Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Bregant

# Operativna transformacija

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Zoran Bosnić

Ljubljana 2014

Originalni izvod izdane teme diplomskega dela...

## Izjava o avtorstvu diplomskega dela

Spodaj podpisani Marko Bregant, z vpisno številko **63080011**, sem avtor diplomskega dela z naslovom:

*Operativna transformacija*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Zorana Bosnića,
- 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. maja 2014 Podpis avtorja:

Zahvala...

Posvetilo...

# Kazalo

# Seznam uporabljenih kratic

**OT** (Operational Transformation) - operativna transformacija

**AJAX** (Asynchronous JS and XML) - asinhroni JS in XML

**API** (Application Programming Interface) - vmesnik za programiranje aplikacij

**JS** (JavaScript) - programski jezik, ki v zadnjem času pridobiva na popularnosti

**XML** (Extensible Markup Language) - razširljiv označelvani jezik

# Povzetek

Aliquam erat volutpat. Mauris porttitor luctus vehicula. Suspendisse vulputate faucibus nulla, eu ultrices libero gravida ut. Nullam aliquet facilisis lacus, ac venenatis dolor pellentesque facilisis. Phasellus ut placerat tellus. Nunc in euismod felis. Pellentesque ullamcorper elementum justo et scelerisque.

Vestibulum eget felis tincidunt, porta massa ut, pretium arcu. Integer ornare tincidunt pharetra. Ut egestas, tortor a viverra adipiscing, nibh lectus elementum sem, et lobortis est lectus ac est. Mauris condimentum nulla tempus bibendum pellentesque. Vivamus auctor massa non neque sodales, eget aliquam nibh pulvinar. Duis pharetra felis in velit elementum vehicula. Phasellus vulputate tellus quis odio tincidunt, in lobortis dolor auctor. Nunc vel blandit nibh.

# Abstract

Nam nec sagittis diam. Quisque dictum lorem vitae urna gravida tincidunt. Sed quam enim, vestibulum quis mattis sed, ultricies id purus. Morbi rhoncus mauris vitae ipsum vulputate, in facilisis lacus cursus. Sed euismod metus eget ligula cursus rhoncus a ut ipsum. Duis eget vulputate purus.

Phasellus volutpat orci elementum quam ultricies, et pellentesque enim bibendum. Ut at nulla sollicitudin, blandit risus vel, aliquam risus. Etiam tristique metus vel libero mattis aliquet. Phasellus quis lorem est. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin iaculis risus vitae facilisis ultrices. Maecenas pellentesque rhoncus turpis a consequat. Proin rutrum a urna eu varius. Etiam nibh diam, congue at lectus at, adipiscing mollis mauris.

# Poglavje 1

# Uvod

Še ne dolgo nazaj se nam je internet zdel počasen. Vse skupaj je delovalo zelo statično. Lahko bi rekel, da smo dve desetletji nazaj splet uporabljali za brskanje, dobesedno le za brskanje. Nato so se počasi pojavile spletne strani, ki so omogočale nekaj malega interaktivnosti. To so bile funkcionalnosti kot so vpisovanje komentarjev, iskalniki, spletne galerije, forumi... Danes se nam zdijo te funkcionalnosti že skoraj integrirane v spletnih aplikacijah. Lahko se ozremo nazaj in rečemo, da so to bili prvi zametki, ki so uporabnikom omogočili, da ustvarjajo splet tak kot ga poznamo danes. Z izboljševanjem internetne infrastrukture so se izboljšale hitrosti prenosa podatkov. Z razvojem spletnih tehnologij se je izboljša celotna uporabniška izkušnja. V zadnjih dvajestih letih smo bili priča razvoju napredka tako na programski kot tudi na strojni opremi. Na tem mestu se je smiseleno vprašati, kaj je bilo tisto bistveno, ki je celotno zadevo izboljšalo in kako se bo izboljševala v prihodnosti? Internet v osnovi še vedno deluje tako kot je dvajest let nazaj. Princip je isti. Imamo odjemalca (ang. client) na eni strani in imamo strežnik (ang. server) na drugi strani. Odjemalec vzpostavi komunikacijo s strežnikom. Od njega zahteva neko akcijo in le ta jo izvrši. Sliši se komično, a gledano poenostavljeno, splet še danes deluje tako. Če pa pogledamo podrobno, se razlike enormne. Najpomembnejša razlika je, da se je komunikacija med odjemalcem in strežnikom, ter obratno, začela dogajati v realnem času.

V okviru diplomske naloge bi radi preučili algoritme in raziskali celoten sistem, ki na spletu omogoča urejanje golega besedila v realnem času. Tega se bomo lotili tako, da bomo pregledali raziskave, ki so bile objavljene v akademski sferi. Prvi zapisi o algoritmih sodelovanja uporabnikov segajo že v leto 1989. Kasneje je pri raziskavah precej pripomogla tudi ustanovitev SIGCE (The Special Interest Group on Collaborative Computing), ki promovira raziskovalce na tem področju. Skratka preleteli bomo nekaj raziskav iz tega področja. Po pridobljenem teoretičnem znanju, bomo poiskali članke,

ki so bili napisani s strani industrije. Radi bi izvedeli kaj od teorije se lahko uporabi v praksi. Največ uporabnih informacij bomo dobili od protokola Google Wave. Ne smemo pa pozabiti, da obstaja mnogo drugih uporabnih orodij (npr. Share.js), od katerih se lahko naučimo praktičnega dela. Cilj diplomske naloge je, da zasnujemo enega izmed algoritmov na plaformi Node.js. Ker hočemo doseči, da je algoritem neodvisen od odjemalcev, mora delovati kot API. Kot smo omenili, je uporabnost take izvedbe ravno v tem, da je neodvisen od odjemalcev, ki se nanj povezujejo, naj si bo spletna aplikacija ali mobilna naprava. Poleg tega tak način izvedbe spodbuja nadaljni razvoj celotnega sistema na različnih odjemalcih.

V nadaljevanju bomo v drugem poglavju najprej podrobno predstavili problem. V tretjem poglavju bomo preučili načine za sodelovanje v realnem času. Izbrali bomo našemu problemu najbolj primernega ter ga v četrem poglavju podrobneje opisali. Nato bomo v petem poglavju raziskali pristope in algoritme, ki stojijo za njimi. Algoritme bomo tudi primerjali. V šestem poglavje se bomo iz teoretičnega znanja preselil na praktično znanje. Nekateri algoritmi, ki so bili na akademskem področju uspešni, so se implementirali v končnih ali v pol produktih. Na kratko bomo predstavili nekaj teh produktov. V sedmem poglavju bomo poskusili tudi sami narediti zasnovo urejevalnika v realnem času. Na koncu v osmem poglavju sledijo še sklepne ugotovitve.

# Poglavje 2

# Konflikti in konsistentnost

Leta 2004 oziroma 2005 se je začela uveljavljati tehnologija AJAX. Njen glavni namen je, da odjemalcu omogoča pošiljanje asinhronih zahtev na strežnik. V praksi je bila razlika videna v osveževanju spletnih strani. V preteklosti je brskalnik osvežil celotno spletno stran za vsako zahtevo, ki jo je naredil na strežnik. Uporaba AJAXa pa omogoča, da so zahteve na strežnik manjše, bolj dinamične in najpomembnejše asinhrone. Namesto celotne strani lahko osvežimo le nek manjši del.

Naslednja pomembna stvar na spletu je protokol WebSocket. Trenutno je še v povojih. Danes naj bi ga podpirali že vsi najnovejši brskalniki, vendar ga uporabljajo le redke spletne aplikacije. Sprememba, ki jo prinaša WebSocket je način komunikacije med odjemalcem in strežnikom. Omogoča dvosmerno komunikacijo. Po novem lahko tudi strežnik pošlje zahtevo odjemalcu.

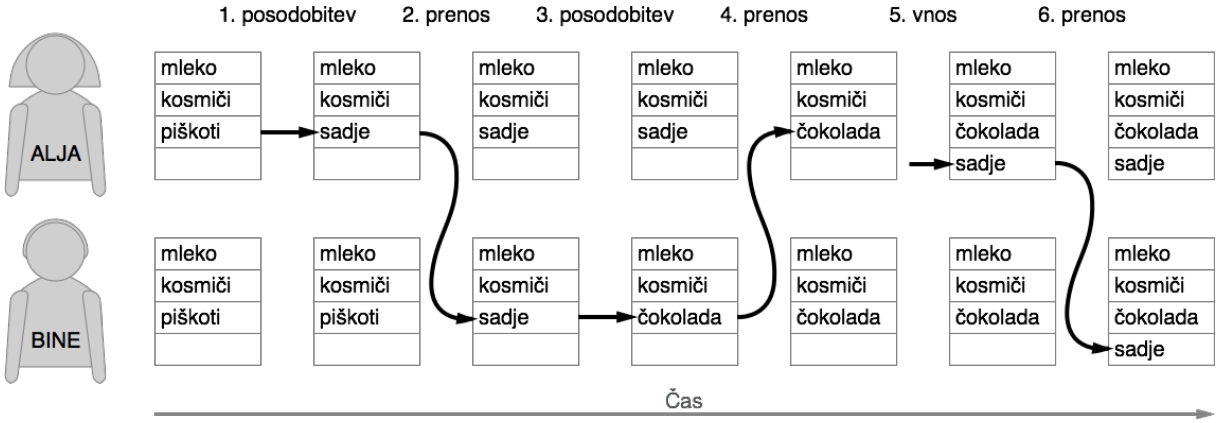
Ti dve tehnologiji omenjam zato, ker sta in bosta po mojem mnenju največ prispevali pri izboljšanju uporabniške izkušnje na spletu. Interakcija med odjemalcem in strežnikom je postala bolj tekoča kot je bila v preteklosti. Z njo nam je bila dana možnost za razvoj orodij za sodelovanje v realnem času (ang. real-time collaboration tools). Obstaja že mnogo orodij, ki preko sodelovanja (ang. collaboration) rešujejo nek specifičen problem. Na podoben problem smo naleteli tudi sami in sicer urejanje besedila v realnem času. Z urejanjem besedila nimamo v mislih označevanje besedila s krepko, ležečo, podčrtano pisavo in tako naprej, ampak za urejanje golega besedila kot takega. Sliši se enostavno. Uporabnik lahko doda črko, pobriše črko, se pravi operira z manjšimi enotami (črke, besede), ki se združujejo v večje enote (stavki, povedi, sporočilo, besedilo, dokument). Sistem mora skrbeti za izmenjevanje nastalega besedila med udeleženci. Na tak način delujejo spletna klepetalnica. Udeleženci v pogovoru si med sabo izmenjujo sporočila. To naj si bodo večja ali manjše enote teksta. Sistem mora le skrbeti, da se le te pravilno prenesejo med udeleženci pogovora. Vendar zadeva ni

niti približno tako enostavna. Bistvena razlika urejevalnika v realnem času v primerjavi s spletno klepetalnico je v obliki hranjenja in operiranja nastalih podatkov. Običajno se pri spletnih klepetalnicah vsako posamezno sporočilo hrani na strežniku kot samostojna enota, ki se v celoti razpošilja med udeleženci. Pri urejevalnikih besedila v realnem času pa je besedilo, ki ga udeležencih urejajo, enotno za vse udeležence hkrati. Lahko bi rekel, da vsi udeleženci urejajo skupni dokument. Posamezne manjše enote besedila, ki se izmenjujejo med udeleženci, so le koščki celotnega dokumenta. Pred tako nastalim dokumentom mora delovati algoritem, ki zna te manjša enote besedila združevati v dokument.

## 2.1 Predstavitev problema na konkretnem primeru

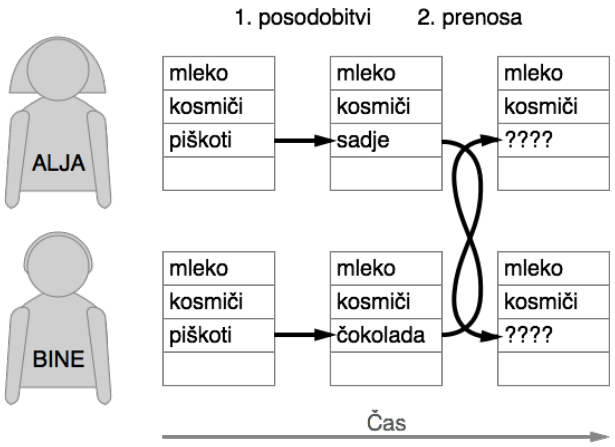
Da bo problem razumljiv še najmanj veščemu uporabniku tehnologij, bomo problem predstavili na konkretnem primeru.

Predstavljajmo si uporabnika Aljo in Bineta, ki za nakupovanje pripravljata skupni nakupovalni listek. Trenutno imata na seznamu mleko, kosmiče in piškote. To so stvari, ki jih običajano kupita vsako soboto. Alja se odloči, da bo ta vikend namesto piškotov kupila sadje, zato uredi seznam tako, da piškote zamenja s sadjem. Aplikacija spremembe pošlje tudi Binetu, ki nato vidi piškota zamenjane s sadjem. Bine so odloči, da bi tokrat raje kupil čokolado, zato temu primerno spremeni seznam. Aplikacija pošlje Binetove spremembe k Alji, ki sedaj vidi sadje zamenjano s čokolado. Nezadovoljna Alja se z Binetom dogovori za kompromis in doda sadje na seznam, tako da Binetova čokolada še vedno ostane na seznamu. Tako Alja kot Bine imata sedaj na seznamu mleko, kosmiče, čokolado in sadje, tako kot je to prikazano na koncu Slike 2.1.



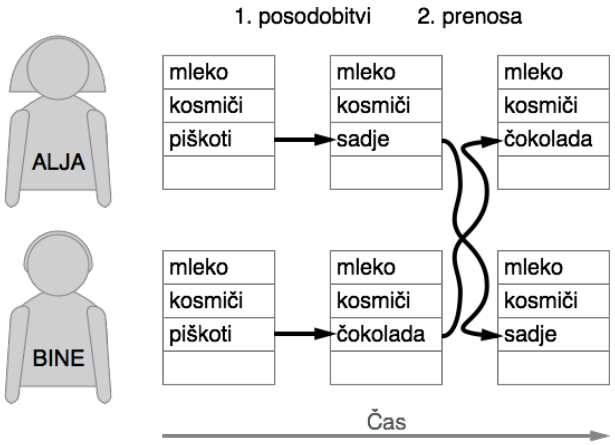

Slika 2.1: Diagram sočasnega urejanja nakupovalnega listka, pri čemer mora Alja dvakrat vpisati svojo željo po sadju.

Pri tako fleksibilni interakciji lahko nastane konflikt. Alja bi lahko spremenila piškote v sadje sočasno kot bi Bine spremenil piškote v čokolado. Oba bi svoje spremembe videla takoj. Vendar spremembe Alje potrebujejo nekaj časa, da pridejo do Bineta. Tudi spremembe Bineta potrebujejo nekaj časa, da pridejo do Alje. Ta zamuda lahko spremeni vrstni red Aljine in Binetove spremembe, kar povzroči nakonsistentnost kot je to prikazano na Sliki 2.2. Alja in Bine bi morala pri sebi imeti vedno enake artikle na nakupovalnem listku.




Slika 2.2: Diagram sočasnega urejanja nakupovalnega listka. Vprašaji v zadnjem koraku pomenijo, da je seznam v konfliktu in mora biti razrešen, da zagotovimo konsistentnost.

Predstavljajmo si enostavno rešitev za posodabljanja nakupovalnega listka, ki Alji in Binetu pokaže vedno zadnjo spremembo, ki je bila narejena na seznamu. Alja bi piškote spremenila v sadje, nato pa bi prišla Binetova sprememba v čokolado. Na drugi strani je Bine piškote spremenil v čokolado, nato pa bi sprejel Aljino sadje. V tem primeru bi Alja in Bine na koncu imela dva različna seznama, česar sploh ne bi opazila. Primer slabega reševanja konfliktov je prikazan na Sliki 2.3.




Slika 2.3: Diagram sočasnega urejanja nakupovalnega listka. Konflikt v zadnjem koraku je rešen na način, da Alja in Bine sprejmeta zadnjo narejeno spremembo. Rešitev je slaba, saj ne zagotavlja konsistentnosti.

Problem nastane še večji pri sodelovanju večih uporabnikov in še z večjo zamudo pri dostavi sprememb na nakupovlnam listku. Recimo, da se Alji in Binetu pridruži še Cene. Bine ima počasen internet. Alja in Cene na seznam dodata in odstranita deset novih artiklov še preden Bine dobi eno spremembo. Medtem ko Bine ureja svoj seznam, so spremembe Alje in Ceneta še na poti k njemu. Za zagotovitev konsistentnosti bi morala aplikacija upoštevati zakasnjene oddaljene spremembe na osnovi prvotne verzije nakupovalnega listka.

# Poglavje 3

# Sodelovanje v realnem času

Kako reševati konflikte in zagotoviti konsistentnost med oddaljenimi uporabniki pri sočasnem urejanju, je eden izmed glavnih izivov moje diplomske naloge. V tem poglavju bomo raziskali protokole, ki omogočajo sodelovanje uporabnikov v realnem času in teoretično rešujejo omenjena problema. Najbolj razširjeni so Operativna transformacija (ang. Operational transformation), Diferenčna sinhronizacija (ang. Differential synchronization) in protokol Brez operativne transformacije (ang. Without operational transformation, bolj znan pod kratico WOOT).

## 3.1 Operativna transformacija

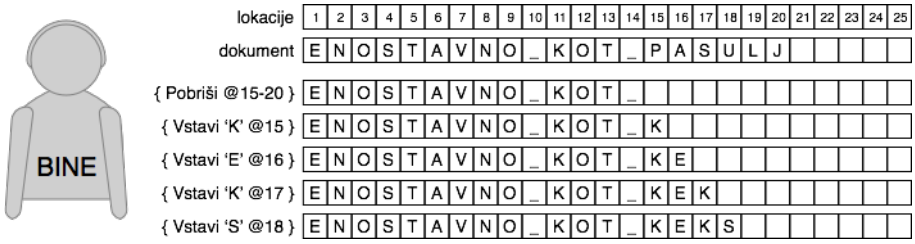
Operativna transformacija se je prvič omenjala v članku Concurrency Control in Groupware Systems. Pri Googlu so Operativno transformacijo vzeli za osnovno pri načrtovanju protokola Wave, ki se uporablja v Google Docsih, kar nakazuje na njeno uporabnost.

Zaradi razumljivosti bomo besedilo, ki ga urejajo uporabniki, poimenovali dokument. Dokument je shranjen kot serija kronoloških sprememb narejenih na dokumentu. Primer spremembe je { `Vstavi 'M' @11` }, ki pomeni "v dokumentu na lokacijo 11 vstavi črko M" ali { `Pobriši @3-7` }, ki pomeni "v dokumentu pobriši vse znake med lokacijo 3 in 7". Obstajajo še druge vrste sprememb kot so oblikovanja besedila, zaklep odstavka, razveljevitev spremembe... Zaradi enostavnosti razlage se bomo osredotočili le na omenjena dva tipa sprememb. Ko uporabnik ureja besedilo, ne spreminja osnovnih znakov, ki bi skupaj predstavljali besedilo kot dokument. Namesto tega se njegove spremembe shranjujejo v revizijski dnevnik. Seveda dokumenta kot zaključene celote znakov obstaja. Vendar pomembne so spremembe, ki so bile narejene na tem

dokumentu. Če se urejanju skupnega dokumenta pridruži nov uporabnik, mu iz revizijskega dnevnika ponovimo vse (od prve do zadnje) spremembe in že lahko sodeluje pri urejanju tako kot ostali uporabniki.

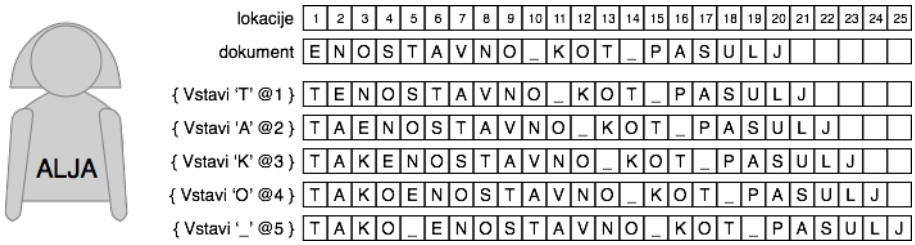
Glede na to, da poznamo revizijo vseh sprememb narejenih na dokumentu, lahko preverimo, kaj je uporabnik imel v svojem urejevalniku, preden je naredil novo spremembo. Na ta način njegovo spremembo pravilno umestimo v skupno besedilo skupaj z ostalimi spremembami, ki so bile narejene med tem. Algoritem, ki skrbi za umeščanje ali združevanje (ang. merging) sprememb, se imenuje Operativna transformacija (v nadaljevanju OT).

Poglejmo delovanje OT na primeru. Predpostavimo, da imamo dokument v katerem se trenutno nahaja stavek "ENOSTAVNO KOT PASULJ" in zasede dvajset lokacij. Urejanju tega dokumenta se pridružita Alja in Bine. Če Bine spremeni stavek v "ENOSTAVNO KOT KEKS", potem je za to moral narediti pet sprememb.



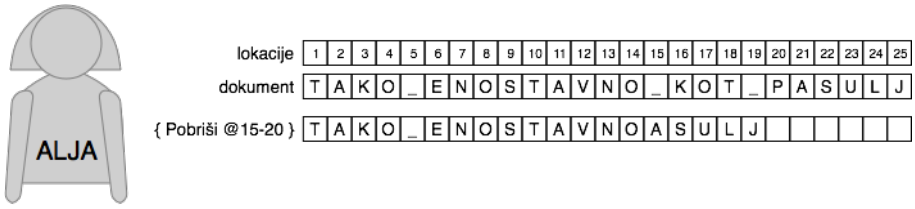

Slika 3.1: Bine naredi pet sprememb.

Predstavljajmo si, da med tem ko Bine tipka, začne stavek spreminjati tudi Alja. in sicer v "TAKO ENOSTAVNO KOT PASULJ". Tudi Alja je morala narediti pet sprememb.



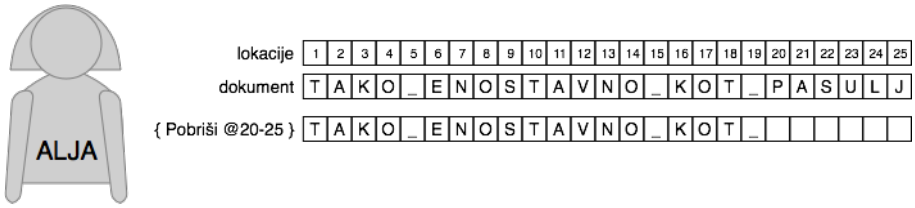

Slika 3.2: Alja naredi pet sprememb.

Če bi Alja v naslednjem koraku naivno sprejela in izvršila Binetovo prvo spremembo, bi v stavku pobrisala napačne črke tako kot je to prikazano na Sliki 3.3.



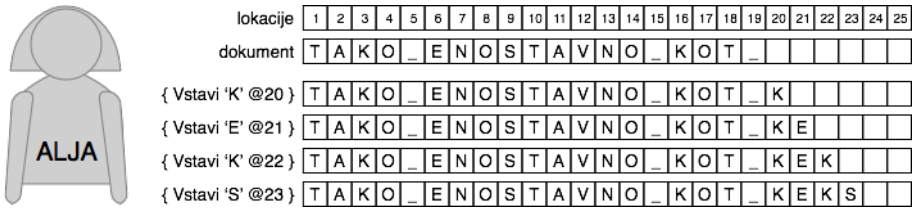

Slika 3.3: Brez transformacije pride do nekonsistentnosti.

Alja je imela pet znakov v začetku stavka, o katerih Bine še ni bil seznanjen. Lokacija Binetove spremembe je zato napačna glede na Aljino vezijo dokumenta. Da bi se izognili temu problemu, mora Alja narediti transformacijo Binetovih sprememb relativno na svoj lokalni dokument. V našem primeru, ko Alja sprejme Binetove spremembe, mora lokacijo spremembe zamakniti za pet znakov, kolikor jih je vpisala na začetku stavka. Ko naredi to transformacijo in izvrši Binetovo prvo spremembo, dobi pravilen stavek.



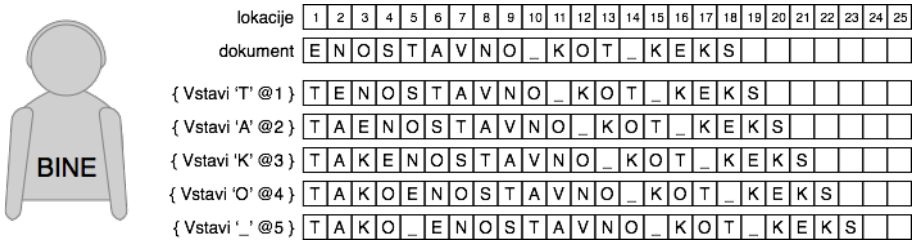

Slika 3.4: Z uporabo Operativne transformacije dobimo pravilen rezultat.

Ko transformira in izvede še ostale štiri spremembe, dobi končno verzijo dokumenta.




Slika 3.5: Končna verzija dokumenta, ki ga vidi Alja.

Včasih spremembe ne povzročajo konfliktov in ni potrebe po transformaciji. Ko Bine prejme Aljine spremembe, ni potrebe po zamikanju lokacij. Bine mora izvesti Aljine spremembe točno take, kot jih je ona izvedla na svojem lokalnem dokumentu.




Slika 3.6: Končna verzija dokumenta, ki ga vidi Bine.

Tako Alja kot Bine v svojem lokalnem dokumentu na koncu vidita stavek "TAKO ENOSTAVNO KOT KEKS". Algoritem, ki smo ga uporabili za zamikanje sprememb, se imenuje Operativna transformacija. Pravilno implementiran algoritem nam garantira, da imajo vsi uporabniki, ko prejmejo vse spremembe, isto verzijo dokumenta.

### 3.1.1 Protokol sodelovanja

Z uporabo OT smo se naučili, kako z zamikanjem lokacij sprememb večim uporabnikom dopustiti urejanje istega dokumenta brez konflikov. Še vedno pa obstaja težava, kako vsako spremembo pravilno združiti z drugimi spremembami, če se le te zgodijo istočano. Zagotoviti moramo, da vsak oddaljeni uporabnik ve, da obstajajo spremembe, ki morajo biti združene. Za to skrbi Protokol sodelovanja (ang. Collaboartion protocol). Tehnologiji OT in Protokol sodelovanja skupaj, znak za znakom, skrbita za sodelovanja v realnem času.

Zavedati se moramo, da za urejanje dokumenta v realnem času skrbijo tako strežnik kot odjemalci. Običajno so to spletni brskalniki uporabnikov. Pri urejanjeju dokumenta mora odjemalec sprocesirati vse spremembe, ki jih naredi uporabnik, in jih poslati na strežnik. Sprocesirati mora tudi vse spremembe drugih uporabnikov, ki mu jih pošlje strežnik. Seveda brez sodelovanja strežnika ne gre.

Najprej si oglejmo, katere podatke si morajo beležiti odjemalci in katere strežnik. Vsak odjemalec si mora beležiti:

- Številko zadnje sinhronizirane revizije. Na Sliki 3.7 so označene s sivim krogcem in številko.
- Čakajoče spremembe. To so spremembe, ki so bile narejene v odjemalcu in niso še bile poslane na strežnik.
- Poslane spremembe. To so spremembe, ki so bile poslane na strežnik, vendar jih strežnik še ni potrdil.

- Trenutno stanje dokumenta kot ga vidi uporabnik.

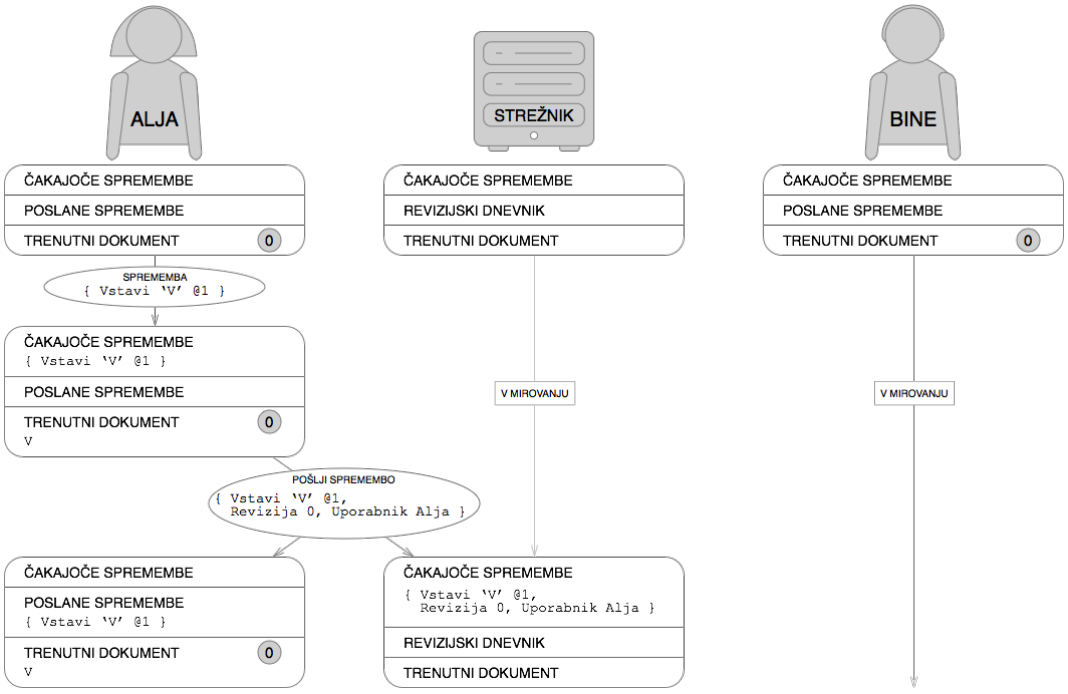
Na strežniku se shranjujejo tri stvari:

- Čakajoče spremembe. To so spremembe, ki jih je strežnik sprejel, a še ni sprocesiral.
- Revizijski dnevnik. Dnevnik v katerem je celotna zgodovina sprememb.
- Trenutno stanje dokumenta, kot bi ga morali v najkrajšem možnem času videti vsi uporabniki.

S hranjenjem in uporabo teh informacij je mogoče zasnovati komunikacijo med strežnikom in odjemalci tako, da so oddaljeni urejevalniki drug od drugega sposobni naglo sprocesirati spremembe.

Oglejmo si kako je v vzorčnem dokumentu poskrbljeno za komunikacijo med strežnikom in odjemalcem. Na Sliki 3.7 zunanja stolpce predstavljata uporabnika Aljo in Bineta, ki urejata skupni dokument. Srednji stolpec je strežnik. Spremembe, ki jih naredita Alja in Bine in se pošljejo na strežnik, so označene v ovalni obliki. OT, ki smo jih spoznali v tem poglavju, bodo na naslednjih slikah označene s peterokotnikom.

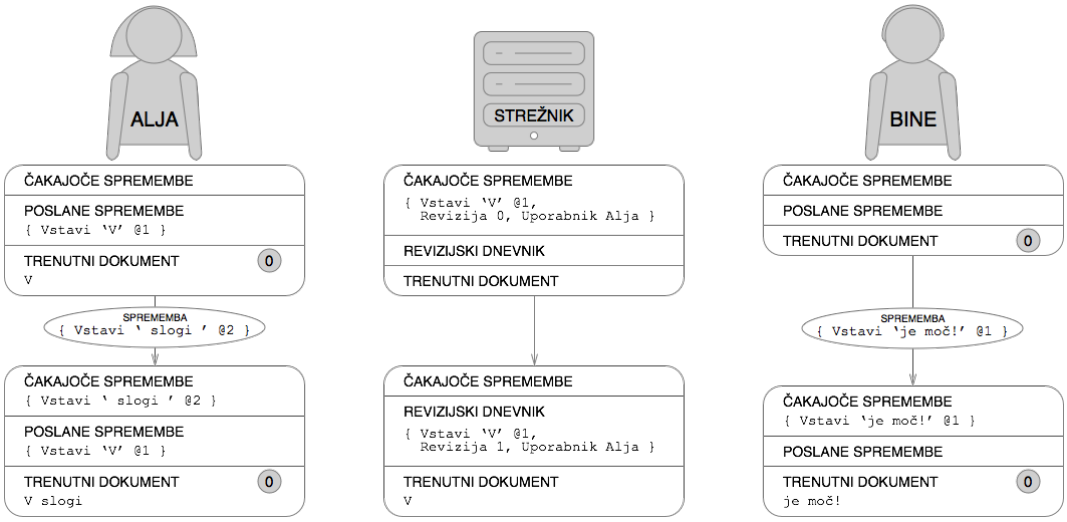
Alja začne tipkati "V" na začetku dokumenta.




Slika 3.7: Alja začne tipkati. Bine je v mirovanju.

Aljin urejevalnik si spremembo { `Vstavi 'V' @1` } shrani med čakajoče spremembe. V naslednjem trenutku se le ta pošlje na server ter se prestavi na seznam poslanih sprememb. Poleg same spremembe, se strežniku pošlje tudi številka revizije in avtorja spremembe (uporabnika). Informacija o uporabniku je pomemba iz vidika avtentikacije. O številki revizije, bom več povedal v naslednjih korakih. Glej Sliko 3.7, strežnik sprejme Aljino prvo spremembo in jo doda med čakajoče spremembe.

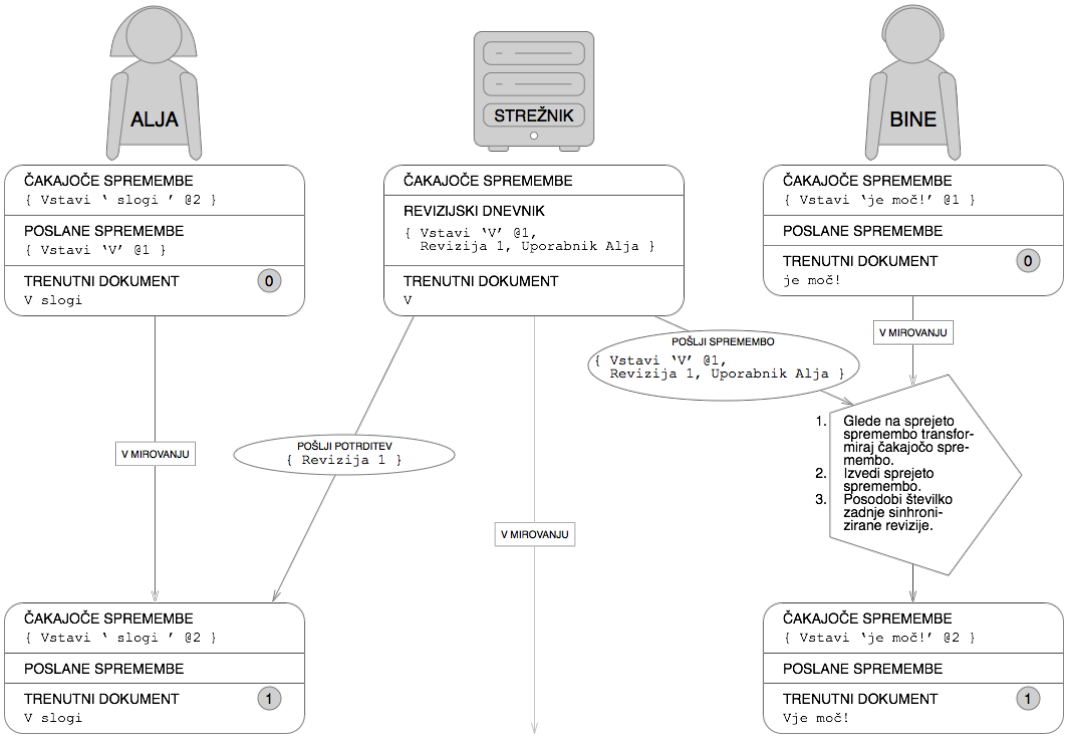
Odjemalec lahko strežniku v enem pošiljanju pošlje tudi več črk za vstavljenje v dokument. Koliko črk se bo poslalo na enkrat je odvisno od implementacije urejevalnika in algoritma za zaznavanje sprememb. Več o tem v naslednjih poglavjih. Na Sliki 3.8 vidimo, da Alja nadaljuje s tipkanjem in doda " slogi " v svoj urejevalnik. Alja v svojem trenutnem dokumentu vidi "V slogi ". V istem času Bine vpiše "je moč!" v svoj prazen dokument. Ne pozabimo, da Bine še vedno ni prejel Aljinih sprememb.




Slika 3.8: Alja nadaljuje s tipkanjem. Bine na drugi strani tudi začne pisati na začetek svojega dokumenta. Strežnik o tem še ni obveščen.

Aljin { `Vstavi ' slogi ' @2` } je bil dodan med čakajoče spremembe in še ni poslan na strežnik. Pravilo je, da nikoli strežiku ne pošiljamo več kot ene čakajoče spremembe na enkrat. Dokler Alja od strežnika ne dobi potrditve prve spremembe, bo njen urejevalnik vse nove spremembe hranil med čakajočimi spremembami. Na Sliki 3.8 lahko opazimo, da si je server Aljino prvo spremembo že shranil v revizijski dnevnik.

V naslednjem koraku, kot je to prikazano na Sliki 3.9, strežnik Binetu pošlje Aljino prvo spremembo ter Alji odgovoril s potrditvijo, da si je zabeležil njeno spremembo v revizijski dnevnik.

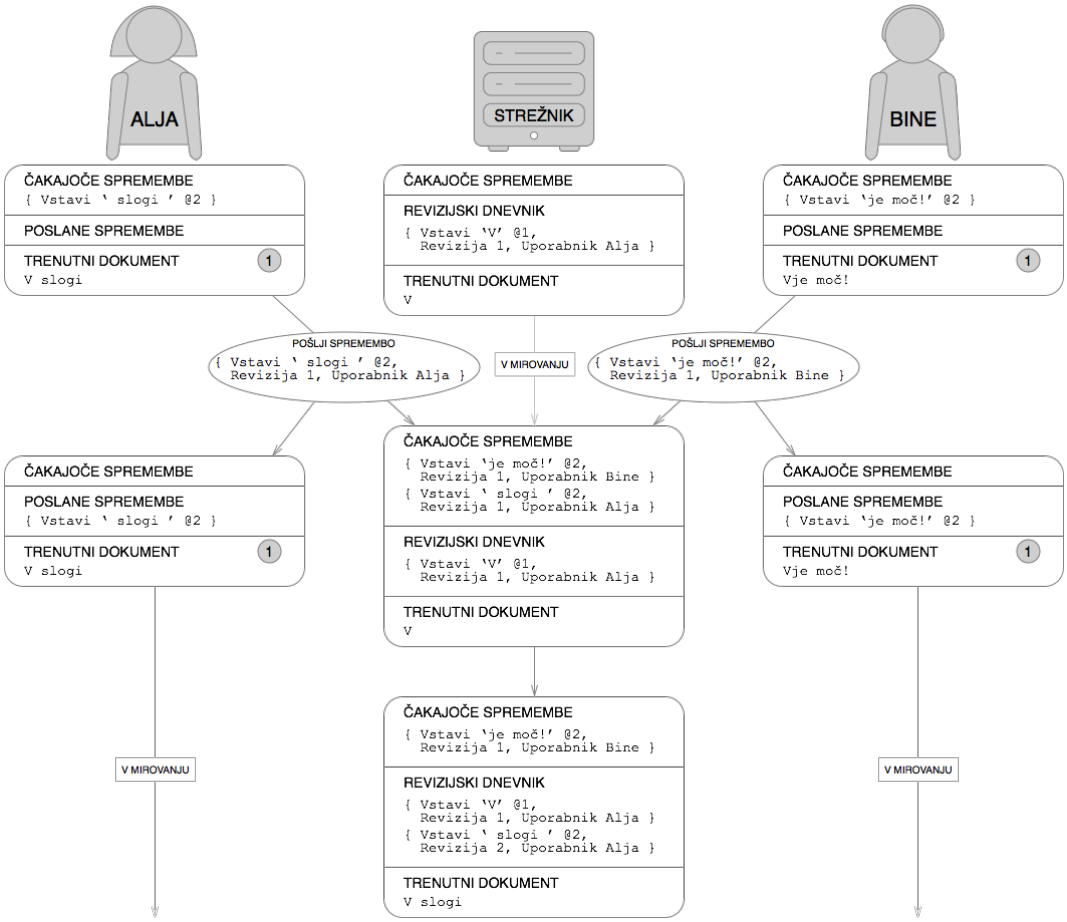



Slika 3.9: Strežnik procesira Aljino prvo spremembo.

Bine sprejme Aljino spremembo od strežnika. Z uporabo OT mora transformirati svojo čakajočo spremembo { `Vstavi 'je moč!' @1` }. Ker je Alja na začetek dokumenta že vpisala "V", Binetov urejevalnik njegovo spremembo zamakne za eno lokacijo. Po tem procesu Binetov urejevalnik v trenutni dokument vstavi Aljino spremembo "V" in posodobi številko zadnje sinhronizirane revizije na 1. Alja je med tem sprejela potrditev iz strežnika. Tudi ona posodobi številko zadnje sinhronizirane revizije na 1. Svojo spremembo odstrani iz seznama poslanih sprememb.

V Aljinem trenutnem dokumentu se nahaj "V slogi ". V Binetovem trenutnem dokumentu se nahaj "Vje moč!", kar nakazuje, da še ni prejel vseh sprememb. Sledi pošiljanje čakajočih sprememb.

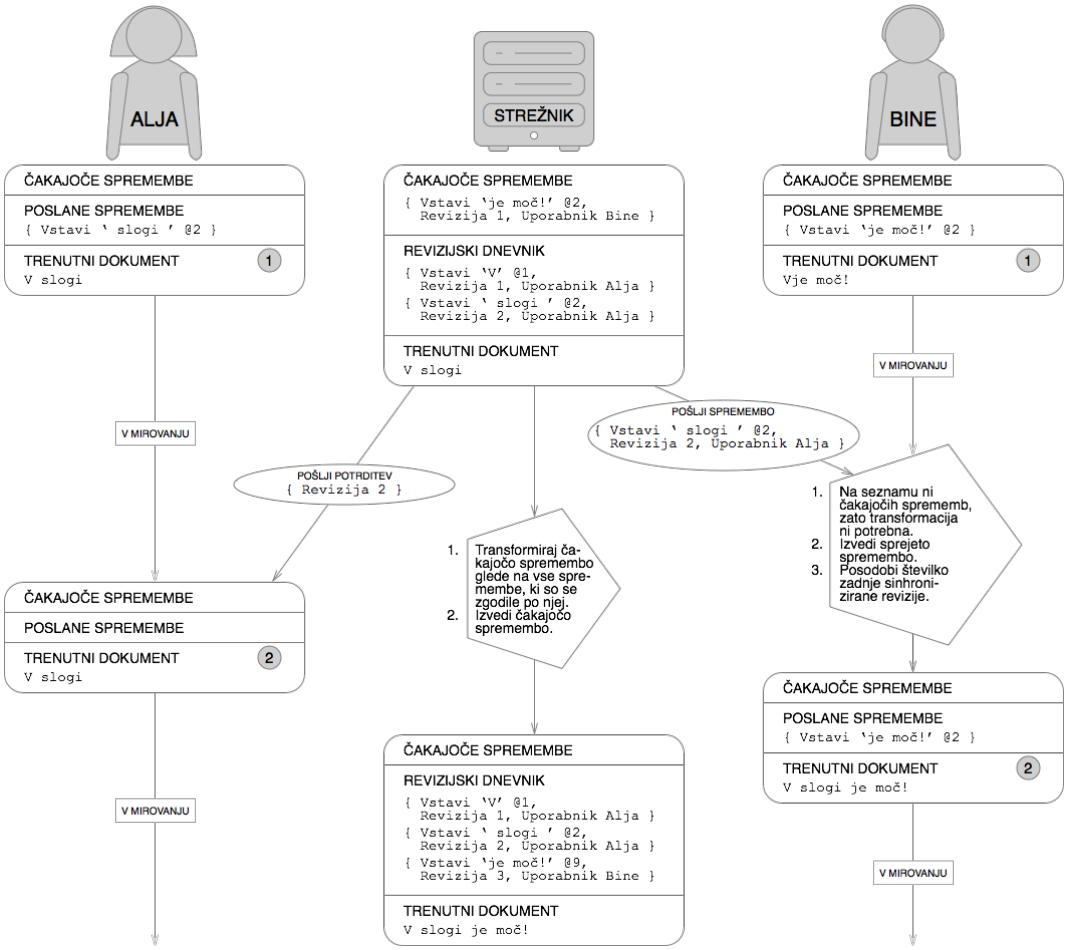
Na Sliki 3.10 se vidi, da oba uporabnika hkrati pošljeta spremembo, vendar strežnik sprejeme Aljino sprejme pred Binetovo in jo zato tudi sprocessira prej. Njeno spremembo iz seznama čakajočih sprememb prestavi v revizijski dnevnik. Pri tem mora le popraviti številko revizije, ki enolično označuje spremembo. Pomnimo, da je Alja pri pošiljanju svoje spremembe poslala številko zadnje sinhronizirane revizije, številko 1. Strežnik na ta način ve, da je njena sprememba narejena na podlagi prve revizije. Ker se v revizijski dnevnik pričakuje sprememba, ki je naslednja po vrsti, lahko njeno spremembo brez težav umesti v trenutni dokument.




Slika 3.10: Istočasno sodelovanje Aljinega in Binetovega urejevalnika preko strežnika.

Sledi potrditev spremembe Alji s številko revizije 2. Binetu se pošlje novo spremembo. Kot na Sliki 3.9 se tudi v primeru na Sliki 3.11 začne izvajati OT v Binetovem urejevalniku. Ker nima nobene čakajoče spremembe, ni potrebe po uporabi OT. Aljina sprememba se vstavi v njegov dokument brez transformacije. Številka zadnje sinhronizirane revizije se Binetu poveča na 2. Ker strežnik Alji odgovori s potrditvijo, se tudi njej poveča številka zadnje sinhronizirane revizije na 2.

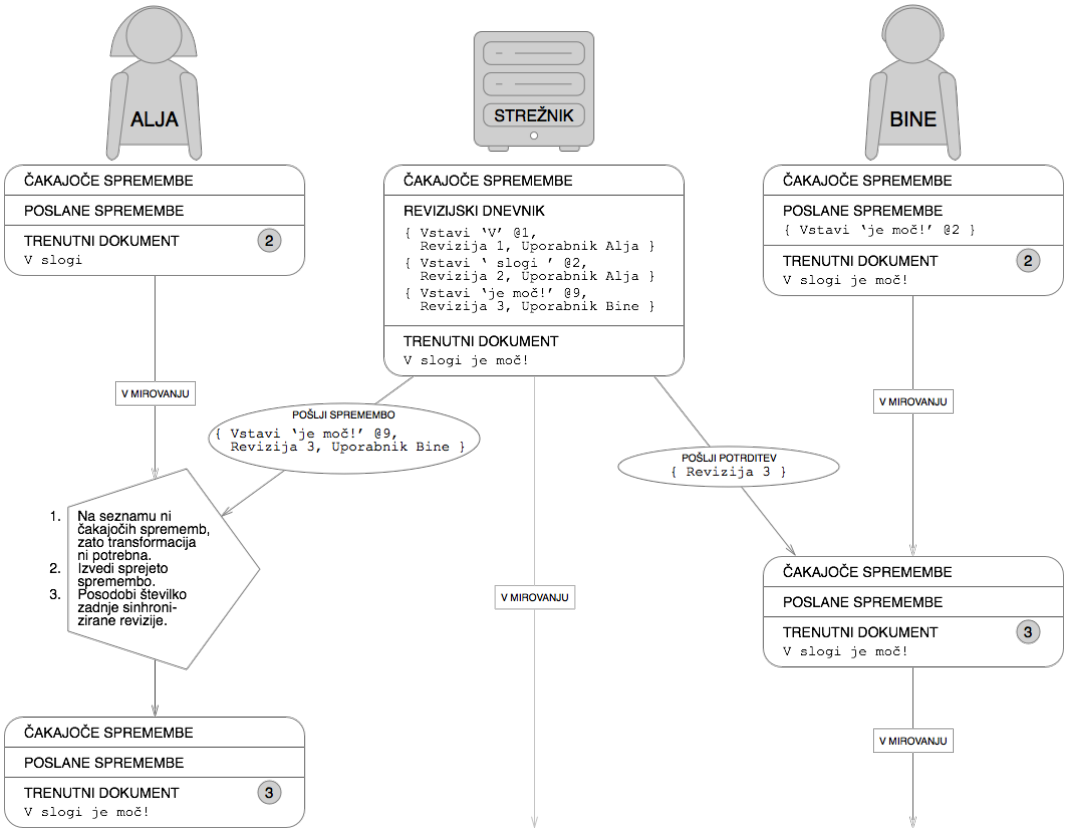
OT se ne dogaja samo pri odjemalcih, ampak je nujna tudi na strežniku. Zakaj bomo ugotovi kmalu. Med tem ko se odjemalca ukvarjata z zahtevami strežnika (potrditev pri Alji in sprememba pri Binetu), je strežnik začel procesirati Binetovo čakajočo spremembo { `Vstavi ‘je moč!’ @2, Revizija 1, Uporabnik Bine` }. Bine je v času pošiljanja spremembe (Slika 3.10) verjel, da bo njegova sprememba nosila zapredno številko revizije 2. Vendar strežnik je že obdelal Aljino spremembo, katero je v revizijski dnevnik zapisal kot drugo spremembo. V tem trenutku mora strežnik z uporabo OT transformirati Binetovo spremembo, da jo bo lahko shranil kot revizijo 3.




Slika 3.11: Strežnik uporabi OT pri obdelovanju Binetove spremembe.

Binetovo spremembo transformira glede na spremembe, ki so bile storjene od kar je Bine zadnjič naredil sinhronizacijo s strežnikom. V našem primeru je bila narejena le Aljina sprememba { `Vstavi ‘ slogi ’ @2` }, ki je povzročila zamik Binetove spremembe za 7 lokacij. Končna sprememba v revizijskem dnevniku izgleda kot { `Vstavi ‘je moč!’ @9, Revizija 3, Uporabnik Bine` }.

Na koncu Bine dobi potrditev svoje spremembe in Alja prejme Binetovo spremembo. Številka zadnje sinhronizirane revizije se obema poveča na 3. V revizijskem dnevniku so 3 revizije. V tem tenutku imajo strežnik in oba urejevalnika isti dokument z vsebino "V slogi je moč!". Glede na to, da uporabnika nimata več čakajočih sprememb je to tudi končna verzija dokumenta, ki sta ga skupaj uredila uporabnika Alja in Bine.




Slika 3.12: Strežnik uporabi OT pri odelovanju Binetove spremembe.

## 3.2 Diferenčna sinhronizacija

Konceptualno enostavni načini sinhronizacije so zaklepanje (ang. locking), prenašanje dogodkov (ang. event passing) in trosmerno združevanje (ang. three way merge).

**Zaklepanje** je najenostavnejši način. Ko uporabnik odpre skupni dokument, se mu dodeli pravica lasnika (ang. ownership). Le on ga lahko ureja. Vsi ostali uporabniki lahko v tistem času dokument gledajo (ang. read-only access). Cilj je delno dosežen. Vsi uporabniki imajo sinhroniziran dokument, vendar urejanje dokumenta se ne izvaja v realnem času. Izboljšava zaklepanja je delno zaklepenje

pri katerem se uporabniku dodeli pravica lastnika le za majhen del dokumenta. Ta način ni najboljši kadar imajo uporabniki slabo povezavo. Paket z informacijo o zaklepanju in odklepanju se lahko izgubi. V dokumentu lahko nastanejo deli nad katerimi ima lahko več uporabnikov pravico lastnika. Pojavijo se tudi druga anomalije.

**Prenašanje dogodkov** temelji na zaznavanju vseh interakacij, ki jih naredi uporabnik z urejevalnikom. V teoriji je to enostavno, saj vsak sistem omogoča zaznavanja tipkanja. V praksi je težko izvedljivo. Poleg tipkanja lahko uporabnik naredi tudi akcije kot so izreži besedilo, prilepi besedilo, povleci in spusti besedilo, zamenjava besedila... Akciji kot sta samodokončanje in pravpisni popravek lahko naredi tudi sam urejevalnik. Kaj narediti v teh primerih? Težava je tudi v tem, da lahko zaradi neke napake ali zamude v komunkaciji pride do dveh popolnom različnih dokumenetov, ki jih ne moremo poenotiti.

**Trosmerno združevanja** uporabljajo programerji pri delu na skupnem projektu. Je zelo robusten sistem. Sestavljen je iz treh korakov. Uporabnik najprej svojo vsebino pošlje na strežnik. Strežnik izvzame narejene spremembe in jih združi s spremembami drugih uporabnikov. Nova kopija dokumenta se sinhronizira vsem uporabnikom. Sistem trosmernega združevanja ima nekaj slabosti. Če uporabnik naredi spremembo na dokumentu medtem, ko je sinhronizacija v teku, mora zavreči vse na novo narejene spremembe. Slabost je, da uporabnik ne dobi nobene povratne informacije (ang. feedback) med tipkanjem. Način trosmernega združevanja bi pri urejanju v realnem času deloval le v dveh primerih. Če bi si lahko privoščili, da bi uporabnika blokirali in sinhronizirali z novo verzijo na strežniku ali pa v primeru, da bi uporabnika sinhronizirali, ko neha tipkati. Ampak nobena od teh dveh variant ni urejanje dokumenta v realnem času.

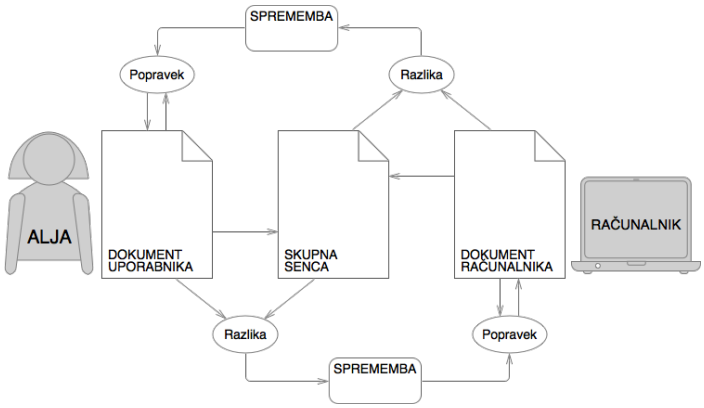
Ker sinhronizacija in ohranjanje konsistentnosti med uporabniki ni enostavna, si oglejmo rešitev imenovano Diferenčna sinhronizacija (v nadaljevanju DF). DF je simetričen algoritem, ki uporablja neskončno ciklov razlik na dokumentih in z njimi popravlja ostale dokumente v ciklu. Najprej bomo preučili osnovno topologijo DS, ki teoretična podlaga za izboljšave z metodo dveh senc in z metodo garantirane dostave.

### 3.2.1 Osnovna topologija

Na Sliki 3.13 je z diagramom prikazan DS. V osnovni topologiji obstajata dokument uporabnika in dokument računalnika. Oba sta locirana na istem sistemu brez internetne

povezave. Med njima se nahaja dokument imenovan skupna senca (ang. common shadow). Predvidimo, da imajo na začetku vsi trije dokumenti isto vsebino. Cilj je, da so dokumenti vseskozi čim bolj enaki. Ostale entitete so še razlika, sprememba in popravek. Razlika je signal, ki pove, da se dokument uporabnika razlikuje od skupne sence. Na ta način vemo, da je uporabnik naredil spremembo na svojem dokumentu. Ni nam potrebno spremljati vsake uporabnikove akcije. Ko se ve, kakšne spremembe so bile narejene, se mora dokument prekopirati v senco. Sprememba se pošlje v smeri računalnika. Na dokumentu računalnika se naredi popravek. Proces se ponovi še v smeri računalnika proti uporabniku. Vsi trije dokumenti so sinhronizirani.

Sistem je zanesljiv. Težava je, da taka zasnova ne omogoča sodelovanja oddaljenih uporabnikov v realnem času saj vse skupaj poteka na enem sistemu.




Slika 3.13: Osnovna topologija Diferenčne sinhornizacije.

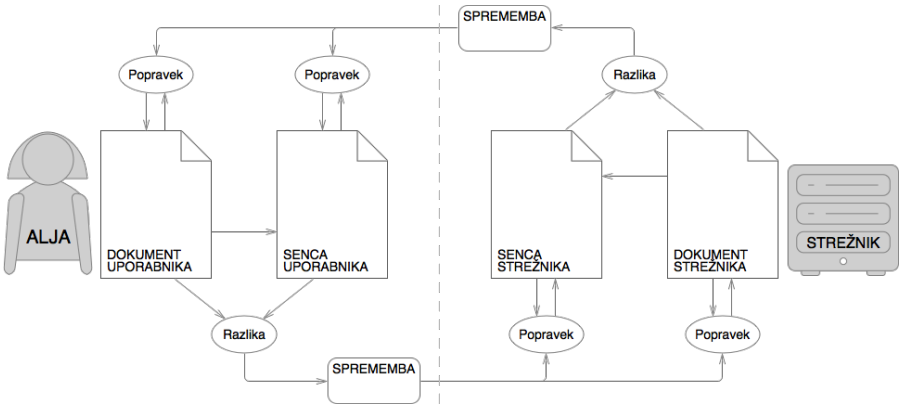
### 3.2.2 Metoda dveh senc

Konceptualno še vedno ostajamo pri istem algoritmu. Razlika je v topologiji. Namesto skupne sence uporabimo senco uporabnika in senco strežnika, ki se s popravki posodabljata ločeno. Taka zasnova DF omogoča sodelovanje oddaljenih uporabnikov. Vsak uporabnik je na svojem sistemu. Uporabnike povezuje centralni strežnik. Zaradi enostavnosti razlage je na Sliki 3.14 prikazan le en uporabnik in strežnik, ki sta ločna s črtkano črto. Po vsakem ciklu morata biti dokument uporabnika in dokument strežnika identična.

Predvidimo, da se na začetku vsi dokumenti v konsistentnem stanju. Uporabnik začne tipkati. Vključi se signal, da je med dokumentom in senco uporabnika razlika. Ko se ve, kakšne spremembe so bile narejene, se mora dokument prekopirati v senco.

Sprememba se pošlje strežniku. Na strežniku se naredita dva popravka, na senci in na dokumentu. Pomembno je, da se popravek na senci izvede brez problemov. Na dokumentu se naredi nejasen (ang. fuzzy) popravek. Nejasen popravek izvira iz sodelovanja med uporabniki. Če eden izmed uporabnikov naredi korenito spremembo na dokumentu, povzroči, da se popravek prvega uporabnika ne umesti v dokument tako kot bi si on želel. Nepredvideno stanje se reši v naslednji polovici cikla. Senca in dokument strežnika sta v tem trenutku različna. Ko se ve, kakšne so spremembe med senco in nepredvidenim dokumentom strežnika, se mora dokument prekopirati v senco. Sprememba se pošlje uporabniku. Ker je bila sprememba dokumenta strežnika narejena na podlagi sence strežnika, ki je bila ista kot senca in dokument uporabnika, se brez težav naredi popravek na senci in na dokumentu uporabnika. Dokumenti so sinhronizirani.

Sistem omogoča sodelovanje oddaljenih uporabnikov, vendar ni zanesljiv. Lahko se zgodi, da uporabnik na nezanesljivi povezavi od strežnika ne dobi nobenega odgovora. Do izgube spremembe lahko pride v prvem ali v drugem delu cikla. Senca uporabnika in senca strežnika nista več sinhronizirani. Edina rešitev za povrnitev nastalega stanje je, da povozimo uporabnikove spremembe. Tega si nihče ne želi.

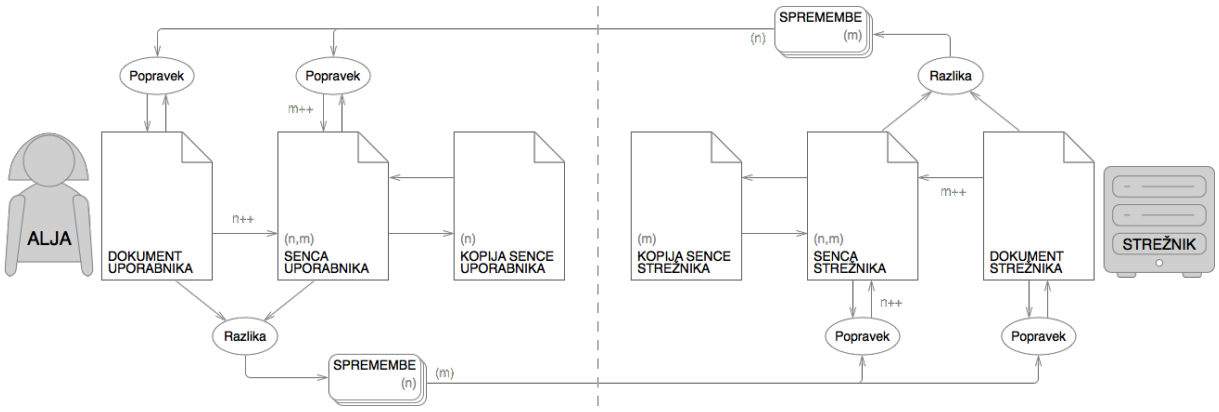



Slika 3.14: Diferenčna sinhronizacija z metodo dveh senc.

### 3.2.3 Metoda garantirane dostave

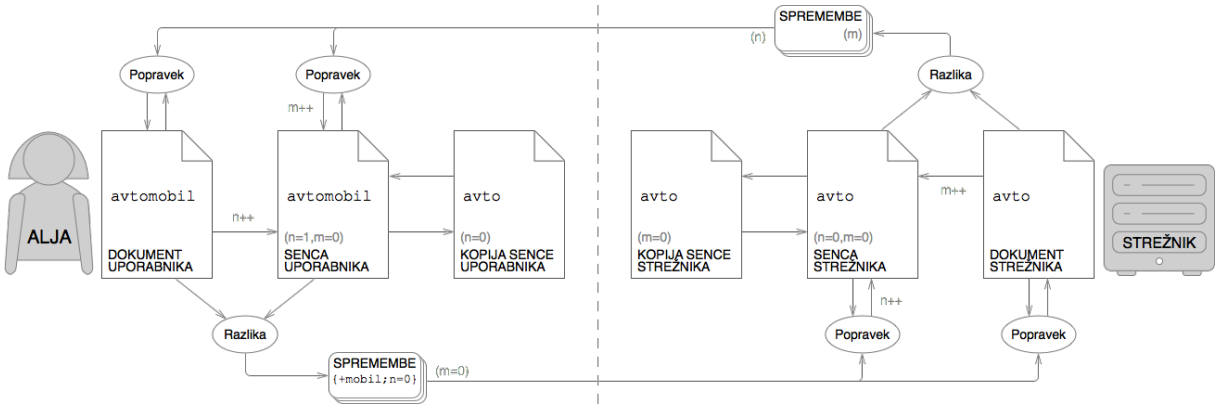
DS z metodo garantirane dostave je nadgradnja metode dveh senc. Iz Slike 3.15 lahko razberemo, da se je topologija razširila še z varnostnima kopijama sence uporabnika in sence strežnika. Tudi tokrat mora biti vseh šest dokumentov v konsistentnem stanju. Namesto ene same spremembe, se lahko pošlje več sprememb na enkrat. Nove v

topologiji so tudi številke verzij. S črko `n` je označena številka verzija uporabnika. S črko `m` je označena številka verzija strežnika. Na začetku sta obe številki verziji enaki `0`.




Slika 3.15: Diferenčna sinhronizacija z metodo garantirane dostave.

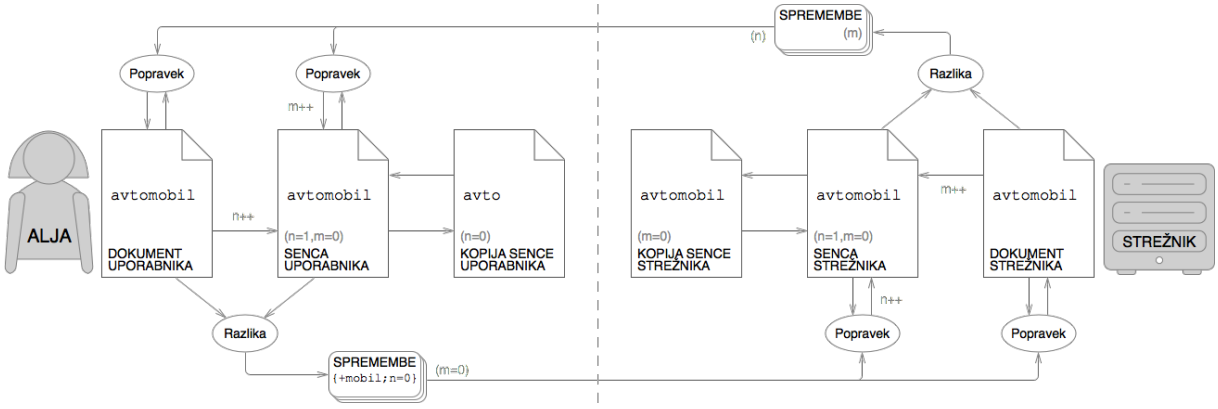
Delovanje DS z metodo garantirane dostave bomo razložili na primeru. Na začetku je v vseh dokumentih beseda `avto`. Alja v svoj dokument dopiše `mobil`. Posledica razlike dokumenta uporabnika od sence uporabnika je sprememba `mobil`. K sami spremembi se shrani številke verzije uporabnika na podlagi katere je bila sprememba osnovana. V seznam sprememb se torej shrani { `+mobil;n=0` }. V tem trenutku se dokument Alje prekopira v njeno senco ter poveča se številka verzije uporabnika na `n=1`. Sprememba se pošlje strežniku. Poleg spremembe se strežniku pošlje tudi številka zadnje sinhronizirane verzije strežnika `m=0`. Stanje je prikazano na Sliki 3.16.




Slika 3.16: Alja v svojem dokumentu vidi besedo avtomobil.

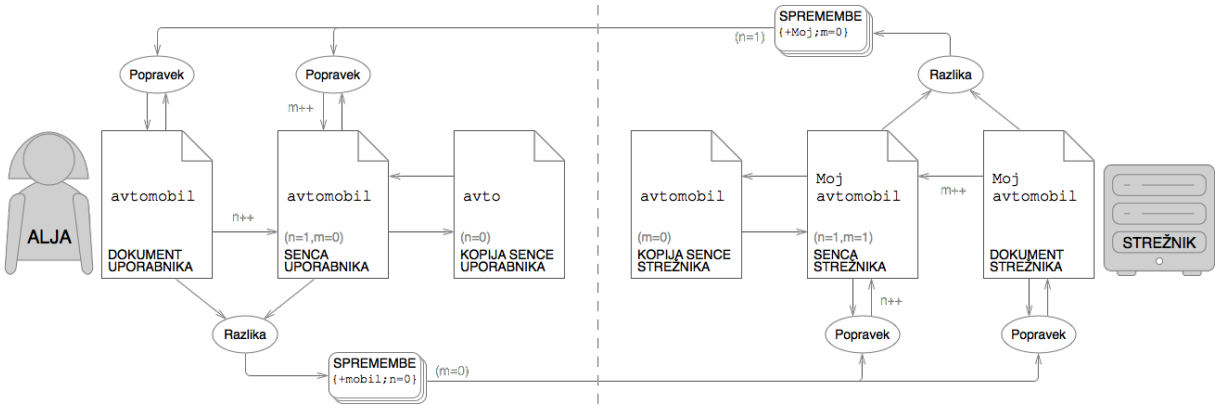
Strežnik primerja številki verzije uporabnika in strežnika spremembe s številkami verzije uporabnika in strežnika v senci strežnika (glej Sliko 3.16). Številki `n` in `m` sta

v obeh primerih `0`, kar dovoljuje, da se naredi popravek na senci strežnika. Številka verzije uporabnika v senci strežnika se poveča na `n=1`. Senca strežnika in senca uporabnika imata enako besedilo in enaki številki `n` in `m`, kar pomeni, da sta sinhronizirani. To se vidi tudi na Sliki 3.17. Popravek se naredi tudi na dokumentu strežnika, kar povzroči, da se senca strežnika prekopira v varnostno kopijo sence strežnika. Zakaj je to potrebno, bomo videli v nadaljevanju.




Slika 3.17: Polovica cikla je končanega. Senci sta konsistentni.

Recimo, da uporabnik Bine na začetek dokumenta vpiše besedo `Moj`. Na isti način kot Aljina sprememba se tudi njegova sprememba sinhronizira v dokument strežnika. Vsebina dokumenta strežnika je tako `Moj avtomobil`. Sedaj lahko nadaljujemo z drugim delom cikla iz smeri strežnika proti Alji.

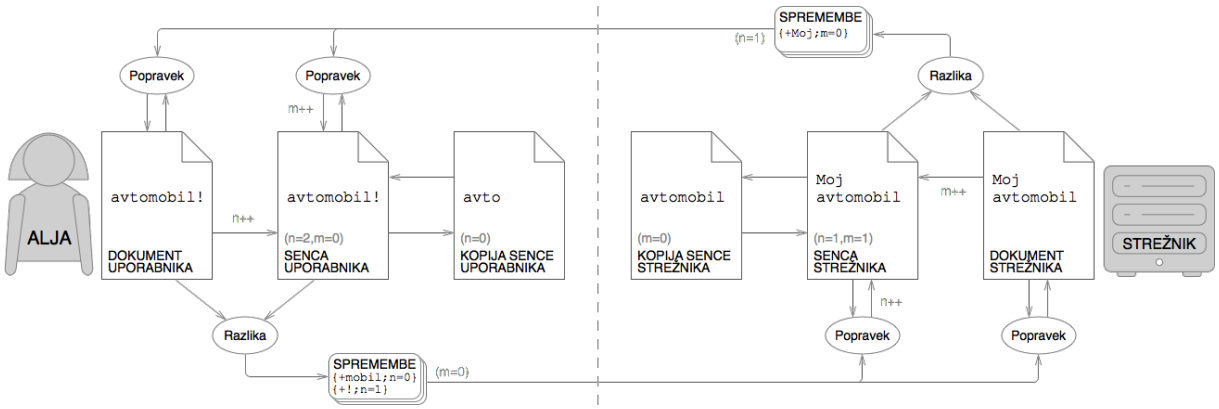



Slika 3.18: Server prejme spremembo od Bineta, ki jo pošlje Alji.

Razlika dokumenta strežnika od sence strežnika je sprememba `Moj` osnovana na številki verzije strežnika `0`. Sprememba { `+Moj;m=0` } se pošlje k Alji. Poleg nje se

pošlje tudi `n=1` s katerim strežnik potrjuje, da je sprejel zadnjo Aljino spremembo. Pred tem se dokument strežnika prekopira v senco strežnika. Številka verzije strežnika `m` v senci strežnika se poveča na 1, Slika 3.18.

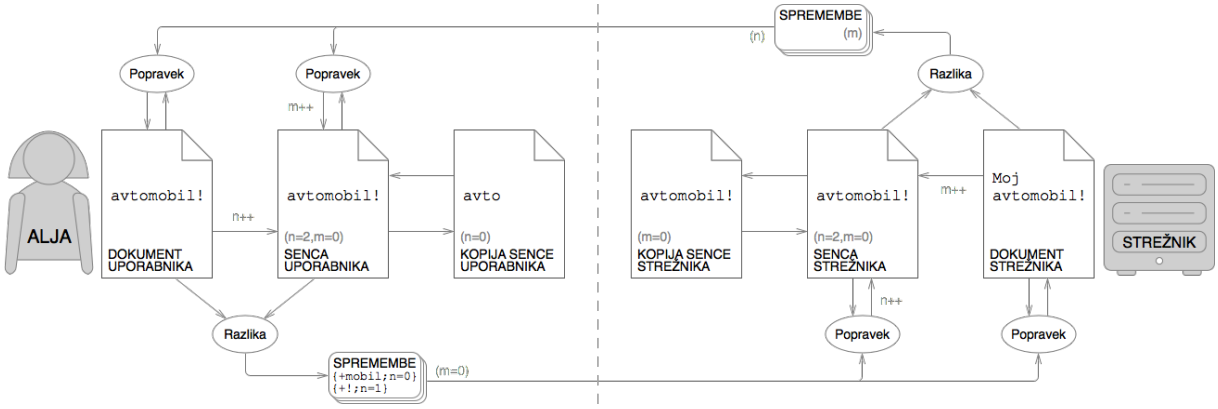
Zaradi slabe povezave Alja ne prejme sprememba { `+Moj;m=0` } iz strežnika. Tega niti ne opazi, ampak tipka naprej. V svoj dokument doda klicaj. Ponovi se postopek podoben prvemu ciklu. K seznamu sprememb se doda { `+!;n=1` }. Dokument Alje se prekopira v njeno senco, `n` se poveča na `2`. Obe spremembi se pošljejo strežniku. Pomnimo, da za prvo spremembo Alja ni dobila nobenega odgovora. Številka zadnje sinhronizirane verzije strežnika je še vedno `m=0`. Slika 3.19 prikazuje nastalo situacijo.




Slika 3.19: Alja ni sprejela strežnikovih sprememb. V svojem dokumentu je naredila novo spremembo.

Strežnik sprejme Aljini spremembi. Najprej poskuša sprocesirati prvo spremembo { `+mobil;n=0` }. Ker se številka zadnje sinhronizirane verzije strežnika `m=0` ne ujema s številko verzije strežnika v senci strežnika, se sence strežnika povrne iz kopije sence strežnika (s številko `m=0`) v katerem se nahaja `avtomobil`. Strežnik poskusi ponovno sprocesirati prvo spremembo. Številki verzije strežnika `m=0` se ujemajo. Številka verzije uporabnika je v prvi spremembi `n=0`, v senci strežnika pa `n=1`. To pomeni, da je strežnik v enem izmed prejšnjih ciklov že obdelal to spremembo, zato jo lahko ignorira. Strežnik nato poskusi sprocesirati drugo spremembo { `+!;n=1` }. Številki `n=1` in `m=0` se tokrat ujemajo. Strežnik lahko naredi popravek na senci strežnika. Številka `n` se poveča za `1` na `n=2`. Popravek se nato naredi tudi na dokumentu strežnika, naredi pa se tudi kopija sence strežnika. Senca uporabnika in senca strežnika sta sinhronizirani.

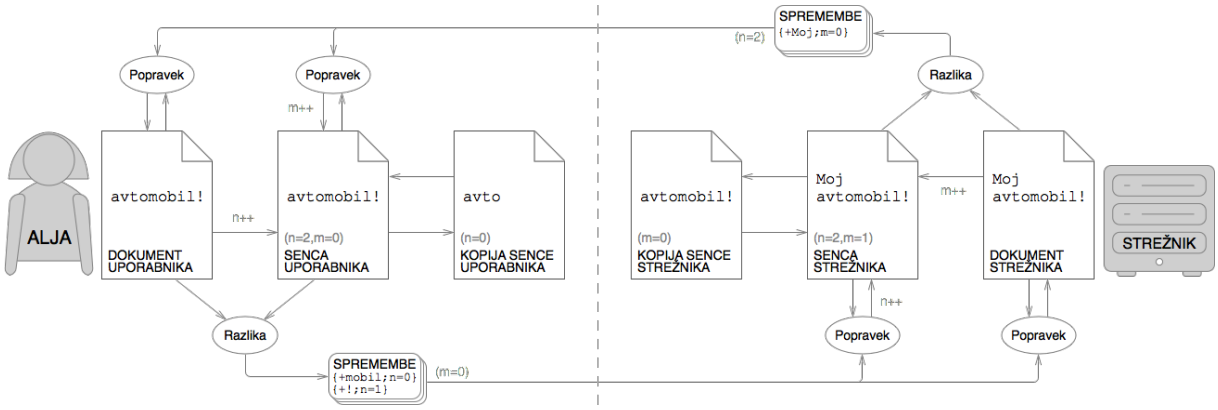
Opomba! Ko se je naredila povrnitev kopije sence strežnika, se je seznam strežniških sprememb pobrisal. Spremembe v dokumentih in številkah `n` in `m` si oglejmo na Sliki 3.20.




Slika 3.20: Alja je strežniku poslala že dve spremembi.

Dokument strežnika in senca strežnika se na Sliki 3.20 razlikujejta. Strežnik pogleda kakšne so spremembe na dokumentu. Tako kot v prvem ciklu najde spremembo { `+Moj;m=0` }. Sprememba se ponovno pošlje Alji. Tokrat je številka verzija uporabnika v senci dnevnika enaka `2`, zato se Alji poleg spremembe pošlje tudi `n=2`. Še prej se dokument prekopira v senco in v senci strežnika se poveča številka verzije strežnika iz `m=0` nazaj na `m=1`.

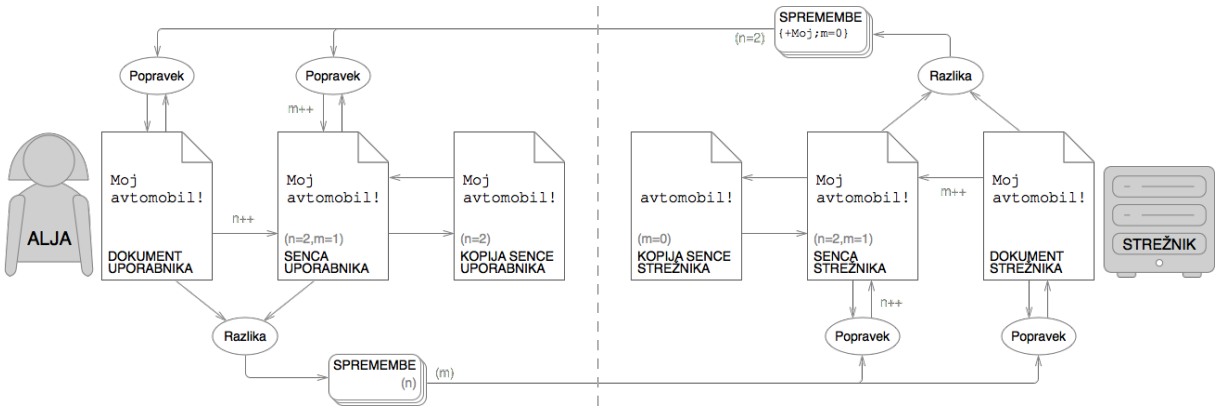
Če bi bil strežnik spet neuspešen pri pošiljanju svojih sprememb, bi se seznam Aljinih sprememb povečeval vse dokler ne bi dobila odgovora od strežnika. Recimo, da tokrat strežniku uspe in Alja prejme spremembo { `+Moj;m=0` }.




Slika 3.21: Strežnik pošilja Alji potrditev njenih spremembe in spremembo Bineta.

Tokrat Alja primerja številke verzij. Številka verzije uporabnika v spremembi in v senci uporabnika je `n=2`. Številka verzija strežnika v spremembi in v senci je `m=0`. Sprememba je primerna za popravek na senci uporabnika. Naredijo se naslednje akcije. Na senci uporabnika se naredi popravek. Številka verzije zadnje sinhronizacije

s strežnikom se v senci uporabnika poveča na `m=1`. Popravek se naredi na dokumentu strežnika. Naredi se kopija sence uporabnika. Številka verzije uporabnika v kopiji sence uporabnika se poveča na `n=2`. Ker je Alja uspešno sprejela potrditev strežnika, da je sprejel vse Aljine spremembe do številke verzije uporabnika `n=2`, se iz seznama Aljinih sprememb pobrišejo vse spremembe, ki imajo številko verzije uporabnika manjše od `2` (`n<2`).




Slika 3.22: Konsistentno stanje dokumentov.

Kjub temu, da je med sodelovanjem uporabnikov in strežnika prišlo do izpada v povezavi, so se dokumenti posinhronizirali. V dokumentu na koncu piše "`Moj avtomobil!`".

## 3.3 WOOT

V tem poglavju bomo opisali protkol Brez operativne transformacije, bolj znan pod kratico WOOT. Bil je narejen kot odgovor na kopleksnost OT. Rekli smo, da se pri OT med uporabnike pošiljajo spremembe in lokacija sprememb, primer { `Vstavi 'M' @11` }. OT skrbi za pravilno transformacijo lokacij sprememb. Pristop WOOT je drugačen in ga je lažje razumeti. WOOT skrbi za pravilno razvrščanje sprememb oziroma operacij, kot se jih običajno poimenuje. Glavna razlika je v podajanju informacij. Lahko si predstavljamo, da na vsak znak v dokumentu kaže unikatni kazalec. Med oddaljene uporabnike, ki sodelujejo pri urejanju dokumenta, se pošiljajo informacije med katerima znakoma oziroma na kateremu znaku je bila narejena sprememba. Posebnost protokola WOOT je, da se znakov v dokumentu nikoli ne briše, le označi se jih kot nevidne.

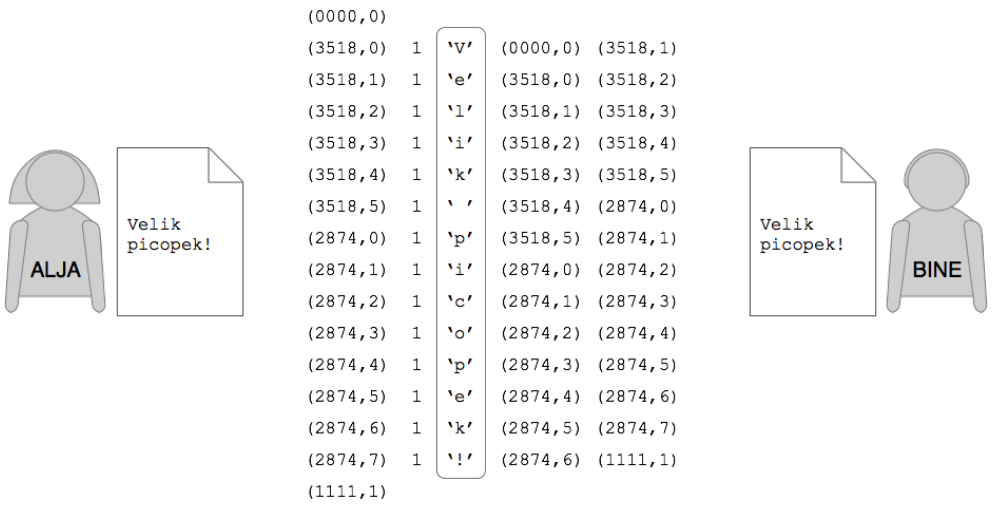
### 3.3.1 Podatkovni model

Dokument pri WOOT-u je shranjen kot zaporedje znakov.

Za vsak znak v dokumentu hranimo 5 informacij: identifikacijo številko, vidnost, vsebino znaka, prejšnji znak in naslednji znak. Identifikacijska številka je kazalec na znak, ki je sestavljena iz unikatne oznake dokumenta uporabnika in lokalne ure ali števca. Vidnost nam pove ali uporabnik določen znak vidi. Pri brisanju, se vidnost postavi na 0. Vsebina znaka je črka ali številka, ki jo znak predstavlja. Prejšnji znak je identifikacijaska številka levega soseda. Naslednji znak je identifikacijska številka desnega soseda.

Poznamo tudi dva posebna znaka, ki označujeta konec in začetek dokumenta.

Alja in Bine na Sliki 3.23 urejata skupni dokument v katerem se trenutno nahaja besedilo "`Velik picopek!`". Dokument bi bil pri pri Alji in Binetu shranjen kot zaporedje znakov od `V` do klicaja.

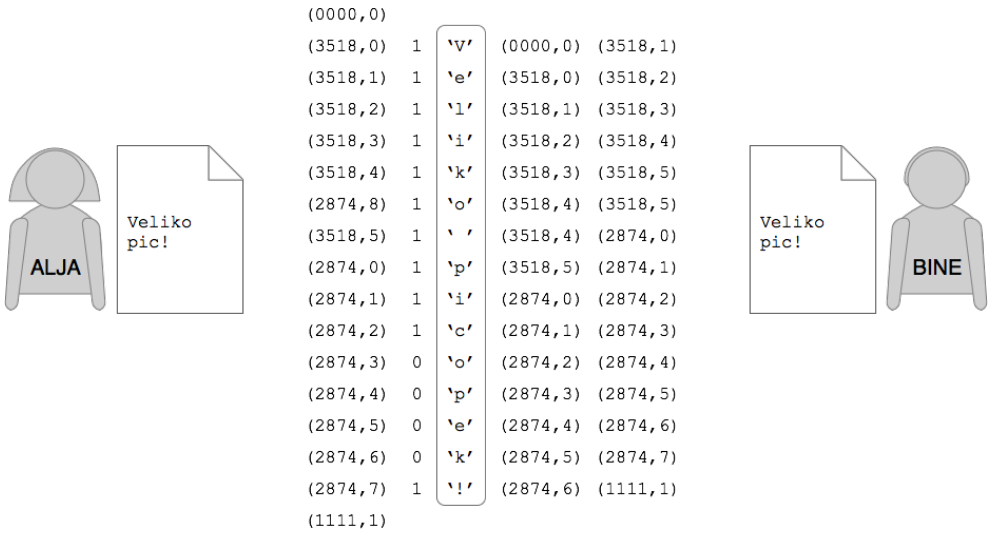



Slika 3.23: Dokument shranjen po protokolu WOOT.

Znaka `(0000,0)` in `(1111,1)` označujeta začetek oziroma konec dokumenta. Za vse ostale znake v stolpcih od leve proti desni hranimo identifikacijsko številko, vidnost, vsebino, prejšnjega ter naslednjega soseda. Iz identifikacijskih številke vidimo, da je prvo besedo skupaj s presledkom napisala oseba katere dokument je označen z unikatno številko `3518`. Drugo besedo skupaj s klicajem je napisala druga oseba. Njen dokument je označen z unikatno številko `2874`. Lahko bi si shranjevali katerega uporabnika predstavlja katere unikatna številka, vendar ni potrebno. Vse črke so vidne. Nobena črka ni bila pobrisana. Glede na oznake sosedov so črke pravilne razvrščene.

Recimo, da Alja in Bine na dokumentu naredita nekaj sprememb. Alja dopiše črko

o, Bine pobriše opek, nastane besedilo "Veliko pic!". Na Sliki 3.24 je prikazano, kako je sedaj shranjen dokument.




Slika 3.24: Nekaj črk v dokumentu je skritih.

Čeprav so v dokumentu shranjene vse črke, ki sta jih napisala Alja in Bine, se njima v urejevalniku kažejo le črke, ki so označene kot vidne. Opazimo lahko tudi, da imata črka k z identifikacijsko številko (3518,4) in črka o z identifikacijsko številko (2874,8) enakega naslednja soseda (3518,5). Ta pojav je normalen in ni napaka. Pravilno zaporedje znakov je določeno s prejšnjim sosedom. Prejšnji sosed črke o je črka k. Lahko bi se zgodilo, da bi dva znaka imela istega prejšnjega soseda in različa naslednjega soseda. Pravilno zaporedje znakov bi bilo v tem primeru določeno z informacijo o naslednjem sosedu.

### 3.3.2 Operacije

Osredotočili se bomo na operaciji Vstavi in Pobriši. Ko uporabnik generira operacijo, se operacija najprej integrira lokalno. Nato se razpošlje vsem oddaljenim uporabnikom, ki sodelujejo pri urejanju. Ko jo sprejmejo, se še njim integrira v dokument.

Potreba po vnaprejšnji lokalni integracija pride iz razvrščanja sprememb. Na primer, ko uporabnik generira spremembo { Vstavi 'Č' @4 }, se le ta v uporabniškem vmesniku prikaže kot črka Č na lokaciji 4. Sprememba se mora pretvoriti v { Vstavi 'O'<'Č'<'I' }, ki pomeni vstavi črko Č, med črko O in I. Lokacija spremembe je tako definirana s svojima dvema sosedoma in ne s konkretno številko lokacije. Pretvorba je prikazana poenostavljeno. V dejanskem algoritmu bi sose-

dnja dva znaka označlil z njunima identifikacijskama številkama. Podobno kot vstavljanje, se mora tudi brisanje pretvoriti WOOT protokolu primerno. Sprememba { `Pobriši @7` } pobriše znak na lokaciji 7. V primeru, da je na lokaciji 7 črka `R`, moramo spremembo pretvoriti v { `Pobriši 'R'` }. Seveda je tudi ta primer poenostavljen, saj je tudi operacija brisanja vezana na identifikacijsko številko in ne na konkretno črko ali številko. Pomembno je poudariti, da znakov pri protokolu WOOT nikoli ne brišemo, ampak jih le skrivamo. Dokument je shranjen kot zaporedje uporabniku vidnih in uporabniku nevidinih znakov. Na ta način je razvrščanje spremembe vedno pravilno, saj je odvisno tudi od nevidnih znakov.

Ko uporabnik sprejme operacijo, mora le ta iti najprej preko preverjanje predpogojev za izvršitev. Če predpogoj ni izpolnjen, se itegracija operacije začasno prestavi nazaj med čakajoče operacije. Primer, da pogoj ni izpolnjen je, ko uporabnik sprejme operacijo { `Pobriši 'B'` }, črka `B` pa v njegovem dokumentu še ne obstaja. Operacija brisanja črke `B` je prehitela vstavljanje črke `B`, zato operacija brisanja trenutno še ne more biti integrirana.

Integracija brisanja pri oddaljenih uporabnikih je enostavna operacija. Pri njej se le postavlja vidnost na 0 ali 1. Pri integraciji vstavljanja pri oddaljenih uporabnikih lahko pride do problemov. Znak, ki se ga integrira, se mora postavi direktno med dva sosednja znaka. Če so se na to mesto pred sprejemom operacije, vrinili tudi že drugi znaki, so potrebne primerjave z vrinjenimi znaki. Pomembno je, da se vedno izvede enaka strategija, ki zagotovi konsistentnost med uporabniki.

### 3.3.3 Točka-točka sodelovanje

Prej omenjena protokola DS in OT temeljita na centralnih strežnikih in sta od njih odvisna. Za destribucijo vsebine so bolj učinkovita točka-točka (ang. peer-to-peer, kratica P2P) omrežja. Ta potencial želimo izkoristiti tudi pri sodelovanju urejanja vsebine. Protokol WOOT je popolnoma decentraliziran. Na primerih bomo videli, da ni odvisen od centralnih strežnikov.

# Poglavje 4

# Modeli konsistence, struktura in pravilnost

Nulla vel lorem erat. Nullam convallis molestie mauris non vulputate. Fusce sagittis purus non mi interdum, ac tempor nunc vulputate. Vivamus tempor diam a laoreet porta. Aliquam adipiscing elit vel risus accumsan, eget lacinia felis rutrum. Mauris ornare mattis vestibulum. Proin adipiscing semper turpis, eleifend sollicitudin tortor aliquam eget.

Nam non odio at arcu tincidunt laoreet. Aenean iaculis magna nec risus semper, eu hendrerit nunc blandit. Vivamus in eros sit amet augue interdum fringilla. Nullam semper volutpat iaculis. Aliquam erat volutpat. Nullam magna magna, adipiscing non leo imperdiet, pharetra facilisis ante. Mauris quis felis sed turpis sagittis varius. Quisque laoreet elementum mauris, ac lacinia magna ultricies ac. Praesent a blandit nibh. Maecenas malesuada velit ac accumsan convallis. Quisque ultrices arcu lacus, vulputate cursus arcu faucibus id. Suspendisse blandit at enim eu condimentum. Praesent laoreet luctus enim, nec ultrices est varius eu. Vestibulum viverra ante eu est ultrices feugiat eget nec mauris. Vestibulum tempus nisl risus, non condimentum ipsum elementum a. Maecenas ligula enim, consectetur sit amet dignissim eu, blandit nec enim. In hac habitasse platea dictumst. Praesent auctor ligula ut justo congue mattis. In mollis porta tempor. Cras tincidunt egestas porta. Donec non tristique arcu. Vestibulum tempor ac nisl non tincidunt. Duis hendrerit risus mi, ut commodo orci eleifend a. Aenean hendrerit leo sed consectetur facilisis. Duis gravida blandit accumsan. Curabitur id volutpat lorem. Nunc ac vulputate massa. Morbi aliquam cursus ligula id malesuada. Nam ut felis orci. Vestibulum tincidunt cursus mi, hendrerit scelerisque orci aliquet sit amet. Quisque placerat nunc sit amet nibh congue tincidunt. Sed vitae hendrerit mauris, in egestas leo. Etiam blandit hendrerit felis a malesuada. Cras ultri-

cies urna sed tincidunt blandit. Donec at gravida purus, eu gravida augue. Curabitur varius nibh nec erat hendrerit blandit. Etiam commodo elit ut luctus elementum.

In hac habitasse platea dictumst. Vestibulum lorem orci, suscipit eget fermentum nec, convallis in odio. Curabitur eu rhoncus eros, faucibus ullamcorper mi. Nam varius sapien tellus, et mollis nisl hendrerit venenatis. Vivamus facilisis lectus ac nisl porttitor, a placerat sem egestas. Aliquam ornare iaculis justo pellentesque venenatis. Donec euismod libero sed neque venenatis ullamcorper. Donec vestibulum quam lorem, id ornare sem sodales nec. Curabitur tristique nibh pellentesque, blandit leo at, hendrerit felis. Donec rhoncus massa non tortor condimentum semper. Nulla varius nulla vel elit vestibulum, nec malesuada nibh posuere. Vestibulum sagittis est nec laoreet lobortis. Proin eget varius quam. Vestibulum sit amet sodales libero, non commodo purus. Sed ullamcorper nulla a imperdiet dignissim. Aliquam sed turpis porta, rutrum erat nec, iaculis purus. Mauris gravida erat non tortor faucibus, vitae malesuada dolor vehicula. Morbi volutpat ipsum non mauris gravida luctus. Nunc venenatis molestie ligula, volutpat bibendum felis scelerisque eget. Phasellus eget lectus vitae nisi bibendum dignissim sit amet eget odio. Fusce sed ipsum in turpis feugiat dictum. Donec sed arcu orci. Nullam ac suscipit urna. Ut sollicitudin sagittis justo id volutpat.

Etiam blandit diam in tincidunt facilisis. Donec mi est, malesuada et felis non, condimentum cursus nisl. Sed sit amet nisi ac massa eleifend commodo ut a enim. Fusce congue at dui at aliquet. Sed et nisi facilisis, dapibus elit nec, lobortis libero. Sed a orci et ante venenatis feugiat. Praesent quis sodales mauris, id tincidunt eros. Suspendisse potenti. Vestibulum leo neque, vulputate mattis mollis ac, ultricies eget mi. Proin tortor ligula, fringilla nec sapien et, rhoncus rutrum augue. Vestibulum ut tellus laoreet, congue elit eget, semper sapien. Cras diam mi, lobortis ut vehicula ut, hendrerit vel nisi. Quisque auctor dui id enim iaculis, ac varius nisl porttitor. Cras in lectus vulputate, rutrum risus ut, ornare nisl. Sed viverra ligula vel tristique sagittis. Donec non leo eu lorem ultricies congue nec nec lacus. Maecenas egestas sed augue nec sodales. Integer ac leo eu risus pharetra auctor vitae non nibh. Cras in purus eu eros imperdiet mattis in sed ligula. Pellentesque id varius sem. Ut et velit mattis lacus sagittis egestas non ac lectus. Proin eget neque mi. Maecenas molestie, sapien vitae tincidunt luctus, felis sapien semper diam, ac consectetur augue lacus vitae augue. Vivamus bibendum libero eu elit lobortis elementum. Integer mollis neque et sapien vestibulum cursus. In eget elit eu ligula aliquam sollicitudin. Curabitur iaculis vel lorem id fermentum. Curabitur ante massa, laoreet in semper eget, interdum ac magna. Vestibulum tincidunt dignissim volutpat. Phasellus urna velit, imperdiet quis semper quis, rhoncus nec est. Aliquam mauris nunc, tincidunt a placerat vitae, scelerisque quis

lorem.

Interdum et malesuada fames ac ante ipsum primis in faucibus. Nam sagittis ipsum nisi, at ultricies nisl interdum sed. Nulla facilisi. Vestibulum tristique nunc ipsum, nec luctus tellus vulputate sit amet. Sed diam justo, imperdiet eu magna ut, convallis accumsan neque. Nulla non dolor volutpat, mollis justo vitae, auctor est. Nunc id mi nulla. Integer aliquet, tellus id dictum lobortis, arcu urna posuere lectus, in rutrum orci sapien in ante. Etiam aliquam turpis sapien, quis varius nisl pretium accumsan. Suspendisse ultrices nibh sed libero fermentum, et scelerisque turpis pretium. Sed lobortis nibh eu ante scelerisque auctor. Donec eu interdum metus, et dignissim nulla. Maecenas vestibulum rutrum lacus. Fusce fringilla metus hendrerit, ultricies velit at, blandit libero. Nulla nisl eros, adipiscing in mauris eu, ullamcorper pretium augue. Ut imperdiet odio vitae velit imperdiet, id consectetur lorem rutrum. Interdum et malesuada fames ac ante ipsum primis in faucibus. Quisque risus sapien, molestie ut tortor sed, sollicitudin pulvinar magna. Donec a quam quam. Proin hendrerit purus in sapien consequat tristique. Donec malesuada quis odio non varius. Mauris lobortis, eros eget euismod commodo, magna odio eleifend mi, sed ullamcorper leo massa vitae sem. Nullam eros enim, luctus nec ante quis, varius molestie lorem. Praesent ullamcorper, ante ac faucibus adipiscing, urna urna porttitor arcu, at luctus velit erat vitae elit. Fusce dui quam, vulputate vel cursus non, pulvinar sed ante. Nullam interdum turpis in libero tempor, non sodales leo consectetur. Donec quis eros non quam fermentum varius et ut neque. Nulla vel dolor velit. Aliquam erat volutpat. Sed fringilla quam a sem fringilla lacinia. In condimentum ante nisl, eget egestas urna pretium vitae. Vivamus facilisis feugiat mi, sit amet congue dui aliquam id. Maecenas luctus sed lorem vitae auctor. Maecenas eu magna sed nisl lobortis commodo.

Sed condimentum nisi augue, vitae scelerisque dui congue at. Pellentesque ac sem sed velit faucibus rutrum. Proin fringilla nunc eu dui rutrum, vel luctus nunc dapibus. Nullam mi enim, convallis ac sapien et, suscipit pellentesque lorem. Duis ornare massa nec quam gravida laoreet. Nam ut semper sapien. Aenean quis massa nisi. Praesent quis ligula vel dolor ornare interdum tempus quis urna. Mauris quis justo id lorem porttitor egestas eget ornare arcu. Vestibulum et turpis arcu. Vivamus et placerat mauris. Proin eget eleifend diam, sit amet pharetra leo. Proin auctor vulputate nisl, at placerat massa. Sed tempor mi at varius facilisis.

# Poglavje 5

# Algoritmi integracije operativne transformacije

Phasellus felis diam, molestie in laoreet ac, vestibulum at nunc. Donec et venenatis leo. Nunc elementum molestie est in tincidunt. Integer suscipit, tortor a sodales fringilla, lacus turpis hendrerit elit, tempor pharetra metus sapien ac nisl. Nam vitae neque ipsum. Cras porttitor enim tortor, ut ultricies mi fermentum vitae. Sed at pretium diam. Phasellus eros felis, lacinia et pellentesque sed, feugiat ac ante. Donec sed imperdiet leo. Pellentesque eu luctus lacus. Integer adipiscing lectus a gravida scelerisque. Nam fermentum turpis et posuere semper. Maecenas sit amet sapien lorem. Aenean mattis auctor nibh, nec congue enim varius id. Quisque eget enim eu lectus accumsan vulputate vitae id urna. Donec non felis nec ipsum semper luctus vel eu nisi.

Cras dapibus lacus metus, id tristique odio feugiat sit amet. Etiam erat odio, ornare et eros semper, lacinia ornare metus. Sed vestibulum dolor id leo faucibus, a egestas massa tincidunt. Morbi libero quam, lacinia ac dapibus vel, scelerisque eu libero. Aenean suscipit purus vel diam congue, eget fringilla orci fringilla. Donec purus augue, varius nec tortor vel, lacinia pulvinar libero. Vestibulum at adipiscing lacus. Phasellus at vestibulum lectus.

## 5.1 Pristop GROVE (decentraliziran)

Cras dictum laoreet mi at lacinia. Sed nec volutpat leo. Nulla facilisi. Sed at orci ante. Integer id dui lectus. Maecenas sagittis, libero molestie varius lobortis, urna sapien congue magna, non commodo est turpis quis lacus. Vestibulum molestie tristique

auctor. Etiam eget interdum diam. Donec at velit adipiscing purus tincidunt placerat. Donec luctus nulla sit amet felis ullamcorper, vel suscipit velit hendrerit. Fusce tempus magna dictum, volutpat metus id, commodo nunc. Etiam eu velit sapien. Phasellus accumsan, nunc convallis suscipit viverra, diam leo sagittis dolor, ac porta nulla libero porta tellus. Donec volutpat, massa nec pulvinar consectetur, eros elit vehicula dui, ut vehicula libero leo ut massa. Sed eget tempor augue.

Nullam sollicitudin nisi sed lacus tempus vehicula. Ut posuere tortor sem, aliquet tempor libero vestibulum vel. Nunc luctus, nibh semper bibendum lacinia, neque nisl dapibus purus, id consectetur lorem arcu a mi. Aenean vulputate non mi non auctor. Integer ultricies arcu scelerisque risus pharetra auctor. Nulla vulputate blandit dui, id euismod eros blandit congue. Curabitur porttitor dolor justo, ac gravida est rutrum tristique. Praesent in mattis diam, at aliquet tellus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Integer commodo purus non est feugiat tincidunt. Integer nisi dolor, porta eget nisi non, blandit feugiat tellus. Nam ac orci sed sapien cursus aliquet vitae in diam. Aliquam pulvinar libero id nibh scelerisque condimentum nec non elit. Nam tristique dui quis enim pretium fringilla.

## 5.2 Pristop Jupiter (odjemalec-strežnik)

Donec sed eros venenatis, facilisis quam non, aliquam libero. Vivamus in vulputate leo. Integer tellus ipsum, venenatis ut sagittis eu, elementum eu nisi. Curabitur sit amet consectetur sem. Nulla in velit sed arcu ultrices fermentum. Fusce in est at sem interdum sagittis vel ac tellus. Suspendisse eget facilisis eros. Nunc non massa nec nisl hendrerit vestibulum non nec leo. Duis sollicitudin dapibus ante eu euismod. Morbi egestas tempor cursus.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Proin tempor, quam pulvinar sagittis scelerisque, libero erat hendrerit dolor, et lacinia ipsum justo sit amet leo. Vestibulum mauris lectus, vestibulum quis porttitor at, pretium et elit. In dictum purus erat, ut imperdiet velit blandit ut. Sed rutrum diam dictum magna blandit suscipit. Sed a ipsum sed risus gravida suscipit a at purus. Cras molestie enim sed auctor porta.

Donec malesuada, tellus quis gravida malesuada, est nisl dapibus odio, non molestie elit velit non mi. Nulla facilisi. Nullam ut diam ac mi vestibulum tristique eget luctus nisi. In euismod diam quis convallis bibendum. Donec laoreet sem ac lectus porttitor, eu porttitor augue euismod. Ut vulputate condimentum orci, ut tincidunt sem rhoncus

eu. Nulla consequat porta ligula, nec lobortis lectus vehicula in. Aenean ac bibendum justo, lacinia pellentesque nulla. Etiam nec cursus ante.

## 5.3 Pristop Google Wave (izboljšava)

Maecenas consequat adipiscing ligula, a imperdiet ligula tincidunt ut. Sed a tristique quam. Sed quis posuere lorem. In suscipit lacus eu velit pellentesque scelerisque. Etiam pretium in mi vel placerat. Nunc at ultrices lacus, id tincidunt magna. Vivamus ac nibh ac quam vulputate luctus. Pellentesque id eros eu sem facilisis tincidunt. Donec varius odio ut elit porttitor, quis lobortis turpis porttitor. Nunc dui dolor, molestie ac odio quis, porta sollicitudin tellus. Donec commodo metus id sodales suscipit. Sed nec lectus adipiscing, feugiat ante id, sollicitudin neque. Aliquam blandit adipiscing neque, eget feugiat urna viverra sit amet.

Aenean porta, ligula ut feugiat porta, metus felis [2] fermentum sem, quis aliquam lectus erat nec lectus. Maecenas vitae est felis. Nunc sit amet pharetra mauris, semper fermentum lorem. Nam lobortis egestas blandit. Donec mattis quis metus sed accumsan. Pellentesque in pulvinar massa. Donec at nisi nec turpis porta molestie. Vestibulum gravida ac lacus sit amet dignissim. Proin risus ligula, rhoncus ac sapien non, dignissim convallis lorem. Suspendisse vehicula et dolor eu ultrices.

# Poglavje 6

# Urejevalniki v realnem času

Quisque ut velit bibendum, convallis urna vitae, tempor felis. Fusce cursus condimentum nisl, accumsan pharetra lorem iaculis sit amet. In ac lectus est. Nulla facilisi. Vivamus ornare justo id neque interdum, auctor ornare turpis cursus. Vestibulum euismod nisl orci. Curabitur consectetur, nibh vel tristique pretium, enim magna hendrerit tortor, non rutrum risus nibh et urna. Vivamus a ornare libero. Integer vitae tortor tincidunt, placerat mauris ac, molestie risus. Pellentesque ipsum neque, molestie nec malesuada blandit, fringilla cursus sapien. Sed sed risus lacinia, tempus tortor at, cursus metus. In lacinia elit vel est varius, feugiat faucibus libero vulputate. Nam molestie enim ut interdum venenatis. Donec dolor arcu, blandit et tempor nec, cursus a leo. Phasellus id sollicitudin sapien. Aenean placerat adipiscing metus eget rutrum. Proin adipiscing ac dui vel rutrum. Aenean sit amet placerat nibh. Nulla tempus tristique tortor in auctor. Fusce volutpat mattis fermentum. Maecenas accumsan nibh ut nisi consequat, eget blandit augue auctor. Aenean ac elementum mi. Aenean eu mattis ipsum. Vivamus dolor ipsum, aliquet id dapibus mattis, pretium vel enim.

# Poglavje 7

# Poskus implementacije OT

Sed nec luctus elit. Etiam feugiat, ipsum at fringilla iaculis, ligula massa auctor lorem, quis viverra ligula quam ut orci. Ut faucibus consectetur velit vel adipiscing. Aliquam nec metus sapien. Morbi nec lobortis ipsum. Etiam vehicula, libero dapibus pulvinar pretium, orci nunc aliquam leo, id bibendum lacus diam id quam. Etiam at lectus ac mi malesuada ultricies. Cras id venenatis nisl. Mauris ullamcorper felis eget arcu rhoncus ullamcorper. Nullam sollicitudin justo in orci iaculis, eget tristique tellus adipiscing. Sed at bibendum felis, viverra mollis est. Proin convallis felis eget augue condimentum, at placerat sem porttitor. Sed turpis nulla, fringilla sit amet ultrices id, condimentum ut massa. Pellentesque nec augue a mauris commodo suscipit et in ante. Praesent mollis et diam nec fermentum.

```
1  function cache_add () {
2    var self = this;
3
4    self.cache.add(self.user._id + '_testing', self.resource('sl', '
       session_dafaq'));
5
6    self.json({
7      status: 'okay',
8      text: self.resource('sl', 'cache_has_been_successfully_set')
9    }); return;
10 }
```

Exampelus 7.1: Interdum pretium

Pellentesque egestas orci quis dui tincidunt 7.1 io linus 4 fringilla. Vivamus ac libero arcu. Mauris hendrerit massa eros. Morbi augue ligula, viverra sed volutpat at, pharetra ut erat. Integer consectetur tristique diam, vulputate placerat urna fermentum eu. Ut feugiat ligula a quam venenatis gravida. Ut elementum fermentum sapien, at

posuere purus lacinia non. Mauris ante nunc, sodales eu pretium vitae, ultrices ac turpis.

# Poglavje 8

# Sklepne ugotovitve

Nam semper augue at rhoncus laoreet. Nunc non aliquet risus. Curabitur dictum, nisl nec interdum pretium, dui quam vehicula urna, vel porta nunc est rutrum tellus. Aliquam condimentum pharetra sem, sed sodales massa viverra id. Nam aliquam faucibus egestas. Cras viverra non augue sed sagittis. Nam et elit a tortor aliquet laoreet. Pellentesque congue, risus ultricies rhoncus tincidunt, quam libero tempor tellus, sit amet bibendum velit augue eleifend lectus. Integer lorem nulla, placerat varius faucibus eu, cursus tempor purus. Cras at suscipit elit. Nam varius in ipsum eget tristique. Nulla ac nulla rutrum, vulputate lacus ac, dignissim ligula. Mauris imperdiet ut leo a tempus. Aliquam tempor condimentum massa, at varius diam egestas nec. Mauris eu sem at odio sagittis porta. Curabitur quis rhoncus magna, vitae dignissim enim.

# Literatura


[1] The Dojo Fundation. Open Cooperative Web Framework. Dostopno na: http://opencoweb.org/ocwdocs/intro/openg.html

[2] J. Day-Richter. What's different about the new Google Docs. Dostopno na: http://googledocs.blogspot.com/2010/09/whats-different-about-new-google-docs_22.html

[3] Neil Fraser. Differential Synchronization. Dostopno na: https://neil.fraser.name/writing/sync/

[4] G. Oster, P. Urso, P. Molli, A. Imine. Real time group editors without Operational transformation. Dostopno na: http://hal.inria.fr/docs/00/07/12/40/PDF/RR-5580.pdf

[5] C. Sun. Operational Transformation Frequently Asked Questions and Answers. Dostopno na: http://cooffice.ntu.edu.sg/otfaq/

[6] C.A. Ellis, S.J. Gibbs. "Concurrency Control in Groupware Systems". Dostopna na: http://www-ihm.lri.fr/ mbl/ENS/CSCW/2012/papers/Ellis-SIGMOD89.pdf

[7] D. A. Nichols, P. Curtis, M. Dixon, J. Lamping. "High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System". Dostopno na: http://lively-kernel.org/repository/webwerkstatt/!svn/bc/15693/projects/Collaboration/paper/

[8] D. Wang, A. Mah, S. Lassen. Google Wave Operational Transformation. Dostopno na: http://www.waveprotocol.org/whitepapers/operational-transform