Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Učebné prostredie pre online kolektívny vývoj agilných programov za pomoci zdieľateľného editoru

Bakalárska práca

2018
Emanuel Tesař

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Učebné prostredie pre online kolektívny vývoj agilných programov za pomoci zdieľateľného editoru

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. František Gyarfaš, CSc.

Bratislava, 2018
Emanuel Tesař

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Emanuel Tesař
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Učebné prostredie pre online kolektívny vývoj agilných programov za pomoci zdieľateľného editoru
Learning environment for online collaborative programming using sharable editor

Anotácia: Cieľom bakalárskej práce je vytvorenie web online editora pre kolektívne riešenie agilných úloh z programovania. Editor s kódom a testmi zdieľajú všetci účastníci skupiny, všetci môžu upravovať kód, pridávať testy, vyberať testy na zbiehanie a zbiehať ich vo virtuálnom prostredí na serveri. Aplikácia umožní pripojeným účastníkom zobrazenie kompilačných chýb, zobrazenie zbehnutých testov ako aj verzionovanie zdrojových súborov na serveri. Program umožní tvorbu skupín a ohodnocovanie úloh učiteľom. Súčasťou bude aj administratívne rozhranie pre zadávateľa úloh pre prípravu zadaní, viditeľné aj skryté testy, podľa ktorých sa automatizovane hodnotia výsledky. Technológie/nástroje: HTML 5, CSS, JavaScript (React), serverový framework (Express), Psql, virtuálny server.

Vedúci: Ing. František Gyarfaš, CSc.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 26.10.2018

Dátum schválenia: 06.11.2018 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: TODO: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstrakt

Cieľom práce bol návrh a funkčná implementácia zdieľateľného editora podporujúca súbežné písanie kódu viacerých používateľov. Používatelia okrem písania kódu, možu pridávať testy, ktoré sa dajú spustiť vo virtuálnom prostredí na serveri. V prípade neskompilovateľného kódu, chyby počas behu programu, prípadne inej chyby, zobrazí chybovú hlášku. Okrem pohodlného prostredia pre používateľa obsahuje aj administrátorské rozhranie, v ktorom sa dajú pridávať neviditeľné testy pre používateľov. Toto rozhranie umožňuje profesorom vytvárať zadania pre študentov, ktorý následne možu úlohu riešit priamo vo webovom prehliadači.

Kľúčové slová: jedno, druhé, tretie (prípadne štvrté, piate)

Abstract

TODO: Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

- Latencia - reakčný čas označújúci dobu akciu a následnou reakciou. Vo webových technológiách sa pod letenciou rozumie čas, medzi poslaním webového request-u a následnej odpovede zo servera.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Kapitola 1

Základné pojmy a definície

Kapitola 2

Editor umožňujúci konkurentné úpravy jedného dokumentu

Myšlienka kolaboratývneho editora bola prvýkrát zaznamenaná už v roku 1968 Douglasom Engelbartom. Avšak do popularity sa dostala až nedávno, približne 20 rokov od prvého záznamu. Kolaboratívny editor umožnuje viacerým používateľom upravovať jeden dokument. Tieto editory sa rozdeľujú na dve kategórie

- v reálnom čase - zmeny dokumentu sa okamžite zobrazia všetkým používateľom
- s oneskorením - zmeny dokumentu sa nedejú okamžite (podobne ako pri verzionovacých systémoch ako Git, Mercurial)

My sa v práci zameriavame editormi so zmenami v reálnom čase, kde treba riešiť synchronizáciu editorových inštancií používateľov a riešenie možných konfliktov.

Problematika kolaboratívnych editorov v reálnom čase sa dá rozdeliť do samostatných zmysluplných podkapitol

- technické výzvy
- algoritmy riešiace konkurentné modifikácie jedného zdroja

2.1 Technické výzvy

Technické výzvy pramenia z asynchrónnej komunikácie po sieti. Teoreticky, keby táto komunikácia bola okamžitá, tak vytvorenie takéhoto editora, by nebolo veľmi odlišné od editora pre jedného používateľa. Algoritmus , rišiaci takýto problém, by mohol fungovať na základe *upravovacieho zámku*. Fungoval by celkom jednoducho:

1. Požiadanie servera o *upravovacý zámok*
2. Počkanie na schválenie zo servera, že sme na rade s úpravou

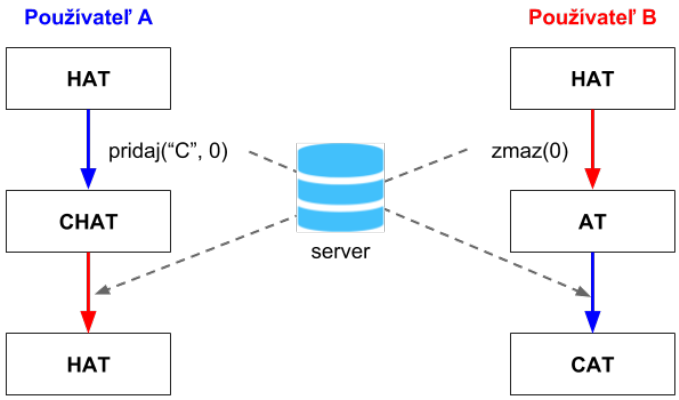
3. Úprava dokumentu

4. Vzdanie sa *upravovacieho zámku*

Avšak rýchlosť komunikácie je obmedzená latenciou siete. To vytvára základnú dilemu: používatelia potrebujú okamžite vidieť vlastné úpravy, ktoré sú do dokumentu zapracované, ale ak sú začlenené okamžite, tak pre latenciu komunikácie musia byť ich úpravy nevyhnutne vložené do rôznych verzií dokumentu.

Jednou s možností by mohlo byť zamedzenie konkurentných úprav zdroja (algoritmus by bol podobný 2.1). Ďalšou, pre používateľa príjemnejšou voľbou je optmistická replikácia, kde sa zmeny v zdroji uplatňujú postupne a prípadné chyby a nekonzistentnosti sa riešia neskôr. Neskôr v texte ukážeme, že existuje spôsob akým docieliť konkurentné úpravy textového dokumentu zaručene bez konfliktov.

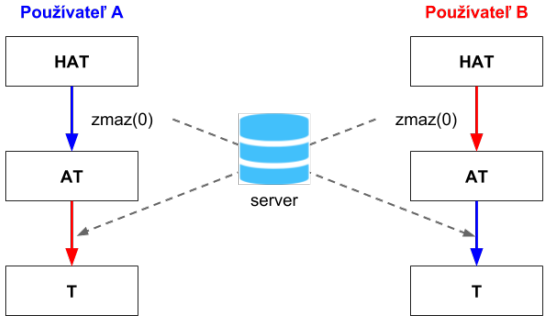
Problém súbežnej modifikácie jedného textového poľa je, že jednoduché textové operácie ako pridať alebo zmazať znak, nie sú komutatívne 2.1 a ani idempotentné 2.2. Keďže používatelia modifikujú dokument cez sieť, nie je zaručené v akom poradí sa modifikácie uskutočnia. [?] Ilustrujme tieto problémy na príklade:



Obr. 2.1: Nekomutativita textových operácii

Výzvou v spolupráci v reálnom čase je teda presne zistiť, ako možno aplikovať úpravy od vzdialených používateľov, ktoré boli pôvodne vytvorené vo verziách dokumentu, nikdy lokálne neexistovali a môžu byť v rozpore s vlastnými miestnymi úpravami používateľa.

Najsofistikovanejšie riešenia vyriešia tento problém spôsobom, ktorý nevyžaduje server, nepoužíva uzamknutie (všetci používatelia môžu voľne upravovať všetky časti dokumentu súčasne) a podporuje ľubovoľný počet používateľov (obmedzený iba zdrojmi počítačov). *UNA* a *SubEthaEdit* sú príklady dvoch programov, ktoré využívajú tento prístup. Tieto programy sú však dostupné iba pre operačných systémoch macOS a využívajú technológie, ako napríklad [1],ktoré sú špecifické pre tento operačný systém.



Obr. 2.2: Neidempotentnosť textových operácii

Zatiaľ čo tieto sofistikované prístupy umožňujú najlepšiu používateľskú skúsenosť, v klientskom serveri môže byť vytvorený aj základný editor pre spoluprácu. Pri klient-server scenári je pri otvorení dokumentu priradená jednej z inštancií editora úloha servera. Tento server zaisťuje, že ostatné editory sú synchronizované. Server obdrží upozornenia na zmeny vykonané v dokumente inými používateľmi. Určuje, ako majú tieto zmeny ovplyvňovať svoju lokálnu kópiu, a vysiela jej zmeny ostatným klientom. V niektorých modeloch sa zmeny na klientovi neodzrkadľujú dovtedy, kým sa zo servera nevráti oficiálna odpoveď, a to aj vtedy, ak boli tieto zmeny vykonané lokálne. Príkladom takéhoto editora je napríklad *Gobby*.

My sme sa v práci rozhodli použiť klient-server model, pričom za synchronizáciu klientov je zodpovedný výhradne server. Podobný prístup používa napr. spoločnost Google v produkte Google dokumenty.

2.2 Algoritmy riešiace konkurentné modifikácie

Na riešenie synchronizácie klientov existujú dva dobre preskúmané typy algoritmov.

1. OT - Prevádzková transformácia (Operational transformation)
2. CRDT - Bezkonfliktné replikovateľné dátové typy (Conflict-free replicated data type)

V práci použijeme CRDT, pretože OT je predchodca CRDT a v praxi často nefunguje tak dobre, ako to autori zamýšľali. Taktiež použitie OT je komplikované a neškálovateľné [8].

Kapitola 3

Bezkonfliktné replikovateľné dátové typy (CRDT)

Súbežné úpravy inštancie jedného zdroja bez koordinácie medzi jednotlivými hostiteľskými počítačmi môže viesť k nekonzistentosti medzi zdrojmi, ktoré vo všeobecnosti nie je možné vyriešiť. Riešenie konzistentnosti v takýchto prípadoch vyžaduje vrátenie niektorch operácii.

V distribuovanom výpočte je konfliktný replikovaný dátový typ (CRDT) dátová štruktúra, ktorá môže byť replikovaná vo viacerých počítačoch v sieti, pričom jednotlivé inštancie je možné aktualizovať nezávisle a súbežne bez koordinácie medzi ostanými inštanciami a kde je vždy matematicky možné vyriešiť nezrovnalosti, ktoré by mohli vyplynúť. Z toho vyplýva, že použitím CRDT nemôže nastať situácia, kde by sme museli nejakú operáciu vrátiť.

Koncept CRDT bol formálne definovaný v roku 2011 osobami Marc Shapiro, Nuno Preguiça, Carlos Baquero a Marek Zawirski [11].

3.1 Typy CRDT

V súčastnosti sú preskúmané 2 typy CRDT:

1. CRDT založené na operáciách
2. CRDT založené na stavoch

3.1.1 CRDT založené na operáciách

CRDT založené na operáciách sú označované ako komutatívne replikované dátové typy alebo *CmRDT*. CmRDT menia zdroj vysielaním iba operácie, ktoré sa potom aplikujú lokálne u všetkých, ktorý zdroj zdieľajú. Operácie musia byť komutatívne, avšak

nemusia byť idempotentné. Je preto nutné zaručiť idempotenciu pomocou vhodnej implementácie komunikácie (správa musí byť vyslaná práve raz). Jednoduchým príkladom, kde je vhodné použiť CmRDT je napríklad aktualizovanie počtu zdieľaní príspevkov na sociálnych sieťach, kde sa server vie postarať o idempotentnosť.

3.1.2 CRDT založené na stavoch

CRDT založené na stavoch sa nazývajú konvergentné replikované dátové typy alebo *CvRDT*. Na rozdiel od CmRDT posielajú celý svoj stav, kde stavy sa potom zlučujú funkciou, ktorá musí byť komutatívna, asociatívna a idempotentná.

Posielanie celého stavu, nie je vždy optimálne. Častokrát stačí posielať zmeny, ktoré sa na zdroji uskutočňujú. Takéto CRDT označujeme ako *Delta CRDTs*

3.2 Princíp fungovania CRDT

CRDT funguje tak, že každý znak v dokumente sa prerobí do objektu so špecifickými vlastnosťami:

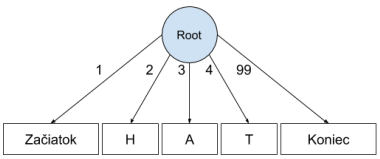
1. znak, ktorý objekt predstavuje
2. relatívna pozícia tohto znaku
3. množina pozícií musí tvoriť úplné usporiadanie
4. priestor tvorený množinou pozícií musí byť hustý

Vzhľadom na to, že každý z týchto objektov je jedinečný a môže byť identifikovaný týmito vlastnosťami, môžeme zabrániť vloženiu alebo vymazaniu znakov viac ako raz. To umožňuje komutativitu a idempotenciu. Nevýhodou tohto prístupu je veľké množstvo metaúdajov. Tým sa zvyšuje spotreba pamäte aplikácie.

Zaujímavým aspektom CRDT, ktorý ho odlišuje od OT, sú relatívne pozície znakov. Tieto pozície majú nasledovné vlastnosti:

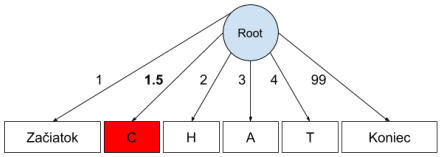
1. žiadne 2 CRDT objekty nemajú rovnakú pozíciu
2. pozícia nejakého objektu sa nidky nezmení
3. ak 2 CRDT objekty A a B operujú na tej istej pozícii v dokumente a objekt A nastane skôr než B, tak relatívna pozícia A musí byť menšia ako B.

Vytvorenie takýchto pozícií je celkom jednduché. Znaky si môžeme predstaviť ako vrcholy na strome, kde každý znak má väčsie číslo ako znak pred ním, no menšie ako znak po ňom. Pozície môžu vyzerať zhruba nasledovne:



Obr. 3.1: Relatívne pozície znakov

Pridanie znaku potom funguje veľmi jednoducho. Nájdeme v strome vrcholy medzi ktoré chceme pridať ďalší znak a ako pozíciu mu dáme priemer relatívnych pozícií daných vrcholov. Napríklad:



Obr. 3.2: Pridanie relatívnej pozície znakov

Komutativita a idempotentnosť CRDT

Ak predpokladáme, že pozície spĺňajú podmienky z 3.2, tak CRDT objekty sú komutatívne a idempotentné.

Keďže všetky relatívne pozície sú unikátne, tak nezáleží na poradí, v ktorom vykonávame dané operácie, teda CRDT sú naozaj komutatívne.

Idempotencia je daná tým, že ak chceme do dokumentu pridať znak s najekou relatívnou pozíciou, a tá sa tam už nachádza, tak opätovné pridanie už nič neurobí. Ak naopak sa snažíme vymazať niečo na pozícii, ktorá sa v dokumente už nenachádza, tak vieme, že uz vymazaná bola. Obe operácie teda zachovávajú idempotentnosť. [10]

3.3 Použitie v praxi

Aplikácie CRDT sa dajú nájsť nielen v oblasti kolaboratívnych editorov, ale na mnohých ďalších miestach. Príkladom sú napríklad databáza *Redis*, *SoundClound* alebo NoSQL dátové úložisko *Riak*, ktoré sa používa na vnútorný chatovací systém v hre *League of Legends*. [2]

Kapitola 4

Funkcia servera

Server má viacero funkcií:

- synchronizácia klientov
- prihlasovanie pre administrátorov
- ukladanie skrytých testov
- zbiehanie kódu, poskytovanie odpovede klientom

4.0.1 Synchronozácia klientov

Najdôležitejšou funckiou servera je synchronizácia všetkých klientov ak nastane nejaká zmena dokumentu. Server na toto nepotrebuje veľkú logiku, ale potrebuje korektne preposielať informácie medzi klientami a reagovať na prípadne komunikačné chyby (väčšinou odosľat správu znovu).

4.0.2 Prihlasovanie pre administrátorov

Súčasťou práce je aj administrátorské rozhranie, do ktorého sa dá prihlásiť a vytvárať v ňom skryté testy.

4.0.3 Prihlasovanie pre administrátorov

Administrátori vedia pridávať skryté testy, na ktorých sa dá spúštať klientský kód. Toto funguje na jednoduchom princípe skupín. Administrátor vytvorí skupinu, do ktorej sa klienti vedia prihlásiť. Ak je klient súčasťou nejakej skupiny, tak vie svoj kód zbiehať voči testom vytvorených administrátorom.

4.0.4 Zbiehanie kódu, poskytovanie odpovede klientom

Ďalšou dôležitou častou je možnosť zbehnúť kód na serveri. Kód sa dá zbehnúť viacerými spôsobmi:

- na vlastných testoch
- na skrytých testoch (iba v prípadne ak je klient členom nejakej skupiny)
- na vlastnom vstupe

Kapitola 5

Izolované spúštanie kódu na serveri

Kód spustený na serveri, nesmie narušiť ostatné programy, prípadne inak narušiť prostredie servera.

Kapitola 6

Použiteľnosť v praxi

Práca môže byť použitá na účely agilného programovania na univerzite, kde učiteľ ako administrátor vie pripraviť úlohy pre študentov.

Záver

Literatúra

[1] Bonjour (software). https://en.wikipedia.org/wiki/Bonjour_(software).

[2] Conflict-free replicated data type. https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type.

[3] Building conclave: a decentralized, real time, collaborative text editor, January 2018. https://hackernoon.com/building-conclave-a-decentralized-real-time-collaborative-text-editor-a6ab43

[4] X. Autor1 and Y. Autor2. *Názov knihy*. Vydavateľstvo, 1900.

[5] X. Autor1 and Y. Autor2. Názov článku (väčšinou z konferencie). In *Názov zborníka (väčšinou názov konferencie spolu s ročníkom)*, pages 1–100, 1900.

[6] X. Autor1 and Y. Autor2. Názov článku z časopisu. *Názov časopisu, ktorý článok uverejnil*, 4(3):1–100, 1900.

[7] X. Autor1 and Y. Autor2. Názov technickej správy. Technical Report TR123/1999, Inštitút vydávajúci správu, June 1999.

[8] Abdessamad Imine, Michaël Rusinowitch, Gérald Oster, and Pascal Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theor. Comput. Sci.*, 351(2):167–183, February 2006.

[9] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *Nie príliš stručný úvod do systému LaTeX2e*. 2002. Preklad Ján Buša ml. a st.

[10] Nuno Preguica, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A commutative replicated data type for cooperative editing. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 395–403, Washington, DC, USA, 2009. IEEE Computer Society.

[11] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*, SSS'11, pages 386–400, Berlin, Heidelberg, 2011. Springer-Verlag.

[12] Univerzita Komenského v Bratislave. Vnútorný predpis č. 12/2013, smernica rektora Univerzity Komenského v Bratislave o základných náležitostiach záverečných prác, rigoróznych prác a habilitačných prác, kontrole ich originality, uchovávaní a sprístupňovaní na Univerzite Komenského v Bratislave, 2013. https://uniba.sk/fileadmin/ruk/legislativa/2013/Vp_2013_12.pdf.