UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Kralj

UČINKOVITA VIZUALIZACIJA DINAMIČNIH PODATKOV NA SPLETU

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: izr. prof. dr. Marko Bajec

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00002/2010
Datum: 04.11.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIHA KRALJ**

Naslov: **UČINKOVITA VIZUALIZACIJA DINAMIČNIH PODATKOV NA SPLETU**

EFFICIENT VISUALISATION OF DYNAMIC DATA USING WEB APPLICATIONS

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V diplomskem delu opišite, s kakšnimi tehnikami lahko rešujemo problem vizualizacije podatkov prek spleta, če gre za podatke, ki se frekventno spreminjajo in običajni pristopi (npr. z osveževanjem slike) zato niso primerni. Izbrane tehnike predstavite na primeru.

Mentor:

prof. dr. Marko Bajec

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Miha Kralj,

z vpisno številko 63960085,

sem avtor diplomskega dela z naslovom:

Učinkovita vizualizacija dinamičnih podatkov na spletu

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  izr. prof. dr. Marka Bajca
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 10.1.2011 Podpis avtorja:

Zahvala

Najprej bi se rad zahvalil staršem, ki so mi vedno stali ob strani in me podpirali. Rad bi se zahvali tudi ženi Miri in otrokoma za kakšno odtegnjeno urico, ki sem jo preživel za računalnikom, da sem lahko spisal to diplomsko nalogo.

Sofiji in Jakobu

Kazalo

Seznam uporabljenih kratic in simbolov

AJAX - Asynchronous Javascript And Xml
APC - Alternative PHP Cache
CSS - Cascading StyleSheet
CDN - Content Delivery Network
CMS - Content Management System
DOM - Document Object Model
ETag - Entity Tag
HMI - Human Machine Interface
JIT - Just In Time Compilation
JS - JavaScript
JSON - JavaScript Object Notation
HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol
PCV - Protocol Converter
PHP - skriptni jezik PHP
RDBMS - Relational Database Management System
RTU - Real Time Unit
SCADA - Supervisory Control And Data Acquisition
SVG - Scalable Vector Graphics
XHR - XmlHttpRequest
XML - eXtensible Markup Language

Povzetek

V sledečih poglavjih so predstavljeni ključni elementi pri izgradnji informacijske infrastrukture za učinkovito prestavitev hitro se spreminjajočih podatkov na spletu. Tako okolje je značilno za SCADA aplikacije, kjer imamo lahko več sto sprememb v sekundi. Te spremembe so večinoma predstavljene kot signalizacije in meritve na enopolnih shemah. Če bi naivno npr. enkrat na sekundo osvežili shemo v obliki bitne slike, bi pri več uporabnikih to pomenilo veliko breme za spletni strežnik. Poleg tega shema ki se tako pogosto osvežuje ni ravno privlačna za uporabnika. Namesto take naivne tehnike lahko uporabimo druge, ki bistveno razbremenijo spletni strežnik in izboljšajo uporabniški vmesnik. Tehnike prestavljene v tem tekstu so XHR oz. AJAX za prenašanje podatkov v ozadju, HTTP podatkovni tokovi za učinkovit prenos več podatkovnih paketov od strežnika do odjemalca znotraj ene HTTP povezave, medpomnilniki za shranjevanje že 'izračunanih' vsebin, ter SVG kot učinkovita predstavitvena plast. Ta diplomska naloga bo pokazala, da je s takim pristopom mogoče predstaviti dejansko stanje naprav kot jih 'vidi' SCADA, brez preskokov in z minimalnim časovnim zamikom. Poleg tega so vse uporabljen tehnologije podprte v vseh modernih brskalnikih brez kakršnih koli vtičnikov.

Ključne besede:

AJAX, SVG, SCADA, XHR, XmlHttpRequest

Abstract

The following chapters explain the reader the key parts of building an information infrastructure for effective representation of highly dynamic data on the internet. Such environment is characteristic for SCADA applications where there can be a few hundred updates per second. This updates are typically visualized as value changes on wiring diagrams. If we naively updated such a diagram as a bitmap image, let say once per second and we had a few diagrams updating at approximately the same time, there would be a lot of network traffic and a lot of processing on the server side. Besides, a bitmap that refreshes once per second in a browser doesn't contribute to the user experience. As an upgrade to such naive technique this paper shall give an overview of other means to achive the same goal, but with a fragment of time and server load. The techniques presented here are XHR (AJAX) for background data transfer, HTTP data streaming for multiple data package transfer from the server to the client whithin a single HTTP transaction, cache managers for caching data until they are updated and finally, SVG as an effective presentational layer. This thesis shall show that the above-mentioned techniques allow for real-time presentation of data from SCADA servers and other sources wih a minimal time lag. Furthermore, all the engaged techniques are natively supported by all major browsers without using plugins.

Key words:

AJAX, SVG, SCADA, XHR, XmlHttpRequest

Poglavje 1

Uvod

Tako kot na preostalih področjih računalništva in informatike se je v zadnjih letih tudi pri spletnih aplikacijah marsikaj spremenilo. Kljub temu, da je komunikacijski protokol ostal isti, torej HTTP, se je uporaba le tega s pojavom XHR razširila. Ravno ta instrument (XHR) je dal pribl. pred desetimi leti nov zagon spletnim aplikacijam.

1.1 XHR in HTTP podatkovni tok

XHR je asinhrono nalaganje podatkov v ozadju. Od tod tudi njegova sopomenka AJAX.[1] Potrebno je povedati, da sta obe okrajšavi, tako XHR kot AJAX nekoliko zavajajoči, ker navajata na to, da se v odgovoru na XHR nahaja niz XML. V primeru da strežnik na XHR povpraševanje ne odgovori s pravo informacijo o tipu dokumenta[2], ali da vsebina ni XML, potem ostane polje responseXML objekta XHR neveljavno. V takem primeru odjemalec sam interpretira vsebino. (glej pogl. 2)

1.2 SVG

SVG je naravna izbira za predstavitev enopolnih shem ker je vektorski format, ker je odprto-kodni in ker je podprt v vseh modernih brskalnikih. S tem so zagotovljene 3 osnovne zahteve:

- shema se skalira z velikostjo ekrana

[1] Zaradi tega ker je XmlHttpRequest izvorno ime, ki ga je Microsoft dal objektu, ki je služil temu namenu, bo v tem besedilu povsod uporabljena kratica XHR.

[2] header: Content-Type=text/xml

- shema je shranjena v odprto-kodnem formatu, tako da so možni uvozi in izvozi v druge formate npr. dxf, dwg, ...
- shema je v brskalniku predstavljena kot objekt DOM, tako da se zlije z ostalimi spletnimi vsebinami

1.3 Medpomnilniki

Po vlogi v spletnih aplikacijah delimo medpomnilnike na dva razreda:

- medpomnilnik za shranjevanje že interpretirane kode PHP (XCache)
- medpomnilnik za shranjevanje podatkov (XCache, memcached)

Prvi je koristen za shranjevanje že interpretirane kode PHP. S tem prihranimo čas, ki ga rabi PHP za interpretacijo skript. Hitrost izvajanja se s tem lahko primerja z ostalimi JIT okolji.

Slednji pa je 'ta pravi' medpomnilnik, ki služi shranjevanju vseh podatkov, ki so potrebni za delovanje informacijskega sistema. Ta medpomnilnik pa lahko ponovno razdelimo v dva razreda:

- medpomnilnik za shranjevanje podatkov o stanju aplikacije (XCache)
- medpomnilnik za shranjevanje podatkov SCADA (memcached)

Poglavje 2

Prenos podatkov: XHR in HTTP tokovi

To poglavje odgovori na vprašanje: "Kako prenašamo podatke?"

2.1 Protokol HTTP

Takoj moramo povedati, da protokol HTTP ni naravna izbira za vizualizacijo dinamičnih podatkov. Transakcija HTTP sestoji iz zahteve in odgovora. Začne se tako, da odjemalec sproži zahtevo in strežnik na to zahtevo odgovori. Tak model prenosa podatkov je tipičen za splet: uporabnik klikne na povezavo[1] in strežnik odgovori z zahtevano vsebino, ki jo brskalnik pokaže uporabniku. S tem je krog sklenjen. Problem se pojavi, ko si poskušamo odgovoriti na vprašanje: "Kaj pa, če se neka sprememba, ki bi zahtevala osveženo sliko podatkov, zgodi nekje na strežniku?"[2] V tem primeru bo uporabnik videl spremembo šele, ko bo sprožil ustrezno zahtevo. V primeru nadzora procesov v realnem času je tak model neuporaben. Edina rešitev do uvedbe XHR (glej pogl. 2.2), je bila samodejno periodično osveževanje strani HTML.

Tak način osveževanja ima sledeče dobre lastnosti:

- je zelo preprost za implementacijo. Zadošča že 1 funkcija, ki periodično osvežuje stran.

```
setInterval(''window.location.reload()'',1000); // 1x / sek.
reload
```

[1] Ali kako drugače sproži zahtevo

[2] Ne nujno na strežniku HTTP. Kot bomo videli se spremembe vršijo na strežnikih SCADA.

- ima koristen stranski učinek, t.j. ponovno nalaganje strani povzroči brisanje programskih smeti, ki so se nabrale (garbage collection)
- spletni uporabnik je navajen nalagati novo vsebino s klikom na povezavo[3] ali kar z gumbom v brskalniku[4]

Kljub temu pa ima tudi lastnosti, ki so za uporabnika moteče. To je:

- velika potrata pasovne širine.[5]
- vizuelno se osveževanje strani vidi kot utripanje, ki je za uporabnika, ki bere podatke moteče.
- veliko število zahtev HTTP obremenjuje tako strežnik kot tudi brskalnik.

Zaradi zgoraj naštetega, moramo pri implementaciji uporabniku prijaznega in hkrati učinkovitega informacijskega sistema uporabtiti programska orodja, ki so opisana v naslednjih dveh poglavjih.

2.2 XHR

XHR ali XmlHttpRequest se razlikuje od navadne zahteve HTTP samo v tem, da se izvrši v ozadju.[6] XHR je objekt. Je programski konstrukt. Najbolje si ga predstavljamo kot brskalnik v brskalniku. Ima lastnosti in funkcije ki programerju omogočajo, da sproži HTTP zahtevo in dobi odgovor, ki je tudi shranjen kot lastnost XHR objekta. To nam omogoča, da zahtevamo podatke od strežnika ne da bi morali za to osvežiti stran, ki jo trenutno gledamo. Tako pridobljene podatke lahko shranimo ali uporabimo za delno posodobitev strani.[7] Tak pristop rešuje prvi dve pomanjkljivosti zgoraj omenjenega samodejnega periodičnega osveževanja strani.

XHR ustvari pogoje za bistveno boljšo uporabniško izkušnjo. Kljub temu pa je XHR še vedno periodično osveževanje. To pa pomeni, da je potrebno še vedno določiti periodo osveževanja. Če je ta predolga, potem je zaupanje v podatke[8] slabo. Če pa je prekratka, potem je obremenitev strežnika velika.

[3] link

[4] Refresh

[5] Nekaj 10kb za vsako zahtevo. Za dober občutek svežine zelo dinamičnih podatkov je potrebno osveževanje vsaj 1x na sekundo. Več je bolje.

[6] V ozadju pomeni, da je uporabniku storitev neviden, programerju pa je seveda viden

[7] Stran posodobimo z operacijami nad DOM drevesom.

[8] npr. sliko enopolne sheme

V skrajnem primeru, če je veliko odjemalcev lahko pridemo s krajšanjem periode osveževanja do obratnega učinka, t.j. da se reakcijski čas[9] celo podaljša. Določiti "pravo" periodo osveževanja pa ni enostavno. Odvisna je od dinamike spreminjanja podatkov[10] in od količine prikazanih podatkov[11].

Na voljo imamo dva načina za določitev periode osveževanja.

- dinamično prilagajanje dolžine periode osveževanja glede na število sprememb. Sledi algoritem.

```
periodaOsv = 1000; //zacetna perioda osv. v ms
periodaStPodatkov = 5; //kriterij za spremembo periode
setTimeout("osveziPodatke();",periodaOsv);
function osveziPodatke() {
 var noviPodatki = getPodatkiFromServer();
 if (count(noviPodatki) >= periodaStPodatkov) {
  periodaOsv = periodaOsv / 2; // podatki so zelo dinamični
 } else {
  periodaOsv = periodaOsv * 2; // podatki so manj dinamični
 }
 setTimeout("osveziPodatke()",periodaOsv);
}
```

  V zgornjem algoritmu je potrebno še določiti spodnjo in zgornjo mejo periode osveževanja. Namreč, če bi bila perioda prekratka, bi preveč obremenjevali strežnik. Če bi pa bila predolga, bi v primeru nenadne spremembe predolgo čakali na osvežitev podatkov na sliki kot jo vidi uporabnik. Ta metoda je dobra če se količina sprememb na časovno enoto relativno zvezno spreminja. V robnih primerih pa ta algoritem odpove. Z določitvijo spodnje in zgornje meje osveževanja se sicer izognemo "neskončnemu čakanju na spremembo" ali preobremenjevanju strežnika. Vseeno pa lahko vidimo, da bomo lahko v najslabšem primeru morali na spremembo na sliki čakati toliko čas, kolikor smo določili zgornjo mejo. V praksi se pokaže, da če imamo na enopolni shemi (glej pogl. 4) vsaj nekaj meritev ali števcev[12], potem tak sistem določanja periode dobro deluje. V nasprotne primeru pa ne tako dobro[13]. Če XHR ne bi omogočal

[9] to je čas od dejanske spremembe podatkov do spremembe podatkov na sliki, kot jo vidi uporabnik

[10] meritve in števci se običajno hitreje spreminjajo kot signalizacije; glej pogl. 4

[11] npr. velikost enopolne sheme; glej pogl. 4

[12] Meritve in števci se dokaj pogosto spreminjajo

[13] Signalizacije se praviloma redkeje spreminjajo

branja podatkov med transakcijo HTTP, potem bi se morali zadovoljiti s takim načinom regulacije dinamike osveževanja.

- HTTP tokovi (glej pogl. 2.3)

2.3 HTTP tokovi

> Podatkovni tok HTTP je neprekinjena sekvenca podatkov, katere izvor je strežnik, ponor pa odjemalec.

Podatkovni tokovi HTTP se že nekaj časa uporabljajo za prenašanje zvočnih in video vsebin preko svetovnega spleta. Od navadne transakcije HTTP se razlikuje le po tem, da se podatki berejo, interpretirajo in posledično vizualizirajo že MED transakcijo HTTP in ne šele ko je transakcija HTTP zaključena.

Na tem mestu je potrebno navesti ključno lastnost objekta XHR brez katere podatkovni tokovi HTTP ne bi obstajali. Objekt XHR se lahko nahaja v petih stanjih[14]:

- 0 - po stvaritvi
- 1 - po uspešni povezavi
- 2 - ko je bilo povpraševanje HTTP poslano in je bila sprejeta glava odgovora
- 3 - ko se nalaga vsebina trupa odgovora
- 4 - po zaključenem prenosu

[2, 3, 4]

Objekt XHR nam omogoča, da pripnemo poljubno funkcijo na spremembo stanja t.j. "onReadyStateChange".

- XHR

```
xhr = new XmlHttpRequest();
xhr.onReadyStateChange = function(readyState) {
 if (readyState == 4) {
  var paket = this.responseText;
```

[14]readystate

```
    // koda za update slike
    ...
  }
 }
```

- XHR + tok HTTP

```
xhr = new XmlHttpRequest();
var xInt, textPos=0, textBuf="", delimiter="\n";
xhr.onReadyStateChange = function(readyState) {
 if (readyState == 3) {
  xInt = setInterval("readResponseText();",50); //20x na sek.
 } else if (readyState == 4) {
  // prenos zakljucen; ni potrebno storiti nicesar
  // ker so ze vsi podatki preneseni
  clearInterval(xInt);
 }
}
function readResponseText() {
 while (true) {
  var delimiterPos = xhr.responseText.substrpos(delimiter, textPos);
  if (delimiterPos == -1) {
   //beremo od pozicije kjer smo v prejsnji iteraciji koncali
   textBuf = textBuf + xhr.responseText.substring(textPos);
   break;
  } else {
   var paket = textBuf + xhr.responseText.substring(textPos,
delimiterPos-1);
   // koda za update slike
   ...
   textPos = delimiterPos+1;
   textBuf = "";
  }
 }
}
```

Ključna razlika med XHR in XHR + tok HTTP je v tem, da v slednjem v stanju 3 ustvarimo funkcijo, ki v nekem kratkem intervalu bere vsebino responseText objekta XHR ko je le ta v stanju 3. V stanju 4 pa le prekinemo proces. Funkcija "readResponseText" samo poskrbi, da se berejo paketi znotraj ene

transakcije HTTP. Paketi so ločeni z nekim ločilom, v zgornjem primeru je to NEWLINE, ki se ne pojavlja znotraj paketa.

Z besedami povedano pa funkcija "readResponseText" deluje tako: Najprej določimo pozicijo prvega najdenega ločila od pozicije textPos naprej. Do pozicije "textPos" smo poskrbeli že v prejšnji iteraciji. V primeru "delimiterPos == -1" ,t.j. ko ni bilo končnega ločila pomeni, da paket še ni bil v celoti poslan naredimo sledeče: v "textBuf" dodamo vse od pozicije "textPos" naprej. "textBuf" hrani del paketa. Ker nismo našli ločila pomeni da je to zadnji paket, zato lahko zapustimo neskončno zanko "while(true)". Sicer pa, če je v nizu "responseText" od pozicije "textPos" končno ločilo, potem združimo del paketa prebranega v prejšnjih iteracijah "textBuf" in del niza od "textPos" do ločila; to je sedaj cel paket, ki ga lahko naprej obravnavamo. Na koncu je potrebno seveda še pomakniti "textPos" na novo pozicijo za prvim ločilom in zbrisati "textBuf". Vse se ponovi.

Za delovanje toka HTTP pa mora biti prilagojen tudi strežnik. Namreč, v prvem primeru, ko gre za navadno XHR transakcijo strežnik odgovor posreduje in zaključi transakcijo. V drugem primeru pa se postavi v zanko in polni izhodni tok s paketi, ki jih potem odjemalec interpretira. Delovanje strežnika, ki je prilagojen za podatkovne tokove HTTP lahko ponazorimo z naslednjim algoritmom:

```
zanka: trenutni_cas - zacetni_cas < maksimalni_cas_zveze_http
paket = nov paket
vpisi paket na izhodni tok podatkov
vpisi locilo na izhodni tok podatkov
konec zanke
```

Podatkovnemu toku lahko z drugimi besedami rečemo tudi podatkovni kanal, ker se v eni transakciji HTTP prenaša več podatkovnih paketov.

* Pri podatkovnih tokovih se lahko pojavi težava. Če uporabimo za podatkovni strežnik katerega od javno dostopnih HTTP strežnikov (npr. Apache+PHP) se lahko zgodi, da tokovi HTTP ne bodo pravilno delovali zaradi vmesnih pomnilnikov (buffer-jev). Posledica je, da se vsebina, ki se pojavi na izhodnem toku strežnika ne prenese takoj do odjemalca ampak šele, ko je vsebina medpomnilnika polna ali ko se skripta (npr. PHP) zaključi. Tako delovanje se pozna kot zakasnitev v prenosu podatkov, kar praktično izniči dodaten trud, ki smo ga vložili za implementacijo podatkovnih tokov. K sreči je mogoče s pravilno konfiguracijo HTTP strežnika tako med-pomnjenje izključiti.[15]

[15] v skriptnem jeziku PHP je to parameter "OutputBuffering = Off" v php.ini; glej dokumentacijo za PHP

Poglavje 3

Razpoložljivost podatkov: medpomnilniki

To poglavje odgovori na vprašanje "Kje se podatki nahajajo?"

3.1 Medpomnilnik za shranjevanje odgovorov HTTP na strani odjemalca

Ne da bi se spuščali v podrobnosti specifikacije HTTP protokola lahko izluščimo naslednje parametre, ki so pomembni za dobro delovanje vsakega sistema.

- Če je nek resurs že prisoten na odjemalčevi strani in je veljaven potem sploh ne izvedemo zahteve.
- Če je nek resurs prisoten na odjemalčevi strani in mu je veljavnost potekla potem s pošiljanjem glave HTTP zahteve brskalnik avtomatsko pridobi podatke o veljavnosti resursa; če je še veljaven potem se trupa sploh ne prenese sicer pa ja. Na tak način se izognemo pošiljanju trupa odgovora na zahtevo HTTP v kolikor to ni potrebno.

Pomembno je še razlikovanje zahteve HTTP med način GET in POST. Če sprožimo zahtevo HTTP v načinu GET potem se sploh omogoča shranjevanje resursov sicer, če uporabimo POST je shranjevanje inherentno onemogočeno. GET ima parametre v glavi in jih uporablja kot ključ v tabeli keširanih resursov. POST pa ima parametre v telesu in zato se keširanje sploh ne izvede. Z uporabo GET/POST reguliramo keširanje resursov.

Uporabljeni parametri glave zahteve HTTP, ki zadevajo medpomnilnik so:
Cache-Control:

- max-age=n - maksimalno št. sekund od prejema odgovora v obdobju katerih je resurs veljaven (zahteva se ne izvrši)
- must-revalidate - pove brskalniku kaj mora storiti ko nek resurs ni več veljaven (mora zahtevati veljavnost)

Z konfiguracijo:"Cache-Control: max-age=0, must-revalidate" dosežemo, da brskalnik preveri veljavnost vsakega resursa, ki smo ga pridobili z XHR. To je zelo pomembno, če hočemo imeti vedno sveže vsebine, kar velja za vse dinamične vsebine. Nekako lahko zaključimo s takim sklepanjem:

Če smo uporabili XHR in podatkovne tokove za pridobivanje vsebine, potem si seveda ne želimo stare vsebine in zato so smiselne zgoraj omenjene nastavitve.

V sledeči poglavjih so opisani medpomnilniki na strani strežnika HTTP.

3.2 Medpomnilnik za shranjevanje binarne kode PHP

Zaradi kompleksnosti sodobnih aplikacij na HTTP strežnikih (lahko več 10000 vrstic) je medpomnilnik (ang. Cache) za shranjevanje binarne kode PHP ključnega pomena pri zagotavljanju kratkih odzivnih časov strežnikov HTTP. To je gonilnik, ki skrbi za to, da se določena skripta PHP ne interpretira vedno znova pri vsaki transakciji HTTP. Interpretira oz. prevaja se "ravno ob pravem času" oz. ang. "Just in Time" - JIT. S tem dosežemo, da se procesorski čas na strežniku smotrno porabi za "prave" operacije in ne za prevajanje kode. Za PHP obstaja več takoh gonilnikov, med njimi APC, XCache, PHP Accelerator, idr.

V implementaciji informacijskega sistema (glej pogl. 5) je uporabljen programski paket XCache[1].

3.3 Medpomnilnik za shranjevanje podatkov

Razen transportne poti je lahko ozko grlo pri vizualizaciji podatkov tudi dostop do podatkov. V idealnem svetu bi se podatki nahajali vedno v dinamičnem spominu računalnika na katerem je tudi strežnik HTTP. V realnem svetu pa temu skoraj nikoli ni tako, razen v trivialnih informacijskih sistemih. V srednjih in velikih sistemih, je strežnik HTTP samostojen računalnik ki mora

[1] xcache.lighttpd.net

na tak ali drugačen način dostopati do raznorodnih podatkov med katerimi so razne konfiguracije, dokumenti, strežniki SCADA (ki jih je navadno več - redundanca), baze podatkov, idr.

Takoj lahko ugotovimo, da očitno ne moremo imeti vseh podatkov v dinamičnem spominu lokalnega strežnika HTTP. Medpomnilniki se ponudijo kot rešitev tega problema. Vstopijo kot vmesni člen.

- brez medpomnilnika

```
-> zahteva za dostop do resursa
dostop do resursa
<- resurs v odgovoru na zahtevo
```

- modpomnilnik

```
-> zahteva za dostop do resursa
če je resurs v medpomnilniku vrni resurs
sicer dostop do resursa
resurs v medpomnilnik
<- resurs v odgovoru na zahtevo
```

3.3.1 Medpomnilnik za shranjevanje podatkov o stanju aplikacije

Stanje aplikacije je vsebina vseh elementov aplikacije oz. informacijskega sistema. Sem spadajo:

- konfiguracija aplikacije (glej pogl. 5)
- podatkovni kanali (glej 2.3)

Kot programsko orodje za implementacijo tega medpomnilnika je uporabljen PHP paket XCache[2].

Pomnjenje tovrstnih spremenljivk je potrebno zaradi narave spletnih aplikacij.

[2]Ker ponuja tudi pomnjenje poljubnih spremenljivk po ključu t.j. ime=vrednost

3.3.2 Medpomnilnik za shranjevanje podatkov SCADA

Podatki SCADA so vsi podatki v realnem času pridobljeni iz strežnikov SCADA, iz podatkovnih baz ali direktno iz perifernih postaj RTU. Lahko jih razvrstimo v naslednje osnovne skupine:

- stanja - stanje podatkovnih točk (trenutno stanje merilnikov oz. instrumentov) [stanje]
- trendi - zgodovina analognih in številčnih meritev [grafi]
- kronologija - zgodovina signalizacij [dnevnik]

V tekoči implementaciji informacijskega sistema je implementirana le 1. točka, torej stanje; trendi in kronologija se berejo iz relacijske baze podatkov (Glej razdelek. 6.1).

Kot programsko orodje za implementacijo tega medpomnilnika je uporabljen zelo razširjen paket memcache[3]. Memcache uporabljajo vsi večji ponudniki internetnih spletnih vsebin, kar nakazuje na kvaliteto vmesnika, zanesljivost, skalabilnost in vsesplošno odzivnost. Njegova osnovna lastnost in razlog za uporabo v SCADA sistemih in v predstavljenem informacijskem sistemu je porazdeljenost.

Memcache je sestavljen iz dveh delov.

- memcached - memcache daemon je program, ki se zažene povsod tam, kjer imamo na razpolago neuporabljen spomin (dinamični)
- memcache client - odjemalec je knjižnica, ki jo uporabimo za interakcijo z vsem pomnilniškim prostorom, ki ga imamo na razpolago

To pomeni, da lahko praktično vsak računalnik v danem omrežju uporabimo kot del gruče, ki se uporablja za hrambo stanj trendov in kronologije. V predstavljenem informacijskem sistemu se uporablja gruča dveh računalnikov, SCADA strežnik A in SCADA strežnik B. Tak pristop omogoča redundanco podatkov. Prisotnost podatkov je tako praktično 100Iz prej omenjenega je razvidno, da je memcache protokol mrežni protokol, natančneje TCP protokol. SCADA strežniki polnijo medpomnilnik, spletna aplikacija na spletnem strežniku pa ga bere. To nakazuje, da v predstavljenem informacijskem sistemu uporabljamo memcache nekoliko drugače kot se sicer uporablja. Navada je, da se nek podatek najprej poskuša prebrati iz medpomnilnika in v kolikor tega podatka ni, se ga prenese od izvora v medpomnilnik in se ga šele

[3] www.memcached.org

potem posreduje odjemalcu. V našem primeru pa se podatki ne prenašajo na povpraševanje ampak so v naprej pripravljeni. Namreč, medpomnilnik za podatke v realnem času v klasičnem smislu ne bi imel nobenega smisla, ker bi skoraj vedno imeli zgrešitev. Čas posameznega dostopa bi se tako gotovo povečal. Zelo dinamične podatke moramo namreč v kratkem času razveljaviti, da zagotovimo svežino. V našem primeru uporabljamo memcache kot hranilnik podatkov dostopen vsem, ki te podatke potrebujejo. Podatki se nikoli ne razveljavljajo (brišejo), se le posodabljajo. Potrebno je povedati da postavitev memcache na SCADA strežnik ni edina možna. Memcache bi lahko bil konec koncev lociran na spletnem strežniku. S tem bi bil dostop do podatkov hitrejši [4]. Proti taki izbiri pa govorijo sledeči razlogi.

- Tipična frekvenca vpisa podatkov v memcached nekaj 100 do 1000 / sek. bi zelo obremenilo lokalno mrežo.
- Hitrost dostopa do podatkov SCADA za spletno aplikacijo je sicer pomembna, je pa vseeno omejena s prepustnostjo zunanjega vmesnika, ki je tipično kar nekaj krat manjša od prepustnosti lokalne mreže.

[4] testiranja so pokazala, da je lokalni dostop pribl. 10x hitrejši kot dostop do oddaljenega memcache, 5000 proti 50000 r/w operacij na sekundo

Poglavje 4

Predstavitev podatkov: tehnološke sheme

Do tega poglavja je bil predstavljen podatkovni vidik informacijskega sistema. V sledečih razdelkih pa bo predstavljen vizualizacijski vidik informacijskega sistema WebSCADA CMS.

Predstavitev podatkov delimo v sledeče sklope:

- tabelarične - spiski meritev, signalizacij, števcev, alarmne liste, dnevniki
- grafične - grafi, tehnološke sheme
- poročila - kombinacija prejšnjih dveh

Ker so tehnološke sheme (enopolne sheme v energetiki) najpomembnejša referenca za spremljanje delovanja sistema in ker so od prej naštetih le te avtorsko delo, bodo v nadaljevanju podrobno predstavljene. Tabelarična predstavitev je del programskega okolja v katerem je implementiran vizualni del informacijskega sistema WebSCADA CMS. To je ExtJS podjetja Sencha[1]. Grafi pa so predstavljeni s pomočjo chart animacije, ki je posebna animacija za predstavitev grafov znotraj tehnološke sheme.

4.1 SVG

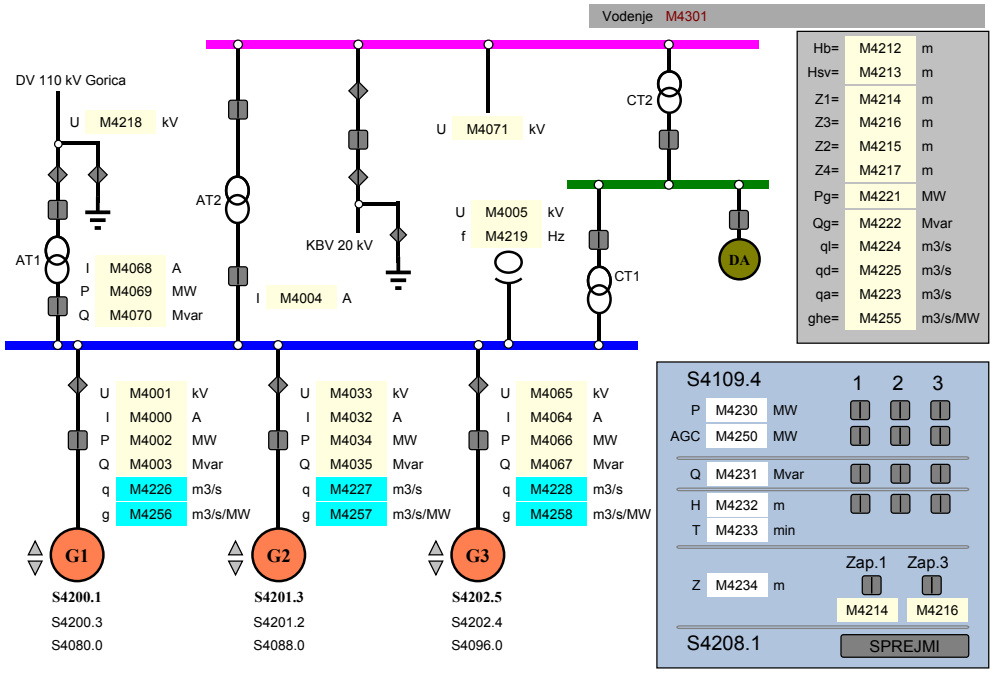
SVG je naravna izbira za predstavitev tehnoloških shem, ker je odprt standard, ker je vektorski format in ker je XML; torej spletu prijazen. SVG standard je star približno 10 let in se je v tem času dovolj prijel med razvijalci spletnih

[1] www.sencha.com

brskalnikov. Danes praktično ni več brskalnika, ki ne bi podpiral SVG. Res je, da noben brskalnik ne podpira SVG standarda verzija 1.1 v celoti, ampak to sploh ni pomembno. Pomembno je, da so podprti osnovni risarski elementi. Specifikacija SVG je dolga več kot 1000 strani in je dostopna bralcu na spletu[2].

4.2 Tehnološke sheme in animacije

Tehnološke sheme so najpomembnejše orodje za spremljanje dogajanja v nekem sistemu. V elektro proizvodnji in distribuciji se imenujejo enopolne sheme. Enopolna shema predstavlja pogled na trenutno stanje aparatur v delu celotnega sistema. Z drugimi besedami, enopolna shema je vizualna predstavitev nekega nabora podatkov v realnem času. Primer je podan na sliki.



Slika 4.1: Enopolna shema HE Solkan.

S pomočjo enopolnih shem dispečerji spremljajo dogajanje v elektrarni v realnem času in ustrezno reagirajo. Vse ostale oblike predstavitev so pomembne za analizo zgodovine, sledenju dogajanja v nekem časovne razdobju, statistiko, odkrivanju napak, preučevanje razlogov za neko odločitev, ipd.

[2]www.w3.org/Graphics/SVG

4.2.1 Prevedba enopolnih shem iz formata matične SCADA aplikacije

Enopolnih shem je v sistemu SENG več kot 100. Vsaka od njih ima od nekaj deset do nekaj sto podatkovnih točk - grafičnih elementov. Ponovno risanje vseh shem bi bilo zelo zamudno. Ocenil sem, da bi porabil 300-500 ur, to je več kot 2 meseca. V matični SCADA aplikaciji so vse sheme že narisane tako da je bilo smiselno vložiti napor v prevedbo teh shem v SVG. Program za prevajanje shem matične SCADA aplikacije ViSpro v SVG: vispro2svg.php je realiziran kot prevajalnik z orodjem FSM Parser Class[3] in je tudi avtorsko delo.

4.2.2 Animacije

Animacije so vse vidne spremembe na enopolnih shemah. Vsak grafični element enopolne sheme, ki se spremeni ob spremembi vrednosti neke podatkovne točke (ali več podatkovnih točk) pravimo da je animiran. Iz povedanega je razvidno, da mora vsak animiran grafični element vsebovati neko informacijo o tem ob katerem dogodku se animacija sproži in in kako se element ob tem dogodku spremeni. To informacijo hranimo v atributu: "linkScadaConfig". Vsebina tega XML atributa je serializiran JSON objekt. Natančen opis sledi spodnjemu primeru. Ključno je spoznanje, da tako kot lahko kompleksne elemente sestavimo iz preprostih, tako lahko kompleksne animacije se stavimo iz preprostih.

Primer. Predstavljajmo si tipičen potenciometer, ki lahko rotira okrog svoje osi in hkrati v svoji sredini prikazuje neko meritev, hkrati pa ta potenciometer spreminja barvo glede na vrednost od zelene preko rumene in oranžne do rdeče. Na prvi pogled lahko tak zgleda kompleksen. Po premisleku pa vidimo, da je tak element v resnici sestavljen iz treh elementov in treh animacij.

Grafični elementi: krog, pika, tekst Animacije:

- rotacija - krog s piko: 0-270 st. (animacija je na grupi krog + pika)
- barva - krog: zelena - rdeča (animacija je na krogu)
- tekst - tekst: vrednost meritve (animacija je na tekstu)

Del SVG zapisa ki ponazarja tak SCADA grafični element:

[3]Avtor: Dmitry A. Kirilin, dostopno na naslovu: http://www.phpclasses.org/package/2807-PHP-Generic-parser-using-finite-state-machine.html

Slika 4.2: Tipičen (malo poenostavljen) potenciometer, kot ga srečamo v HMI/SCADA aplikacijah.

```
<g id="potenciometer1">
 <g>
  <circle cx="50" cy="50" r="40" style="fill:darkgreen; stroke:black;
stroke-width:2"/>
  <circle cx="30" cy="75" r="3" style="fill:silver; stroke:white;
stroke-width:1"/>
 </g>
 <text x="50" y="56" style="fill:silver; stroke:white; stroke-width:1;
font-size:16px; font-weight:800; text-anchor:middle">M1000</text>
</g>
```

Skupaj z atributom "linkScadaConfig"pa:

```
<g id="potenciometer1">
 <g linkScadaConfig="{rotatef:{tag:\"M1000\", name:\"rotateAngle\",
params:[0,270]}}">
  <circle linkScadaConfig="{colorf:{tag:\"M1000\", name:\"rotateColor\",
params:[0.75, \"green\", 0.85, \"yellow\", 0.95, \"orange\", \"red\"]}}"
cx="50" cy="50" r="40" style="fill:darkgreen; stroke:black; stroke-width:2"/>
  <circle cx="30" cy="75" r="3" style="fill:silver; stroke:white;
stroke-width:1"/>
 </g>
 <text linkScadaConfig="{textf:{tag:\"M1000\", name:\"getTagValue\",
params:[]}}" x="50" y="56" style="fill:silver; stroke:white; stroke-width:1;
font-size:16px; font-weight:800; text-anchor:middle">M1000</text>
</g>
```

```
//
// funkcija: rotateAngle
// opis: vrne kot potenciometra 0-270 st. glede vrednost in na omejitve
pod. točke
// parametri:
// - min_angle - minimalni kot (kot pri dp.min)
// - max_angle - maksimalni kot (kot pri dp.max)
//
function rotateAngle(tag,min_angle,max_angle) {
var dp = getDp(tag); // podatkovna tocka
var perc = (dp.value-dp.min)/(dp.max-dp.min);
return (min_angle + perc*(max_angle-min_angle)); // izračunan kot
glede na omejitve pod. točke
}

//
// funkcija: rotateColor
// opis: vrne barvo potenciometra glede na vrednost pod. točke in
omejitve
// parametri:
// - prva meja izrazena v delu celote
// - zacetna barva
// - druga meja izrazena v delu celote
// - druga barva
// ...
// - zadnja barva
//
function rotateColor() {
var arg, tag = arguments[0];
var dp = getDp(tag); // podatkovna tocka
var perc = (dp.value-dp.min)/(dp.max-dp.min);
var stop = 0;
for (var i=1;i<arguments.length;i++) {
arg = arguments[i];
if (typeof arg == 'number') {
stop = arg;
} else {
if (perc<=stop) {
```

```
return arg;
}
}
}
return arg;
}

//
// funkcija: getTagValue
// opis: vrne vrednost pod. točke
// parametri:
//
function getTagValue(tag) {
var dp = getDp(tag); // podatkovna tocka
return dp.value;
}
```

Sledi spisek vseh animacij, ki se pojavljajo v enopolnih shemah s kratkim opisom.

- color animacija barve polnila grafičnega elementa glede na 4 stanja dvobitne signalizacije
- colorf animacija barve polnila grafičnega elementa glede na poljubno funkcijo, ki vrne barvo polnila (glej primer zgoraj)
- visibility animacija vidnosti gr. el. glede na 4 stanja dvobitne signalizacije
- visibilityf animacija vidnosti gr. el. glede poljubno funkcijo ki vrne stanje vidljivosti "visible"ali "hidden"
- blink animacija utripanja gr. elementa glede na stanje signalizacije; parametri so tip: 0 - utripanje z atributom visibility, 1 - utripanje z atributom fill, stanje: stanje v katerem gr. el. utripa, interval: hitrost utripanja v milisekundah, za utripanje z atributom fill pa še barva 1 in barva 2
- blinkf animacija utripanja gr. elementa glede na poljubno funkcijo ki vrne stanje v katerem element utripa; ostalo je enako kot blink
- rotate animacija kota rotacije gr. elementa okrog svoje osi glede na stanje dvobitne signalizacije

- rotatef animacija kota rotacije gr. elementa okrog svoje osi glede na poljubno funkcijo ki vrne kot rotacije (glej primer zgoraj)
- text animacija teksta gr. el. (veljavna samo za SVG ¡text¿) glede na vrednost dvobitne signalizacije ali meritve, vr. 0-9
- textf animacija teksta glede na poljubno funkcijo ki vrne poljuben tekstovni niz (glej primer zgoraj)
- bar animacija stolpca glede na stanje signalizacije ali vr. meritve; parametri so: min - minimalna vrednost (stolpec prazen, ni polnila), max - maksimalna vrednost (stolpec poln); animacija je primerna za sode, rezervoarje, ipd.
- barf animacija stolpca glede na poljubno funkcijo ki vrne delež polnosti stolpca 0-1
- execute to dejansko ni animacija; implementira dogodek ob kliku z miško na gr. el.
- tooltip to dejansko ni animacija; implementira dogodek ko je kurzor miške pozicioniran nad gr. el. Uporabljamo za hint; statični tekst
- tooltipf to dejansko ni animacija; implementira dogodek ko je kurzor miške pozicioniran nad gr. el. Uporabljamo za hint; dinamični tekst
- chart implementira graf kot animacijo; na tak način lahko prestavimo grafe znotraj enopolnih shem in obratno (več v razdelku 4.2.4)

Izkaže se da lahko s temi osnovnimi animacijami ustvarimo zelo kompleksne in "vizualno privlačne"enopolne sheme.

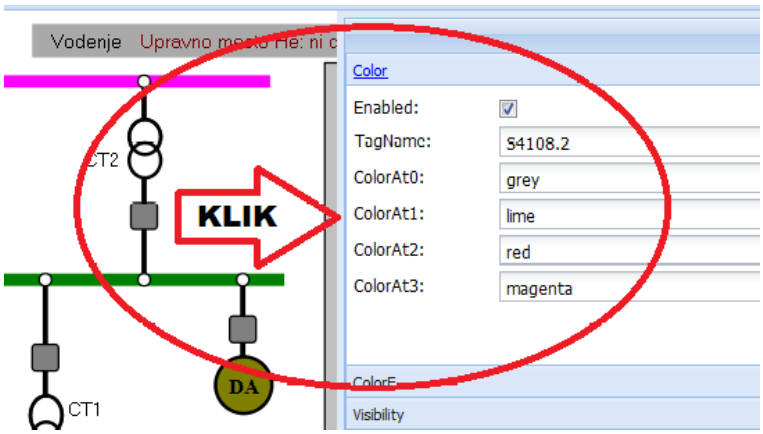
4.2.3 Urejanje enopolnih shem

V prejšnjem razdelku je podan opis delovanja animacij. Nič pa ni bilo omenjeno kako z animacijami "opremimošvg sliko da le-ta postane "živa"enopolna shema. Seveda lahko atribute "ročno"vpišemo tako, da svg sliko[4] odpremo v poljubnem tekstovnem urejevalniku in tam na ustrezno mesto dodamo "linkScadaConfig"atribut z ustrezno vsebino. S tem načinom ni nič narobe, je pa potrebno natančno poznati vse animacije in JSON. Pisanje JSON objektov v

[4]SVG sliko lahko narišemo v katerem od dostopnih aplikacij za vektorsko risanje: Inkscape, QCad, ipd.

serializirani obliki ravno tako ni prav posebno zabavno. Zaradi omenjenega je bil v sklopu enopolnih shem razvit vmesnik za lažje urejanje enopolnih shem po sistemu:

- izbor elementa - klik na gr. el. na SVG sliki, ki ga želimo "oživiti"
- urejanje elementa - izbor vseh animacij "obešenih"na ta element
- shranjevanje sprememb



Slika 4.3: Dodajanje animacije nekemu elementu na enopolni shemi.

S kombinacijo risanja v katerem od vektorskih risarskih programov in opremljanja z atributi je možno sestaviti vsako enopolno shemo. Če želimo enopolno shemo po tem ko smo jo že opremili z nekaterimi animacijami ponovno grafično urejati, moramo iz WebSCADA CMS izvoziti ustrezno ".svg"datoteko , jo popraviti in ponovno uvoziti. Možne nadgradnje takega načina urejanja so opisane v razdelku 6.2.

4.2.4 Grafi

HMI/SCADA sistemi poznajo mešane predstavitve podatkov; včasih je na enopolnih shemah priročno imeti še grafično predstavitev nekega trenda. Tako lahko hkrati spremljamo tekoče stanje in še malo zgodovine. Ideja je, da se razširi enopolne sheme s še eno animacijo tipa chart, ki bo s podano konfiguracijo v "linkScadaConfig"atributu znala narisati graf nekaj trendov. Z velikostjo in postavitvijo grafa in ostalih elementov je mogoče doseči učinek grafa v shemi ali sheme (lahko samo nekaj signalizacij ali meritev) v grafu.

V tem razdelku bom podrobno opisal kako se ta animacija vklaplja v enopolno shemo.

Graf je realiziran kot SVG element "svg" znotraj enopolne sheme. Umestimo ga z atributi x,y ter width in height. Poleg teh osnovnih atributov ima še atribut "linkScadaConfig"kot vsi ostali elementi-animacije. Izpostaviti moramo dve funkciji: "init()"in "run()". Funkcija "init()"inicializira graf na podlagi podatkov v atributu "linkScadaConfig", funkcija "run()"pa na podlagi (s strežnika) pridobljenih podatkov ustrezno nariše graf in po potrebi popravi skalo.

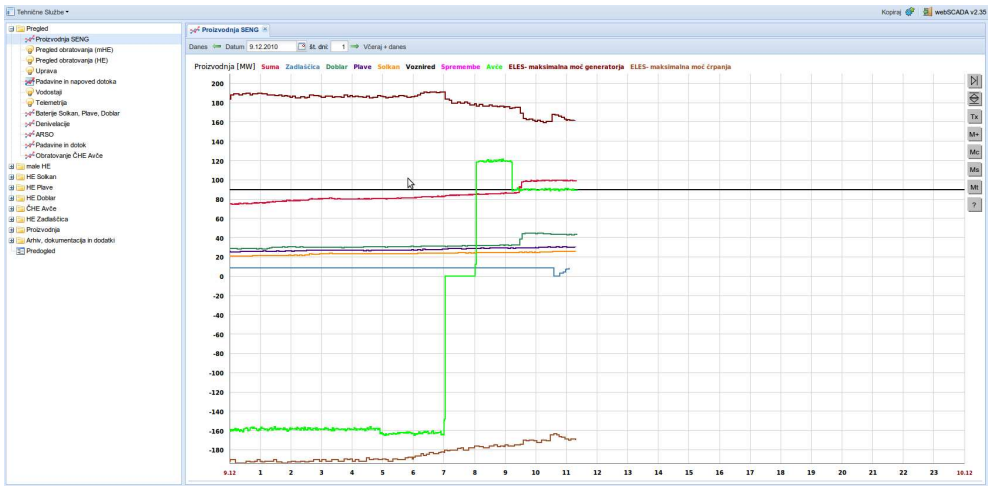
Vizualizacija: Avtomatsko skaliranje, sledenje podatkom, zoom, pan in ostale funkcije grafa

Na sliki 4.4 je prikazan graf kot samostojna enota. Tak graf je dejansko prazna enopolna shema z enim samim elementom z animacijo tipa chart; ta element ima še to posebnost, da ima atribute x=0, y=0 in width ter height postavljene na "100%", da se graf raztegne čez celo enopolno shemo.

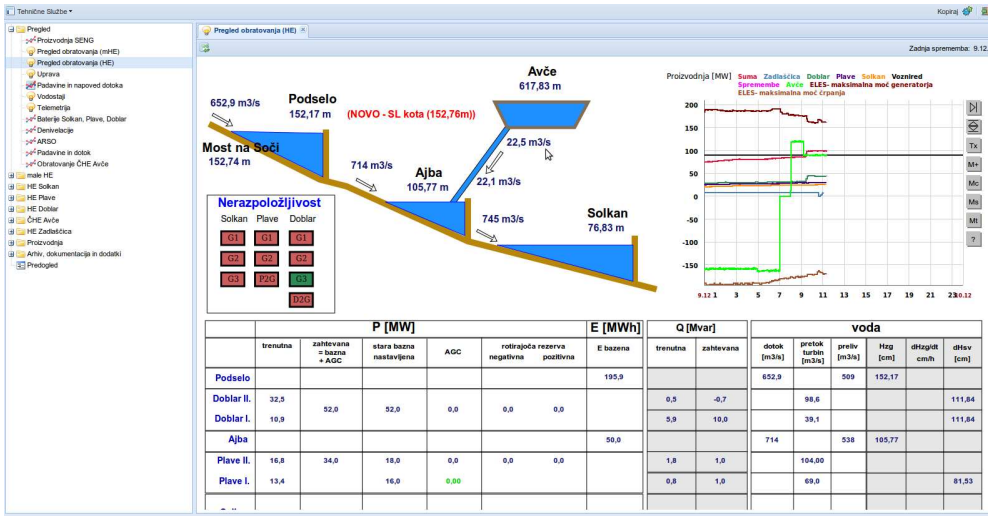
Na sliki 4.5 vidimo enopolno shemo z absolutno pozicioniranim grafom.

Funkcije grafa (ikone na desni strani od zgoraj navzdol):

- Sledenje podatkom - ob vklopu te funkcije je zagotovljeno da so novi podatki vedno vidni
- Skaliranje na podatke - ob vklopu te funkcije graf skalira glede na minimalno oz. maksimalno vrednost podatkov in ne na vrednost min oz. max določeno v konfiguraciji
- Tx točke na serijah - ob vklopu te funkcije se prikažejo točke na serijah, ki ponazarjajo podatke (majhni krogci oz. markerji)
- M+ izbor markerjev - ob vklopu te funkcije s klikanjem po grafu izbiramo odlikovane vrednosti, ki jih lahko kopiramo v odložišče in naprej obdelujemo
- Mc brisanje markerjev - brisanje izbranih markerjev
- Ms markerji na skalo - avtomatsko postavljanje markerjev na trenutne odlikovane vrednosti skale na x osi (npr. na vsako uro)
- Mt tabela markerjev - predogled izbranih markerjev
- ? pomoč



Slika 4.4: WebSCADA CMS - prikaz grafa.



Slika 4.5: WebSCADA CMS - prikaz grafa znotraj enopolne sheme.

Poglavje 5

Infrastruktura novega informacijskega sistema

V grobem lahko informacijski sistem razdelimo na dva pogleda. Informacijski pogled podaja informacijo o temu kako se informacije prenašajo, kaj je izvor, kaj ponor in kje se informacije nahajajo, kako se pretvarjajo iz ene oblike v drugo, ipd. Upravni pogled pa je pogled upravitelja informacijskega sistema; to so enopolne sheme, preglednice, grafi, poročila, ukazi.

5.1 Informacijski pogled

Elementi informacijskega pogleda so podrobno opisani v poglavjih 2 in 3. Na tem mestu je podan samo slikovni povzetek.

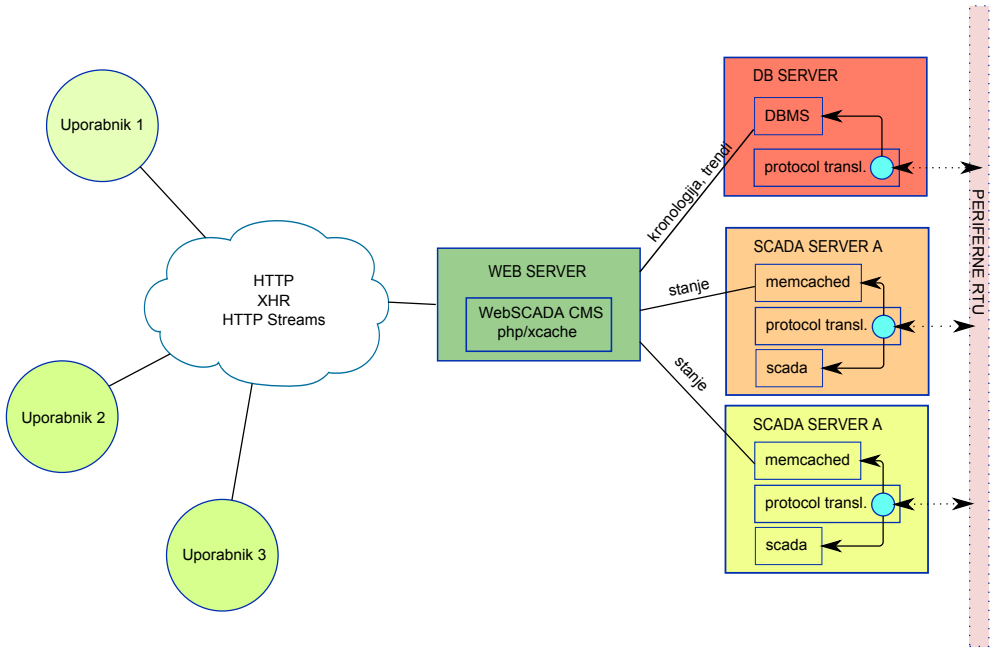
5.2 Upravni pogled

Enopolne sheme so podrobno opisane v pogl. 4. Ostali elementi upravnega pogleda, to so preglednice, poročila in drugi so namenoma izpuščeni. V nadaljevanju je podana grafična ponazoritev upravljanja.

Nekaj pojasnil v zvezi s sliko 5.2. Upravne poti:

- WebSCADA CMS - RDBMS WebSCADA CMS omogoča urejanje vseh lastnosti podatkovnih točk; me njimi so opis, format izpisa, koeficienti za pretvorbo iz surovih vrednosti, kot prihajajo iz RTU postaj in drugo.

- WebSCADA CMS - PCV XY WebSCADA CMS omogoča konfiguriranje



Slika 5.1: Infrastruktura novega informacijskega sistema - informacijski pogled.

oddaljenih RTU postaj ki jih upravlja PCV SCADA[1]

5.2.1 WebSCADA CMS

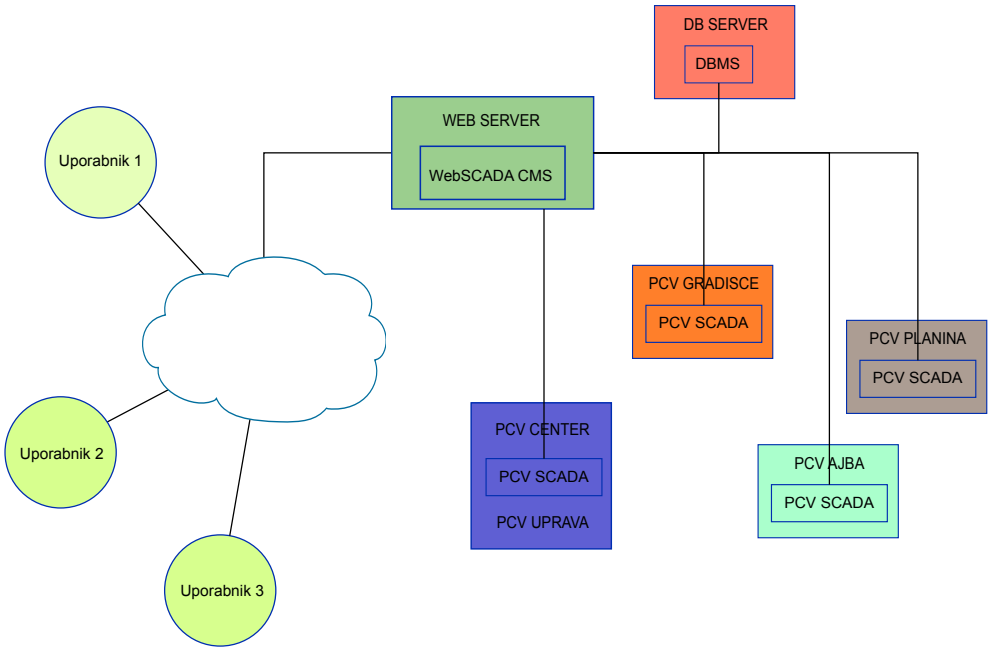
WebSCADA CMS je sistem za upravljanje z vsebinami.

Podatki in relacije

WebSCADA CMS v osnovi vsebuje 5 tabel z medsebojnimi relacijami kot prikazuje slika 5.3

- Tabela distribucija določa katere vsebine so na nekem določenem področju.
- Tabela vloge določa katere vloge ima nek uporabnik na določenem področju. Poleg tega vsebuje še nastavitve uporabnika za določeno področje.

[1]PCV SCADA je ravno tako spletna aplikacija, nekoliko poenostavljena WebSCADA



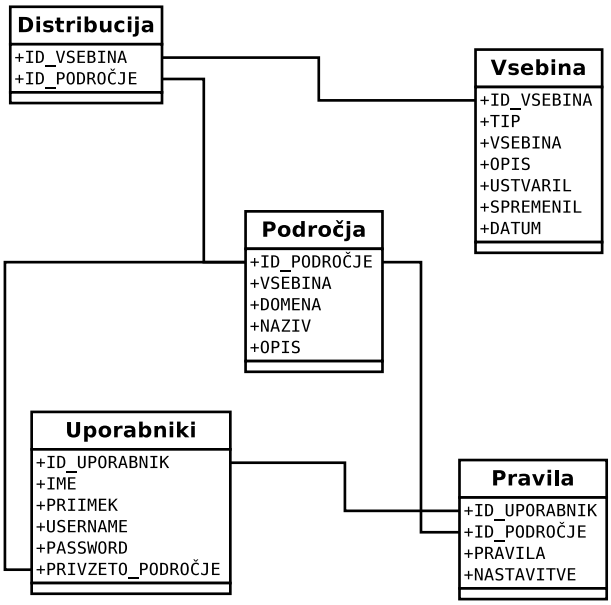
Slika 5.2: Infrastruktura novega informacijskega sistema - upravni pogled.

- Tabela področij vsebuje drevo vsebin, torej strukturo ki določa način pregledovanja vsebin.
- Tabela uporabniki vsebuje uporabnike; polje PRIVZETO_PODROČJE določa v katero področje bo uporabnik ob prijavi prišel. Uporabnik ima možnost prijaviti se v različna področja; če ima uporabnik določeno "view"vlogo na nekemu področju (tabela vloge) potem ima dostop do tega področja in se lahko prijavi.
- Tabela vsebina vsebuje vse vsebine CMS.

Uporabniški vmesnik - administracija

Administracija je razdeljena na 3 dele:

- Administracija vsebin; kjer popravljamo, brišemo in dodajamo nove vsebine.
- Administracija uporabnikov; kjer popravljamo, brišemo in dodajamo nove uporabnike.

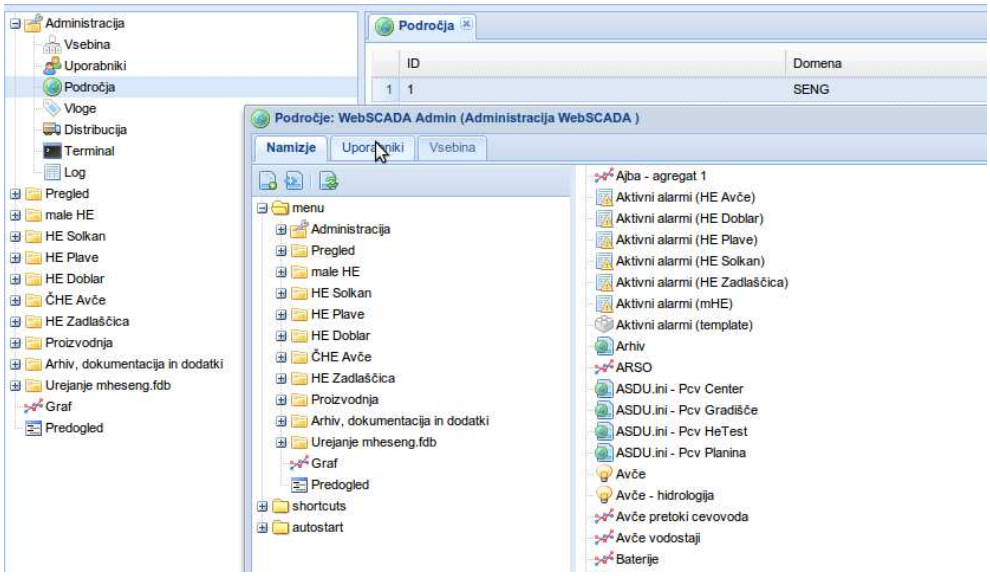


Slika 5.3: Relacijski model aplikacije WebSCADA CMS.

- Administracija področij; kjer popravljamo, brišemo in dodajamo področja. Administracija področja zajema še administriranje namizja, vlog, ter distribucije vsebin. Administriranje namizja je prikazano na sliki 5.4. Na levi strani je drevo; vozlišče menu je korensko menija namizja. Na desni strani so vsebine. S pomikanjem vsebin od desne na levo stran dodajamo vsebine na neko določeno mesto v menu. Administriranje uporabnikov je prikazano na sliki 5.5. Na levi strani so uporabniki na tem področju, na desni strani so uporabniki, ki niso na tem področju; s pomikanjem uporabnika z desne strani na levo dodamo uporabnika na področje. V ospredju pa je dialog, ki omogoča nastavitev vlog in nastavitev za določenega uporabnika. Administriranje vsebin je prikazano na sliki 5.6. Na levi strani so vsebine na tem področju, na desni pa ostale vsebine.

5.3 Optimizacija

Optimizacija spletnih aplikacij zadeva hitrost in odzivnost. V zvezi s tem je zelo zanimivo dejstvo, da je spletni velikan Google razkril, da je njihova uvodna stran namenoma tako okleščena in brez reklamnih sporočil. Iz ankete



Slika 5.4: WebSCADA CMS - Administracija namizja.

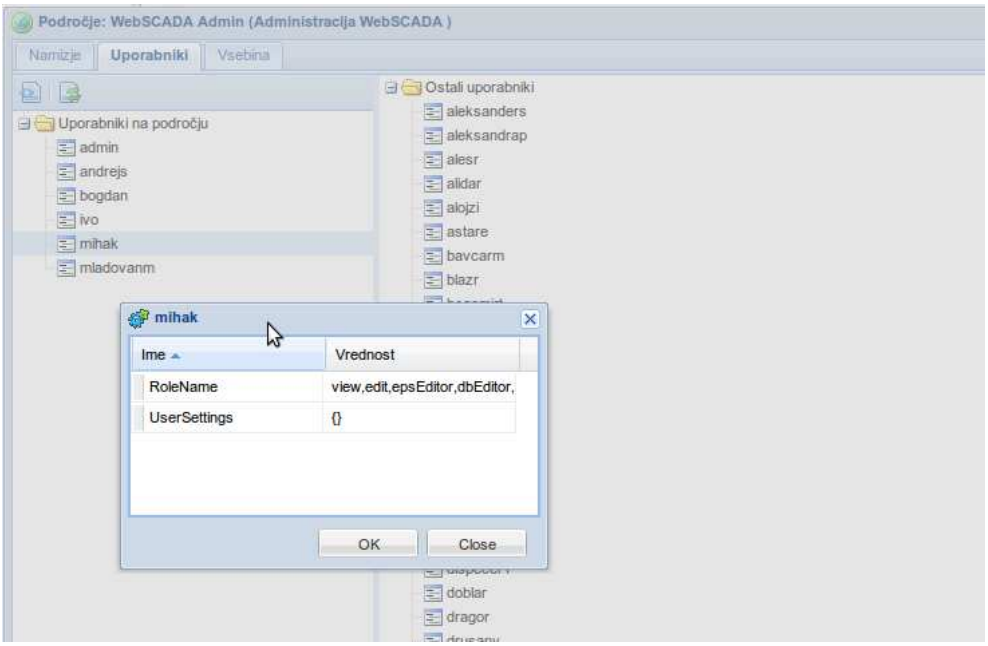
ki so jo izvedli je bilo razvidno, da je veliki večini uporabnikov pomembno prav to. Da je uvodna stran kar se da odzivna. Učinkovita spletna aplikacija pa terja premišljeno načrtovanje. Optimizirati pa je mogoče praktično vse dele aplikacije oz. širše - infrastrukture. Nekateri vidiki optimizacije so bili že predstavljeni v sklopu prejšnjih poglavij (glej pogl. 2 in 3), ostali bodo opisani v sledečih razdelkih.

5.3.1 14 pravil za hitrejši spletni uporabniški vmesnik

Sledijo pravila za učinkovito implementacijo uporabniškega vmesnika (frontend) povzeto po [7] in [1].

1 Čimmanj zahtev HTTP (Make fewer HTTP requests) Vsaka zahteva HTTP vzpostavi novo povezavo. Vzpostavljanje nove povezave HTTP pa je relativno draga operacija tako za odjemalca, predvsem pa za spletni strežnik. To pravilo je pravzaprav skoraj samoumevno.

2 Uporabi CDN (Use a CDN) CDN je vsak sistem za posredovanje vsebine. Jasno je, da če posredujemo samo vsebino in ne oblike dobimo bolj odziven sistem. Z drugimi besedami to pomeni. Spletno aplikacijo delimo na dva dela. Staitčni in dinamični del. Statični del je t.i.



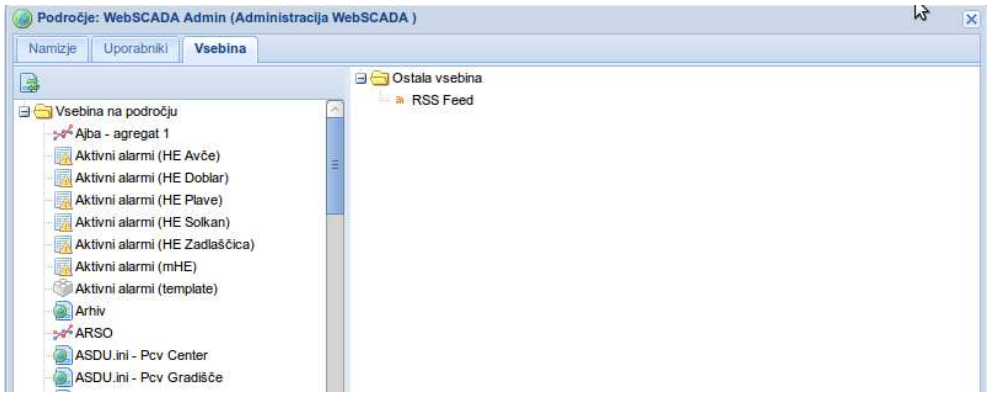
Slika 5.5: WebSCADA CMS - Administracija uporabnikov.

forma programa, ki jo sestavljajo knjižnica za vizualno predstavitev podatkov (JS,CSS,PNG), potem enopolne sheme (SVG), ter ostali resursi aplikacije. Dinamični del pa predstavljajo podatki (v JSON notaciji), ki jih zahtevamo vsakokrat ko želimo osvežiti statično vsebino: sheme, grafe, tabele, ... Statična vsebina se le redkokdaj spremeni (npr. ko poravimo enopolno shemo); naložimo jo samo v primerih ko se spremeni. Dinamična vsebina pa se nalaga vsakokrat ko želimo osvežiti podatke, oz. kot je bilo navedeno v razdelku 2.3 ob spremembi podatkov.

3 Uporabi "Expires"glavo HTTP (Add an Expires header) "Expires"glava se uporablja ko je znano kdaj bo potekel rok trajanja nekega dokumenta in ga bo potrebno ponovno naložiti s strežnika. Ker takih dokumentov v WebSCADA CMS ni je to pravilo brezpredmetno.

4 Stisni komponente (Gzip components) Stiskanje pred prenosom je (kot sem ugotovil pri delu na tej diplomski nalogi) ključni faktor pri izboljšanju performans celotnega sistema. Zato si zasluži nekoliko podrobnejšo razlago (glej razdelek 5.3.2).

5 CSS na vrh strani (Put CSS at the top) To je potrebno zato da ni po-



Slika 5.6: WebSCADA CMS - Administracija vsebin.

trebno ponovno risanje (najprej brez CSS, nato še z CSS).

6 JS na dno strani (Move JS to the bottom) To je potrebno zato, da se naloži najprej vizualni del aplikacije in šele potem funkcijski del. Za WebSCADA CMS je brezpredmetno, ker se vse naloži v JS.

7 Izogibaj se izrazom CSS (Avoid CSS expressions) Z veseljem.

8 Uporabi zunanje JS in CSS (Make JS and CSS external) Zunanji JS in CSS omogočajo optimalno nalaganje in keširanje s strani spletnega brskalnika.

9 Omeji DNS vpoglede (Reduce DNS lookups) DNS vpogled se prišteva k času dostopa do resursa.

10 Pomanjšaj JS (Minify JS) Pomanjšanje JS datotek dodatno pripomore k zmanjšanju porabe pasovne širine in celo k hitrosti prevajanja JS. Poleg tega ima tudi stranski učinek zakrivanja izvorne kode, kar zmanjšuje tveganje, da nam ukradejo intelektualno lastnino.

11 Izogibaj se preusmeritvam (Avoid redirects) Brezpredmetno.

12 Odstrani podvojene skripte (Remove duplicate scripts) To je povezano s točko 1.

13 Izklopi ETag-e (Turn off ETags) ETag identificirajo nek resurs; če se ETag spremeni, potem to pomeni da se je spremenil resurs in ga je potrebno ponovno naložiti. ETag-i so komplementarni Last-Modified, If-Modified-Since paru opisanemu v razdelku 6.1.2.

14 Uporabi keširanje podatkov prenesenih preko XHR (Make AJAX cacheable and small) Preko XHR se v WebSCADA CMS prenašajo tako statični podatki (vsebine) kot tudi dinamični (podatkovne točke, historične vrednosti, ...). Vse kar se lahko kešira, se tudi kešira. Keširajo se vse vsebine (content). Način kako se kešira statične in ne kešira dinamične resurse je opisan v razdelku 3.1

5.3.2 Stiskanje resursov

Stiskanje resursov sledi delitvi resursov na statične in dinamične. Statične resurse lahko predpripravimo in shranimo na trdem disku ali v predpomnilniku, medtem ko dinamične resurse stiskamo tik preden jih posredujemo odjemalcu.

- Posredovanje statičnih stisnjenih resursov. Ustrezno nastavimo Apache strežnik. Sledi izsek iz datoteke httpd.conf, ki služi temu namenu.

```
# Če se datoteka konča z .js.gz, potem nastavimo ustrezen Content-Encoding
t.j. kako je vsebina zakodirana
# ForceType pove da se v zakodirani vsebini skriva javascript
<FilesMatch "\.js\.gz$">
  Header set Content-Encoding "gzip"
  ForceType "text/javascript; charset=utf-8"
</FilesMatch>
# ForceType pove da se v zakodirani vsebini skriva css
<FilesMatch "\.css\.gz$">
  Header set Content-Encoding "gzip"
  ForceType "text/css; charset=utf-8"
</FilesMatch>
# ForceType pove da se v zakodirani vsebini skriva svg
<FilesMatch "\.svg\.gz$">
  Header set Content-Encoding "gzip"
  ForceType "image/svg+xml; charset=utf-8"
</FilesMatch>
# V kolikor so zahtevane datoteke tip css ali js ali svg pogledamo
če odjemalec
# sprejema stisnjene podatke (Accept-Encoding) in če seveda obstaja
stisnjena
# oblika datoteke, potem posredujemo tako datoteko sicer pa nestisnjeno.
<FilesMatch "\.(css|js|svg)$">
```

```
  RewriteEngine On
  RewriteCond %{HTTP:Accept-Encoding} gzip
  RewriteCond %{REQUEST_FILENAME}.gz -f
  RewriteRule ^(.*)$ $1.gz [L]
</FilesMatch>
```

- Posredovanje dinamičnih stisnjenih resursov Dinamične resurse stisnemo tik preden jih posredujemo. V progrmskem jeziku PHP izgleda tako:

```
// $response je nestisnjena vsebina odgovora
$len = strlen($response);
// stiskamo samo če so resursi daljši od xxx, ker bi sicer porabili
več časa za stiskanje, kot bi prihranili
// pri nalaganju zaradi zmanjšane velikosti resursa
if ($len>10000) {
  $response = gzdeflate($response,9); // stiskanje defate = gzip
brez headerja
  header("Content-Encoding: deflate");
}
```

5.3.3 Dodatne možnosti za optimizacijo kot jih predlaga Google Page Speed in Yahoo YSlow

Poleg zgoraj omenjenih so na razpolago še dodatne možnosti za izboljšavo odzivnosti spletne aplikacije. Nekatere bi lahko še implementiral ampak performančni dobitek le-teh je vprašljiv. Pri optimizaciji je potrebno pač nekje potegniti črto, kaj se še splača in kaj bi predstavljalo preveč truda za premalo rezultata.

- Združevanje JS datotek (Combine external JavaScript) V trenutni implementaciji aplikacije so dve JS datoteki. Ena predstavlja tuje knjižnice, ki so večinoma statične. Ta datoteka je največja in je smiselno, da je zapakirana posebej, ker se prenaša le redkokdaj. Druga datoteka je knjižnica vseh JS datotek, ki se nanašajo na aplikacijo in se pogosteje spreminjajo - ko se aplikacija nadgrajuje. To je minimalna smiselna količina, ki zagotavlja prožnost aplikacije.

- Optimiziraj bitne slike (Optimize images) Bitne slike, ki se uporabljajo v spletnih aplikacijah so večinoma ikone in je njihova velikost majhna ne glede na to ali so stisnjene ali ne. Drugače je, če nalagamo večje bitne

slike. Potem je stiskanje (JPEG) obvezno. Z izjemo ene take velike bitne slike v aplikaciji WebSCADA CMS optimizacija ni potrebna.

- Minimiziraj HTML (Minimize HTML) WebSCADA CMS ima samo eno HTML datoteko dolžine 20 vrstic, zato je taka optimizacija brezpredmetna.

- Ostanejo nam še drugi, bolj agresivni pristopi do optimizacije. Npr. Knjižnica ExtJS je velika in nekaterih delov se ne uporablja. Zato bi morda lahko brisali dele kode, kateri se ne uporabljajo. Ravno tako bi lahko brisali dele CSS datotek ki se ne uporabljajo. Teoretično bi morda tak pristop bil mogoč, problemi pa bi se pojavili, ko bi npr. hoteli dodati neko funkcionalnost aplikaciji, potem bi morali znova in znova preučevati katere dele knjižnice potrebujemo in katere ne. Zato sem tako vrsto optimizacije opustil. Glede na to, da je aplikacija obširna bi bilo verjetno zmanjšanje knjižnice glede na celoto zanemarljivo. Slabih 300k kolikor je sedaj velika celotna stisnjena knjižnica pa za današnje povezave tudi ni ravno ne vem kakšen zalogaj. To knjižnico prenašamo namreč le enkrat in ob spremembah, ki pa so redke.

- Združevanje ikon v CSS Sprit-e (Combine images into CSS sprites) Ta zadnja omenjena možnost optimizacije predlaga združevanje vseh uporabljenih ikon v eno bitno sliko. Tako združevanje je smiselno, ker nalaganje vsakega resursa vključuje eno zahtevo HTTP. Te pa so, kot sem prej omenil, drage. S pozicioniranjem na delček (združene) bitne slike in omejevanjem velikost lahko z uporabo CSS pravil dosežemo, da se naloži le ena velika bitna slika, ki združuje vse ikone v uporabi. Ta možnost je zanimiva ampak zahteva poseg v osnovno knjižnico ExtJS. Glede na to, da se ti resursi naložijo le redko[2] bi bilo vprašljivo ali bi se tak poseg sploh splačal.

5.3.4 Rezultati in zaključek

Po vseh omenjenih izboljšavah se je odzivnost aplikacije bistveno izboljšala. Google Page Speed je podelil aplikaciji 91 od 100 točk. Bistveno pa je to, da je sedaj aplikacija WebSCADA CMS skoraj tako odzivna kot bi bila namizna aplikacija.

[2]največ enkrat na leto - nastavitev v httpd.conf; veljavnost jim poteče šele čez leto dni, tako da se ne preverja

Bistveno pa je tudi dejstvo, da so uporabniki aplikacije znani in jo vsakodnevno uporabljajo. To pa pomeni, da imajo večino resursov že naloženih v predpomnilniku brskalnika. To pa naprej pomeni, da ni vsakokratnega nalaganja aplikacije, kot je v primeru, ko imamo opravka z naključnimi uporabniki na spletu. To dejstvo in dejstvo, da se aplikacija večinoma (ne pa izključno) uporablja v notranjem omrežju omogoča, da lahko nekatere vidike optimizacije brez velike škode izpustimo.

Poglavje 6

Zaključek

Opisani informacijski sistem WbeSCADA CMS je nastajal zadnji dve leti, je operativen in bo z letom 2011 zamenjal obstoječ HTML vmesnik v podjetju Soške Elektrarne Nova Gorica - SENG. Aplikacijo bi brez večjih težav lahko umestili v katero koli okolje. Vse kar bi se spremenilo bi bil vmesnik s podatkovno bazo. In seveda vsebine.

6.1 Problemi, ki so se pojavili pri izpeljavi projekta

V sledečih razdelkih so predstavljeni nekatere težave, in načini kako so bili rešeni oz. kako sem jih zaobšel.

6.1.1 Pravilna nastavitev PHP

Za pravilno delovanje XHR podatkovnih tokov je potrebno zagotoviti da se vsebina HTTP ogovora NE zadržuje v medpomnilniku PHP ali Apache strežnika. V PHP to zagotovimo z ustrezno nastavitvijo parametra "OutputBufferning = Off"v datoteki php.ini. Apache strežnik je bolj problematičen. Eksperimentalno sem ugotovil, da novejše verzije PHP distribucij hkrati z nastavitvijo parametra "OutputBuffering"v PHP okolju avtomatično nastavijo velikost izhodnega medpomnilnika Apache strežnika. Tako da "outputBuffering = Off"res pomeni: izklopi vse izhodne medpomnilnike. Žal to ne velja npr. za verzijo PHP 5.0.5, ki je predinstalirana na npr. MOXA industrijskih računalnikih, ki se uporabljajo kot lokalne SCADA postaje po objektih. Nadgradnja na teh sistemih je skoraj nemogoča, zato je bilo najbolj smiselno ne uporabljati

XHR podatkovnih tokov in stopiti korak nazaj. Druga rešitev ki se ponuja je napihovanje paketov do kritične meje. Na primer, če na nekem sistemu ugotovimo, da je izhodni medpomnilnik velikosti 4k, napihnemo vsebino paketa do te velikosti. Ampak po drugi strani to porodi vprašanje o smiselnosti uporabe XHR tokov nasploh. Ugotovil sem namreč, da bi bili paketi tipično 10x večji, zato sem tak način uporabe XHR tokov ovrgel kot možno rešitev problema izhodnega pomnjenja.

6.1.2 Pravilna nastavitev HTTP glave odgovora pri posredovanju enopolnih shem

Vsebine posebnega pomena kot so enopolne sheme ne smejo nikoli biti zastarele. To pomeni, da se mora vedno preverjati ali so predstavitve na odjemalčevi strani sveže. V primeru da dostopamo do resursa direktno, npr. http://domena/shema.svg to zahtevo izpolnimo tako da vstavimo sledeče nastavitve na ustrezno mesto[1].

```
# NEVER CACHE
<FilesMatch "\.(svg)$">
Header set Cache-Control "max-age=0, must-revalidate"
</FilesMatch>
\end{verbatim}
V primeru da posredovanja vsebine preko PHP pa to zahtevo izpolnemo
tako da vstavimo sledeči delček kode:
\begin{verbatim}
header("Cache-Control: max-age=0, must-revalidate"); // vedno zahtevamo
preverjanje svežine predstavitve
$last_modified_ts = $response->extdata->lastModified; // čas spremembe
resursa
// v primeru da smo prejeli HTTP If-Modified-Since v glavi HTTP vprašanja
pomeni da imamo preverjanje svežine
if (isset($_SERVER['HTTP_IF_MODIFIED_SINCE']) &&
strtotime($_SERVER['HTTP_IF_MODIFIED_SINCE']) >= $last_modified_ts)
{
  // če ugotovimo, da je resurs svež vrnemo ustrezen HTTP odgovor
  header('HTTP/1.1 304 Not Modified');
  exit;
```

[1] Za podrobnosti glej dokumentacijo o mod_headers na naslovu http://httpd.apache.org/docs/current/mod/mod_headers.html

```
}
// če je predstavitev zastarela vrnemo nov Last-Modified čas, ki se
bo ob naslednji HTTP zahtevi pretvoril
// v If-Modified-Since; s tem je krog zaključen
$lastModified = date(DATE_RFC2822,$last_modified_ts);
$header("Last-Modified: $lastModified");
```

Uporabne informacije o pravilni nastavitvi in uporabi HTTP/1.1 predpomnilnika in ostale napotke za učinkovito implemetacijo sem našel na naslednih naslovih:

- Caching Tutorial - http://www.mnot.net/cache_docs/
- Speed up your site with Caching and cache-control - http://www.askapache.com/htaccess/speed-up-your-site-with-caching-and-cache-control.html
- Hypertext Transfer Protocol HTTP/1.1 - http://www.w3.org/Protocols/rfc2616/rfc2616.html

6.1.3 Problemi v zvezi s SCADA strežnikom ViSpro

Težave v zvezi s SCADA strežnikom izvirajo predvsem iz slabo oz. nedokumentiranih API funkcij in orodij. Pri tovrstnih aplikacijah je bilo namerno skrivanje oz. preprečevanje dostopa do shranjenih informacij nekaj vsakdanjega. Navsezadnje so posli sklenjeni za avtomatizacijo nekega sistema pomenili obstoj ali propad nekega podjetja. Oteževanje prehoda me enim in drugi SCADA sistemom je glavno orodje pri zadrževanju strank. Žal je taka praksa, še posebno pri velikih sistemih botrovala počasnejšemu razvoju in togosti aplikacij. Konkretno je SCADA ViSpro (izbrana je bila zaradi tega ker deluje v okolju linux) dandanes zelo toga glede izbora operacijskega sistema; nezdružljiva je z novejšimi linux jedri, utf-8, grafičnimi knjižnicami (ViSpro še vedno uporablja GTK1), namizji kot so GNOME, KDE. Vse to povečuje težave povezane z upravljanjem sistema.

6.2 Kaj bi se dalo še izboljšati

Od možnih izboljšav obstoječega informacijskega sistema WebSCADA CMS bi izpostavil predvsem tri ključne.

6.2.1 Trendi in kronologija iz memcache

Trendi in kronologija se trenutno berejo iz relacijske baze podatkov, ki služi kot sekundarni arhiv historičnih podatkov SCADA. Bolje bi bilo, če bi bi se neposredno (preko memcache) prenesli iz strežnika SCADA. V memcache bi hranili zadnjih n dni trendov in kronoloških dogodkov[2]. Ostalo bi se še vedno bralo iz sekundarnega arhiva; vseh podatkov namreč ni mogoče hraniti v dinamičnem spominu. Implementacija pomnjenja trendov in kronologije v medpomnilniku je nekoliko bolj zapletena zaradi dodatne dimenzije, t.j. čas. Ključ za shranjevanje trendov in kronologije bi moral biti sestavljen iz datuma in imena trenda oz. kronološkega arhiva. Dogodki s točnim časom bi se potemtakem dodajali na konec takega zapisa. Dodatna težava je v tem, da obstaja več trendov "obešenih"na neko podatkovno točko, ker potrebujemo različno frekvenco zapisov za različne namene. Tako bi morali z neko konfiguracijsko postavitvijo določiti kateri trend se bo uporabil za vpis v memcache. Zaradi velike obremenitve podatkovne baze bo verjetno to ena izmed bodočih nadgradenj informacijskega sistema.

6.2.2 TouchSCADA: WebSCADA za prenosne telefone, in ostale naprave na dotik

Aplikacijo WebSCADA CMS bi bilo mogoče nadgraditi s posebnim namizjem (glej pogl. 5.2.1), ki bi bilo prilagojeno velikosti ekranov manjših formatov. Ravno tako bi morali prilagoditi tudi ostale vsebine kot so preglednice, enopolne sheme. Kot razvojno okolje bi lahko uporabili kar produkt podjetja Sencha, to je Sencha Touch, ki razvija tudi ExtJS na kateremu temelji uporabniški vmesnik WebSCADA CMS. Sencha Touch je nekako okleščena verzija ExtJS. Pravzaprav je to varianta, ki je izpeljana iz novega HTML 5 standarda in je namenjena prav napravam na dotik.

6.2.3 Integrirano risanje tehnoloških shem znotraj WebSCADA CMS

Sestavljanje (risanje) enopolnih shem je sedaj prepuščeno zunanjim orodjem ki podpirajo vektorsko grafiko in znajo izvoziti vsebino v SVG. Tako sliko potem uvozimo v informacijski sistem WebSCADA CMS in jo ustrezno opremimo z animacijami. Nerodnost je v tem, da moramo vsakokrat ko želimo popraviti karkoli sliko ponovno uvoziti. Neka stopnja možnosti popravljanja shem bi

[2]verjetno bi zadostoval n=3 - da se pokrije vikend

bila dobrodošla znotraj WebSCADA CMS; vsaj tiste najbolj pogoste operacije kot so: premikanje elementov, brisanje, mogoče dodajanje predpripravljenih elementov iz neke knjižnice HMI component narisanih v SVG. Neka možnost popravljanja bo vsekakor vgrajena v naslednje verzije informacijskega sistema.

Slike

Literatura

[1] Souders Steven, Even Faster Web Sites: Performance Best Practices for Web Developers O'Reilly Media; 1 edition (June 10, 2009)

[2] XmlHttpRequest Dostopno na:
http://www.w3.org/TR/XMLHttpRequest

[3] XmlHttpRequest History Dostopno na:
http://en.wikipedia.org/wiki/XMLHttpRequest

[4] XmlHttpRequest Call Dostopno na:
http://ajaxpatterns.org/XMLHttpRequest_Call

[5] HTTP Streaming Dostopno na:
http://ajaxpatterns.org/HTTP_Streaming

[6] Scalable Vector Graphics (SVG) 1.1 (Second Edition), W3C Working Draft 22 June 2010 Dostopno na:
http://www.w3.org/TR/SVG/

[7] Best Practices for Speeding Up Your Web Site Dostopno na:
http://developer.yahoo.com/performance/rules.html