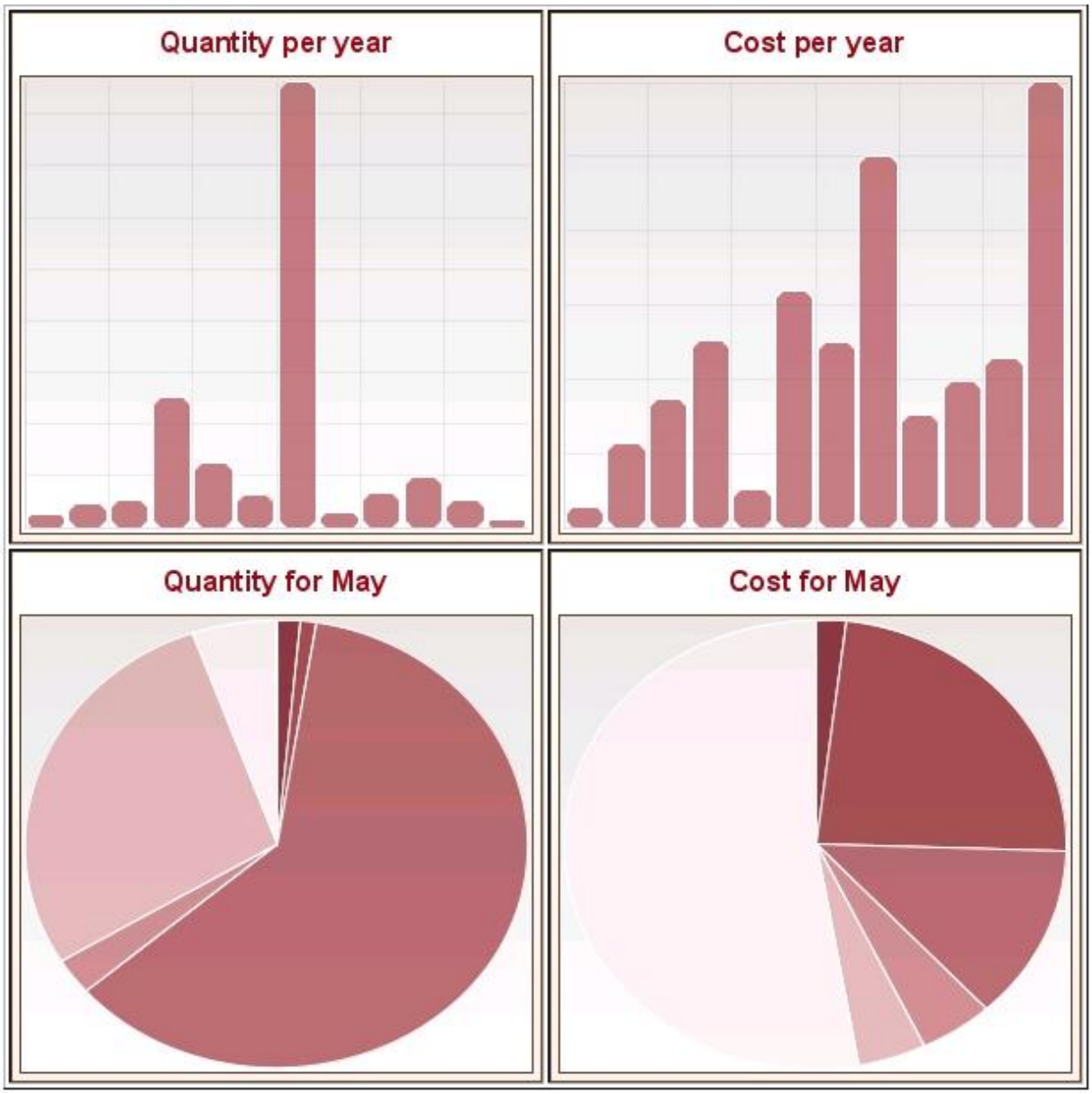
Poleg izboljšanja zmogljivosti bi si želeli še zamenjavo preprostega mehanizma z gumbi, ki spreminja datumski razpon, v bolj prilagodljivo in informativno kontrolo v obliki grafa, ki prikazuje količino storitev na dan, in drsnika, ki bi omogočal izbiro datumskega razpona do dneva natančno.

Take so torej zahteve prototipa. Vse definirane funkcionalnosti sicer niso bile realizirane v vseh treh tehnologijah. Nekaterih preprosto ni bilo mogoče realizirati, za druge bi potrebovala preveč časa in sem se jim zato odpovedala, posledično pa je orodje na tistem področju dobilo slabšo oceno, za nekatere pa je bilo preprosto jasno, da je to mogoče in zato z njimi nisem izgubljala časa.

2.5.2.2. Izdelava prototipa v Infragistics NetAdvantage for JSF 2008

Pri izdelavi prototipa v NetAdvantage-u sem se najprej lotila poročila »Podrobni podatki o izvajalcu zdravstvenih storitev. Ker so tabele s podatki na vrhu poročila trivialne, sem se raje

osredotočila na grafe in povezavo med njimi. Dodajanje grafa in določanje tipa grafa (stolpčni, črtni, ipd.) je zelo enostavno, saj se knjižnica integrira v Eclipse ali RAD tako, da svoje komponente tudi doda v paleto poleg ostali HTML, JSP in JSF komponent. Potrebno je torej le izbrati komponento »chart« in jo postaviti na želeno mesto. Za izbrano komponento se v spodnjem delu razvojnega orodja prikažejo opcije, ki ji jih lahko nastavljamo. Tu je tudi tip grafa. V našem primeru za dva izmed grafov izberemo tip »column«, za druga dva pa »pie.« Naslednji korak je določanje vira podatkov. Tu pa so se začele prve težave. V primerih z navodili je vir podatkov XML datoteka, nikjer pa ni kakšnih podrobnih navodil, kako uporabiti kak drug vir. Uradna dokumentacija je namreč sestavljena le iz primerov in navodil, kako narediti primere ter iz Java API-ja. Uradna spletna stran produkta sicer ponuja tudi forum, preko katerega si uporabniki lahko medsebojno pomagajo, vendar je spletna skupnost tega produkta tako majhna, da je forum praktično neuporaben pri reševanju kakršnihkoli težav. Povsem drugače je pri verziji NetAdvantage-a za .NET, kjer je uporabnikov na forumu ogromno, vendar sta produkta tako različna, da si tudi s tem nisem mogla pomagati. Ob nakupu te programske opreme sicer dobimo pravico do spletne podpore, s »premium« paketom pa tudi do štiriindvajseturne telefonske podpore, vendar sem pri izdelavi prototipa uporabila preizkusno različico, zato te možnosti nisem imela. Tudi izven Infragisticsove spletne strani praktično ni ničesar, kar bi mi lahko pomagalo. Večina zapisov, ki jih vrnejo glavni spletni iskalniki se nanaša na internetne prodajalne, ki izdelek tržijo. Dokumentacija se tako že takoj na začetku izdelave prototipa izkaže kot šibka točka NetAdvantage-a. Preostalo ni nič drugega kot preizkušanje oz. t.i. obratni inženiringa osnovi danih primerov.
Spletna stran, ki uporablja NetAdvantage komponente je tipa JSP z dodanimi povezavami na knjižnice z JSF in NetAdvantage komponentami. Poleg tega so v vsakem NetAdvantage projektu še Java razredi, ki podpirajo delovanje strani. Izkazalo se je, da je potrebno vsakemu grafu določiti javanski razred, iz katerega naj črpa podatke, natančneje pa še objekt tega razreda, ki jih vsebuje. Ni pomembno, kakšen vir podatkov uporabimo, pomembno je, da jih pridobimo v okviru tega javanskega razreda in pretvorimo v obliko, primerno izbranemu tipu grafa. V primeru stolpčnega grafa, je to objekt tipa List, ki vsebuje toliko podatkov, kolikor je stolpcev. Če imamo na istem grafu več nizov, moramo imeti polje, katerega polja vsebujejo objekte tipa List za posamezne nize. Med značilnostmi grafa je potrebno poudariti tudi, kateri od podatkov je vrednost stolpca, ostali so namreč lahko tudi oznake, zaslonski namigi in podobno. V primeru tortnega grafa imamo vedno le eno vrednost za vsak niz in vedno več nizov. Za tak graf potrebujemo poljuben objekt, ki vsebuje vse potrebne podatke. V primeru poročila »Podrobni podatki o izvajalcu zdravstvenih storitev« imamo dva stolpčna grafa, katerih podatke dobimo s klicem EJB metode getMspColumnGraph. Metoda vrne podatke v obliki Java objekta tipa List, ki vsebuje objekte tipa MspColumnGraphData. Vsak izmed teh objektov ima podatek o količini in delu storitev za dani mesec. Podatkovni vir bo torej za oba grafa enak, prav tako podatki na x osi, razlika bo le v podatku znotraj objekta, ki predstavlja y vrednost. Za tortne grafe dobimo podatke s klicem EJB metode getMspPieGraph, ki nam vrne objekt tipa List, ki vsebuje objekte tipa MspPieGraphData, ti pa vsebujejo šifre in opise storitev ter njihove procentualne vrednosti glede na količino in ceno. Mesec oziroma datumski razpon, ki ga potrebujemo ter izvajalec zdravstvenih storitev, za katerega iščemo podatke, podamo obema EJB metodam v obliki parametrov. S tem so izrisani vse štirje grafi, sedaj pa je potrebno poskrbeti, da se bo ob kliku na poljuben stolpec zgornjih dveh grafov, izvedel klic EJB-ja z novimi parametri, ter da se bosta spodnja dva grafa osvežila in prikazala z novimi podatki.



Slika 3.5.2.2.1: Grafi poročila »Podrobni podatki o izvajalcu zdravstvenih storitev« narejeni z Infragistics NetAdvantage for JSF 2008

Poleg knjižnic in dokumentacije je v evaluacijskem paketu še demo aplikacija, ki prikazuje vse NetAdvantage elemente. Pri grafih je takoj opaziti, da je v večini primerov dodana možnost klikanja na stolpce, točke oz. rezine grafa, ob tem pa se v oznaki poleg grafa izpiše vrednost točke, na katero smo kliknili. Ponovno po principu obratnega inženiringa sem pogledala v izvorno kodo demo aplikacije in ugotovila, da imajo taki grafi definiran atribut dataPointListener. V javanskem razredu moramo definirati objekt tipa DataPointListener ter mu dodati referenco v JSP strani po istem principu kot izvor podatkov, torej ime_razreda.ime_objekta. Za ustvarjeni DataPointListener lahko nato implementiramo več različnih metod za različne tipe dogodkov, na katere čakamo. V našem primeru je to onClick metoda, ki se sproži, ko kliknemo na enega od stolpcev grafa. Znotraj te metode je treba sedaj implementirati spremembo podatkov tortnih grafov. Izkaže se, da imamo lahko na vseh elementih tudi atribut »binding.« V Java razredu naredimo objekt tipa Chart z imenom

quantityPie ter enega z imenom costPie. Na pripadajočih grafih izpolnimo atribut »binding« po že znanem pravilu ime_razreda.ime_objekta in tako sta graf na JSP strani ter tisti na Java strani zvezana. Kar počnemo z objekti quantityPie in costPie, se torej odraža preko sprememb na strani. Ob kliku na stolpec dobimo v metodi onClick DataPoint, ki pripada objektu DataPointListener, ki se je odzval na naš dogodek (klik). DataPoint vsebuje podatke o točki, na katero smo kliknili, vendar iz njega nikakor ne moremo dobiti podatka o x osi, torej meseca. Kot najhitrejša rešitev se je izkazal naslednji postopek: v eno izmed mnogih oznak, ki jih lahko obesimo na posamezen stolpec zapišemo še podatek o mesecu. Oznake so zapisane med atributi, atribute pa zlahka preberemo iz DataPoint objekta. Ko preberemo mesec, pokličemo EJB z ravno prebranim parametrom, podatke priredimo objektu, ki ga graf uporablja kot vir podatkov in osvežimo graf. To naredimo za oba tortna grafa.
Zaradi narave aplikacije omogoča razvojno orodje le statičen predogled strani. Da bi preizkusili delovanje strani v celoti, je potrebno aplikacijo naložiti na aplikacijski strežnik. Projekt znotraj Rational Application Developerja je tipa »Dynamic Web Project«, zato imamo več možnosti. Lahko ga izvozimo v obliki WAR datoteke in ga ročno namestimo na strežnik preko administrativne konzole. Druga opcija je, da ustvarimo še EAR projekt in mu dodamo naš projekt kot modul. Tak EAR projekt lahko nato bodisi izvozimo v obliki EAR datoteke in namestimo na strežnik preko administrativne konzole, bodisi ga dodamo na strežnik preko povezave na strežnik znotraj Rational Application Developerja. V tem primeru se bo aplikacija, če le tako želimo, ponovno namestila ob vsaki spremembi in bomo imeli na strežniku vedno zadnjo verzijo, kar je za razvojne strežnike najboljša opcija. Po vsaki taki spremembi v spletnem brskalniku le osvežimo že odprto stran in takoj vidimo spremembe. Izbrala sem seveda zadnji način. Sama namestitev je še kar hitra, vendar stran javi napako, da ne more najti Java razreda, od koder črpamo podatke za grafe. Po natančnem pregledu primerov z navodili iz dokumentacije sem ugotovila, da je potrebno vse uporabljene Java razrede zabeležiti v datoteki managed-beans.xml v naslednji obliki:

```
<managed-bean>
<description>Page bean for the grid</description>
<managed-bean-name>servicesDAO</managed-bean-name>
<managed-bean-class>si.prototype.na.ServicesDAO</managed-bean-class>
<managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```

Na ta način natančno povemo JSP strani, katero ime je referenca na kateri razred in v katerem paketu ga je mogoče najti. S popravljeno managed-beans.xml datoteko se stran prikaže brez napak. Tudi s samim delovanjem sem zelo zadovoljna, saj se tortni grafi osvežijo z neverjetno hitrostjo v primerjavi z našo prejšnjo rešitvijo. Dodati je potrebno le še značilnost, da tortnih grafov na začetku, ko prvič odpremo stran, ni. To najlažje dosežemo z atributom »visible.« Vrednost mora biti privzeto »false.« V metodi onClick pa jo vedno znova popravimo na »true« (po prvem kliku tako ni več spremembe). Pri tem poročilu sem se na tem mestu ustavila. Ostale elemente bi bilo namreč nesmiselno implementirali, saj s tem ne bi pridobila dodatnega znanja o produktu in bi s tem po nepotrebnem tratila čas.

Naslednje je poročilo »Spisek storitev.« Zanj sem ustvarila novo JSP stran z imenom services.jsp. Tudi pri tem poročilu sem zgornjo tabelo s podatki o izvajalcu zdravstvenih storitev izpustila, saj ne vsebuje posebnih funkcionalnosti, njena implementacija pa bi bila trivialna. Posvetila sem se torej tabeli s spiskom storitev in jo uporabila kot primer zmogljivosti NetAdvantage-a na področju tovrstnih gradnikov.

Iz palete izberemo komponento gridView in jo postavimo na stran services.jsp. Izpolnimo atribut dataSource kot objekt razreda, kjer so zapisani podatki. Ta razred tudi dodamo v managed-beans.xml. Iz primerov v demo aplikaciji pa tudi iz primerov z navodili iz dokumentacije sem hitro ugotovila, da je potrebno vsak stolpec tabele deklarirati, mu določiti naslov in lastnost objekta, ki se bo v njem prikazala. Da bi omogočili sortiranje, je potrebno pri vsakem stolpcu, na katerem bo sortiranje omogočeno, izpolniti atribut sortBy. Njegova vrednost mora biti enaka vrednosti, po kateri bi radi sortirali. Sortiranje je že implementirano v komponenti sami, deluje pa na osnovi lastnosti objekta, ki predstavlja vrstico. Iz primerov v demo aplikaciji vidimo tudi, da lahko z enako preprostostjo implementiramo tudi paginacijo. Med atribute gridView-ja enostavno dodamo še pageSize in mu določimo število vrstic, ki jih želimo na eni strani. To je vse. Če tega atributa ne določimo, se vsi zapisi preprosto prikažejo na isti strani. Zamenjave strani so izjemno hitre, prav tako sortiranje, zato lahko na tem mestu NetAdvantage le posebno pohvalim. Nadaljujmo z implementacijo iskanja. Zanj potrebujemo vnosno polje in gumb. Oba elementa povlečemo iz palete na želeno mesto nad tabelo. Gumbi ne potrebujejo posebnega objekta tipa Listener, izpolniti jim je potrebno le atribut actionListener, saj jim ta objekt avtomatsko pripada, njegova vrednost pa je ime metode, ki se mora izvesti ob kliku nanj. Oblika zapisa je seveda že poznana. Za implementacijo sortiranja je treba torej le še napisati metodo onClick (ime je lahko sicer poljubno). V metodo se prenese objekt ActionEvent, iz katerega lahko preberemo vrednost vnosnega polja v trenutku klika. To vrednost nato primerjamo z vsemi vrsticami tabele in ustvarimo nov objekt tipa List, ki vsebuje le tiste vrstice, ki to vrednost vsebujejo. Novo ustvarjeni seznam priredimo objektu, ki predstavlja podatkovni vir tabele, ter ju ponovno povežemo. S tem povzročimo tudi osvežitev tabele, ki mora sedaj prikazovati le vnose, ki vsebujejo iskalni niz. Pred tem moramo definirati še objekt tipa gridView in ga zvezati s tabelo preko atributa »binding«, saj le tako lahko dostopamo do njenega podatkovnega vira in ga popravimo. Izkaže se, da je iskanje ravno tako hitro kot sortiranje in paginacija.


Click here to export to CSV file | tok | Search

Storitve

	Zavarovanec	Šifra storitve	Opis storitve	Datum začetka	Datum konca	Količina	Cena storitve	Dejavnost OZZ
☐	[redacted]	86040	TH DIADINAM INTERFER TOK	14.11.2007	14.11.2007	1	4,1175 €	101
☐	[redacted]	86040	TH DIADINAM INTERFER TOK	14.11.2007	14.11.2007	1	4,1175 €	101
☐	[redacted]	86040	TH DIADINAM INTERFER TOK	14.11.2007	14.11.2007	1	4,1175 €	101
☐	[redacted]	86040	TH DIADINAM INTERFER TOK	15.11.2007	15.11.2007	1	4,1175 €	101
☐	[redacted]	86040	TH DIADINAM INTERFER TOK	22.11.2007	22.11.2007	1	4,1175 €	101
☐	[redacted]	86040	TH DIADINAM INTERFER TOK	22.11.2007	22.11.2007	1	4,1175 €	101
☐	[redacted]	86040	TH DIADINAM INTERFER TOK	22.11.2007	22.11.2007	1	4,1175 €	101

Slika 3.5.2.2.2: Funkcija iskanja po tabeli izdelana z Infragistics NetAdvantage for JSF 2008

Po principu iskanja bi lahko na enak način implementirali še filtriranje. Preostane nam še izvoz podatkov v CSV obliko. Za to dodamo še en gumb, ki izvoz sproži. Kot pri iskanju potrebujemo metodo, ki se bo klicala ob kliku na gumb. Sam izvoz podatkov je že implementiran, treba je poklicati metodo export, ki pripada objektu Grid in ji določiti, katere podatke bomo izvozili ter v kakšni obliki naj bodo. Če nikakor ne označimo vrstic, ki bi jih radi imeli v izvoženi datoteki, se bodo enostavno izbrale vse. Da bi imeli še možnost izbiranja vrstic, na začetek tabele dodamo še stolpec columnSelectRow, ki ima po eno potrditveno stikalo za vsako vrstico. Če mu dodamo še atribut selectAll z vrednostjo true, namesto naziva stolpca dobimo še dodatno potrditveno stikalo, s katerim lahko izberemo vse vrstice. Če damo pri klicu metode export kot parameter »selected« namesto »current«, bomo s tem izvozili vse

vrstice. Brskalnik nam po kliku na gumb »Export to CSV« ponudi odpiranje ali shranjevanje tekstovne datoteke, ki vsebuje vrednosti naše tabele, ločene s podpičji.

Slika 3.5.2.2.3: Izvoz podatkov v CSV obliko z Infragistics NetAdvantage for JSF 2008

Med brskanjem po primerih demo aplikacije, je opaziti še dva atributa gridView-ja, ki ju do sedaj nisem uporabila in sicer resizableColumns, ki sprejema vrednosti »true« in »false« ter fixedColumnCount, ki kot vrednosti sprejema cela števila. S preizkušanjem sem ugotovila, da resizableColumns določa, ali je omogočeno prilagajanje širine stolpcev tabele med ogledovanjem strani. Če poleg tega izpolnimo še atribut fixedColumnCount, bo prvih n stolpcev fiksnih (pri čemer je n število, ki smo ga priredili atributu), preostale pa bo še vedno mogoče prilagajati. Če je resizableColumns nastavljen na »false«, se vrednost fixedColumnCount ignorira.
Poročilu »Spisek storitev« pa bi radi kot že rečeno dodali še element, s katerim bi bilo mogoče določiti datumski razpon prikazanih storitev. V ta namen smo si želeli dodati drsnik, s katerim bi grafično prikazali izbran razpon. Izvedba samega drsnika je preprosta, saj lahko na internetu najdemo pestro izbiro tovrstnih gradnikov, mnogi so prosto na voljo za uporabo.

Slika 3.5.2.2.4: Poskus implementacije izbire datuma z drsnikom pri Infragistics NetAdvantage for JSF 2008

Problem nastane v posredovanju izbranih vrednosti. Če želimo prilagoditi vrednosti v tabeli, moramo to storiti s spremembo njenega podatkovnega vira in sicer v javanskem razredu. Tja moramo torej pripeljati izbrane parametre, za to pa potrebujemo pripadajoči objekt in izpolnjen parameter »binding.« Za zunanje elemente pa pripadajočih Java objektov ni. Poskušala sem s prenašanjem vrednosti v NetAdvantage gradnike na JSP strani s pomočjo čistega JavaScripta, saj bi iz teh gradnikov lahko prebrala vrednosti na Java strani, vendar običajne JavaScript funkcije na teh elementih očitno ne delujejo in je bilo prenašanje neuspešno. V nobenem izmed ponujenih primerov seveda ni uporabe zunanjih komponent, spletna skupnost pa je kot že rečeno za to orodje tako majhna, da na to temo na svetovnem spletu praktično ne najdemo zadetkov. Ob tem se pojavi velika skrb o razširjenosti NetAdvantage-a, saj je očitno uporaba zunanjih komponent ali razvoj lastnih komponent praktično nemogoč, to pa je za tovrstno orodje ogromen minus. Če se je NetAdvantage doslej z izjemo dokumentacije izkazal kot zelo dobro orodje s preprostim razvojnim ciklom in širokim naborom funkcij, smo sedaj naleteli na oviro, ki splošno mnenje o tem orodju drastično spremeni.

Kljub temu sem za potrebe odločitvenega procesa nadaljevala z izdelavo prototipa. Ostalo nam je namreč še poročilo »Odstopanje cene zdravila.« Implementacija grafa je bila preprosta, saj stolpčne grafe že poznam iz poročila »Podrobni podatki o izvajalcu zdravstvenih storitev.« Vse, kar je treba storiti, je določiti kot tip grafa »Stacked100,« da se vrednosti prikažejo v razmerjih ter vsako vrednost, ki jo preberemo iz objekta MedicinePriceCheckGraphData prirediti drugemu nizu. Pomembno je, da so nizi zapisani v takem vrstnem redu, da se ne prekrivajo, zadnji vsebuje vrednost »price« in mu je treba določiti tip »line,« da bo predstavljal zaračunano vrednost zdravila. Poleg grafa postavimo tabelo in jo napolnimo s podatki po že znanem postopku.

Slika 3.5.2.2.4: Poročilo »Odstopanje cene zdravila« izdelano z Infragistics NetAdvantage for JSF 2008

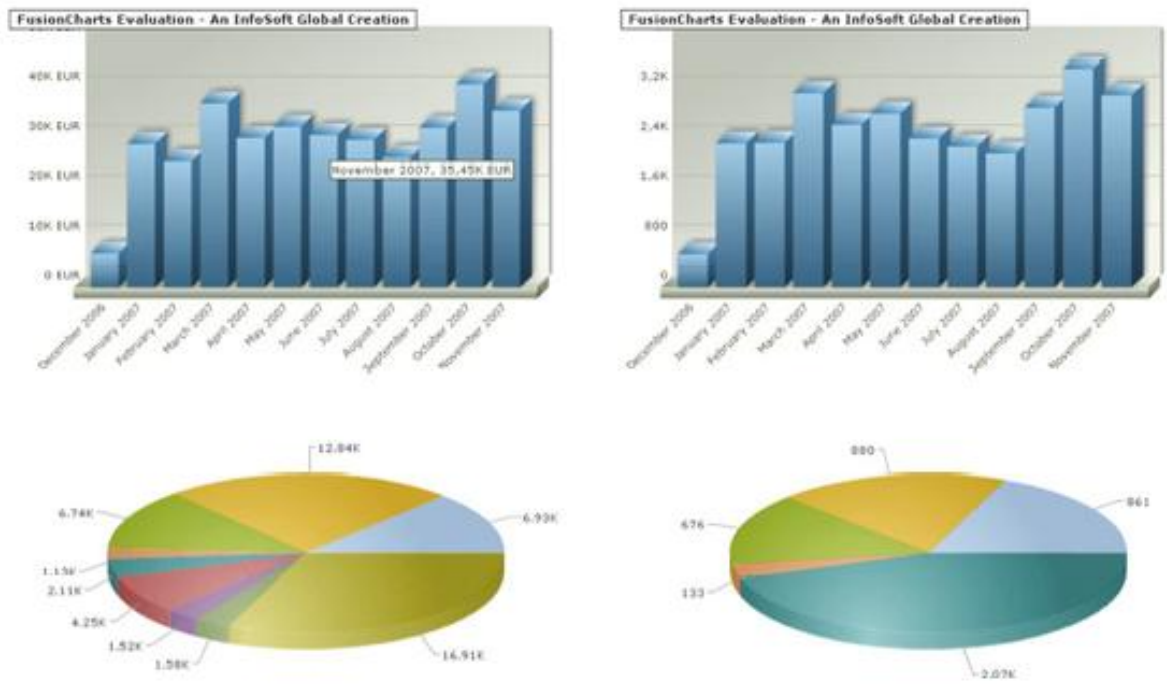
Preostane nam še klikanje po tabeli, ki naj bi sprožilo spremembo položaja črte, ki označuje ceno zdravila. Osveževanje grafov in spreminjanje podatkovnih virov je že poznano, potrebno je še poiskati mehanizem, ki bi omogočal klikanje po tabeli. Pri brskanju po Java API-ju sem našla objekt TableRowListener in poskusila z njim. Ustvarimo mu pripadajočo onClick metodo kot pri DataPointListener-ju. Tudi tu se pojavi problem, da ni mogoče ugotoviti, na katero vrstico smo kliknili, zato preberemo kar vrednost zadnjega stolpca, ji priredimo lastnosti Price objekta, ker predstavlja naš podatkovni vir in osvežimo graf. To je vse, kar potrebujemo in s tem je prototip v NetAdvantage-u končan.

2.5.2.3. Izdelava prototipa v Google Web Toolkitu

Google Web Toolkit je ogrodje, ki vsebuje poseben prevajalnik, ki Java kodo namesto v class datoteke prevede v JavaScript kodo. Za razvoj lahko uporabljamo katerokoli orodje, ki podpira Javo, zato Sem ostala pri IBM-ovem Rational Application Developer-ju. Obstaja veliko GWT dodatkov za orodja, ki temeljijo na Eclipse platformi, nekatera so zastonjska, druga plačljiva, ena pomagajo le pri oblikovanju projektov in prevajanju, druga ponujajo tudi grafično oblikovanje. Ker se dodatki, ki ponujajo grafično oblikovanje izkažejo za bolj ali manj neuporabne, če bi radi dodajali tudi elemente iz zunanjih knjižnic, sem izbrala Cypal Studio. Cypal Studio je zelo enostavno orodje, ki omogoča preprosto kreacijo Google Web Toolkit modulov in oddaljenih storitev. Za vsak nov modul mu je treba določiti le ime vstopne točke in osnovnega paketa, Cypal Studio pa ustvari še client, public in server podpakete, prazen Java razred za vstopno točko in *.gwt.xml konfiguracijsko datoteko. Naj še omenim, da so Google Web Toolkit projekti, projekti tipa Dynamic Web Project. Glede kreiranja samega projekta in nameščanja na strežnik torek velja isto kot za NetAdvantage. Po tem, ko ustvarimo projekt in modul, ki bo vseboval poročilo »Podrobni podatki o izvajalcu

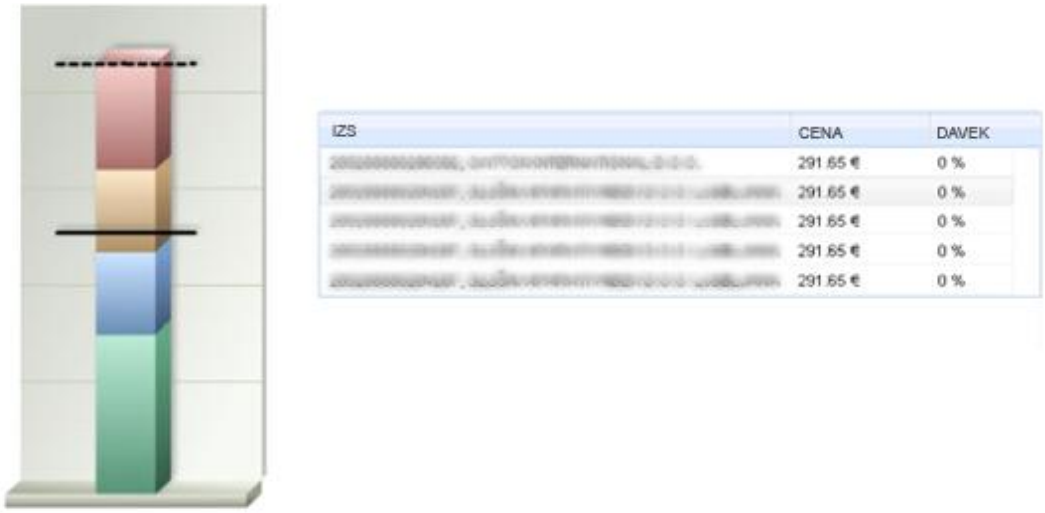
zdravstvenih storitev,« moramo dodati še knjižnice, ki jih bomo uporabljali. Nesmiselno bi bilo namreč razvijati vse komponente in funkcije na novo. Kot najbolj popolna zbirka komponent se je izkazala GWT-Ext. To je GWT vmesnik za znano knjižnico JavaScript komponent ExtJS. Z njegovo pomočjo lahko te komponente uporabimo v jeziku Java namesto JavaScript. GWT-Ext sicer vsebuje tudi nekaj grafov, vendar so bili dodani šele v zadnji verziji in ne podpirajo večino funkcionalnosti, ki jih potrebujemo. Zato je bilo potrebno izbrati še programski paket, s katerim bo mogoče izdelati grafe. Sicer obstajajo tudi paketi, ki jih lahko neposredno vgradimo v Google Web Toolkit kodo, vendar vizualno niso bili preveč prepričljivi, zato smo iskali dalje. FusionCharts je paket Flash grafov, ki jih lahko krmilimo preko JavaScripta, zato so primerni tudi za uporabo z Google Web Toolkitom. Po istem principu kot pri NetAdvantage prototipu sem tudi tu izdelala grafe in interakcijo med njimi, implementacijo tabel pa prihranila za poročilo »Spisek storitev.«

Po navodilih vstavimo povezave do štirih grafov v štiri celice tabele, ki jo ustvarimo v HTML datoteki modula, poleg tega v projektu potrebujemo še SWF datoteke za stolpčne in tortne grafe in XML datoteke, ki bodo za grafe predstavljale podatkovni vir. V ta namen lahko uporabimo JSP strani, ki z določenimi parametri kličejo že omenjene EJB metode in dobljene podatke pretvorijo v XML obliko. Do prikaza grafov s pomočjo navodil in danih primerov v evaluacijskem paketu FusionCharts pridemo v izjemno kratkem času, izgled pa je krasen. Preostanejo nam dogodki. Da bi spremenili parametre, s katerimi kličemo JSP, in osvežili graf, uporabimo JavaScript funkcijo setDataURL. Klicali bi jo lahko direktno iz druge JavaScript funkcije, ki pripada stolpčnim grafom, vendar si želimo na poročilo nekoč dodati Google Web Toolkit elemente, zato je potrebno v prototipu pokazati, da lahko taki klici potekajo tudi preko Google Web Toolkit kode. V ta namen je na voljo tako imenovani JSNI (JavaScript Native Interface). JSNI omogoča implementacijo JavaScript metod direktno v Javi. Ustvari namreč Java signaturo okoli JavaScript kode, kar omogoča klicanje Java metod iz JavaScripta in obratno. Ustvariti je torej potrebno JSNI metodo, ki jo bo klical onClick dogodek na stolpcih obeh grafov ter ji posredoval parameter, glede na katerega se bo iz JSNI metode najprej klicala JavaScript metoda setDataURL, ki bo spremenila podatkovni vir tortnih grafov. Celoten postopek ne povzroča večjih težav. Primer JSNI metode najdemo že na uradni GWT-jevi strani, kodo za dogodke in osveževanje grafov pa najdemo v primerih evaluacijskega paketa FusionCharts.



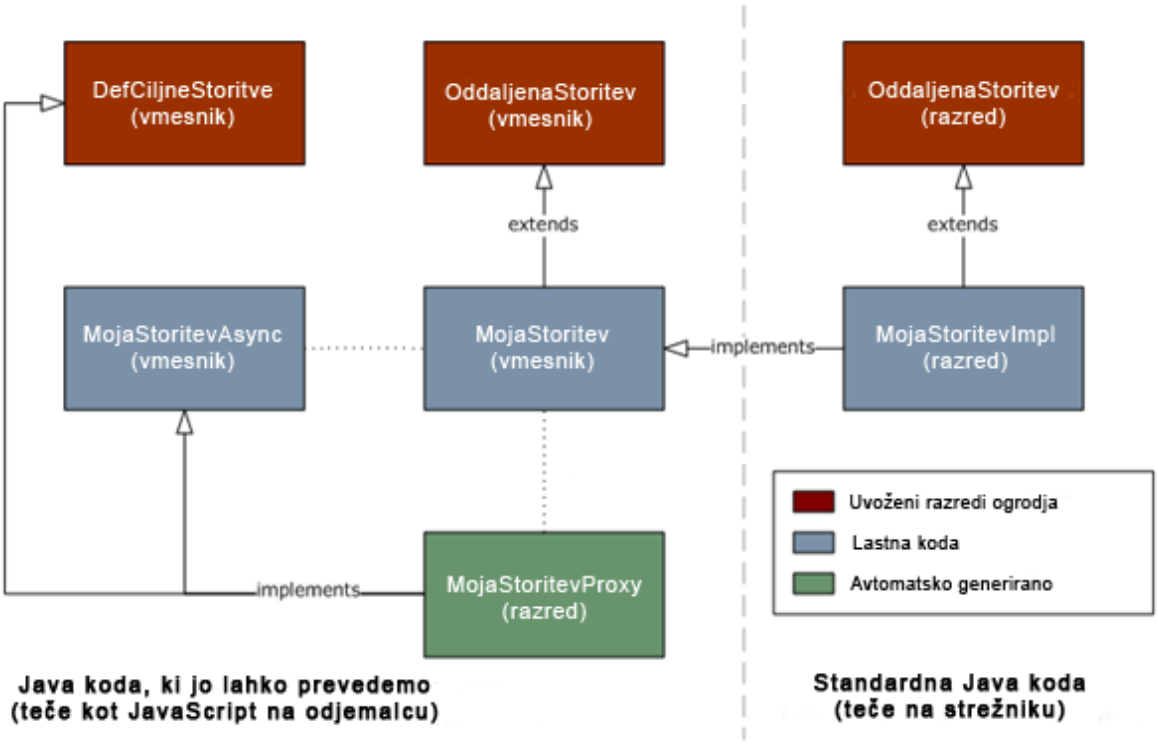
Slika 3.5.2.3.1: Poročilo »Podrobni podatki o izvajalcu zdravstvenih storitev« izdelano z grafi paketa FusionCharts in Google Web Toolkit podlago.

Naslednje je poročilo »Odstopanje cene zdravila.« Sestavlja ga GridView tabela in posebno prilagojen stolpčni graf paketa FusionCharts. Začnimo z grafom.
Kombiniranje različnih grafov v FusionCharts ni problem. Tu sem uporabila Flash komponento StackedColumn3DlineDY.swf ter ji dodala štiri nize za prikaz različnih območij cene nato pa še dva niza tipa črtni graf, ki označujeta referenčno ceno ter ceno izbranega zdravila. Da bi se črte pravilno prikazale, moramo v XML datoteki, ki predstavlja podatkovni vir grafa, simulirati tri različne točke na x osi z istimi y vrednostmi, da je črta, ki se potegne med njimi lepo vidna. Elementi, ki predstavljajo podatke stolpčnih nizov, morajo prav tako vsebovati tri različne točke na x osi, vendar le drugi izmed njih priredimo izračunano vrednost in s tem dosežemo, da je stolp na sredini črt. Način za osveževanje podatkov v grafu že poznamo iz prvega poročila, potrebno je le najti način, kako prenašati klike iz tabele v JavaScript funkcijo.
Za prikaz tabele uporabimo preprost GWT-Ext objekt tipa GridPanel, ki mu dodamo GridCellListener. Objekt tega tipa doda tabeli funkcionalnost klikanja nanjo. S tem ob vsakem kliku ustvarimo dogodek, ki kliče neko funkcijo, ki jo določimo. Znotraj funkcije dobimo podatek o tem, katera vrstica in celo kateri stolpec je bil izbran. V našem primeru potrebujemo le številko vrstice, iz katere lahko iz dvodimenzionalnega polja objektov, ki predstavlja podatkovni vir tabele, dobimo podatek o ceni izbranega zdravila in ga priredimo grafu preko že poznane setDataUrl funkcije, ki ji kot parameter damo novo ceno.



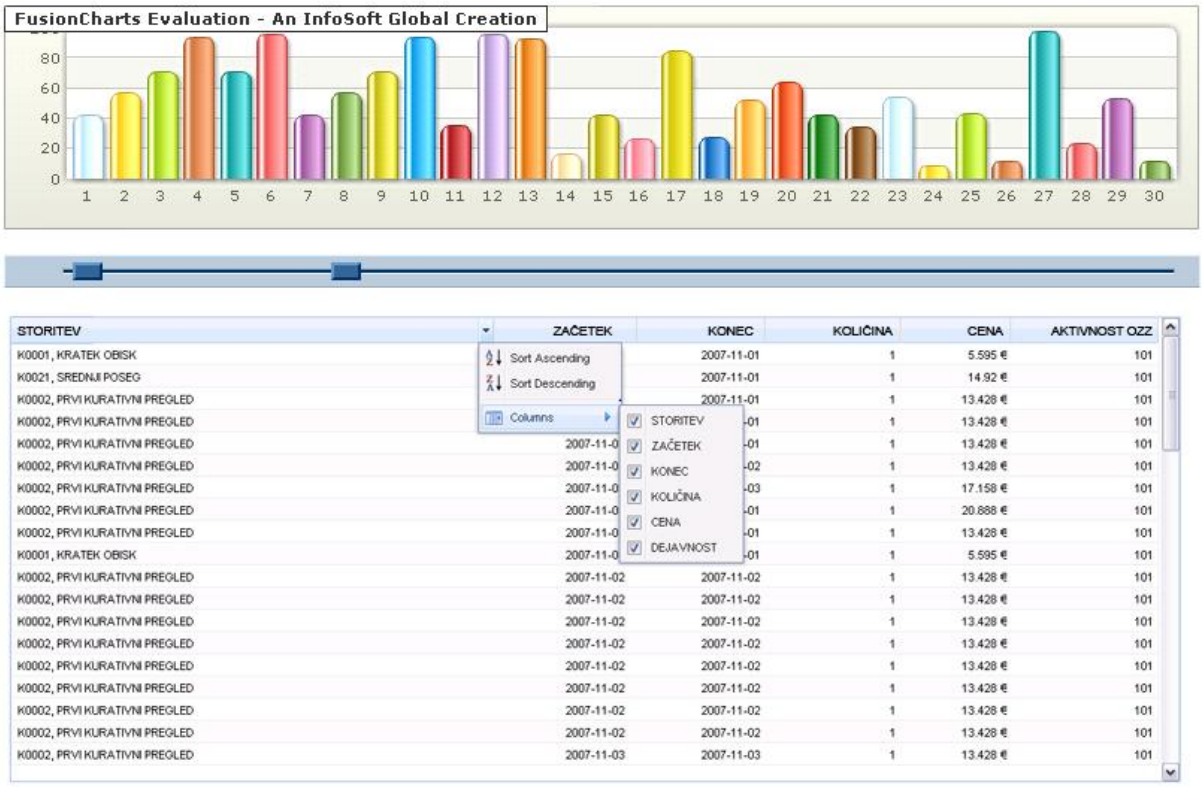
Slika 3.5.2.3.2: Poročilo »Odstopanje cene zdravila« izdelano z Google Web Toolkitom, GWT-Ext knjižnico komponent in FusionCharts grafom

Sledi poročilo »Spisek storitev,« ki je sestavljeno iz gradnika GridPanel. Ta gradnik že sam po sebi omogoča sortiranje, prilagajanje širine stolpcev ter izbiro stolpcev, ki naj bodo prikazani. Vsi tipi mrež v GWT-Ext potrebujejo nek objekt tipa MemoryProxy. Če pa želimo implementirati tudi paginacijo, potrebujemo PagingMemoryProxy. Na mrežo dodamo še objekt tipa PagingToolbar, ki priskrbi gumbe za navigacijo, opcijsko pa lahko dodamo še objekt tipa NumberField, ki prikazuje trenutno stran, število vseh strani ter omogoča vnos in direkten skok na želeno stran. Celotno izvorno kodo lahko skopiramo iz demo prikaza vseh GWT-Ext gradnikov, ki ga najdemo na njegovi uradni strani. Potrebujemo le še podatkovni vir. Ker je ta v našem primeru objekt, ki ga vrača EJB metoda, je potreben oddaljeni postopek, ki bo to metodo klical, vrnil pa objekt, ki implementira IsSerializable. Cypal Studio nam ob zahtevi po novem oddaljenem postopku ustvari tri datoteke in sicer ServicesService.java, ServicesServiceAsync.java ter ServicesServiceImpl.java. Prva dva se nahajata v client paketu, zadnji pa v server paketu. Poleg tega doda definicije za t.i. servlet oz. spletni programček v web.xml datoteko projekta in povezavo na razred storitve v gwt.xml datoteko modula.



Slika 3.5.2.3.1: Prikaz delovanja klica oddaljenega postopka pri Google Web Toolkitu

Sam klic EJB metode se izvede na strežniški strani znotraj ServicesServiceImpl, kot rezultat pa dobimo objekt tipa ServicesListData. PagingMemoryProxy pa potrebuje podatke v obliki dvodimenzionalnega polja, zato ustvarimo nov razred ServiceData z enim elementom, dvodimenzionalnim poljem. Po klicu EJB metode ustvarimo nov tak objekt in ga napolnimo s prebranimi podatki ter vrnemo na odjemalčevo stran. S tem je zaključena implementacija tabele. Potrebujemo še drsnik in graf za spreminjanje datumskega razpona. Uporabimo kar JavaScript gradnik, ki smo ga poskusili uporabiti na primeru NetAdvantage-a. Vstavimo da v HTML datoteko na vrh strani in napišemo JavaScript funkcijo, ki ob spremembi vrednosti drsnika nove vrednosti pošlje JSNI funkciji v Google Web Toolkit kodi, od tam pa jih lahko uporabimo za nov klic oddaljenega postopka, ki nam vrne podatke za tabelo. Tabelo je treba na koncu le še osvežiti. Za še hitrejše delovanje, lahko le filtriramo že pridobljene podatke, tako ponovni klici oddaljenih postopkov niso potrebni in je delovanje s tem hitrejše. Seveda tak pristop deluje le, če datumski razpon zmanjšamo, pri povečevanju je vedno potreben nov klic oddaljenega postopka. Edina težava v celotnem postopku je prilagajanje grafa, ki naj bi predstavljal količino storitev po dnevih z razponom drsnika. Da sem prišla do pravilnega razmerja dan-vrednost, je bilo potrebno kar nekaj preizkušanja.



Slika 3.5.2.3.2: Poročilo »Spisek storitev« izdelano z Google Web Toolkit-om in FusionCharts vključno z izbiro datumskega razpona z drsnikom

2.5.2.4. Izdelava prototipa v Flexu

Razvoj Flex projektov poteka v Flex builderju. To je Adobe-ovo plačljivo orodje, ki temelji na Eclipse platformi. Specializirano orodje bi lahko jemali kot minus, če že uporabljamo drugo razvojno okolje in s tem dodamo novega. Po drugi strani pa Flex Builder delo neverjetno olajša. Flex namreč uporablja ActionScript in MXML, ki sta vsem razvijalcem našega kolektiva tuja. Začetek dela je tako predstavljal pravi izziv. Brez uporabe Flex Builderja pa bi bilo delo še toliko težje. Izmed vseh preizkušenih orodij je namreč to edino, ki nudi kakovostno paleto komponent. V zavihku »Components« le izberemo želeno komponento in jo postavimo na zanjo predvideno mesto na strani. To funkcionalnost ima sicer tudi NetAdvantage, ki se združi s katerimkoli Eclipse okoljem in v paleto doda svoje komponente, Flex Builder pa ima na tem področju še dodatno prednost. Ko sem projektu dodala še paket grafov ILOG Elixir, se je namreč v zavihku »Components« pojavila nova mapa z njegovimi grafi.
Obratno od ostalih dveh orodij Flex nima zbirke primerov uporabe vseh svojih komponent. Večino navodil za izdelavo tega prototipa sem zato dobila iz knjig in raznih forumov na spletu.
Kot pri ostalih prototipih sem tudi tu začela s poročilom »Podrobni podatki o izvajalcu zdravstvenih storitev.« Izdelava tega poročila je bila pri Flexu izmed vseh treh najenostavnejša. Zanj potrebujemo štiri grafe in sicer dva stolpčna ter dva tortna. Flex že

vsebuje nekaj svojih grafov, dodatno pa imamo zaradi pomanjkljive funkcionalnosti še paket Flex grafov Elixir podjetja ILOG. Da bi jih primerjali, dodamo na stran po en graf vsakega tipa (stolpčni, tortni) in vsake knjižnice (Flex, Elixir). Večino kode ustvari Flex Builder sam, v oknu lastnosti pa lahko grafom določimo morebitne dodatne parametre kot so na primer prikaz oznak, naslov in podobno. Flex graf ustvari v MXML kodi značko mx:ColumnChart, Elixir pa ilog:ColumnChart3D. Dodati je torej potrebno le še podatkovni vir in kodo, ki bo podprla klikanje po stolpčnih grafih in posledično spreminjanje podatkov na tortnih.
Podatkovni vir je sicer mogoče določiti tudi v oknu za določanje lastnosti, vendar v tem primeru v sami MXML kodi potrebujemo statični objekt, na katerega se sklicujemo. V našem primeru pa so podatki dinamični, dobimo jih s klicem EJB metode glede na prebrane parametre. Podatkovni vir za oba tipa grafov mora biti XML datoteka, ki jo dobimo s klicem JSP strani. V ta namen je bilo potrebno izdelati dodaten projekt tipa »Dynamic Web Project.« Znotraj tega projekta je za vsak potreben podatkovni vir JSP datoteka, ki kliče pripadajočo EJB metodo, podatke pretvori v XML obliko in jih prikaže. Postopek je hiter, saj sem ga spoznala že pri izdelavi Google Web Toolkit prototipa, kjer sem ta način uporabila za FusionCharts grafe, ki prav tako zahtevajo XML vir podatkov. Uporabila bi sicer lahko tudi vrsto drugačnih pristopov, vendar se Flexovi uporabniki po svetu strinjajo, da je to najhitrejši način. V MXML datoteki je potrebno torej poklicati JSP stran. To storimo z uporabo značke mx:HTTPService na naslednji način [5]:

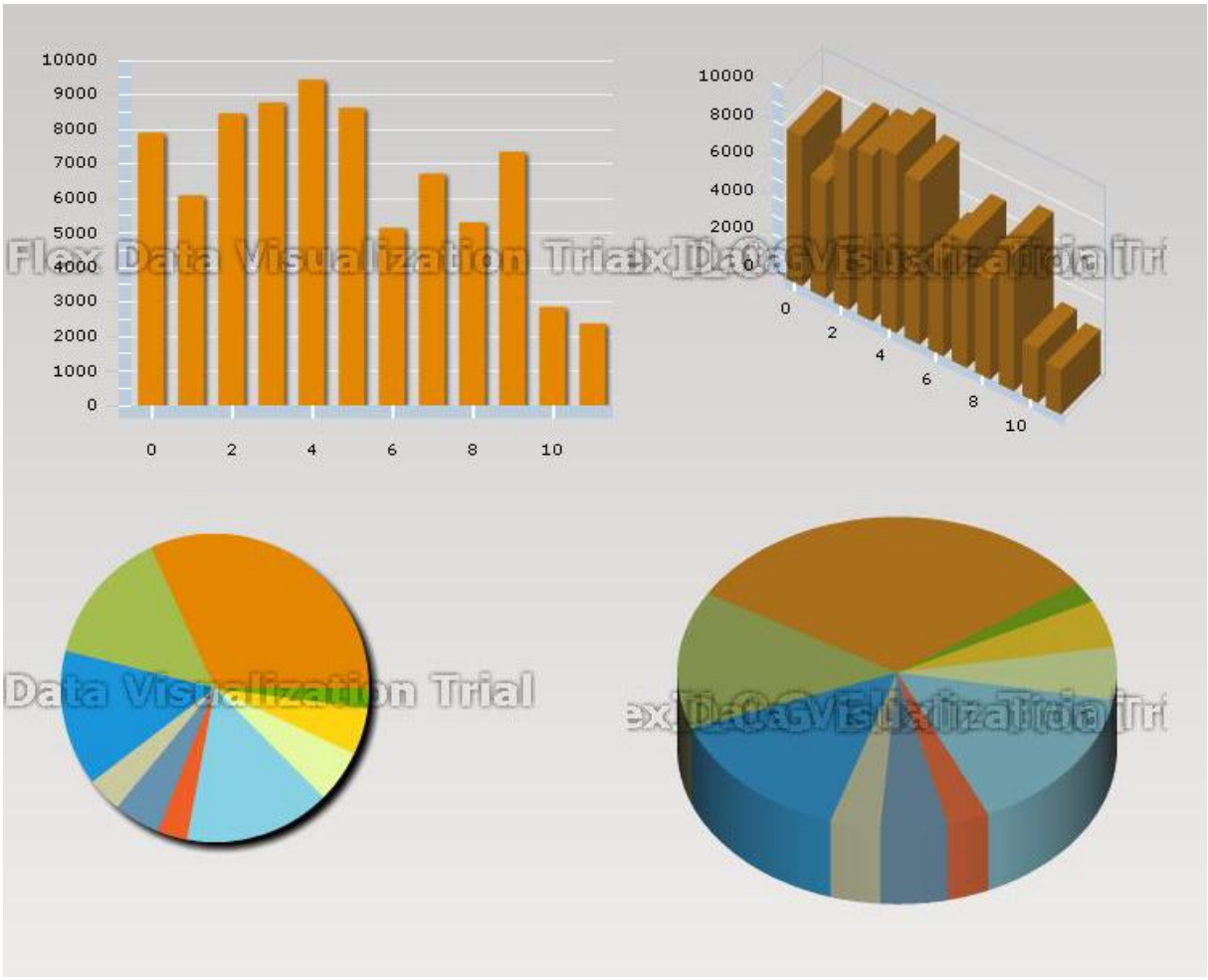
```
<mx:HTTPService id="jspService"
      url="http://localhost:9081/flexData/mspReportData.jsp"
      result="handleObject(event)"
      fault="handleFault(event)"
      resultFormat="e4x">
      <mx:request>
            <mspUnqId>{getParam(mx.core.Application.application.url,
      »mspUnqId«}</mspUnqId>
            <dateFrom>{getParam(mx.core.Application.application.url,
      »dateFrom«}</dateFrom>
            <dateTo>{getParam(mx.core.Application.application.url,
      »dateTo«}</dateTo>
            <month>{{getParam(mx.core.Application.application.url,
      »month«}}</month>
      </mx:request>
</mx:HTTPService>
```

Ta koda generira HTTP zahtevo na podan naslov in doda parametre, ki jih definiramo znotraj mx:request značke. HandleObject in handleFault sta tu metodi, ki se avtomatsko pokličeta, ko dobimo odgovor na zahtevo. V handleFault naj bo koda za preprost izpis morebitne napake, v handleObject pa koda, ki bo vsakemu izmed grafov pripisala tiste podatke iz prejetega objekta, ki jih potrebuje. Podatke za vse štiri grafe imamo lahko v isti datoteki, saj Flex zna delati z XML podatki in mu lahko določimo, katere elemente naj uporabi za posamezen graf s preprosto hierarhijo tipa

```
costColumnChart.dataProvider = event.result.costColumn.elements("value");
```

Za določanje podatkovnega vira je torej potrebna le ena vrstica za vsak graf.

Preostane le še klikanje po stolpčnih grafih in posledično spreminjanje podatkov v tortnih. Flex grafi imajo lastnost itemClick, ki ji lahko priredimo funkcijo, v kateri ponovno kličemo JSP stran z novimi parametri za mesečne podatke, jih priredimo spodnjima grafoma in določimo, da sta vidna. Enako je pri Elixir grafih, le da je tu potrebno izpolniti atribut chartClick, uporabimo pa seveda isto funkcijo.
S tem je prvo poročilo v Flexu zaključeno.



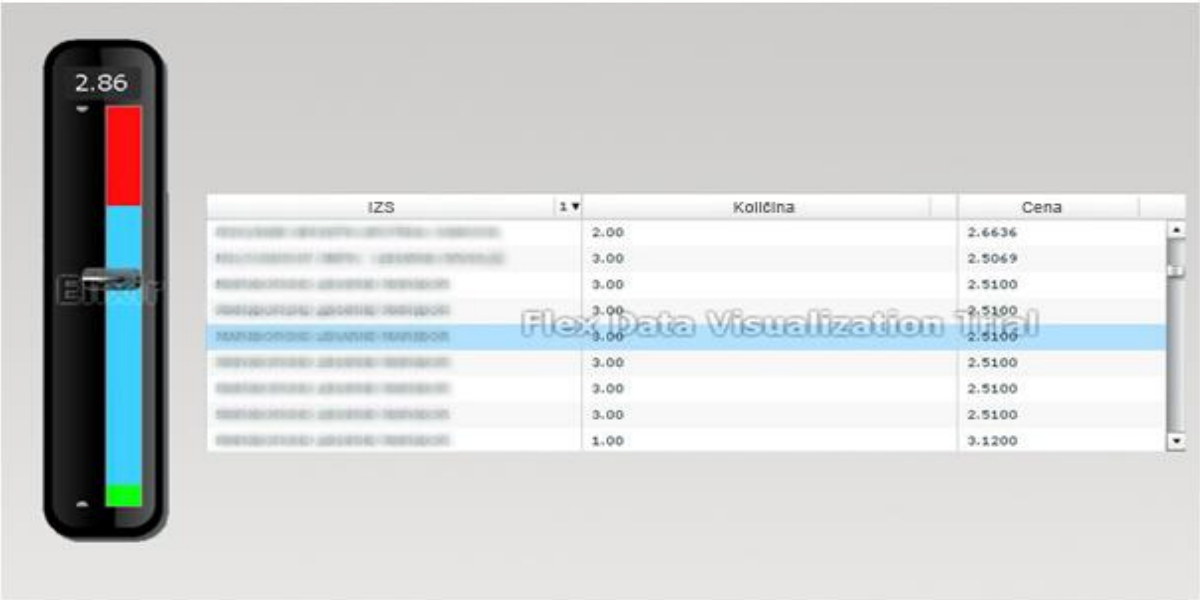
Slika 3.5.2.4.1: Poročilo »Podrobni podatki o izvajalcu zdravstvenih storitev.« Na levi so originalni Flex grafi, na desni pa ILOG Elixir grafi.

Naslednje je ponovno poročilo »Odstopanje cene zdravila.« Tu sem za prikaz spiska izvajalcev zdravstvenih storitev uporabila Flex komponento Advanced Data Grid ki v MXML datoteki ustvari značko mx:AdvancedDataGrid. Določiti ji je potrebno stolpce, ki jih bo tabela vsebovala ter atribute headerText in dataField za vsakega izmed njih. Prvi atribut nam pove, kakšen napis bo vsebovala naslovna vrstica tabele in je statična vrednost, drugi pa, katero izmed vrednosti v podatkovnem viru naj prikaže. Klic JSP strani in določanje podatkovnega vira v metodi handleObject sta že poznana iz implementacije poročila »Podrobni podatki o izvajalcu zdravstvenih storitev.«
Za graf, ki prikazuje območja cen izbranega zdravila sem uporabila Elixir komponento Black Vertical Gauge, ki v MXML datoteki ustvari značko ilog:BlackVerticalGauge. Znotraj le te vstavimo še značko ilog:GradientEntries, v kateri lahko z elementi mx:GradientEntry

določimo barve in intervale, ki jih želimo, Elixir pa te barve združi, da se lepo prelivajo. Višino, na kateri naj se nahaja kazalec, določimo z atributom »value« v funkciji itemClickEvent [13]. Kot v primeru grafov na prejšnjem poročilu lahko tudi tabelam določimo atribut itemClick, ki določa funkcijo, ki se pokliče ob kliku na katerokoli področje v tabeli. V določeni funkcijo nato dostopamo do izbrane cene na naslednji način:

```
price = grid.selectedItem.elements(»price«);
```

Tu je »grid« ime tabele (njen atribut »id«), »price« pa ime stolpca, katerega vrednost želimo prebrati (atribut »dataField«). Prebrano vrednost le še pripišemo grafu in s tem je poročilo zaključeno.

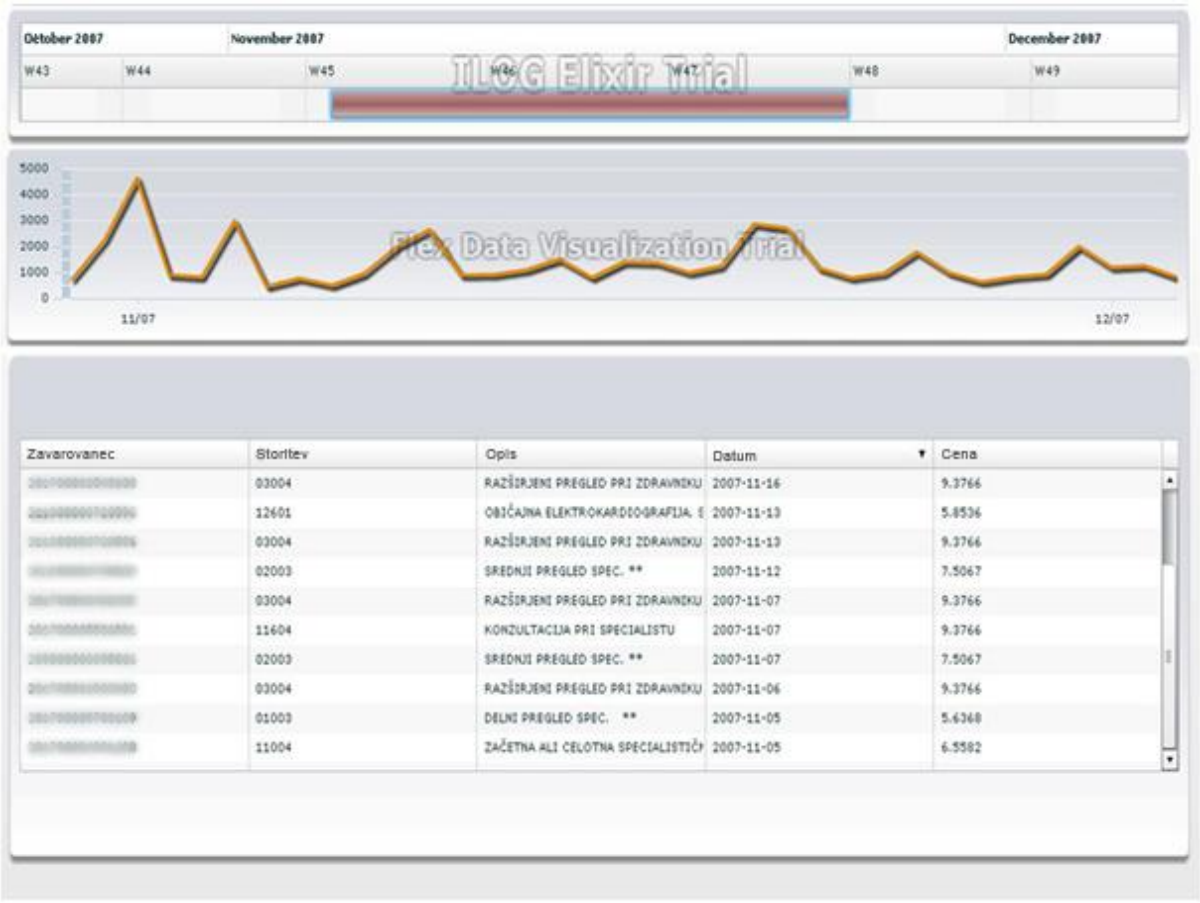


Slika 3.5.2.4.2: Poročilo »Odstopanje cene zdravila« izdelano s Flexom in ILOG Elixir knjižnico za grafe.

Če je bila implementacija prejšnjih dveh poročil razmeroma enostavna, to za poročilo »Spisek storitev« vsekakor ne velja. Tabela pri »Odstopanje cene zdravila« ne potrebuje posebnih funkcionalnosti, tu pa je podatkov veliko več, zato potrebujemo čim več načinov, ki bi nam omogočali kar se da enostavno pregledovanje. Flexov Advanced Data Grid kljub imenu nima širokega nabora že implementiranih funkcij. Sortiranje sicer že obstaja vendar le tekstovno. Če želimo, da bo tudi sortiranje po datumih in cenah delovalo pravilno, moramo napisati dodatne komparatorje. Poleg tega lahko avtomatsko prilagajamo širino stolpcev, ne moremo pa kakšnega izločiti iz prikaza. Tudi funkcije kot so sortiranje in filtriranje bi morali implementirati samostojno. Sam postopek dodajanja tabele je opisan že pri poročilu »Odstopanje cene zdravila« in je tu identičen.
Poleg same tabele imamo na poročilu »Spisek storitev« seveda še komponente, ki naj bi nam omogočale izbiro datumskega razpona storitev, ki jih želimo prikazati. To vključuje graf, ki prikazuje količino storitev po dnevih. Ker so se pri »Podrobni podatki o izvajalcu zdravstvenih storitev« Flex grafi izkazali za lepšo rešitev pri enostavnih grafih, sem tudi tu uporabila enega izmed njih in sicer črtni graf. Namesto drsnika pa sem uporabila del

Ganttovega grafa ki ga vsebuje paket Elixir. Ta graf, ki ga v MXML kodi predstavlja značka ilog:ResourceChart namreč že vsebuje časovni prikaz (skalo dni) in drsnik, ki ga lahko ne le premikamo levo in desno temveč mu lahko tudi spreminjamo širino. Vsebuje pa še marsikaj kot na primer tabelo možnih izvajalcev in graf opravil, ki sledijo časovnemu prikazu. Da bi lahko uporabili časovno os z drsnikom na našem primeru, je potrebno ostale dele celotne komponente odstraniti. Nikjer nisem našla nobenega primera, kako bi lahko uporabili le ta del komponente, zato je bilo potrebno kar nekaj časa in preizkušanja, da sem izločila vso nepotrebno kodo. V pomoč pa je bila sicer razlaga namembnosti vsake značke tega gradnika [12]. Naslednji korak je prebiranje parametrov startTime in endTime, ki določata razpon drsnika. Komponenta ResourceChart ustvari množico funkcij, ki podpirajo delovanje Ganttovega diagrama, med drugim tudi onGanttSheetItemEditResize in onGanttItemEditMove. Prva se avtomatsko pokliče ob vsaki spremembi širine drsnika, druga pa ob premiku drsnika v desno ali levo. Obe funkciji vsebujeta kodo, ki prilagodi vrednosti startTime in endTime spremenljivk, potrebno pa je dodati kodo, ki bo na podlagi teh vrednosti spremenila podatkovni vir tabele storitev. DataProvider vsebuje že implementirano funkcijo filterFunction, ki filtrira vnaprej definiran podatkovni vir katerekoli Flex komponente. Na začetku moramo torej prebrati podatke za celoten razpoložljiv časovni interval, kar pa je lahko zamudno. Filter, ki ga podamo filterFunction metodi je prav tako metoda, ki vrača logično vrednost glede na to, ali podatek spada v filtrirano skupino ali ne. Metoda je avtomatsko definirana, vendar moramo sami napisati njeno vsebino. Vsak datum v podatkovnem viru tabele moramo torej primerjati s startTime in endTime atributi Elixirjeve komponente [7].

Postavitev samega drsnika in implementacija prenosa atributov je bil najbolj zahteven del celotnega prototipa v Flexu. Na srečo ima ILOG Elixir kar izčrpno uradno dokumentacijo vključno s kar bogatim forumom na njihovi uradni spletni strani, ki mi je pri implementaciji zadnjega poročila še najbolj pomagal.



Slika 3.5.2.4.3: Poročilo »Spisek storitev« izdelano s Flexom. Izbor datumskega razpona je izdelan z ILOG Elixir komponento.

2.6. Vrednotenje in analiza variant

2.6.1. Vrednotenje

Vsako varianto je potrebno oceniti po kriterijih, ki so v listih drevesa odločitvenega modela. Vse ostale ocene dobimo preko odločitvenih pravil in tako pridemo do končne odločitve v korenu drevesa.
Nekatere ocene oz. razlogi zanje so bili predstavljeni že v poglavju 3.5 Opis variant. Na tem mestu pa bom predstavila še ostale.

Pri funkcionalnosti tabel, sta NetAdvantage in Google Web Toolkit dobila oceni dobro, Flex pa srednje, saj na trgu ni velike izbire že implementiranih tabel za Flex, osnovna različica pa ne ponuja dovolj funkcionalnosti, morali bi jih torej implementirati sami.
Funkcionalnost grafov sem povsod ocenila kot dobro, saj vse tri knjižnice, s katerimi sem izdelala grafe, ponujajo dovolj funkcij in lep izgled.
Pri dodatnih komponentah je NetAdvantage dobil oceno slabo in sicer zaradi že omenjene težave z vključevanjem JavaScript komponent. Dodatnih komponent, ki bi bile specifične za uporabo z NetAdvantage-em na trgu ni. Flex si je pri tem kriteriju pridobil oceno srednje, saj na spletu lahko dobimo nekaj že izdelanih Flex komponent, implementacija lastnih pa je precej zapletena in navadno zahteva tudi znanje Flasha. V primeru Google Web Toolkita imamo vrsto že implementiranih Widgetov, lahko pa uporabimo tudi navadne JavaScript komponente, zato ima Google Web Toolkit na tem mestu oceno dobro.
Enostavnost uporabe je bila pri vseh treh orodjih ocenjena z dobro, saj nobeno ni posebno zahtevno za uporabo, ko se spoznamo z njim.
Pri krivulji učenja ni bilo pričakovati, da bi katero izmed orodij dobilo oceno enostavna, saj so vsa tri članom naše ekipe in prav tako meni popolnoma nepoznana in zato normalno potrebujejo malce večji začetni vložek, da razvoj steče. NetAdvantage-u in Google Web Toolkitu sem zato dalo oceno strma, Flexu pa celo zelo strma, saj sta mi Java in JSP že poznana, Flex pa uporablja ActionScript in MXML, kar povzroča še dodatne preglavice.
Uporabniški vmesnik je pri NetAdvantage-u in Flexu že poznan, imamo razvojno orodje, ki temelji na Eclipse-u v njem pa paleto, od koder lahko izbiramo komponente ter jim določamo nekatere parametre. Ocena je zato v tem primeru dobro. Pri Google Web Toolkitu tudi uporabljamo Eclipse ali sorodno orodje, vendar palete, v katero bi lahko vključili tudi GWT-Ext komponente ni, zato je dobil oceno slabo.
Pri ekonomskih kriterijih ima NetAdvantage pozitivne ocene, saj je njegova licenca razmeroma poceni, ne potrebujemo dodatnih knjižnic, obnavljanje licence pa je še cenejše kot prvi nakup. Edino stroški in čas izobraževanja imajo slabšo oceno, saj za NetAdvantage ni knjig, ki bi nam bile lahko v pomoč, obstajajo le organizirani tečaji, ki pa seveda stanejo precej več. Tudi Flex in Google Web Toolkit imata razmeroma nizke stroške posodabljanja, vendar je pri njiju začetna cena višja, na tem mestu ima Google Web Toolkit celo najslabšo oceno. So pa vsaj stroški in čas izobraževanja ocenjeni z nizki, saj je za obe omenjeni orodji na voljo vrsta literature, ki nam lahko pomaga pri razvoju.
Eden najpomembnejših kriterijev je odzivni čas, zato je vzpodbudno, da so bila na tem področju vsa tri orodja ocenjena zelo pozitivno. Vsa poročila iz prototipov so namreč imela odzivni čas krajši od treh sekund.

Tudi pri podpori različnim podatkov virom so ocene zelo pozitivne, le Flex je bil ocenjen s srednje, saj je podatke potrebno pretvarjati v XML obliko in jih brati iz JSP datotek. To sicer velja tudi za grafe izdelane s FusionCharts, vendar je za nekaj grafov takega dela precej manj kot za celotno aplikacijo.
Namestitev za potrebe razvoja je v vseh treh primerih zelo enostavna, namestitev na strežnik pa celo identična, saj imamo vedno projekt tipa »Dynamic Web Project.« Ocene so tudi pri zahtevnosti namestitve torej vse dobre.
V sklopu dokumentacije imata Flex in Google Web Toolkit vse tri ocene enake. Njuna spletna skupnost je namreč ogromna in ponuja vrsto odgovorov in komponent, na trgu pa je tudi pestra ponudba literature, ki te dve orodji obravnava. Malce slabšo oceno sta Flex in GWT dobila le pri uradni dokumentaciji, ki sicer ni ravno slaba, lahko pa bi bila precej boljša. Popolnoma drugače je pri NetAdvantage-u. Ena njegovih največjih pomanjkljivosti je ravno dokumentacija. Spletna skupnost je absolutno premajhna, da bi bila uporabna, knjig na to temo praktično ni, tako da se je potrebno zanesti na uradno dokumentacijo, ki je v splošnem dokaj skopa, vendar jo rešujejo primeri vseh gradnikov s kodo.

Ko vse te ocene zberemo, dobimo naslednjo tabelo.

Option	Infragistics NetAdvantage for JSF 2008	Adobe Flex 3	Google Web Toolkit 1.5
funkcionalnost tabel	dobro	srednje	dobro
funkcionalnost grafov	dobro	dobro	dobro
dodatne komponente	skope	srednje	bogate
enostavnost uporabe	dobro	dobro	dobro
krivulja učenja	strma	zelo strma	strma
uporabniški vmesnik	dobro	dobro	slabo
stroški posodabljanja	0-500€ letno	0-500€ letno	0-500€ letno
cena licenc	0-500€ letno	500-1000€ letno	več kot 1000€ letno
stroški in čas izobraževanja	visoki	nizki	nizki
odzivni čas	manj kot 3 sekunde	manj kot 3 sekunde	manj kot 3 sekunde
podpora različnim podatkovnim virom	dobro	srednje	dobro
zahtevnost namestitve	dobro	dobro	dobro
spletna skupnost	slabo	dobro	dobro
uradna dokumentacija	srednje	srednje	srednje
knjižni viri	slabo	dobro	dobro

Slika 3.6.1: Ocene kriterijev v listih odločitvenega modela

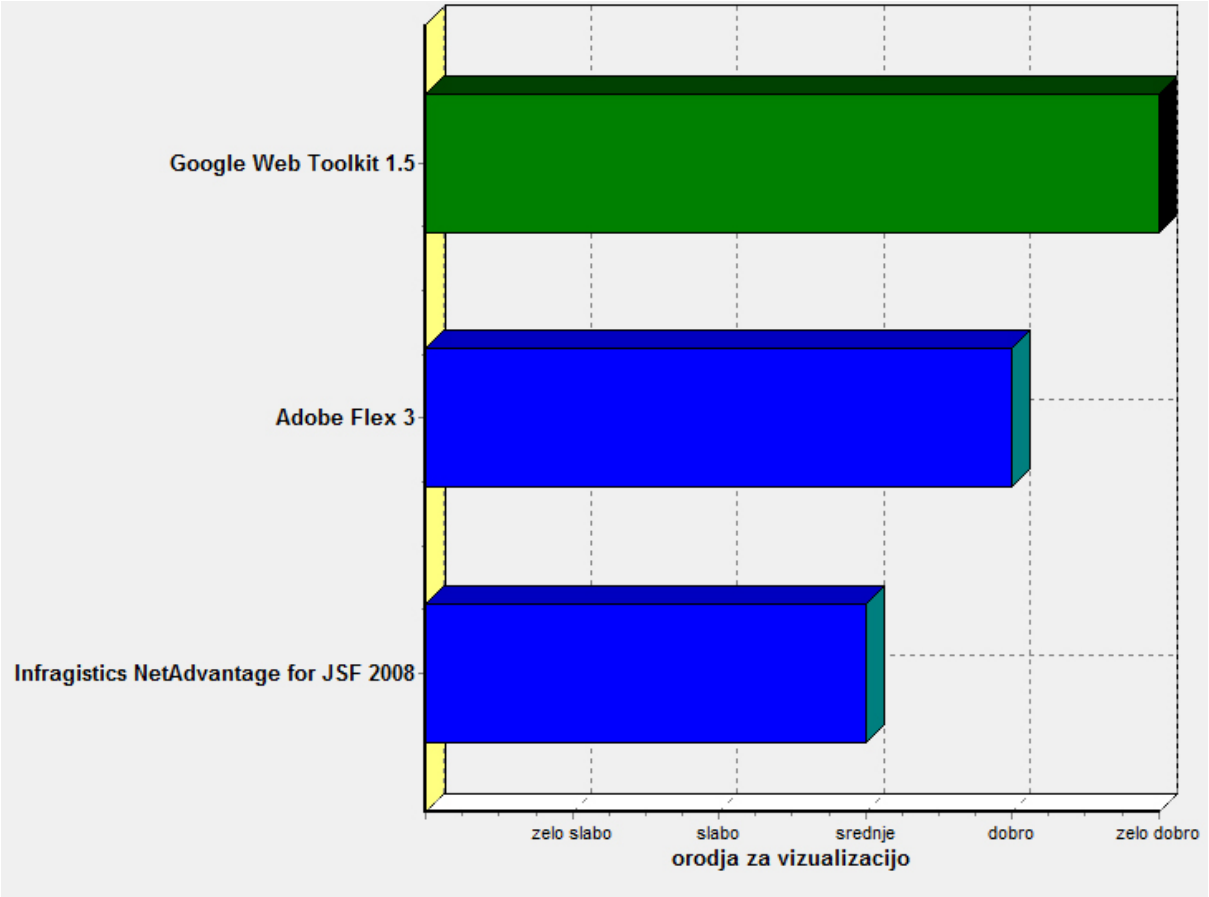
DeXi iz definiranih odločitvenih pravil nato sestavi tudi ocene za celotno drevo, rezultat pa je končna odločitev, ki jo prinese funkcija koristnosti v korenu drevesa odločitvenega modela.

Option	Infragistics NetAdvantage for JSF 2008	Adobe Flex 3	Google Web Toolkit 1.5
. orodja za vizualizacijo	**srednje**	**dobro**	**zelo dobro**
. . uporabniški kriteriji	**srednje**	**srednje**	**dobro**
. . . funkcionalnost	**srednje**	**srednje**	**dobro**
. . . . funkcionalnost tabel	dobro	srednje	dobro
. . . . funkcionalnost grafov	dobro	dobro	dobro
. . . . dodatne komponente	skope	srednje	bogate
. . . zahtevnost	**dobro**	**dobro**	**srednje**
. . . . enostavnost uporabe	dobro	dobro	dobro
. . . . krivulja učenja	strma	zelo strma	strma
. . . . uporabniški vmesnik	dobro	dobro	slabo
. . ekonomski kriteriji	**dobro**	**dobro**	**srednje**
. . . stroški posodabljanja	0-500€ letno	0-500€ letno	0-500€ letno
. . . cena licenc	0-500€ letno	500-1000€ letno	več kot 1000€ letno
. . . stroški in čas izobraževanja	visoki	nizki	nizki
. . tehnični kriteriji	**dobro**	**dobro**	**dobro**
. . . odzivni čas	manj kot 3 sekunde	manj kot 3 sekunde	manj kot 3 sekunde
. . . podpora različnim podatkovnim virom	dobro	srednje	dobro
. . . zahtevnost namestitve	dobro	dobro	dobro
. . dokumentacija in viri	**slabo**	**dobro**	**dobro**
. . . spletna skupnost	slabo	dobro	dobro
. . . uradna dokumentacija	srednje	srednje	srednje
. . . knjižni viri	slabo	dobro	dobro

Slika 3.6.2: Ocene vseh vozlišč drevesa glede na ocene listov in funkcij koristnosti

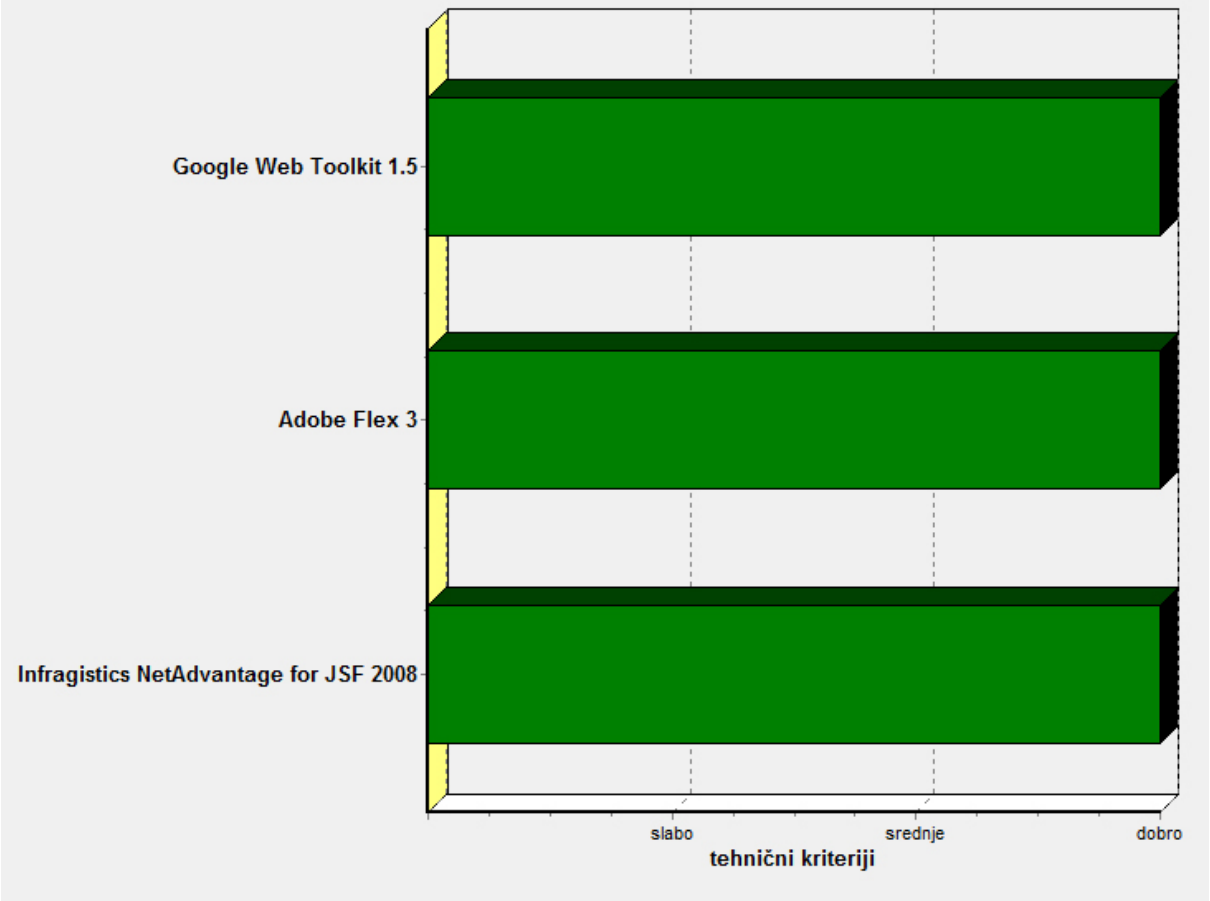
Iz slike je razvidno, da se je najbolje odrezal Google Web Toolkit. V primerjavi s Flexom, ki mu sledi, je dobil boljše ocene predvsem na področju uporabniških kriterijev, tabele GWT-Ext-ja so namreč bolj funkcionalne od Flexovih, JavaScript komponent pa je na medmrežju prav tako več kot komponent za Flex. Funkcionalnost tabel in dodatne komponente sta torej kriterija, ki Google Web Toolkitu prinašata prednost v primerjavi z Adobe-ovim Flexom. Glavne pomanjkljivosti NetAdvantage-a so kot že rečeno nezmožnost uporabe zunanjih JavaScript komponent ter pomanjkljiva dokumentacija, zato se je kljub odličnim ostalim ocenam in nizki ceni znašel šele na tretjem mestu.

Poglejmo ocene orodij še v grafični obliki.



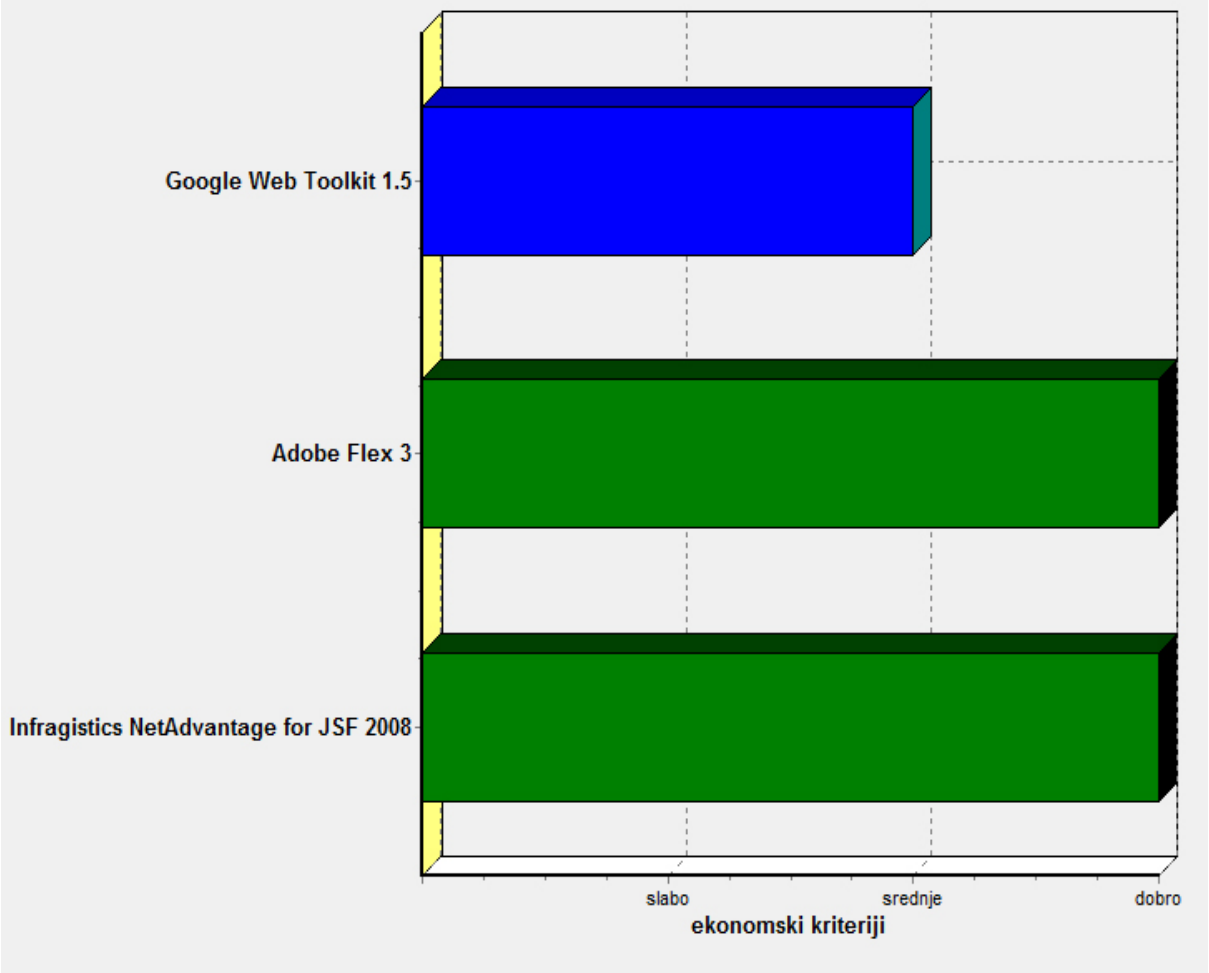
Slika 3.6.3: Grafični prikaz končnih ocen

Kot je razvidno iz zgornjega grafa, je končna odločitev odločno v prid Google Web Toolkita. Da bi videli zakaj, je potrebno pogledati podrobnejše prikaze za vsako skupino kriterijev posebej.



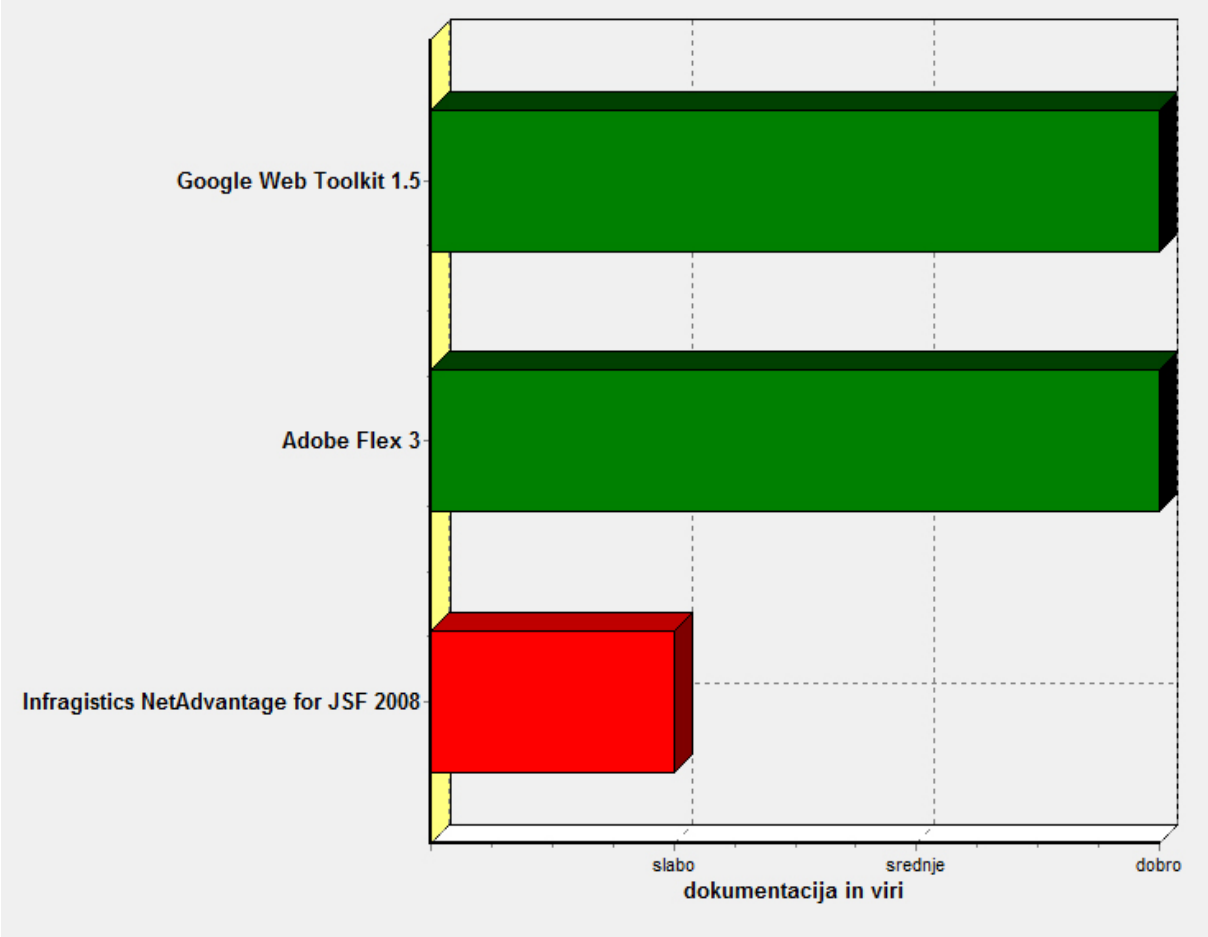
Slika 3.6.6: Grafični prikaz rezultatov funkcije koristnosti tehničnih kriterijev

V skupini tehničnih kriterijev so se vsa orodja odlično odrezala, zato te ocene praktično niso vplivale na končni rezultat.



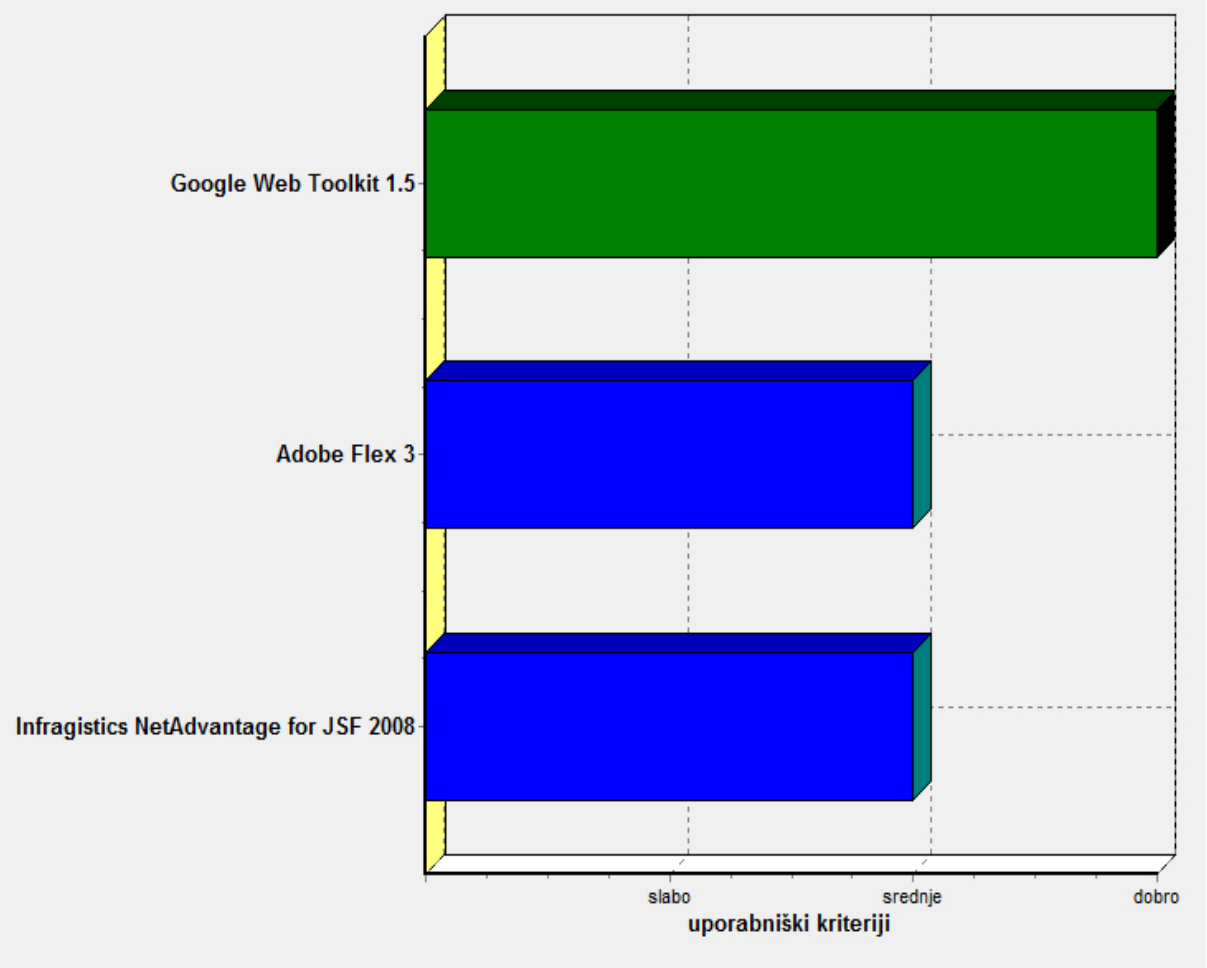
Slika 3.6.4: Grafični prikaz rezultatov funkcije koristnosti ekonomskih kriterijev

V skupini ekonomskih kriterijev ima NetAdvantage prednost pred konkurentoma, saj ne potrebuje dodatnih knjižnic, ki bi celotni rešitvi zvišale ceno. Kljub temu pa je tudi Flex dobil zelo pozitivne ocene, saj sto stroški njegove nadgradnje minimalni. Nasploh pa cene rešitev niso tako visoke, da si jih ne bi mogli privoščiti, zato ta kriterij ni pretehtal v korist NetAdvantage-a.



Slika3.6.7: Grafični prikaz rezultatov funkcije koristnosti dokumentacije in virov

Iz primerjave orodij po skupini kriterijev dokumentacija in viri je razvidno, zakaj ima Flex prednost pred NetAdvantage-em. V ostalih ocenah sta si namreč veliko bolj blizu, slaba dokumentacija oz. predvsem praktično neobstoječa spletna skupnost NetAdvantage-a pa je povzročila, da je njegova končna ocena nižja od Flexove.



Slika 3.6.5: Grafični prikaz rezultata funkcije koristnosti uporabniških kriterijev

Zgornja slika nazorno prikazuje, kaj je bilo odločilno pri zmagi Google Web Toolkita. Skupina uporabniških kriterijev namreč vsebuje funkcionalnost tabel, funkcionalnost grafov in dodatne komponente, ti kriteriji pa so ključnega pomena za zagotavljanje prijaznega in čim bolj funkcionalnega uporabniškega vmesnika. Ker se je GWT na tem področju najbolj izkazal, poleg tega pa nima drugih izjemno slabih lastnosti, ki bi mu končno oceno znižale, je tudi končna ocena temu primerna.

2.6.2. Kaj-če analiza

Na začetku se je NetAdvantage zdel idealna rešitev, saj ponuja vse želene kvalitete v enem samem produktu poleg tega pa je tudi poceni. Težave so nastale, ko sem poskušala skupaj z NetAdvantage komponentami uporabiti navadne JavaScript komponente. V primeru, da bi našli nadomestno rešitev, bi ocena kriterija ostale komponente postala bogate. Ta ocena posledično spremeni funkcionalnost v dobro, s tem pa se tudi ocena uporabniških kriterijev spremeni v dobro in končna ocena prav tako.
Kljub izboljšavi na področju uporabniških kriterijev, NetAdvantage še vedno ne zmaga, vendar si deli drugo mesto s Flexom. Če bi NetAdvantage for JSF imel tako veliko spletno

skupnost kot njegova ASP različica, bi se njegova ocena na tem področju spremenila v srednje, s tem bi se tudi ocena celotne skupine dokumentacija in viri spremenila v srednje, končna ocena NetAdvantage-a pa v zelo dobro. S tem bi se izenačil z Google Web Toolkitom, ki pa je dražji, zato bi NetAdvantage postal zmagovalna varianta.
Primerjajmo še Flex in Google Web Toolkit. Glavna razlika je v uporabniških kriterijih, kjer ima Flex pri funkcionalnosti tabel in ostalih komponentah. Če bi na trgu že obstajala kakšna implementacija Flex tabele, ki bi podpirala več funkcij, bi se ocena funkcionalnosti spremenila v dobro, s tem pa tudi uporabniški kriteriji. Končna ocena Flexa bi s tem postala zelo dobro in bi se izenačila z Google Web Toolkitom. Flex pa bi v tem primeru še vedno imel boljše ocene na področju uporabniškega vmesnika in cene ter s tem postal zmagovalec.

3. Sklepne ugotovitve

Spletne aplikacije vedno pogosteje nadomeščajo namizne aplikacije. Najpogosteje so v uporabi seveda spletni vmesniki za upravljanje z elektronsko pošto, tu pa so še vedno bolj popularni blogi, razne spletne strani za shranjevanje slik in foto albumov, storitve za izdelovanje in shranjevanje Wordovih, Excelovih ter podobnih dokumentov in še in še. Ta porast pa ni viden le pri uporabniških aplikacijah, tudi vedno več poslovnih aplikacij se seli na splet. Z večanjem povpraševanja pa se povečuje tudi ponudba in to ne le glede raznolikosti storitev, temveč tudi glede tehnologij, ki omogočajo razvoj takih aplikacij. Veliko tovrstnih tehnologij je še v povojih, zato je izbira prave toliko težja hkrati pa zelo pomembna. Mlade tehnologije se lahko izkažejo kot zelo inovativne in perspektivne, vendar to na vse večjem trgu programske opreme ni dovolj. Veliko vlogo igra na primer tudi marketing in zato se lahko zgodi, da se določen produkt čez nekaj let ne bo obdržal. Tu se naša izbira, Google Web Toolkit, ponovno izkaže. V primeru, da bi z leti njegov razvoj opustili, bi bil prehod obstoječe rešitve na čisti JavaScript razmeroma neboleč. Potrebno bi bilo le najti novo orodje za nadaljnji razvoj v JavaScriptu, GWT-Ext pa bi zamenjali z njegovim JavaScript ekvivalentom ExtJS-jem.

Po zaključenem diplomskem delu naravno sledi implementacija želene aplikacije v izbranem orodju. Prvi korak bo seveda prevedba obstoječe Birt aplikacije na novo tehnologijo, nato pa se bo razvoj nadaljeval v smeri izkoriščanja novih možnosti, ki jih nudi GWT, seveda pa vsakem koraku sledi tudi optimizacija. Google Web Toolkit ponuja še veliko možnosti na področju JSNI vmesnika, ki za potrebe odločitvenega modela niso bile v celoti raziskane. JSNI bo poleg tega še naprej uporaben za izkoriščanje že implementiranih čistih JavaScript komponent kot se je izkazalo na primer že pri uporabi FusionCharts grafov in drsnika za izbiro datumskega razpona pri poročilu »Spisek storitev.«

Prav tako se ponuja ogromno možnosti za implementacijo lastnih komponent iz obstoječih osnovnih GWT komponent z dodano v Javi napisano dodatno logiko. Lastne komponente lahko z Google Web Toolkitom izdelamo tako, da kombiniramo osnovne HTML elemente kot so gumbi, tabele, vnosna polja in podobno ter jim dodamo nove funkcionalnosti. Tako izdelane komponente lahko združimo v poseben objekt Widget in ga nato kjerkoli uporabljamo po istem principu, kot to počnemo z osnovnimi Google Web Toolkit komponentami. Na ta način bi lahko morda realizirali filtriranje podatkov pri tabelah, razne menije za izbor in prilagajanje prikaza vsebine, svojo pravo moč pa bi Google Web Toolkit na tem področju pokazal v primeru, da bi se odločili, da nanj prevedemo ne le poročila temveč tudi portal, ki je srce naše spletne aplikacije in od koder do samih poročil tudi dostopamo. Na portalu so namreč elementi bolj raznovrstni, potrebovali bi menije narejene po meri, maske za vnos podatkov in podobne elemente, ki jih pri poročilih ne srečamo.

Ker je v času pisanja tega dela razvoj aplikacije z Google Web Toolkitom že stekel, lahko dodam, da se je tudi kasneje GWT izkazal kot učinkovito orodje. Bilo je seveda nekaj

začetnih težav pri elementih, ki pri prototipu niso bili uporabljeni, prav tako je kar nekaj dela zahtevalo pisanje CSS datotek, ki skrbijo, da se izgled nove aplikacije ujema s celostno grafično podobo stare. Nekaj težav je bilo tudi s samim razvojnim okoljem, kar je povzročalo zamude oz. nepravilne ocene trajanja razvoja. Tovrstne težave pa so sedaj bolj ali manj odpravljene, zato lahko Google Web Toolkit pohvalim kot orodje, ki prinaša izjemne prihranke pri implementaciji JavaScript funkcij, ki podpirajo AJAX tehnologijo in s tem omogočajo razvijalcem izdelavo rešitev, ki uporabniku nudijo prijaznejšo, hitrejšo in bolj funkcionalno uporabniško izkušnjo.

Literatura

[1] A. Kompare, M.Stražišar, T. Vec, I. Dogša, N. Javšovec, J.Curk, Psihologija – spoznanja in dileme, Ljubljana: DZS, 2001

[2] D. Best, Web 2.0 Next Big Thing or Next Big Internet Bubble? Dostopno na: http://page.mi.fu-berlin.de/best/uni/WIS/Web2.pdf

[3] J. Spolsky, User interface design for programmers, New York: Apress 2001

[4] M. Bohanec, Odločanje in modeli, Ljubljana: DMFA 2006

[5] M. E. Davis, J. A. Phillips, Flex 3: A Beginner's Guide, McGraw-Hill Osborne Media 2008

[6] Različni avtorji, Veliki splošni leksikon: v osmih knjigah, Ljubljana: DZS 1998, sedma knjiga

[7]T. Ahmed, J. Hirschi, F. Abid, Flex 3 in Action, Manning Publications 2008

[8] T. O'Reilly, Web 2.0 Compact Definition: Trying Again. Dostopno na: http://radar.oreilly.com/2006/12/web-20-compact-definition-tryi.html

[9] W. O. Galitz, The essential guide to user interface design, An introduction to GUI design principles and techniques, Third Edition, Indianapolis: Wiley Publishing 2007

[10] Flex overview. Dostopno na: http://www.adobe.com/products/flex/overview/

[11] Google Web Toolkit, Product overview. Dostopno na: http://code.google.com/webtoolkit/overview.html

[12]ILOG Elixir 1.0 Developer's guide. Dostopno na: http://www.ilog.com/documentation/elixir10/en-US/Content/Visualization/Documentation/Elixir/_pubskel/index.html

[13]ILOG Elixir Forum. Dostopno na: http://forums.ilog.com/elixir/

[14] NetAdvantage for JSF, AJAX enabled JSF components. Dostopno na:
http://www.infragistics.com/java/netadvantage/jsf.aspx#Overview

[15] Rich internet application. Dostopno na:
http://en.wikipedia.org/wiki/Rich_Internet_application

[16]Web 2.0. Dostopno na:
http://en.wikipedia.org/wiki/Web_2

Izjava o avtorstvu

»Izjavljam, da sem diplomsko delo izdelala samostojno pod vodstvom mentorja doc. Dr. Marka Bajca. Izkazano pomoč drugih sodelavcev sem v celoti navedla v zahvali.«

Nina Krmac