SVEUČILIŠTE U ZAGREBU – GEODETSKI FAKULTET

ZAVOD ZA KARTOGRAFIJU I FOTOGRAMETRIJU

Nikola Kranjčić

SERVISNO ORIJENTIRANA GENERALIZACIJA GEOINFORMACIJA ZA TURISTIČKE SVRHE

Diplomski rad

Zagreb, srpanj 2015.

I. Autor

Ime i prezime:	Nikola Kranjčić

II. Diplomski rad

Naslov:	Servisno orijentirana generalizacija geoinformacija za turističke svrhe
Mentor:	doc. dr. sc. Dražen Tutić
Komentorica:	prof. dr. sc. Nada Vučetić
Voditeljica:	prof. dr. sc. Nada Vučetić

III. Ocjena i obrana

Datum zadavanja zadatka:	14. siječnja 2015.
Datum obrane:	03. srpnja 2015.
Sastav povjerenstva pred kojim je branjen diplomski rad:	doc. dr. sc. Dražen Tutić prof. dr. sc. Nada Vučetić prof. dr. sc. Miljenko Lapaine

Zahvala

Zahvaljujem mentoru doc. dr. sc. Draženu Tutić na stručnim savjetima, strpljenju i razumijevanju pri izradi diplomskog rada.

Zahvaljujem komentorici i voditeljici diplomskog rada prof. dr. sc. Nadi Vučetić na smjernicama i velikoj pomoći u nabavljanju neophodne literature za izradu diplomskog rada.

Zahvaljujem svim prijateljima i kolegama koji su mi svojim društvom uljepšali studentske dane.

Na kraju, neizmjerno hvala mojim roditeljima, bratu, baki i djedu za potpunu podršku, potporu i savjete tijekom studiranja.

Sažetak

Široka rasprostranjenost prostornih podataka u geoinformacijskim sustavima (GIS) povlači razvijanje različitih servisa za generalizaciju kako bi se korisniku omogućio bolji prikaz i jednostavnija upotreba karata. Uz brzi razvoj različitih servisa javlja se potreba za standardom kako bi se spriječilo gomilanje nerazumljivih servisa. U ovom diplomskom radu dan je pregled standarda web-servisa za obradu podataka (Web Processing Service, WPS), kao i pregled servisa za generalizaciju. Također dan je tablični pregled tri različita web servisa za generalizaciju s pripadnim svojstvima. U praktičnom dijelu razmotrena je implementacija generalizacije putem WPS-a putem slobodnih softvera kao što su Geoserver i OpenLayers 2.13.

Ključne riječi: web-servis za obradu podataka (WPS), generalizacija, generalizacijski servisi, web servisi, Geoserver, OpenLayers 2.13

Abstract

Widespread use of geospatial data in geoinformation systems (GIS) influence the development of different services for generalisation so that the end user gets better representation and easier use of the map. The fast development of different services indicates the need for standardized services to avoid accumulation of nonstandard services. This master thesis will provide an overview of WPS standard, as well as overview of generalisation services. Also, the overview of three different web generalisation services with corresponding features is given. In practical part generalisation via WPS through open services like Geoserver and OpenLayers 2.13 is discussed.

Key words: Web Processing Service (WPS), generalisation, generalisation services, web services, Geoserver, OpenLayers 2.13.

SADRŽAJ

Popis slika

Popis tablica

Popis kratica

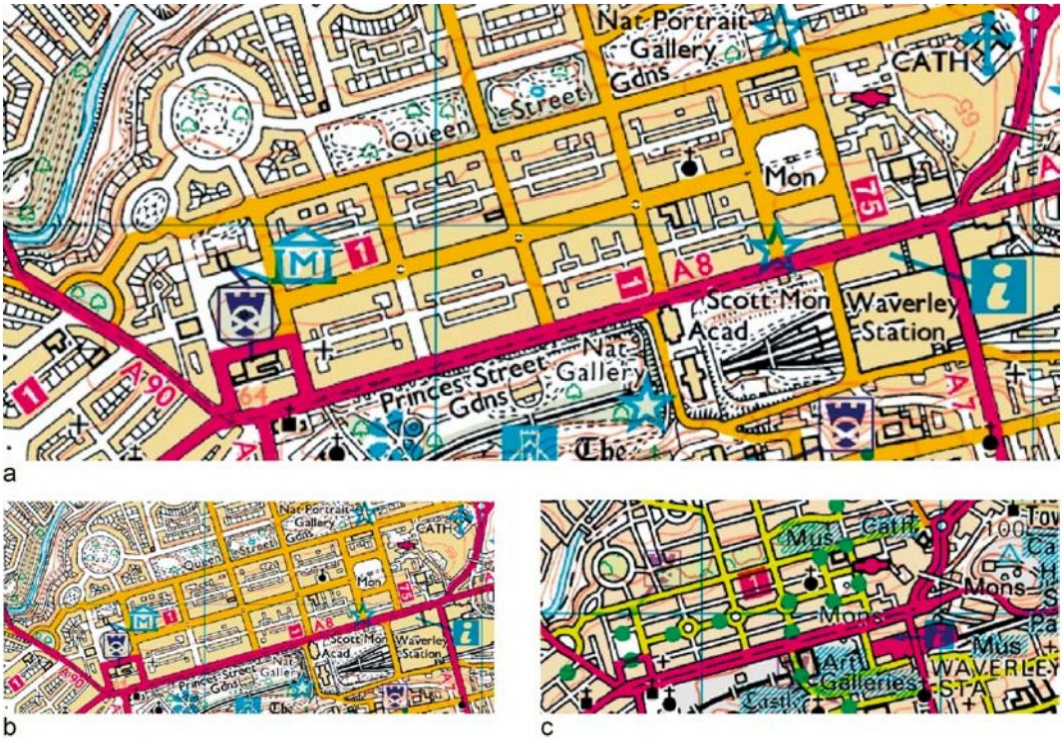
API	Application Programming Interface
BPEL	Business Process Execution Language
CORBA	Common Object Request Broker Arhitecture
GIS	Geografski informacijski sustav
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
Java RMI	Java Remote Method Invocation
MRDB	Multi Resolution Database
NMA	National Mapping Agency
OGC	Open Geospatial Consortium
Pcoll	Point collections
POI	Point of interest
RPC	Remote Procedure Calls
SOAP	Simple Object Access Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
URN	Uniform Resource Name
W3C	World Wide Web Consortium
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Mapping Service
WSDL	Web Service Definition Language
XML	Extensible Markup Language

1. UVOD

Sve širom rasprostranjenošću prostornih podataka dolazi do potrebe njihove kvalitetne, brze i učinkovite obrade. S novim tehnologijama razvija se poveći broj novih aplikacija i servisa za obradu prostornih podataka, te se pojavila potreba za definiranjem standarda kojima se treba voditi u stvaranju novih servisa. Uz pojavu novih, nesređenih, množinski prikupljenih prostornih podataka pojavila se potreba za web servisima koji će izvoditi automatsku generalizaciju nad tako prikupljenim podacima. Budući da se u diplomskom radu razmatraju prostorne informacije vezane uz turističke objekte razmotrena su tri postojeća web servisa, s preglednom tablicom njihovih karakteristika, te su ocijenjeni po upotrebljivosti za turizam. Većina rada temelji se na razmatranju web-servisa za obradu podataka (Web Processing Service, WPS), njegovim mogućnostima u generalizaciji, mogućnostima ulančavanja s drugim servisima, njegovim prednostima, nedostacima te predloženim promjenama. Također razmatraju se mogućnosti OpenLayersa i Geoservera u definiranju korisničkih procesa u svrhu generalizacije putem WPS-a. Cijeli rad temelji se na razmatranju slobodnih softvera jer su oni, naspram komercijalnih softvera, pristupačniji korisnicima.

2. GENERALIZACIJA

Generalizacija se može definirati kao proces kojim se smanjuje količina prostornih informacija prilagođena mjerilu prikaza (slika 2.1) i/ili namjeni prostornih informacija. Sam proces može se sastojati od jednog ili više postupaka (podprocesa, operatora) čija se implementacija provodi različitim algoritmima.



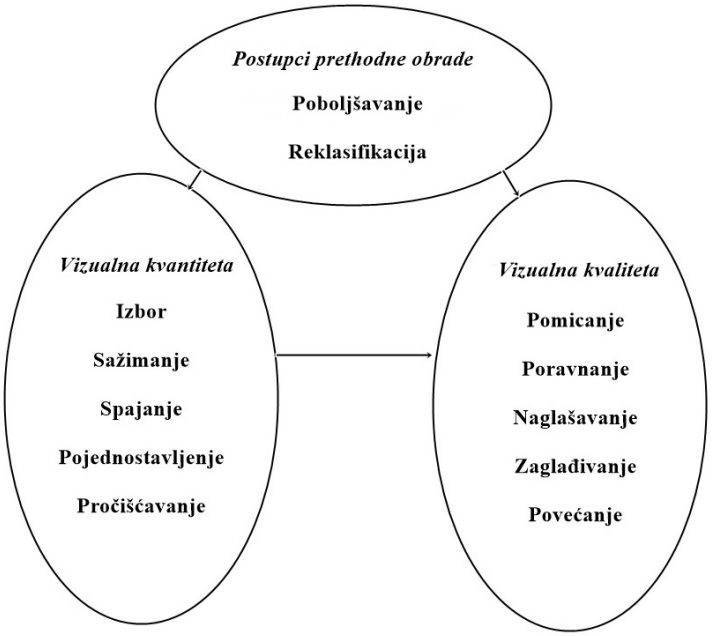
Slika 2.1 - Generalizacija (preuzeto iz Mackaness i Chaudhry, 2008.):

a) Izvorni prikaz (prikaz izvornih podataka)
b) Smanjeni izvorni prikaz
c) Generalizirani izvorni prikaz

Kako se u ovom radu razmatraju podatci u vektorskom obliku, u nastavku se ograničavam na generalizaciju vektorskih podataka.

2.1. Vektorski orijentirani postupci

Stanislawski i dr. (2014.) dijele postupke generalizacije u tri kategorije (slika 2.2) ovisno o potrebnim izlaznim podacima. Prije provedbe postupaka koji utječu na količinu detalja (kvantitativna vizualizacija) ili estetiku (kvalitativna vizualizacija) vektorski se podaci mogu prethodno obraditi postupcima poboljšavanja (*enrichment*) ili reklasifikacije (*reclassification*). Postupkom poboljšavanja objektima se pridodaju atributi za opisivanje geometrije, tj. oni atributi i lokalne geometrije koji nisu eksplicitno pohranjeni u izvornoj bazi podataka. Reklasifikacijom se grupiraju objekti temeljem postojećih i obogaćenih atributa, te je to uobičajeni postupak kojim se olakšava daljnja primjena generalizacijskih postupaka. Na slici 2.2 prikazan je preporučeni slijed postupaka generalizacije. Prvo se generalizacija provodi postupcima koji smanjuju količinu objekata ili detalja objekata, a zatim postupcima čiji je cilj poboljšanje kvalitete prikaza. U postupke smanjenja količine objekata ili detalja objekata ubrajaju se izbor (*selection, elimination*), sažimanje (*aggregation*), spajanje (*merge*), pojednostavljenje (*simplification*), pročišćavanje (*refinement*), a u postupke koji utječu na vizualnu kvalitetu spadaju pomicanje (*displacement*), poravnanje (*alignment*), naglašavanje (*exaggeration*), izglađivanje (*smooth*), povećavanje (*enhancement*).



Slika 2.2 - Klasifikacija generalizacijskih postupaka (prema Stanislawski i dr., 2014.)

2.2. Generalizacija točkastih objekata na webu

Budući da se u daljnjem radu govori o potrebama web-generalizacije za turističke svrhe posebna naznaka stavit će se na generalizaciju točkastih objekata. Prema Bereuter i Weibel (2010. i 2013.) čimbenici koji utječu na generalizaciju točkastih objekata kako za mobilne uređaje tako i web su sljedeći:

- **Podaci kartografske podloge naspram podataka karte**
  Podaci karte su podaci koji su korisnički definirani i njima se pridaje najviše pozornosti. Podaci kartografske podloge ponašaju se kao osnovna karta, te takvi podaci pružaju vizualnu pozadinu i prostornu referencu (npr. pomoć mobilnim korisnicima u orijentaciji).
- **Tipovi točkastih podataka**
  Bereuter i Weibel razlikuju dva osnovna tipa podataka: točke od interesa (POI – point of interest) i skupovi točaka (PColl – point collections). Primjeri točaka interesa su kafići, apartmani, hoteli, plaže, odnosno oni objekti kojima možemo lako i bez dvojbe definirati prostorni položaj. Primjeri skupova točaka uključuju podatke koji postoje u velikim skupovima, poput podataka opažanja životinjskih staništa. Točke interesa su od posebnog značenja za određenu namjenu, te zbog toga moraju zadovoljiti određene prostorne uvjete. Ne smije se dogoditi da objekt koji je smješten na desnu obalu rijeke nakon primjene nekog postupka generalizacije bude položen na lijevu ili obrnuto. Nastavno, skupovi točaka se mogu slobodnije generalizirati izborom i sažimanjem sve dok su sveukupni prostorni odnosi zadržani. Točke interesa mogu imati više pridruženih atributa nego skupovi točaka.
- **Uvjeti za točkaste podatke**
  Uvjeti za podatke kartografske podloge i za podatke karte: podatak karte je uvjetovan prostornim ograničenjima podataka kartografske podloge. Ovdje pripadaju topološke relacije, gdje točka od interesa ne bi smjela promijeniti stranu ceste, rijeke ili nekog drugog linijskog objekta, odnosno površinskog objekta.
  Uvjeti između podataka karte: budući da se radi o točkastim objektima, topološki odnosi bi se mogli činiti nevažnim, ali zapravo su neizmjerno važni jer jedna točka

predstavlja jedan podatak koji može biti generalizirani površinski objekt prikazan kartografskim znakom.

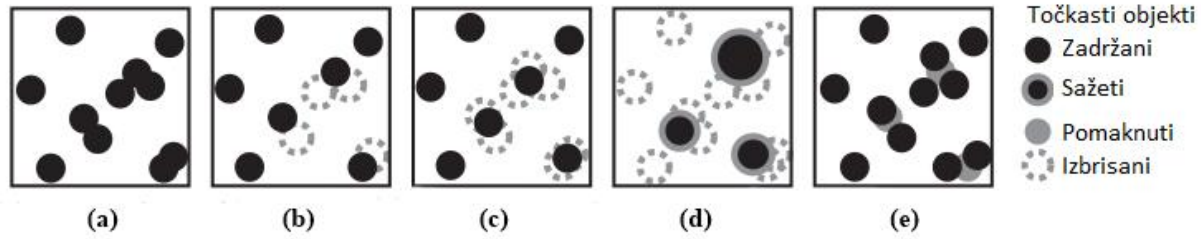
Kartografski uvjeti: zadržavanje prostornih odnosa jedan je od ključnih pravila bilo koje generalizacije i stoga čini temeljni skup kartografskih uvjeta. Kod postupaka generalizacije također se mora voditi računa o dodatnim kartografskim uvjetima vezanim za strukturu, uzorak i semantiku skupa točaka, poput gustoće, prostornog rasporeda ili poretka po važosti.

- **Nivo interaktivnosti**

  Može se pretpostaviti da korisnik ima mogućnosti interaktivnog prilagođavanja prikaza karte, na primjer, upotreba različitih razina zuma, promjenu prikaza, paljenje i gašenje određenih slojeva. Što je veća interaktivnost, više se može podesiti i uređivati po karti. Uz veću interaktivnost dolazi i do potrebe za bržim postupcima generalizacije jer korisnik želi podatke u realnom vremenu.

2.2.1. Postupci generalizacije

Bereuter i Weibel (2010.,2013.) razlikuju četiri postupka generalizacije točkastih objekata: izbor, pojednostavljivanje, sažimanje i pomicanje (slika 2.3). Prva tri postupka smanjuju broj točkastih objekata dok ih četvrti samo pomiče. Izborom se iz izvornog skupa točaka izabire podskup točaka na temelju atributa, dok se pojednostavljenjem podskup točaka izabire na temelju geometrije.

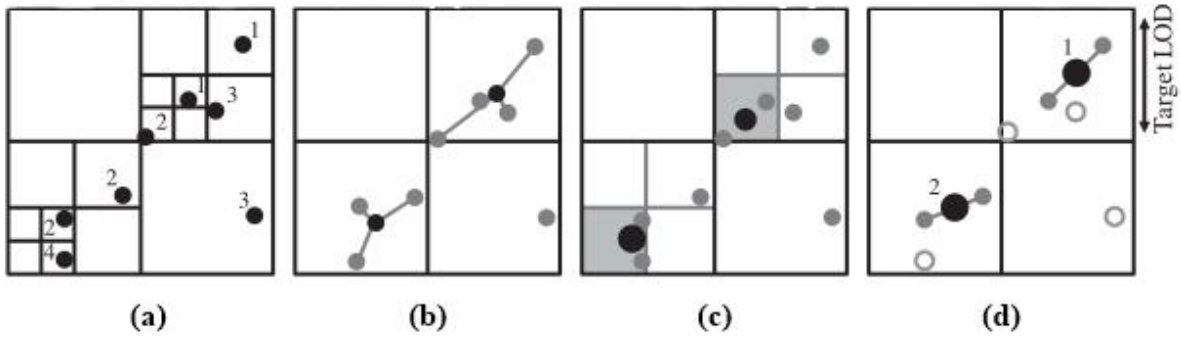


Slika 2.3 – Generalizacija točkastih objekata (prema Bereuter i Weibel, 2013.):

a) izvorni podaci; b) izbor; c) pojednostavljivanje; d) sažimanje; e) pomicanje

Sažimanje zamjenjuje više bliskih objekata s objektom povećanih dimenzija, kao što je zamjena puno točkastih objekata s jednim površinskim objektom ili jednom točkom povećane

dimenzije. Glavna svrha sažimanja je smanjenje razine detalja grupiranjem objekata prema sličnosti njihovih atributa i/ili prema prostornoj bliskosti. Za razliku od izbora, kod sažimanja se stvara novi objekt. Sažimanje se provodi na dva načina (Bereuter i Weibel 2010). Prvo, putem grupe algoritama gdje se putem hijerarhije izvršavaju postupci generalizacije, a drugo putem hijerarhijskih struktura poput kvadratnog stabla. Budući da ćemo se sažimanjem detaljnije baviti, ovdje će se prikazati i mogući algoritmi sažimanja, prikazani u Bereuter i Weibel (2013). Sažimanje u središnju točku (slika 2.4b) sažima točkaste objekte prema srednjoj točki od točaka koje se nalaze u regiji i za to se koristi samo položaj točaka. Na grupama zasnovano sažimanje vraća točku obzirom na broj točaka u podregiji, što je prikazano na slici 2.4c. Na grupama zasnovano sažimanje se može provoditi po geometriji i po atributima. Sažimanje po metodi kolokacije generalizira područje temeljeno na pravilu kolokacije, poput pojavljivanja objekata pripadnih istim klasama. Primjer za kolokaciju bi bio pronalazak parkirališta u blizini određenog hotela u gradu. Za svaku regiju algoritam provjeri sadrži li hotele i parkirališta, te koliko ih je, kako bi se sažimanje provelo ili jednim znakom ili s dva znaka, što je vidljivo na slici 2.4d.



Slika 2.4 - Vrste sažimanja (preuzeto iz Bereuter i Weibel, 2013.) : a) izvorni podaci i kvadratno stablo; b) sažimanje u središnju točku; c) na grupama zasnovano sažimanje; d) sažimanje po metodi kolokacije

3. WEB-SERVISI ZA GENERALIZACIJU

Web-servisi su razvijeni da se računalima omogući pristup servisima, pa u tom pogledu web-servisi koriste računalno-orijentirano sučelje i standardizirani jezik. Web-servis se može definirati na sljedeći način:

"Web-servis je programski sustav dizajniran da omogući interoperabilnost interakcije "od računala–prema računalu" preko mreže. Web servis ima sučelje opisano u računalno-obrađenom formatu (poput WSDL – Web Sevice Definition Language). Ostali sustavi uzajamno djeluju s web servisima na način propisan u njihovim opisima koristeći SOAP (Simple Object Access Protocol) poruke, obično prenesene koristeći HTTP s XML serijama sa sponama s ostalim web-povezanim standardima" (W3C 2004). Upotreba takvih web servisa može biti ili potpuno automatska ili web-servis može biti pozvan na korisnički zahtjev.

Kod web-servisa možemo razlikovati dva pristupa, interaktivni servis korisnik-poslužitelj ili interaktivni servis poslužitelj-poslužitelj. Interaktivni servisi korisnik-poslužitelj dizajnirani su za ljudsku upotrebu i fokusiraju se na krajnjeg korisnika što predstavlja standard OGC-a (Open Geospatial Consortium). Interaktivni servisi poslužitelj-poslužitelj koriste se za obradu podataka, utjecaj krajnjeg korisnika je minimalan, a svrhu nalaze u daljinskoj obradi podataka kada lokalna obrada traje predugo. Prema Burghardtu i dr. (2005.) potencijalno najrašireniji servis bila bi generalizacija, odnosno, različit stupanj zuma u servisima web-karata, koji bi se koristio kao spona (*middleware*) između WMS-a i klijenta, te je u takvom slučaju tip i struktura podataka poznata servisu. Drugi obećavajući servis bio bi servis s povećanom brzinom obrade podataka i s najnovijim algoritmima, što bi smanjilo potrebu za lokalnom obradom podataka i potrebu za učestalim ažuriranjem lokalno instaliranih softvera. Budući da bi kod ovog servisa korisnici trebali učitati svoje podatke, javlja se potreba za standardom kako bi servisi funkcionirali na željeni način.

3.1. Svojstva web-servisa

Za svaki web-servis mogu se definirati sljedeća glavna svojstva (Burghardt i dr., 2005.):

- Neovisnost o platformi
- Registar servisa
- Web-API – sučelje (API – Application Programming Interface)
- Slobodna komunikacija

Prilikom pretraživanja weba i pristupanja servisima korisnicima mora biti omogućen pristup bez obzira na platformu, odnosno operativni sustav koji koriste, stoga je neovisnost o platformi, ne samo poželjno, nego i neophodno svojstvo web-servisa. Također poželjno je da su svi servisi uvedeni u registar po principu objavi-pronađi-poveži (*publish-find-bind*). Web-servisi su objavljeni i mogu biti nađeni kroz registar web-servisa, korisnici ih mogu pronaći i zatim se klijent može povezati sa servisom. Web-API je sučelje servisa koji se može pozvati iz drugih programa, pa takvo sučelje mora biti osnovano na standardima kako bi se moglo pozvati iz gotovo svakog programa. Slobodna komunikacija je potrebna kako bi se programima omogućilo automatsko povezivanje sa sučeljem i najčešće se takva komunikacija odvija putem HTTP-a i XML-jezika koji mogu prenijeti gotovo svaki tip podataka.

U slučaju servisa za generalizaciju koji podržavaju učitavanje i obradu korisničkih podataka postoje posebni zahtjevi za geometriju prostornih podataka. Svi ulazni podaci bi trebali biti prevedeni u zajednički format koji servis može razumjeti, dok bi za korisnika najprikladniji format bio sličan *Shapefile-u* ili GML-u. Kod kompleksnih servisa za generalizaciju trebaju postojati dva dodatna svojstva koja nisu ugrađena u web-servise. Ova svojstva su sposobnost da se zadrži programsko stanje izvršavanja algoritama i održavanje stanja transakcije. Zadržavanje stanja izvršavanja algoritma je bitno kada komuniciranje klijenta sa servisom traje duže od predviđenog. Jednom kad klijent učita početne parametre servis počinje s izvršenjem algoritma, pa ukoliko algoritam traži dodatne podatke klijentu se šalje odgovor s pripadajućim zahtjevom. Kod izvršavanja te komunikacije servis mora zadržati već dobivene informacije i već izračunate promjene. Tehnologije za tu svrhu obično su već ugrađene u mnoge web-servise. Održavanje stanja transakcije izuzetno je važno kod ulančavanja nekoliko servisa, što je ključan korak prema operaciji koja bi generalizirala cijelu kartu automatski i

osiguravala proizvod spreman za upotrebu. U takvim slučajevima kvaliteta je manje bitna naspram brzine i stabilnosti.

3.2. Podjela servisa prema OGC/ISO-vom modelu arhitekture (02-112)

OGC/ISO-ov model arhitekture dijeli servise na:

- servisi obrade
- servis upravljanja modelom/informacijama
- servis upravljanja tijekom rada/zadataka
- servis ljudske interakcije
- komunikacijski servis
- servis upravljanja sustavom

Servisi obrade se većinom koriste kod automatiziranih postupaka generalizacije gdje moraju povezati servise koji izvode veliki broj postupaka, pa je poželjno da u servisima obrade bude implementirana *on-the-fly* generalizacija, poput prilagodbe sadržaja kod zumiranja. Budući da je kod *on-the-fly* generalizacije brzina od izuzetne važnosti, prihvatljiva je i niža kvaliteta generalizacije što znači da će se u realnom vremenu generalizirati postupcima izbora i sažimanja, dok će se generalizacija po atributima zanemariti. Na taj način dobivene karte su pogodne za mobilne uređaje, jer se zbog ograničenosti veličinom ekrana ne može prikazati čitav niz detalja, što je korisnicima prihvatljivo dok god se prikaz dobije u realnom vremenu. Servise upravljanja modelom/informacijama možemo smatrati skupom servisa koji dopuštaju razvoj, obradu i pohranu metapodataka i skupova podataka. Servisi upravljanja tijekom rada/zadataka pomažu u definiranju, pozivanju, statusu i kontroli ulančavanja servisa. Servisi ljudske interakcije dopuštaju interakciju tijekom procesa generalizacije, primjerice, izabiranje klase objekata koji će se generalizirati. (OGC, 2002.)

OGC (2002.) definira mehanizam za kontrolu generalizacijskog procesa putem tri arhitekture za ulančavanje servisa u OpenGIS® Web arhitekturi servisa (OGC, 03-025) koja može biti korištena ovisno o namjeni i kompleksnosti karte:

- Korisnički definirano (transparentno) ulančavanje: korisnik potpuno kontrolira tijek rada. Za kompleksne generalizacijske procese za koje ne postoje prikladni modeli.

- Ulančavanje upravljanim tijekom rada (djelomično transparentno): korisnik poziva servis upravljanja tijekom rada koji kontrolira ulančavanje i korisnik je svjestan pojedinih servisa, te korisnik može definirati redoslijed izvršavanja servisa. Primjenjiv je za srednje kompleksne generalizacijske procese koji mogu biti definirani tijekom rada.
- Netransparentan skup servisa: korisnik poziva servis koji ulančava druge servise pri čemu korisnik nije upoznat sa svim servisima u lancu. Primjenjiv je za relativno jednostavne generalizacijske procese.

U skladu sa OGC-jevom preporukom, Burghardt i dr. (2005.) predlažu servise za generalizaciju kao dopunu postojećim, gore navedenim servisima. Predloženi servisi su poredani po mogućnostima generalizacijskih postupaka. Prvi takav servis koristio bi u podržavanju generalizacije i putem njega bi se mogli obavljati jednostavni postupci poput stvaranja buffer zona oko točkastih podataka. Drugi servis predstavlja nadopunu na prvi spomenuti servis i njime bi se mogli obaviti složeniji postupci sažimanja, pomicanja ili pojednostavljenja. Treći predloženi servis bi integrirao prva dva servisa i takav servis bi bio najbliži automatskoj generalizaciji, jer bi mogao ulančavati različite servise i obavljati najsloženije generalizacijske postupke.

3.3. Realizirani web generalizacijski servisi i njihova svojstva

U tablici 3.1 prikazan je pregled trenutno javno dostupnih web-servisa za generalizaciju sa pripadajućim općim karakteristikama i podržanim postupcima kartografske generalizacije. Svojstva WebGen–WPS-a su preuzeta iz Jezdić i Tutić (2013.), za MapShaper iz Bloch i Harrower (2006.), a za GiMoDig iz Sester i dr. (2004.).

Tablica 3.1 Pregled web-servisa i njihove specifikacije

NAZIV SERVISA	OPĆE KARAKTERISTIKE	PODRŽANI POSTUPCI KARTOGRAFSKE GENERALIZACIJE
WebGen – WPS	– Implementacija WPS standarda – Dostupan putem dodatka OpenJump temeljenog na Javi – Sukladan OGC standardima/Open source – Ručno ili automatsko pozivanje – Čitanje *shapefile* datoteka i jednostavnih GML-dokumenata – Analiza topologije i operacije preklapanja	– Pojednostavljivanje građevina – Zaglađivanje linija – Razmještanje građevina – Uklanjanje površinskih objekata – Tipizacija pravokutnicima – Buffer zone (koridori) – Generiranje centroida – Jednostavno povećavanje površina – Tipizacija zgrada

MapShaper	– C++ kôd – Brzo i efikasno izvođenje – Konvertiranje *shapefile* datoteka u topološke podatke – Više postupaka pojednostavljivanja – Više postupaka zaglađivanja – Neograničeno vraćanje koraka („undo“) – Rezultat u *Shapefileu*, GeoJSON-u i TopoJSON-u – Open source – Potreban Adobe Flash Player	– Pojednostavljivanje linija s tri različite metode: Douglas-Peuckerova, Visvalingham-Wyattova i Weighted Visvalingham metoda – Dvije vrste zaglađivanja linija: putem matematički definirane neprekidne krivulje (Bezierova krivulja) ili smještaj čvorova na polilinije da se aproksimiraju glatke krivulje (rezultati u *Shapefile* formatu)
GiMoDig (Geospatial info-mobility service by real-time data integration and generalization)	– XML/GML jezik – Generalizacija u stvarnom vremenu – Pogodno za mobilne uređaje	– Pojednostavljivanje linija s dvije metode: Douglas-Peuckerov i Langov algoritam – Zaglađivanje linije s Gaussovim algoritmom

	– Pristup temeljen na MRDB – Konačna karta u SVG formatu (Scalable Vector Format)	– Algoritmi filtriranja po površini/dužini – Algoritmi izrezivanja objekata – Izbor linija izohipsi – Pojednostavljenje građevina – Sažimanje građevina – Tipizacija zgrada – Pojednostavljivanje cestovne mreže – Različit stupanj zuma

Sa sve većim rastom kvalitete prostornih podataka i sve boljim performansama mobilnih uređaja javlja se potreba za snažnim algoritmima obrade prostornih podataka koji će tražene informacije korisnicima pružiti brzo, efikasno i pouzdano. Od navedenih web servisa GiMoDig je prvi predstavio generalizacijske postupke namijenjene mobilnim uređajima te implementirao različite stupnjeve zuma što je značajno poboljšanje naspram ostala dva spomenuta servisa. Budući da su ostala svojstva slična u sva tri navedena web-servisa, zaključujem da je GiMoDig, zahvaljujući prilagođenosti mobilnim uređajima, trenutno najpodobniji web-servis za turističke zajednice.

4. WPS – WEB PROCESSING SERVICE

Web Proccesing Service (u daljnjem tekstu WPS) definira mehanizam putem kojeg klijent može podnijeti zadatak serveru na kojem će se on izvršiti. Servis na serveru predstavlja cjelinu koja pruža jedan ili više procesa, ili pojedinačnu obradu zadataka (npr. spajanje dva rasterska podatka mogao bi biti jedan proces). U takvom duhu, bilo koji server može izvesti višebrojne različite i ne nužno povezane procese. Specifikacija WPS-a ukazuje da se za svu komunikaciju treba koristiti XML (eXtensible Markup Language). XML-dokumenti sastavljeni su od pojedinačnih elemenata, pa tako element može sadržavati drugi element, a bilo koji element može sadržavati atribute koji opisuju taj element. Jednostavan primjer XML dokumenta bi mogao biti:

<krajolik>

<ime_krajolika=“Park Maksimir“/>

<tip_drvo=“Breza“

visina=“5“/>

</krajolik>

Ovaj XML dokument opisuje krajolik, što je označeno elementom nazvanim „krajolik“. Atribut nazvan „ime_krajolika“ upućuje da je riječ o parku Maksimir. Podelement nazvan „tip_drvo“ opisuje vrstu drva i njegovu visinu. Element je zatvoren, odnosno završen, ponavljanjem imena elemenata s kosom crtom ispred. Prema Michael i Ames (2007.) XML je dizajniran da bude „direktno korišten preko Interneta“, „ljudima čitljiv i jasan“, „formalan i sažet“, i „jednostavan za kreiranje“.

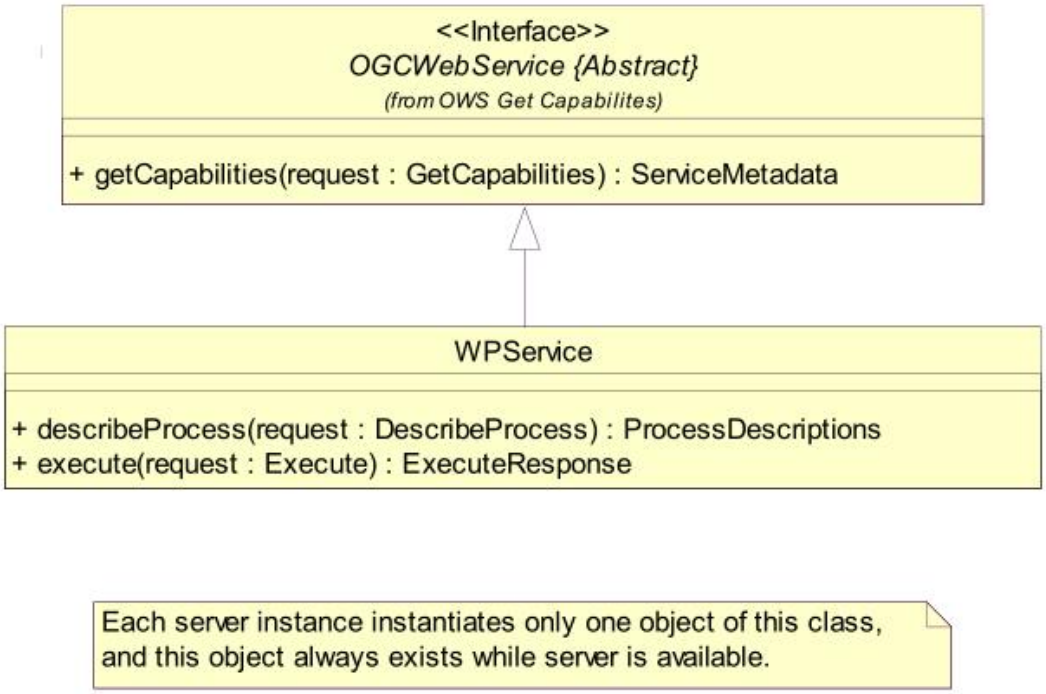
Glavni cilj WPS-a je definirati komunikaciju između udaljenih procesa. Postoje tri ključna upita na WPS server (Michael i Ames, 2007.): *GetCapabilities* (dohvati mogućnosti), *DescribeProcess* (opiši proces) i *Execute* (izvrši). Prvi upit traži od servera popis individualnih procesa koji su dostupni na tom serveru, uključujući kratke sažetke i ključne

riječi. Jednom kad je identificiramo proces putem odgovora, možemo poslati *DescribeProcess* upit. Odgovor na taj upit uključuje informacije o procesu dobivenom iz *GetCapabilities* upita, plus detaljnije informacije o potrebnim ulaznim parametrima za proces, bilo da je riječ o jednostavnom unosu (npr. jednostavan broj poput 24) ili složenom unosu (npr. datoteka s podacima). Ako odgovor *DescribeProcess* upita ukazuje da je to proces koji korisnik želi izvršiti poziva se treći upit. *Execute* upit zahtjeva da server zapravo izvrši neku operaciju. Potrebni parametri uključuju ime procesa kao i potrebne unose za pojedini proces. Odgovor na *Execute* upit je *ExecuteResponse* dokument, XML-dokument koji ukazuje na status procesa, indicira koji su ulazni parametri korišteni, i pruža ili jednostavne doslovne vrijednosti izlaza ili poveznice na kompleksne izlaze. Status procesa može biti *ProcessAccepted* koji nam ukazuje da je proces zaprimljen i čeka na izvršenje; *ProcessStarted* upućuje da se proces izvršava; *ProcessSucceeded* znači da je proces završen; ili *ProcessFailed* iskazuje da je u tijeku procesa došlo do pogreške. Ukoliko je status *ProcessAccepted* ili *ProcessStarted* status je popraćen s atributom koji upućuje gdje se sljedeći *ExecuteResponse* dokument može pronaći. U tom slučaju korisnik može provjeriti status procesa upitom za sljedećim *ExecuteResponse* dokumentom. U slučaju statusa *ProcessStarted* dostupni mogu biti i poruka sa statusom, te postotak napretka. Ako je status procesa *ProcessFailed*, *ExecutionResponse* dokument sadrži kod pogreške ugrađenog u XML *ExceptionReport* element, koji može biti jedan od pet kodova pogrešaka:

1. *MissingParameterValue* – nedostaje vrijednost parametra
2. *InvalidParameterValue* – unešena je pogrešna vrijednost parametra
3. *NoApplicableCode* – nema primjenjivog kôda
4. *ServerBusy* – server je zauzet
5. *FileSizeExceeded* – premašena veličina datoteke

Ukoliko je proces uspješan, izlazni dokument će također uključiti izlazne vrijednosti (u slučaju jednostavnih vrijednosti) odnosno, URL poveznice na kompleksne izlazne vrijednosti (poput datoteke s rasterskim podacima). Ukoliko je proizvedena jedna kompleksna izlazna vrijednost, onda takva vrijednost može biti direktno vraćena umjesto *ExecuteResponse* dokumenta. Zajedno, ove tri operacije upita i odgovora tvore većinu funkcionalnosti WPS-a.

Slika 4.1 je jednostavan UML-dijagram koji opisuje WPS-sučelje. Ovaj dijagram pokazuje da WPS-sučelje nasljeđuje *getCapabilities* operacije od OGCWebService sučelja i dodaje *DescribeProcess* i *Execute* operacije.



Slika 4.1 – UML dijagram WPS sučelja (preuzeto iz OGC, 2007)

Uzmimo u obzir jednostavan slučaj procesa koji presijeca dvije linije. Odgovor na *getCapabilities* upit može ukazati da WPS podržava operaciju zvanu *„intersect“* (presjek) i da je ta operacija ograničena na presjek linije sa linijom. Odgovor na *DescribeProcess* upit za „intersect“ proces ukazuje da su potrebna dva unosa, imenom: „Linija_1“ i „Linija_2“, i ti unosi moraju biti poslani u GML-u. Proces ima jedan izlaz i može biti dostupan na webu. Korisnik pokreće proces pozivanjem *Execute* operacije i može izabrati dvije linije koje želi obraditi i zatražiti izlaz pohranjen kao resurs dostupan putem weba. Nakon izvršenja, proces vraća *ExecuteResponse* XML-dokument koji identificira ulazne i izlazne podatke, ukazuje na eventualne pogreške i reference na izlazne podatke dostupan na webu.

WPS profili opisuju kako WPS treba biti konfiguriran da zadovolji procese standardizirane putem OGC-a. Profil aplikacija sastoji se od:

1. OGC URN koji jedinstveno identificira proces (obavezno) (URN – Uniform Resource Name – Jedinstveno ime resursa)
2. Odgovor na *DescribeProcess* upit za taj proces (obavezno)
3. Dokument koji opisuje proces i njegovu implementaciju (neobavezno, ali preporučeno)
4. WSDL opis tog procesa (neobavezno)

Profili mogu definirati svaki jedinstveni proces unutar skladišta podataka i svaka WPS-instanca se može odnositi na taj URN. Trenutna specifikacija potpuno podržava ovaj pristup standardiziranim servisima.

4.1. Ulančavanje servisa WPS-om

Ulančavanje WPS procesa olakšava stvaranje procesa koji se trebaju višestruko koristiti. WPS procesi mogu biti integrirani u lanac servisa na nekoliko načina:

1. Putem BPEL-a se mogu ulančavati servisi koji uključuju jedan ili više WPS procesa (BPEL – Business Process Execution Language)
2. WPS-proces može biti podešen da poziva druge WPS-procese i tako predstavlja sustav za ulančavanje servisa
3. Jednostavni lanci servisa mogu biti uključeni u *Execute* upit. Takav lanac servisa može biti čak izvršen i putem *Get* zahtjeva.

4.2. WPS i SOAP/WSDL

WPS je kompatibilan i sa WSDL-om i sa SOAP-om (SOAP – Simple Object Access Protocol) i definicije koje koristi WPS s ovim standardima nalaze se u specifikaciji OGC-ja. SOAP se može iskoristiti za pakiranje WPS-upita i odgovora. SOAP opisuje mehanizam razmjene poruka koji se sastoji od glavnog elementa, ali sadržaj tog elementa nije opisan. WPS opisuje mehanizam razmjene poruka koji se može koristiti kad SOAP nije potreban (za sigurnost kao što je autorizacija), ali definira kako bi pojedini element trebao izgledati. Slični

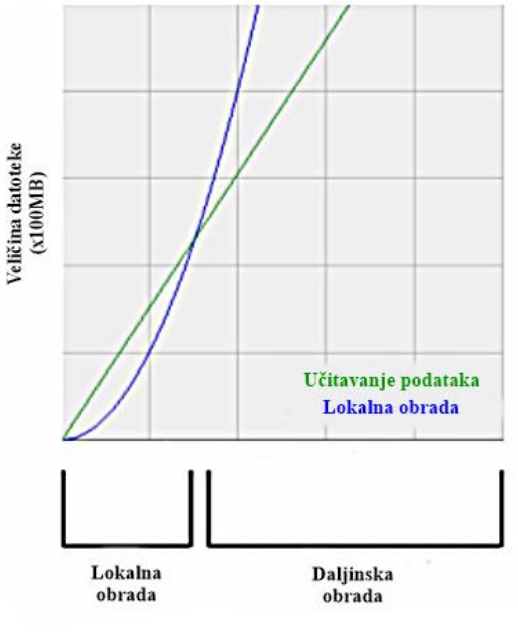
elementi su u WPS-specifikaciji pojednostavljeni te takva standardizacija dramatično pojednostavljuje količinu kodiranja potrebnu za implementaciju sučelja bilo kojeg novog servisa. WSDL samo opisuje servis, dok WPS definira općeniti izgled svakog servisa. WSDL nudi manje opsežan mehanizam definiran putem specifikacije WPS sučelja. WPS nudi više dokumentacije i naprednije mogućnosti ulančavanja servisa. WSDL traži posebno povezane informacije koje se mogu naći jedino u WPS profilima, te je stoga nemoguće generirati samo jedan, opći, WSDL dokument koji bi opisao svaku WPS implementaciju. WPS je moguće koristiti bez WSDL-a samo ukoliko je traženo dinamičko vezivanje na dobro poznate servise. WSDL je potreban kako bi olakšao dinamičko vezivanje na dinamičke servise. (OGC, 2007.)

WPS nudi sljedeće prednosti u odnosu na trenutne SOAP/WSDL specifikacije: (OGC, 2007.)

1. Podržava OGC-jevu *GetCapabilities* konstrukciju, koja pojednostavljuje njezinu prilagodbu unutar geoprostorne zajednice koja je već prihvatila OGC specifikacije
2. Za jednu vrijednost rezultata, podržava direktno vraćanje bez XML dokumenta
3. Specificira kako determinirati status procesa, koji omogućuje podršku dugotrajnim procesima
4. Detaljno specificira kako opisati ulaz i izlaz, što olakšava razvoj softvera i klijenata
5. Specificira spremanje izlaznih podataka procesa, što olakšava ulančavanje servisa
6. Specificira kako referencirati web-resurse poput ulaza/izlaza, što olakšava ulančavanje servisa
7. Specificira kako definirati kompleksne ulazne podatke, što olakšava razvoj višestruko upotrebljivih procesa
8. Nudi mehanizam otkrivanja servisa koji može biti iskorišten bez složenosti WSDL-a, dok istovremeno podržava opciju korištenja WSDL-a kad je potrebno olakšati povezivanje
9. Olakšava ulančavanje servisa, budući da WPS može biti konstruiran da poziva druge servise, uključujući i druge WPS-e
10. Definira standardne poruke pogrešaka, koje pojednostavljuju rukovanje pogreškama za višestruke procese

4.3. Upotreba WPS-a

Zbog funkcionalnosti geoprocesiranja predložena je neograničenost u okviru ili prirodi WPS-a, te je prijedlog obećavajući za korištenje softvera bez da korisnik brine o mogućim bugovima ili o eventualnoj nadogradnji softvera. Bez obzira na neograničenost okvira, mnoge operacije se mogu izvršiti puno brže lokalno (na korisnikovom osobnom računalu) nego daljinski (na centralnom serveru), pogotovo nakon što se uzme u obzir vrijeme potrebno za učitavanje ulaznih podataka i preuzimanje rezultata. Pri određivanju hoće li se koristiti lokalna ili daljinska obrada podataka, treba uzeti u obzir puno više faktora od veličine uključenih datoteka. Također značajnu ulogu igra složenost izračuna. Ukoliko proces traje nekoliko sati na maloj datoteci, bolje je obraditi podatke daljinski. Ako pak zadatak nije složen i glavnina posla leži u prolaženju kroz velik volumen lokalno pohranjenih podataka, lokalna obrada često može biti efikasnija. Kao što je prikazano na slici 4.2 daljinska obrada ima značajno mjesto u modernom računalstvu. Veća snaga obrade na velikim serverima može bez problema smanjiti trošak vremenski zahtjevnih i složenih zadataka, posebice u kombinaciji s brzim mrežama. Podaci bi trebali biti poslani na server s većom moći obrade umjesto korištenja lokalnog, sporijeg računala, pogotovo kada bi vrijeme obrade lokalno, bilo veće nego kombinirano vrijeme slanja podataka, daljinske obrade i ponovnog preuzimanja. (Michael i Ames, 2007.)



Slika 4.2 - Lokalna obrada i daljinska obrada (prema Michael i Ames 2007.)

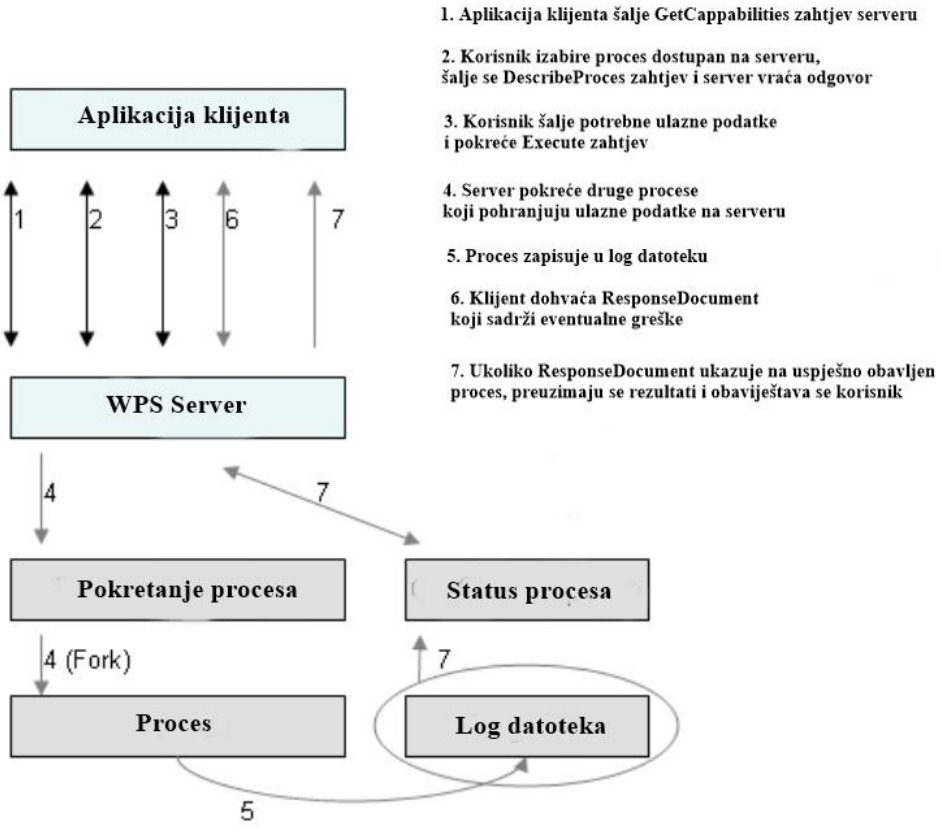
Daljinska obrada je također idealan izbor ukoliko je algoritam relativno nov i pod aktivnim razvojem. U tom slučaju, korisnik ne mora svako malo nadograđivati softver s novim dodacima. Jedino server zahtjeva nadogradnju kako bi odrazio novi algoritam ili kôd. U konačnici svi korisnici automatski dobiju pristup najnovijoj i najtočnijoj verziji procesa koristeći servis, pa smatram da je daljinsku obradu podataka poželjno koristiti i za datoteke manjih veličina, kako bi se izbjegle moguće pogreške zbog neažurnosti softvera.

4.4. Mogućnosti implementiranja WPS-a

Mnoge operacije potrebne od strane WPS servera su jednostavne operacije na metapodacima: pružanje informacije o individualnim procesima (npr. potrebni ulazi) i lista procesa dostupnih na serveru. WPS server će u idealnom slučaju učitati informacije o dostupnim procesima ili iz konfiguracijske datoteke (vjerojatno jedna po procesu) ili iz baze podataka, kojima se kod napisan za WPS server čini ponovno upotrebljiv dodajući dodatne procese u konfiguracijsku datoteku ili u bazu podataka. (Michael i Ames, 2007.)

Pregled predloženog toka komuniciranja između klijenta i servera je prikazano na slici 4.3. Početno, aplikacija klijenta šalje upit na server, sa *GetCapabilities* zahtjevom. WPS server

tada vraća XML dokument u skladu sa WPS shemom. Klijent pruža korisniku listu dostupnih procesa na serveru. Korisnik tada izabire jedan od tih procesa, te klijent zahtijeva dodatne informacije o tom procesu slanjem *DescribeProcess* upita. Klijent pomaže korisniku prikupljanje i unos potrebnih ulaznih parametara za uspješno izvršenje procesa i tada inicira proces na serveru slanjem *Execute* upita. Zbog bržeg izvršavanja procesa i zbog sigurnosti poželjno je započeti potpuno novi proces, pa ukoliko se proces obustavi, neće se zaustaviti i cijeli WPS server, što povlači da je pokretanje novog procesa dobro rješenje.



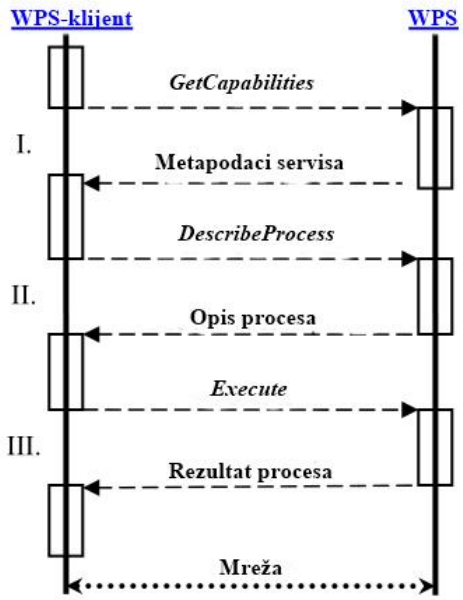
Slika 4.3 - Preporučeni tijek komunikacije između klijenta i servera (prema Michael i Ames, 2007.)

Kako je prije objašnjeno, početni *ExecuteResponse* dokument obavještava klijenta gdje može pronaći status procesa u tijeku (slijedeći *ExecuteResponse*), kao i obavijesti o uspjehu, neuspjehu, prihvaćenom ili odbijenom poslu. Idealan dizajn WPS-servera bi bio takav da se *ExecuteResponse* dokument može pronaći na stalnoj lokaciji, po mogućnosti koristeći isti URL za najnovije dokumente, pa bi dobro rješenje bilo stvaranje posebne web-stranice na kojoj bi se taj dokument mogao pronaći. Jednom kad *ExecuteResponse* dokument ukaže da je

proces gotov, server mora vratiti ili poruku s pogreškom ili informaciju gdje se generirani podaci mogu preuzeti ili zadržati. Klijent tada pohranjuje podatke i sa time se WPS proces prekida.

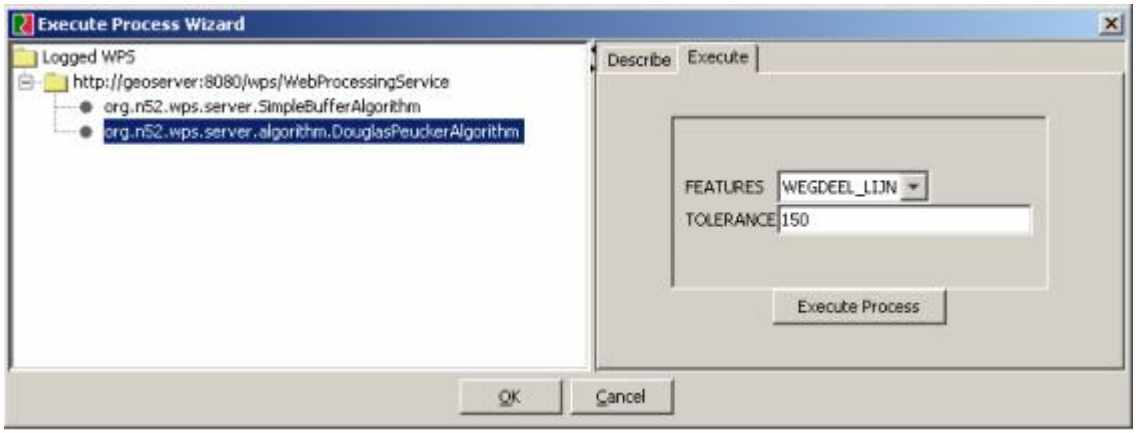
4.5. Generalizacija WPS-om

U ovom poglavlju bit će prikazana generalizacija korištenjem WPS-a. Pretpostavimo da moramo upotrijebiti algoritam pojednostavljivanja poput Douglas-Peuckerovog algoritma na cestovne podatke. Moramo pristupiti WPS-u i proći kroz klijent-server komunikaciju kao što je prikazano u dijagramu na slici 4.4.



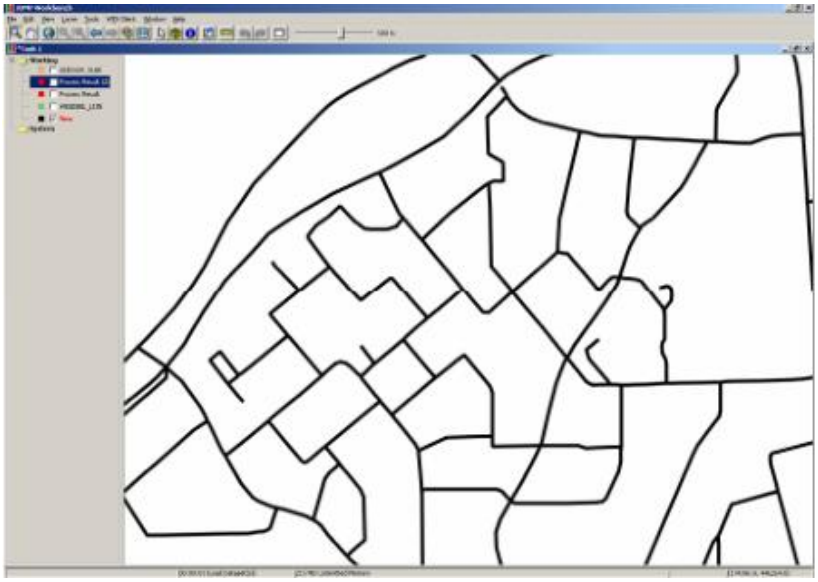
Slika 4.4 - Osnovna klijent-servis komunikacija (prema Foester i Stoter, 2006.)

Kako bi koristili WPS moramo proslijediti URL od WPS-a do klijenta. Klijent će primiti metapodatke servisa kroz *GetCapabilities* operaciju (korak I.). Kroz operaciju *DescribeProcess* klijent prima informacije o procesu nazvanom Douglas-Peucker algoritam (korak II.). Temeljem vraćenog opisa procesa klijent je u mogućnosti generirati formu prikazanu na slici 4.5. Douglas-Peuckerov algoritam će biti pokrenut s traženim parametrima (tolerancija: 150 metara) i originalnim podacima s *Execute* operacijom (korak III.) Nakon toga, WPS vraća obrađeni rezultat.

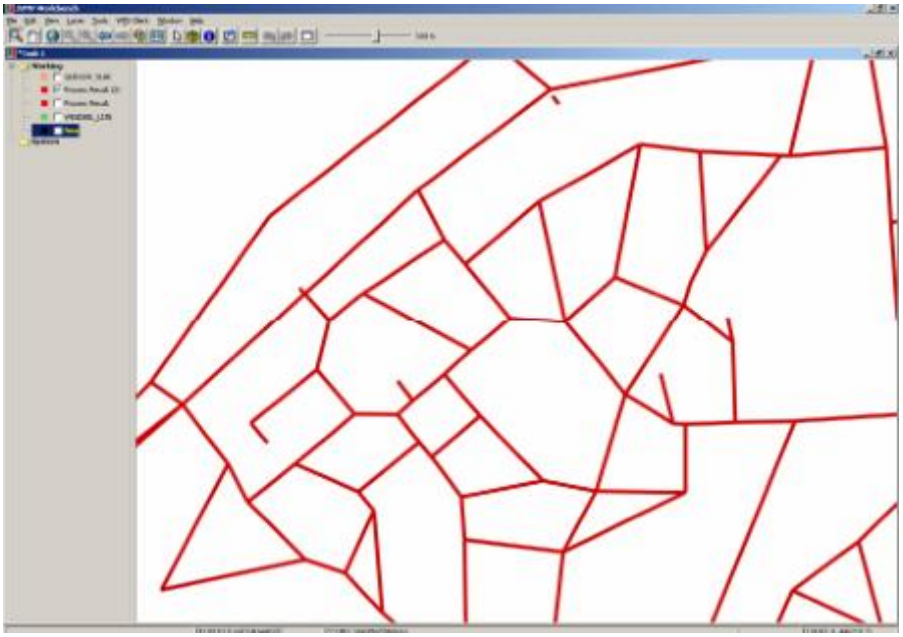


Slika 4.5 - Jednostavan oblik konfiguracije generalizacijskog procesa (preuzeto iz Foester i Stoter, 2006.)

Prikaz izvornih podataka i prikaz dobiven Douglas-Peuckerovim algoritmom, koji je pružen od WPS-a su prikazani na slici 4.6 i 4.7. Iako se javljaju neke pogreške zbog loše topološke strukture ulaznih podataka, ovaj kratak primjer pokazuje da je WPS-om moguće izvršiti generalizacijske postupke nad linijskim podacima.



Slika 4.6 – Prikaz izvornih cestovnih podataka (preuzeto iz Foester i Stoter, 2006.)



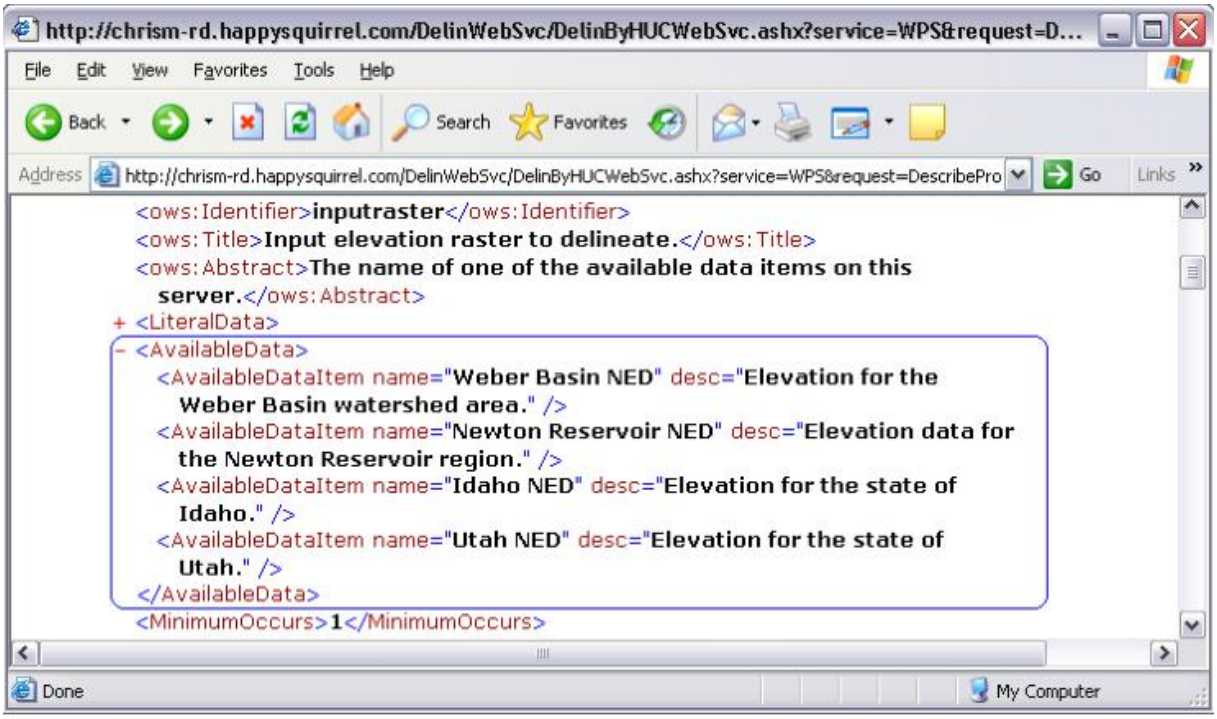
Slika 4.7 - Pojednostavljeni izvorni podaci (preuzeto iz Foester i Stoter, 2006.)

4.6. Predložena poboljšanja, problemi i potencijalna rješenja

WPS bi mogao biti poboljšan sa šest ključnih promjena. Ove promjene uključuju dva dodatna elementa u *DescribeProcess* odgovoru dostupnom od servera, koji opisuje ulaz i izlaz danog procesa, kao i mehanizam putem kojeg klijent može otkazati upit koji je u tijeku. Potencijalne promjene također uključuju ispravljanje nekonzistentnosti u ponašanju, pružajući dodatne formate za rukovanje pogreškama i postojanje jedino jedne ulazne točke za svaki proces i po mogućnosti i za svaki server.

Prva predložena promjena je dodavanje elementa u XML dokument koji je vraćen *DescribeProcess* zahtjevom. Trenutno, traženi ulazni podaci su prikazani u tom dokumentu bez jasnog opisa u kakvom formatu korisnik te ulazne podatke mora poslati. Potrebni podaci mogu biti doslovne vrijednost poput "23", datoteke zadanog tipa ili deseci drugih metoda za prikupljanje podataka. U testnoj implementaciji, Michael i Ames 2007. predstavili su novi XML element nazvan *PromptMethod* za prikaz tog problema. Ovaj element može sadržavati vrijednosti poput *browseforvector, browseforraster, getboundingbox, getmacthingregex.* Ove vrijednosti bi trebale korisnicima olakšati definiranje ulaznih podataka.

Prema Michaelu i Amesu, 2007. druga predložena promjena je dodavanje još jednog retka u *DescribeProcess* odgovor. Trenutno ne postoji mehanizam po kojem server može dati listu podataka dostupnih na serveru za upotrebu danim procesom. Odličan primjer je u procesu u definiranju riječnih tokova temeljenih na rasterskim podacima o visinama. Takve rasterske datoteke su obično velike, te je nepoželjno, a ponekad i nemoguće učitati cijelu datoteku na server. Takvi rasterski podaci se ne mijenjaju često i mogu biti spremljeni na serveru radi uštede vremena. U testnoj implementaciji Michaela i Amesa, 2007. predstavljen je XML element nazvan *AvailableData*, čija je funkcija prikaz svih dostupnih podataka na serveru. Predloženi element vidljiv je na slici 4.8. Ne samo da taj element ubrzava obradu uklanjanjem potrebe za učitavanjem ulaznih podataka, već i kreira sredstva s kojima WPS server može predstavljati skladište podataka kao i njihovu obradu i vraćanje obrađenih podataka.



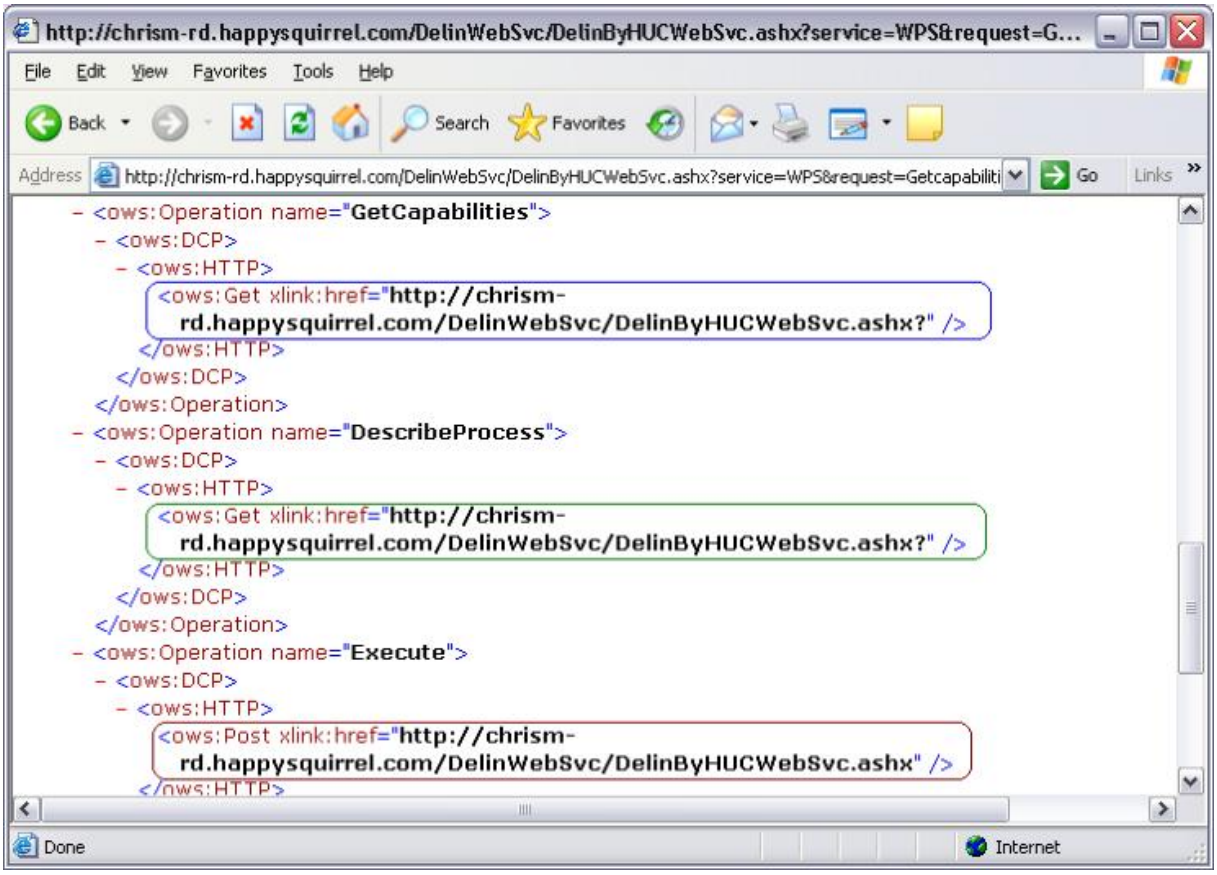
Slika 4.8 - Predloženi *AvailableData* element u *DescribeProcess* odgovoru (preuzeto iz Michael i Ames 2007.)

U trenutnoj strukturi WPS-a postoji nekonzistentnost: što će se dogoditi nakon slanja zadatka serveru, odnosno korisnik nije siguran da li će mu željeni proces vratiti *ExecuteResponse* dokument. Ukoliko će proces rezultirati jednim odgovorom, a *Execute* upit je napravljen tako da ne mora nužno vratiti spremljene podatke, WPS će dozvoliti procesu trenutno vraćanje

rezultata, prije nego se stvori XML-dokument *ExecuteResponse*. Ako postoji više od jednog rezultata, ili je procesu zadano spremanje rezultata tada će se *ExecuteResponse* dokument stvoriti. Takvo ponašanje je nekonzistentno jer bilo koji proces može vratiti ili višestruke rezultate ili jedan rezultat, ovisno o zadanim parametrima pruženim procesu. Umjesto toga, jednostavnije je da se uvijek vraća *ExecuteResponse* dokument koji sprema bilo kakve izlazne podatke na server dok ih klijent ne preuzme. Treća predložena promjena je da se uvijek vraća *ExecuteResponse* dokument u svrhu pružanja veće konzistentnosti.

Nakon podnošenja zadatka na WPS server, ukoliko korisnik shvati da je izabrao pogrešan proces, poželjno bi bilo da se zahtjevana operacija otkaže, što u trenutnoj WPS strukturi nije dostupno. U testnoj implementaciji (četvrta predložena promjena) Michael i Ames 2007. predlažu upotrebu cancel request URL u *ExecuteResponse* dokumentu, uz postojeći URL koji ukazuje gdje će se slijedeći *ExecuteResponse* dokument pronaći. Sa takvom dopunom WPS-a ako korisnik želi obustaviti proces, treba pristupiti URL-u da se obustava zahtijevanog procesa pokrene, sprječavajući trošenje vremena na serveru na neželjene procese.

Peta predložena promjena WPS strukture je da ima jednu ulaznu točku za svaki mogući zahtjev (*GetCapabilities*, *DescribeProcess*, *Execute*) na danom procesu. Idealno, jedinstvena ulazna točka bi bila korištena ne samo za svaki zahtjev na proces, nego i na svaki proces dostupan na tom serveru. Dakle svaki od zahtjeva koje proces podržava može imati različiti URL za izvođenje zahtjeva, kao što je prikazano na slici 4.9. Na toj slici *GetCapabilities* URL je zaokružen plavom, *DescribeProcess* URL zelenom, a *Execute* URL crvenom bojom. Ovdje je svaki URL jednak, što čini održavanje i implementaciju lakšom, iako je prihvatljivo postojanje različitih URL-ova za svaku operaciju. Doduše, lako mogu nastati zabune i pogreške s različitim URL-ovima za takve operacije. To se može spriječiti traženjem jedinstvenog URL-a za sva tri zahtjeva koja proces mora podržati.



Slika 4.9 - Mogućnost postojanja različitih URL-ova za svaki zahtjev na isti proces (preuzeto iz Michael i Ames 2007.)

Posljednja, šesta predložena promjena je implementirati bolje strukturiran sustav pogrešaka. Trenutno postoji samo nekoliko tipova pogrešaka (*MissingParameteValue, InvalidParameterValue, NoApplicableCode, ServerBusy* i *FileSizeExceeded)* koji su ograničavajući, posebice kad se pokušava problem razlučiti automatski. S malim brojem tipova pogrešaka ne postoji općeniti način za rukovanje pogreškama bez uplitanja korisnika. Složenija struktura pogrešaka bi omogućila klijentima razumijevanje prirode pogreške koja je nastupila, te moguće automatsko uklanjanje pogreške. Mogućnost izoliranja korisnika od grešaka je važan dio bilo kojeg standarda i navedeno bi bilo moguće s podrobnijom hijerarhijom pogrešaka.

5. IMPLEMENTACIJA

Cilj praktičnog dijela diplomskog rada je bio izraditi prototip modela kartografske generalizacije podržan WPS-om na izabranom području Republike Hrvatske. Izabrano je područje centra grada Zagreba, s objektima vezanim uz turizam (kafići, restorani, hoteli). Na internetu su pronađeni primjeri generalizacije točkastih objekata te je odlučeno te algoritme primijeniti putem WPS-a. Primjeri su dostupni na adresama: http://bombsight.org/, http://onionmap.io/,http://openlayers.org/en/v3.5.0/examples/earthquake-clusters.html
Odlučeno je da će se točkasti objekti preuzeti s OpenStreetMapa (OSM). Preuzeti podaci s OSM-a su korisnički definirani, te su često nepotpuni, netočni, odnosno neupotrebljivi. Kako bi se u daljnja razmatranja ušlo s potpunim podacima, oni su obrađeni u QGIS-u, tako da su metodom izbora izdvojeni samo oni podaci koji imaju definiran naziv. Budući da je sustav trebao biti zasnovan na WPS-servisu nakon razmatranja dostupnih otvorenih sustava odlučeno je da se za WPS server koristi Geoserver (vidi poglavlje 5.1.), a za WPS klijenta OpenLayers 2.13 (vidi poglavlje 5.2).

5.1. Geoserver

Geoserver je na Javi temeljen softver koji omogućava korisnicima prikaz i uređivanje prostornih podataka. Geoserver je moćan alat za kreiranje karata i posebice za dijeljenje podataka jer se temelji na OGC-standardima. Uz implementiran Web Map Service (WMS) standard, Geoserverom se mogu izraditi karte u različitim izlaznim formatima. OpenLayers (vidi poglavlje 5.2.) su integrirani u Geoserver što čini izradu karata brzom i jednostavnom. Geoserver također sadrži Web Feature Service (WFS) standard, koji dozvoljava dijeljenje i uređivanja podataka korištenih kod izrade karte, što omogućuje ostalim korisnicima dijeljenje podataka na njihovim internetskim stranicama i aplikacijama. Geoserver može prikazati podatke na bilo kojoj popularnoj aplikaciji za kartiranje poput Google Maps, Google Earth, Yahoo Maps i Microsoft Virtual Earth, te se može povezati s tradicionalnim GIS sustavima poput ESRI-jevog ArcGIS-a. U Geoserveru se kao ekstenzija može dodati Web Processing Service (WPS) što je za potrebe rada i najzanimljivije. Prema Iacovella i Younblood (2013) glavna prednost WPS-a na Geoserveru naspram samostalnog WPS-a je u direktnoj integraciji

s ostalim servisima i podacima na Geoserveru, što omogućuje stvaranje procesa na podacima već pohranjenim na Geoserveru, pa se smanjuje broj potrebnih upita za izvršenje procesa. Uz to je moguće i rezultate procesa pohraniti kao novi sloj u skladištu podataka na Geoserveru, tako da WPS možemo shvatiti kao potpuni alat za daljinsku obradu podataka, sposobnog za učitavanje i ispis podataka na Geoserveru.

Uz pretpostavku da je Geoserver instaliran lokalno, ekstenzija za WPS se preuzme sa službene stranice Geoservera, te se podaci spreme u posebnu datoteku unutar paketa Geoservera. Ukoliko su svi postupci točno izvršeni nakon ponovnog pokretanja Geoservera ekstenzija za WPS bi trebala biti vidljiva u koloni *Service Capabilities*. Kao što je u prethodnim poglavljima spomenuto, WPS definira tri operacije: *GetCapabilities*, *DescribeProcess* i *Execute*. Obavezni parametri kod definiranja *GetCapabilities* operacije na Geoserveru su: *service=WPS, version=1.0.0* i *request=GetCapabilities*. Primjer *GetCapabilities* upita je:

```
http://localhost:8080/geoserver/ows?service=WPS&version=1.0.0&request=GetCapabilities
```

Za definiranje *DescribeProcess* operacije potrebno je na primjer iznad dodati dio u kojem se definira upit ključnom riječi *identifier*. Moguće je definirati upite za višestruke procese, stvaranjem upita odvojenog zarezom (identifier=JTS:buffer, gs:Clip), te je važno napomenuti da mora postojati minimalno jedan proces u upitu. Odgovor na upit stvara XML-dokument s metapodacima o svakom željenom procesu, uključujući sljedeće:

- Ime procesa, naslov i sažetak
- Za svaki ulazni i izlazni parametar: ime, naslov, sažetak, podržane tipove podataka i format.

Sljedeći primjer je dan za proces JTS:buffer :

```
http://localhost:8080/geoserver/ows?service=WPS&version=1.0.0&request=DescribeProcess&identifier=JTS:buffer
```

Upitom dobiven odgovor spremljen u XML-dokumentu sadrži:

Naslov: „Buffers a geometry using a certain distance“ / „Stvaranje buffer zone oko geometrije koristeći određenu udaljenost“

Ulazni parametri: **geom**: Geometrija oko koje se stvara buffer zona (obavezan podatak)

distance: Udaljenost buffer zone od geometrije (obavezan podatak)

quadrant segments: Broj kvadratnih segmenata. >0 za okrugle spojeve, 0 za ravne spojeve, a <0 za spojeve pod 45° (neobavezan podatak)

capstyle: Stil bufer zone – okruglo, pravokutno ili kvadratno (neobavezan podatak)

Izlazni formati: GML 3.1.1, GML 2.1.2 ili WKT

Execute operacija izvršava proces sa zadanim ulaznim vrijednostima i traženim izlaznim podacima. Upit se može kreirati kao GET ili POST zahtjev s XML-dokumentom. S obzirom na složenost strukture upita učestalije se koristi POST oblik upita. Ulazni i željeni izlazni podaci ovise o procesu koji se želi izvršiti. Geoserver nudi razne procese za obradu geometrijskih podataka, objekata i podataka o pokrovu. Takvi procesi su već implementirani u Geoserver i popis takvih procesa se može pronaći na web stranicama Geoservera, a budući da želimo implementirati vlastiti WPS proces koji će generalizirati točkaste objekte po geometriji i atributu trebamo se voditi uputama navedenim u nastavku. Prije implementacije vlastitog procesa pokušano je implementirati primjer dostupan na adresi: docs.geoserver.org/latest/en/developer/programming-guide/wps-services/implementing.html. Kod pokušaja implementacije navedenog primjera naišlo se na probleme. Uz nedovoljno jasnu dokumentaciju sustav je u kontinuiranom razvoju pa se online biblioteke u nazivu i sadržaju, te je gotovo nemoguće pronaći ispravne datoteke potrebne za implementaciju spomenutog procesa. Ipak ti problemi su uz pomoć programera Tomislava Poljaka uspješno riješeni. Za uspješnu implementaciju treba slijediti navedene korake. Prije svega potrebno je instalirati Geoserver s mogućnostima daljnjeg razvijanja ("Geoserver for developers"). Također potrebno je instalirati Apache Maven softver za upravljanje projektima i Java Developer Kit. Naravno da takav Geoserver mora imati WPS ekstenziju koja se u Mavenu

dodaje naredbom u naredbenom retku: `mvn -P allExtensions`. Geoserver treba biti dograđen s –Pwps profilom. Prvi korak kod stvaranja novog modula WPS procesa je stvaranje Maven projekta. Za ovaj primjer projekt će se nazvati "hello_wps". Prvi korak je kreiranje lokalnog direktorija nazvane hello_wps. Zatim se stvara datoteka pom.xml unutar hello_wps direktorija, koja mora izgledati ovako:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      http://maven.apache.org/maven-v4_0_0.xsd ">
<modelVersion>4.0.0</modelVersion>
<groupId>org.geoserver</groupId>
<artifactId>hello_wps</artifactId>
<packaging>jar</packaging>
<version>2.5-SNAPSHOT</version>
<name>hello_wps</name>
<dependencies>
      <dependency>
            <groupId>org.geotools</groupId>
            <artifactId>gt-process</artifactId>
            <version>12-RC1</version>
      </dependency>
      <dependency>
            <groupId>org.geoserver</groupId>
            <artifactId>gs-main</artifactId>
            <version>2.8-SNAPSHOT</version>
      </dependency>
      <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
```

```
            <scope>test</scope>
      </dependency>
      <dependency>
            <groupId>com.mockrunner</groupId>
            <artifactId>mockrunner</artifactId>
            <version>0.3.1</version>
            <scope>test</scope>
      </dependency>
</dependencies>
<build>
<plugins>
      <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
            <source>1.7</source>
            <target>1.7</target>
            </configuration>
      </plugin>
</plugins>
</build>
<repositories>
      <repository>
            <id>maven2-repository.dev.java.net</id>
            <name>Java.net repository</name>
            <url>http://download.java.net/maven/2</url>
      </repository>
      <repository>
            <id>osgeo</id>
            <name>Open Source Geospatial Foundation Repository</name>
            <url>http://download.osgeo.org/webdav/geotools/</url>
```

```
      </repository>
       <repository>
            <id>boundless</id>
            <url>http://repo.boundlessgeo.com/main</url>
            <snapshots>
                  <enabled>true</enabled>
             </snapshots>
      </repository>
</repositories>
</project>
```

Potom se unutar direktorija *hello_wps* stvori direktorij *src*, pa unutar nje direktorij *main* i na kraju unutar nje direktorij *java*. Unutar direktorija *java* se stvara paket koji će sadržavati željeni WPS proces. U ovom primjeru stvaramo paket imena *org.geoserver.hello.wps* i unutar njega Java klasu naziva HelloWPS.java koja mora sadržavati definiran paket, učitane procese i parametre i željene rezultate kao u primjeru:

```
package org.geoserver.hello.wps;
import org.geotools.process.factory.DescribeParameter;
import org.geotools.process.factory.DescribeProcess;
import org.geotools.process.factory.DescribeResult;
import org.geoserver.wps.gs.GeoServerProcess;
@DescribeProcess(title="helloWPS", description="Hello WPS Sample")
public class HelloWPS implements GeoServerProcess {
   @DescribeResult(name="result", description="output result")
   public String execute(@DescribeParameter(name="name", description="name
to return") String name) {
         return "Hello, " + name;
   }}
```

Unutar direktorija *main* se stvori novi direktorij *resources* i u njoj se kreira datoteka *applicationContext.xml* sa sljedećim sadržajem:

```
<?xml version="1.0" encoding="UTF-8"?>

  <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

  <beans>

      <bean id="helloWPS" class="org.geoserver.hello.wps.HelloWPS"/>

  </beans>
```

Zatim se u direktoriju *hello_wps* pokrene Maven naredbom `mvn clean install`. Ovom naredbom se kompajlira kod i stvara se JAR datoteka unutar direktorija *target*. Takva se datoteka kopira u /WEB-INF/lib direktorij Geoservera i ponovno se Geoserver pokrene. Ukoliko je cijeli postupak uspješno izvršen proces bi trebao biti vidljiv unutar sekcije "WPS request builder" na Geoserveru pod nazivom *gs:HelloWPS*.

5.2. OpenLayers 2.13

OpenLayers je prema klijentu orijentirana JavaScript biblioteka otvorenog kôda za kreiranje interaktivnih web karata, vidljiva u bilo kojem web pregledniku. OpenLayers je alat kojim možemo vrlo jednostavno izraditi kvalitetne prikaze karata na webu. Važno je spomenuti da OpenLayers radi na svim platformama što uvelike olakšava njegovu upotrebu. OpenLayers omogućuje izradu karte od najjednostavnijeg prikaza informacija do složenijih, te nudi razne mogućnosti uređivanja poput dodavanja slojeva, kontrola i događaja na kartu. Kod dodavanja slojeva u OpenLayersu razlikujemo dvije vrste slojeva: osnovne i dodatne. Osnovni sloj je poseban sloj koji je uvijek vidljiv i putem njega su određena svojstva poput projekcije i razine zuma. Dodatni slojevi se polažu na osnovni sloj te se prema potrebama korisnika mogu uključiti odnosno isključiti. Perez (2012.) preporuča stalno korištenje osnovnog sloja, iako se u nekim slučajevima može izbjeći njegova upotreba. Kao osnovni sloj u radu koristit će se podloga preuzeta s OpenStreetMap (https://www.openstreetmap.org/), a kao dodatni slojevi koristit će se vektorski slojevi. Zbog lakšeg daljnjeg praćenja teksta treba spomenuti da se OpenLayers zasniva na objektno-orijentiranom programiranju. Kod upotrebe vektorskih slojeva koristimo klasu *Vector* i klase *Format, Protocol i Strategy*. Također klasa *Vector* poziva klasu *Object* i klasu *Geometry*. Klasa *Vector* može pozvati različita svojstva iz drugih klasa, a nama su za potrebe rada nazanimljivije klase *Filter*, *Protocol*, *Strategy* i *Style* pa će ovdje biti opisane.

Klasa *Protocol* upravlja komunikacijom vektorskog sloja s izvorom podataka. Kad koristimo klasu *Protocol* zapravo koristimo dvije podklase od klase *Protocol*. Učestalo korištene podklase su *Protocol.HTTP* i *Protocol.WFS*. *Protocol.HTTP* podklasa nam dopušta izravnu komunikaciju s izvorom vektorskih podataka dok nam *Protocol.WFS* podklasa dopušta komunikaciju s WFS-om.

Klasa *Strategy* kontrolira postavke upita na server i na koji način se koriste podaci vraćeni sa servera. Kao i kod klase *Protocol* kod klase *Strategy* također možemo pozvati podklase. Postoji lepeza podklasa koje možemo koristiti, a ovdje će biti opisane *Strategy.Cluster*, *Strategy.Filter*. Podklasa *Strategy.Cluster* ima svojstva udaljenosti i praga prikazivanja. Svojstvo udaljenosti definira udaljenost pri kojoj se dva ili više objekata smatraju jednim klasterom. Svojstvo udaljenosti možemo smatrati generalizacijskim operatorom sažimanja ugrađenim u OpenLayers. Svojstvo praga prikazivanja definira prag ispod kojeg će se na sloj dodati originalni objekt umjesto klastera. Podklasa *Strategy.Filter* ima svojstvo filtriranja, koje možemo poistovjetiti s generalizacijskim operatorom izbora. Važno je napomenuti da je svojstvo filtriranja unutar klase *Strategy* poprilično jednostavno, te za naprednije operacije izbora treba koristiti klasu *Filter*.

Klasa *Filter* kontrolira logiku kod usporedbe atributa objekata prilikom odlučivanja treba li se određeno pravilo primjeniti ili ne. Može se koristiti jednostavan filter ili možemo više filtera kombinirati zajedno, pa na taj način možemo gradove s više od 100 000 stanovnika ili manje od 50 000 prikazati određenim stilom. Izbor se može izvršiti po geometrijskim odnosima, poput provjere da li se dva objekta presjecaju ili da li su dvije točke preblizu za pregledan prikaz. Klasa *Filter* ima šest podklasa, od kojih će se ovdje pojasniti *Filter.Comparison*, *Filter.FeatureId*, *Filter.Logical* i *Filter.Spatial*. *Filter.Comparison* se temelji na usporedbi zadanih parametara između objekata od interesa. U ovoj podklasi mogu se uspoređivati objekti po tipu, svojstvu i vrijednosti atributa. Podklasa *Filter.FeatureId* izabire objekte temeljem njihovih ID vrijednosti. Takav izbor može biti koristan ukoliko znamo ID određenih objekata te njih možemo prikazati u određenom stilu ili ih uopće ne prikazati na karti. *Filter.Logical* nam dopušta upotrebu logičkih operatora na filterima, te na taj način možemo stvoriti kompleksne filtere veoma jednostavno. Postoje tri tipa logičkih operatora koje možemo koristiti: I („AND“), ILI („OR“) i NE („NOT“). *Filter.Spatial* omogućuje nam izbor

objekata temeljem njihove geografske informacije. Ova podklasa se koristi kod kreiranja vektorskog sloja. Kod stvaranja prostornog filtera moramo zadati tip i vrijednost. Vrijednost po kojoj se ovisno o tipu filtrira može biti *OpenLayers.Bounds* ili *OpenLayers.Geometry*. Neki tipovi dozvoljavaju dodatne parametre poput *distance* („udaljenost“) i *distanceUnits* („mjerne jedinice udaljenosti“). Neki od tipova su:

- *OpenLayers.Filter.Spatial.BBOX* – odabire objekte koji se nalaze unutar odgovarajućeg graničnog okvira definiranog putem objekta *OpenLayers.Bounds* u svojstvu vrijednosti
- *OpenLayers.Filter.Spatial.CONTAINS* – koristi se za provjeru da li određeni objekt sadrži drugi objekt
- *OpenLayers.Filter.Spatial.WITHIN* – koristi se za provjeru da li se objekt nalazi u potpunosti unutar željenog objekta

Klasa *Style* se koristi za definiranje stilova koje bi pojedini vektorski sloj trebao koristiti. Unutar klase *Style* veoma lako se dodaju različita oblikovanja vektorskim slojevima od kojih vrijedi izdvojiti *fillColor* koji dodaje ispunu objektu, *fillOpacity* koji mijenja transparentnost objekta, te *pointRadius* koji specificira kojom će se veličinom piksela prikazati točkasti objekti.

Klasa *WPSClient* je posebno značajna zbog teme diplomskog rada. Unutar ove klase se lako može kreirati WPS klijent koji se može spojiti na željeni server. Klasa se poziva naredbom *OpenLayers.WPSClient*. Sljedeći primjer prikazuje stvaranje buffer zone oko presjeka geometrije i objekta, postignuto ulančavanjem dva procesa, uz pretpostavku da su podaci prethodno unešeni na kartu.

Kod pozivanja klase WPS.Client obavezno se mora definirati server na koji se ta klasa spaja,

```
client = new OpenLayers.WPSClient({
        servers: {
            opengeo: 'http://demo.boundlessgeo.com/geoserver/wps'
        }
    });
```

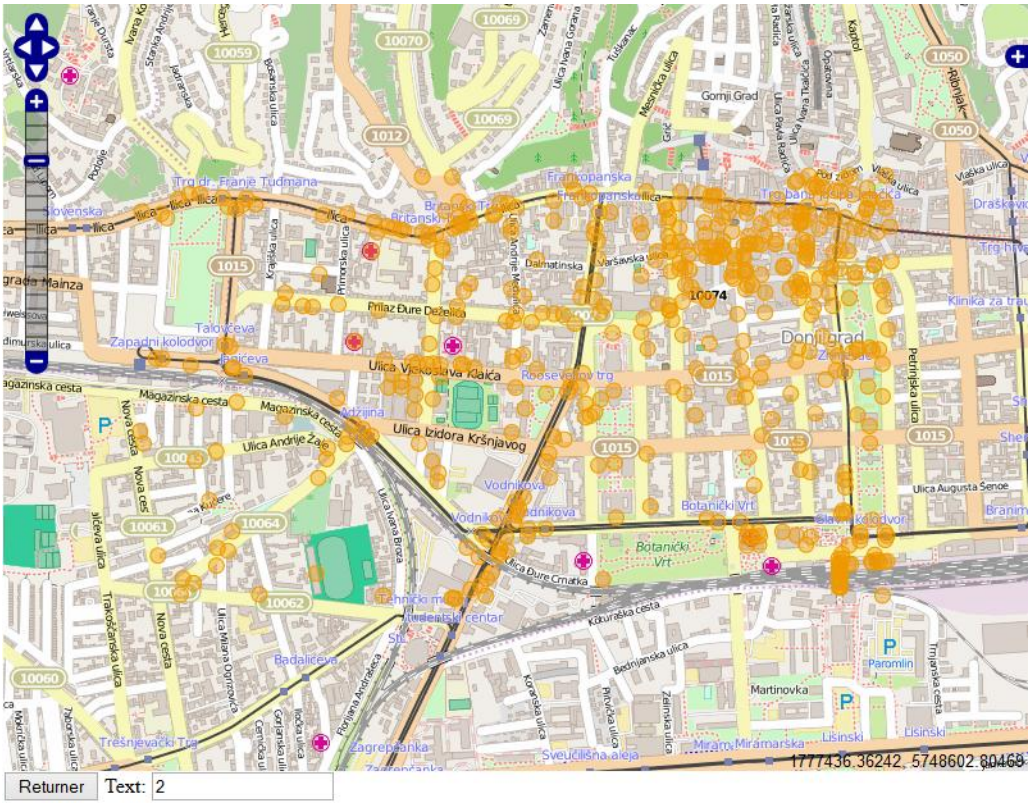
Također treba se definirati proces koji želimo izvršiti,

```
  intersect = client.getProcess('opengeo', 'JTS:intersection');
    intersect.configure({
// ulazni podaci mogu biti objekti, geometrija ili niz objekata i
//geometrija
                inputs: {
            a: features,
            b: geometry
        }
    });
```

I nakon toga definirati još jedan proces koji će se povezati s prethodnim i izvršiti ga:

```
    buffer = client.getProcess('opengeo', 'JTS:buffer');
    buffer.execute({
        inputs: {
            geom: intersect.output(),
            distance: 1
        },
        success: function(outputs) {
               // outputs.result je objekt ili niz objekata za procese na
               //prostornim podacima
            map.baseLayer.addFeatures(outputs.result);
        }
    });
```

Budući da je OpenLayers poprilično jednostavan sustav, do početne karte (slika 5.1.) sa željenim podacima došlo se bez većih problema.



Slika 5.1 Početna OpenLayers karta

Iako je na prvi pogled WPS klijent dobro definiran u pokušaju spajanja na Geoserver naišlo se na prve probleme. Detaljnim istraživanjem otkriveni su brojni problemi vezani uz WPS klijent OpenLayersa. Kako u OpenLayers 2 postoji mogućnost WPS klijenta, a u OpenLayers 3 ne postoji, može se zaključiti da zbog slabog korištenja i lošeg rješenja nije došlo do daljnjeg razvoja WPS klijenta. Pretraživanjem različitih foruma, dolazi se do spoznaje da su razni korisnici nailazili na probleme slične problemima u pokušaju implementacije u okviru diplomskog rada. Problemi su se pojavili kod izvršavanja procesa, u nekonzistentnosti sustava, budući da kod nekih korisnika određeni kôd funkcionira besprijekorno dok kod ostalih ne funkcionira. Naišlo se na primjer gdje je korisnik sam pokušao kreirati WPS klijent u okviru OpenLayersa, te po njegovim tvrdnjama funkcionira bez ikakvih poteškoća. Taj WPS klijent je dostupan na adresi (zadnje pristupljeno 22.06.2015.) http://pywps.wald.intevation.org/documentation/course/_static/WPS.js. Taj WPS klijent trebao je biti dorađen, promjenom nekoliko redaka. U retku 513 kôd

```
var frmts =
OpenLayers.Format.XML.prototype.getElementsByTagNameNS(formatsNode,this.ows
NS,  "MimeType")[0].firstChild.nodeValue;
```
treba zamijeniti kôdom

```
var frmts =
formatsNode.getElementsByTagName("MimeType")[0].firstChild.nodeValue;
```

Redak 521 također treba izmijeniti, pa se kôd

```
var format =
OpenLayers.Format.XML.prototype.getElementsByTagNameNS(supportedFormats[i],
this.owsNS,  "MimeType")[0].firstChild.nodeValue;
```
treba zamijeniti kôdom

```
var format =
supportedFormats[i].getElementsByTagName("MimeType")[0].firstChild.nodeValu
e;
```

Takav izmijenjeni kôd primijenjen je putem PyWPS servera, te je primjer dostupan na adresi: http://wps.kartografija.hr/. PyWPS je implementacija OGC-jevog WPS standarda, napisana u programskom jeziku Python. PyWPS pruža okruženje pogodno za programiranje vlastitih procesa. U navedenom primjeru se koristi algoritam za generalizaciju linija. Karta se sastoji od osnovnog sloja koji prikazuje kartu svijeta i vektorskog sloja koji prikazuje granicu Europe, koja se generalizira. WPS proces se pokreće pritiskom na gumb *WPS demo*, te se nakon nekog vremena na prikazu dobije i generalizirana linija. Nakon uspješne generalizacije putem PyWPS-a razmatrala se mogućnost generalizacije točkastih objekata putem Geoservera, te se za upotrebu na Geoserveru početni WPS klijent trebao dopuniti, pa se sadržaj retka 827

```
output.setValue(OpenLayers.Format.XML.prototype.getAttributeNS(reference[0]
,this.xlinkNS, "href"));
```
trebao zamijeniti sa sadržajem
```
output.setValue(reference[0].getAttribute("href"));
```

Također zbog problema u strukturi XML-a trebalo je nadopuniti kôd od retka 1314 s kôdom:

```
OpenLayers.WPS.executeRequestTemplate = '<?xml version="1.0" encoding="UTF-
8" standalone="yes"?>'+
'<wps:Execute service="WPS" version="1.0.0" '+
```

```
      'xmlns:wps="http://www.opengis.net/wps/1.0.0" '+
      'xmlns:ows="http://www.opengis.net/ows/1.1" '+
      'xmlns:xlink="http://www.w3.org/1999/xlink" '+
      'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" '+
      'xmlns:gml="http://www.opengis.net/gml" '+
      'xmlns="http://www.opengis.net/wps/1.0.0" '+
      'xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
      http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">'+
'<ows:Identifier>$IDENTIFIER$</ows:Identifier>'+
'<wps:DataInputs>'+
      "$DATA_INPUTS$"+
'</wps:DataInputs>'+
'<wps:ResponseForm>'+
      '<wps:ResponseDocument>'+
            "$OUTPUT_DEFINITIONS$"+
      '</wps:ResponseDocument>'+
'</wps:ResponseForm>'+
'</wps:Execute>';
```

Nakon dopune WPS klijenta, napisan je kôd kojim se WPS klijent spaja na Geoserver i obavlja proces *vec:BufferFeatureCollection* koji stvara bufer zonu zadanog promjera oko točkastih objekata. Spomenuti kôd izgleda ovako:

```
<!DOCTYPE html>
<html lang='en'>
<head>
      <meta charset='utf-8' />
      <title>Diplomski rad</title>
      <!-- Map DOM element -->
      <div id="ch3_gml" style="width: 100%;
            height:100%;"></div>
```

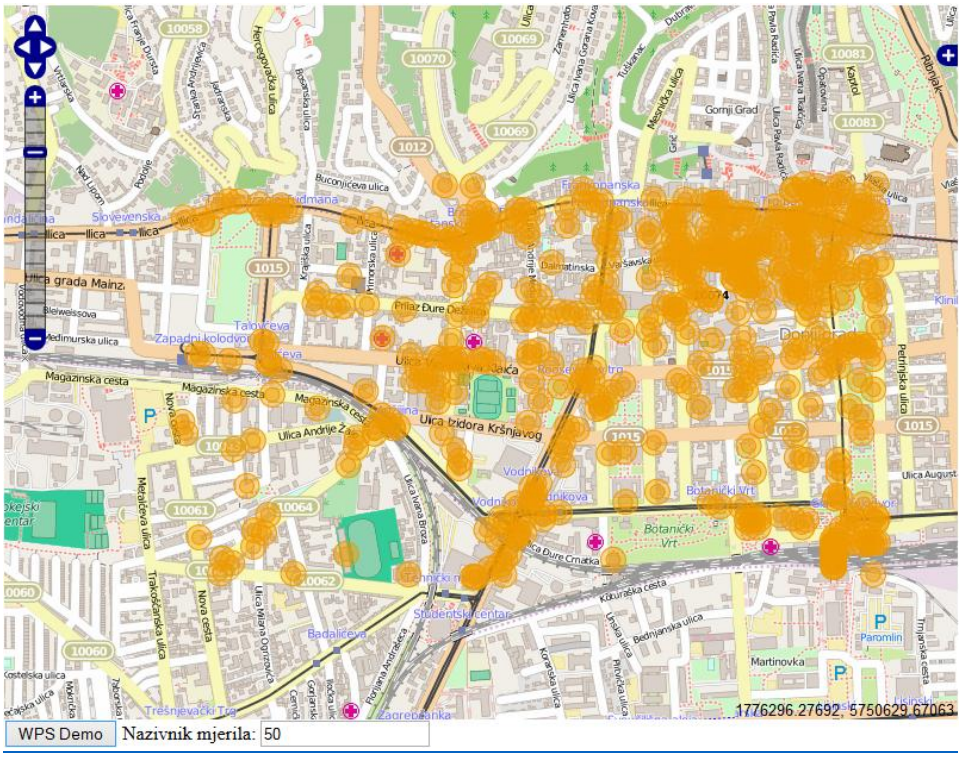
```
      <script type='text/javascript'
src='http://openlayers.org/api/OpenLayers.js'></script>

      <script src="WPS.js"></script>

      <script type='text/javascript'>

            var map, europa, wps;

            function onWPSClicked()  {

                wps = new
OpenLayers.WPS("http://localhost:8080/geoserver/wps",{onSucceeded:
onWPSSussceeded});

                var podaci =
OpenLayers.Format.GML.prototype.write(gml.features);

                var dataInput = new OpenLayers.WPS.ComplexPut({

                    identifier:"features",

                    value:  podaci,

                    format: "text/xml"

                    });

                var literalInput1 = new OpenLayers.WPS.LiteralPut({

                        identifier:"distance",

                        value: document.getElementById("text").value

                    });

                var literalInput2 = new OpenLayers.WPS.LiteralPut({

                        identifier:"attributeName",

                        value: ""

                    });

                 var complexOutput = new OpenLayers.WPS.ComplexPut({

                            identifier: "result",

                            asReference: true,

                            format: "text/xml"

                        });

               var returnerProcess = new OpenLayers.WPS.Process({

                        identifier: "vec:BufferFeatureCollection",

                        inputs: [dataInput,literalInput1],
```

```
                    outputs: [complexOutput] });
            console.debug(returnerProcess);
             wps.addProcess(returnerProcess);
             wps.execute("vec:BufferFeatureCollection");
}
        function onWPSSussceeded(process)  {
             console.debug(process);
             var text = "Uspio sam!"+process.outputs[0].value;
         var proj900913 = new OpenLayers.Projection("EPSG:900913");
             var projWGS84 = new OpenLayers.Projection("EPSG: 4326");
             rezultat = new OpenLayers.Layer.Vector("GML", {
             projection: proj900913,
             protocol: new OpenLayers.Protocol.HTTP({
                 url: process.outputs[0].value,
                 format: new OpenLayers.Format.GML(),
            }),
             strategies: [new OpenLayers.Strategy.Fixed()],
             });
          map.addLayer(rezultat);
             alert(text);
        }
        function init(){
              gml= new OpenLayers.Layer.Vector("GML", {
             protocol: new OpenLayers.Protocol.HTTP({
                 url: "./oi.xml",
                 format: new OpenLayers.Format.GML()
             }),
             strategies: [new OpenLayers.Strategy.Fixed()]
        });
               europa= new OpenLayers.Layer.Vector("GML", {
```

```
                protocol: new OpenLayers.Protocol.HTTP({
                    url: "./europa.gml",
                    format: new OpenLayers.Format.GML()
                }),
                strategies: [new OpenLayers.Strategy.Fixed()]
            });
                  var osm= new OpenLayers.Layer.OSM();
                  var navigation_control= new
OpenLayers.Control.Navigation({});
                  var controls_array = [
                        navigation_control,
                        new OpenLayers.Control.PanZoomBar({}),
                        new OpenLayers.Control.LayerSwitcher({}),
                        new OpenLayers.Control.MousePosition({})
                  ];
                  map = new OpenLayers.Map('map_element',{
                        controls: controls_array
                  });
                  var wms = new OpenLayers.Layer.WMS('OpenLayers WMS',
'http://vmap0.tiles.osgeo.org/wms/vmap0',{layers: 'basic'});
                  map.addLayer(osm);
                  map.addLayer(gml);
                  map.addLayer(europa);
                  europa.events.register('loadend', europa,
function(evt){map.zoomToExtent(europa.getDataExtent())})
      }
      </script>
</head>

<body onload='init();'>
      <div id='map_element' style='width: 800px; height:600px;'>
      </div>
```

```
        <div>

            <input type="button" value="WPS Demo"
onclick="onWPSClicked()"></input>

            Nazivnik mjerila: <input type="text" value="0.001"
id="text"></input>

        </div>

</body>

</html>
```

Nakon pokretanja, u polje *Nazivnik mjerila* se upiše željeni promjer buffer zone te se klikom na gumb *WPS demo* pokreće se WPS proces. Izvorni podaci prikazani su na slici 5.1. Nakon nekog vremena buffer zona se pojavi na osnovnoj karti kao što je prikazano na slici 5.2.



Slika 5.2 Buffer zona promjera 50

Budući da je ovim primjerom pokazana mogućnost spajanja WPS klijenta na Geoserver u daljnja razmatranja uzeti su dostupni WPS procesi na Geoserveru koji bi se mogli upotrijebiti kod generalizacije točkastih objekata za turističke svrhe. Tako su testirani procesi *geo:ConvexHull*, *gs:Aggregate* i *gs:CollectGeometries,* ali su ti procesi loše definirani te ne

daju željeni i očekivani rezultat. Sljedeći korak u implementaciji bio bi razvoj kôda koji bi generalizirao točkaste, množinski prikupljene podatke po geometriji i atributu, no uz trenutni stupanj razvoja postojećih besplatnih sustava takav primjer nije moguće definirati.

6. ZAKLJUČAK

Standardizacijom Web Processing Servicea utvrđen je teorijski temelj za upotrebu WPS-a u različite svrhe, poput ulančavanja servisa ili za potrebe generalizacije. Budući da živimo u dobu naprednih tehnoloških mogućnosti pojavljuju se mnogobrojni prostorni podaci koji su često neupotrebljivi i onemogućuju kvalitetan prikaz na karti te ukoliko ih želimo koristiti neophodno je izvršiti generalizacijske postupke nad tim podacima ovisno o daljnjoj namjeni tih podataka. Putem WPS-a moguća je generalizacija, što su Foester i Stoter (2006) pokazali na generalizaciji linija Douglas-Peuckerovim algoritmom.

Razvojem različitih slobodnih servisa omogućila se šira rasprostranjenost upotrebe WPS-a. OpenLayers je prema klijentu orijentirana JavaScript biblioteka otvorenog koda za kreiranje interaktivnih web karata, vidljiva u bilo kojem web pregledniku. OpenLayers nudi mogućnost WPS klijenta te se putem njega može spajati na odgovarajući WPS server. WPS server je dostupan na također slobodnom servisu Geoserveru koji omogućuje korisniku upotrebu definiranje vlastitih procesa ili upotrebu već definiranih WPS procesa. Ipak bez obzira na dobar teorijski okvir za upotrebu WPS na tim sustavima u praktičnom dijelu još uvijek postoji veliki broj problema. Budući da su Geoserver i OpenLayers besplatni sustavi takvi problemi su očekivani, ali isto tako može se očekivati brzo rješenje tih problema jer se velik broj korisnika susreće s tim problemima i kontinuirano se ti sustavi razvijaju i nadograđuju.

Zaključno, WPS-om je omogućena generalizacija, ali trenutne nesavršenosti sustava preko kojih bi korištenje WPS-a bilo najjednostavnije i najbrže, kao što su Geoserver i OpenLayers, onemogućuju daljnji razvoj generalizacijskih postupaka i stvaranje prikaza prilagođenih određenim skupinama korisnika. Okviri za upotrebu postoje, generalizacijski postupci za sažimanje točkastih podataka su definirani te ih samo treba prilagoditi za upotrebu putem WPS-a. S obzirom na navedene probleme ne preporuča se upotreba WPS-a u generalizaciji za turističke svrhe, budući da korisnici očekuju rezultate u realnom vremenu i brzo što s trenutnim stupnjem razvoja nije moguće. Ipak treba spomenuti da su današnje mogućnosti WPS-a slabo prepoznate, te se koristi u manjem krugu korisnika, no za očekivati je daljnji

razvoj WPS-a i softvera koji podržavaju korištenje WPS-a, te veću iskorištenost teorijski odlično definiranog servisa u budućnosti.

7. LITERATURA

Bereuter, P.; Weibel, R. (2010.): Generalisation of point data for mobile devices: A problem-oriented approach. 13th ICA Workshop on Generalisation and Multiple Representation – 12th, 13th of September 2010, Zürich, http://generalisation.icaci.org/images/files/workshop/workshop2010/genemr2010_submission_25.pdf (08.03.2015.)

Bereuter, P.; Weibel, R. (2013.): Real-time generalization of point data in mobile and web mapping using quadtrees, Cartography and Geographic Information Science, Vol. 40, No. 4, 271-281.

Bloch, M.; Harrower, M. (2006.): MapShaper.org: A map generalization web service, Proceedings of AUTOCARTO 2006, http://www.cartogis.org/docs/proceedings/2006/bloch_hharrower.pdf (09.03.2015.)

Burghardt, D.; Neun, M.; Wiebel, R. (2005.): Generalization Services on the Web – A classification and an initial prototype implementation, Cartography and Geographic Information Science, Vol. 32, No. 4, 257-268.

Foerster, T.; Stoter, J. (2006.): Establishing an OGC Web Processing Service for generalization processes, Workshop of the ICA Commission on Map Generalisation and Multiple Representation – June 25th 2006 http://generalisation.icaci.org/images/files/workshop/workshop2006/ICA2006-foerster_stoter.pdf (15.03.2015.)

Geoserver: http://docs.geoserver.org/ (05.05.2015)

Iacovella, S.; Younblood, B. (2013.): GeoServer, Beginner's Guide, PACKT Publishing, Birmingham

Jezdić, K.; Tutić, D. (2013.): WebGen-WPS, Web-servis za kartografsku generalizaciju, Kartografija i geoinformacije, Br. 19, Vol. 12, 160-165.

Mackaness, W.A.; Chaudhry, O. (2008.): Generalization and Symbolization. In: Shekhar, S.; Xiong, H.(ed.): Encyclopedia of GIS, Springer, New York, 330-331.

Michael, C.; Ames, D. (2007.): Evaluation of the OGC web processing service for use in a client-side GIS, http://www.osgeo.org/files/journal/final_pdfs/OSGeo_vol1_OGC.pdf (20.03.2015)

OGC, Open Geospatial Consortium, Inc. (2002.) : The OpenGIS Abstract Specification, Topic 12: OpenGIS Service Architecture, Version 4.3, portal.opengeospatial.org/files/?artifact_id=1221 (20.03.2015.)

OGC, Open Geospatial Consortium Inc. (2007.): OpenGIS Web Processing Service, Open Geospatial Consortium, OpenGIS® Standard http://portal.opengeospatial.org/files/?artifact_id=24151 (24.03.2015.)

OpetStreetMap: https://www.openstreetmap.org/ (30.05.2015.)

Perez, S.,A. (2012.): OpenLayers Cookbook, PACKT Publishing, Birmingham

Sester, M.; Sarjakoski, L. T; Harrie, L.; Hampe, M.; Koivula, T.; Sarjakoski, T.; Lehto, L.; Elias, B.; Nivala, A.M.; Stigmar, H. (2004.): Real-time generalisation and multiple representation in the GiMoDig mobile service, http://gimodig.fgi.fi/pub_deliverables/GiMoDigD711_721_731_generalisation.pdf (05.05.2015.)

Stanislawski, L.V.; Buttenfield, B.P.; Bereuter, P.; Savino, S.; Brewer, C.A. (2014.): Generalisation Operators. In: Burghardt, D.; Duchêne, C.; Mackaness, W. (ed.): Abstracting Geographic Information in a Data Rich World, Metodologies and Applications of Map Generalisation, Springer, New York, 159-162.

W3C (2004.): Web Services Architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ (14.06.2015.)