Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/05BE35Stipic

PROTOKOL JEZIČKIH SERVERA

LANGUAGE SERVER PROTOCOL

Bojan Stipić, Zorica Suvajdžin Rakić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Cilj ovog rada jeste analiziranje protokola jezičkih servera. Prikazana je istorija i motivacija za razvoj protokola jezičkih servera, kao i sami tehnički detalji. Opisan je programski jezik miniC. Implementiran je jezički server za programski jezik miniC. Prikazana je integracija jezičkih servera sa tekst editorima Vim i Atom.*

Ključne reči: *Protokol jezičkih servera, LSP, miniC*

Abstract – *Goal of this thesis is to analyze the Language Server Protocol. The history and motivation for developing Language Server Protocol as well as the technical details themselves are presented. The miniC programming language is described. A language server for the miniC programming language is implemented. The integration of language servers with Vim and Atom text editors is presented.*

Keywords: *Language Server Protocol, LSP, miniC*

UVOD

Osnovni alat svakog programera je tekst editor. Tekst editori poseduju brojne funkcionalnosti koje olakšavaju život programera. Neke od često dostupnih funkcionalnosti su: dijagnostike (*diagnostics*), dovršavanje reči (*word completion*), isticanje sintakse (*syntax highlighting*), skok na definiciju (*go to definition*), pronalaženje referenci (*find references*)...

Različiti korisnici imaju različite potrebe pri korišćenju tekst editora, pa stoga i potreba za velikim brojem istih. Takođe, pored velikog broja tekst editora, postoji i veliki broj različitih programskih jezika. Implementacija gore navedenih funkcionalnosti zavisi od programskog jezika koji se koristi. Prema tome, svaki tekst editor je tradicionalno morao da implementira sve funkcionalnosti za svaki programski jezik posebno.

Bilo bi pogodnije kada bi postojali zasebni jezički alati koji pružaju "pametne" funkcionalnosti za pojedinačne programske jezike. Tada bi tekst editori mogli da komuniciraju sa tim alatima i tako pruže podršku za sve programske jezike.

Upravo to i jeste ideja Protokola jezičkih servera (Language server protocol) [1] [2] [skraćeno LSP]. LSP ima za cilj da definiše standardizovan način komunikacije između jezičkih alata (servera) i samih tekst editora.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Zorica Suvajdžin Rakić.

Na ovaj način problem `m*n` kompleksnosti svodi se na `m+n` kompleksnost.

1. PROTOKOL JEZIČKIH SERVERA

1.1 Istorija i motivacija

LSP je kreiran od strane Microsoft-a kako bi se definisao zajednički jezik kojim jezički alati komuniciraju. Nastao je 2016. godine, a pored Microsofta su se udružile i druge kompanije, kao što su Red Hat, Codenvy i Sourcegraph, kako bi podržali njegov rast.

Inicijalno, Microsoft je razvio OmniSharp projekat sa ciljem da pruži "pametne" funkcionalnosti za programski jezik C# u tekst editorima. OmniSharp je inicijalno koristio HTTP protokol za razmenu JSON poruka. Ubrzo je integrisan u nekoliko tekst editora, a jedan od njih bio je VS Code, koji je takođe razvijen od strane Microsoft-a.

U istom vremenskom periodu, Microsoft je počeo sa razvojem još jednog jezičkog servera. Ovaj jezički server je imao za cilj da podrži programski jezik TypeScript. Koristio je standardni ulaz/izlaz kao transportni protokol, a poruke su bile u JSON formatu po uzoru na V8 debugger protokol.

Kada je tim zadužen za razvoj VS Code editora integrisao i TypeScript jezički server, uvideli su da je konzumacija jezičkih servera suvišno komplikovana. Kako bi pojednostavili integraciju, počeli su da razmišljaju o opštem zajedničkom protokolu [3]. Zajednički protokol omogućio bi da se integracioni kod napiše samo jednom, i da se koristi za sve jezičke servere koji pričaju zajednički jezik.

Zajednički protokol je kao osnovu koristio protokol TypeScript jezičkog servera, koji je uopšten i napravljen da bude jezički neutralan. U početku, protokol je podržavao poruke dijagnostike, a kasnije je obogaćen ostalim "pametnim" funkcionalnostima koje jedan server može da pruži.

Danas je LSP podržan od strane velikog broja tekst editora i jezičkih zajednica. LSP je otvoren protokol i kompletna specifikacija je slobodno dostupna na GitHub-u[https://github.com/Microsoft/language-server-protocol].

Razvojem standardizovanog protokola, postalo je dovoljno da jezička zajednica napravi jedan jezički server koji pruža sve "pametne" funkcionalnosti za dati programski jezik. Svi editori koji razumeju jezički protokol tada automatski dobijaju podršku za taj programski jezik.

1.2 Tehnički detalji

Tekst editor se ponaša kao klijent koji komunicira sa jezičkim serverom. Komunikacija se odvija slanjem poruka.

U osnovi, poruka se sastoji iz **zaglavlja** (*header*) i **sadržaja** (*content*), slično kao i HTTP protokol. Zaglavlje i sadržaj su razdvojeni `\r\n` karakterima.

Specifikacijom nije određen transportni protokol za razmenu poruka.

U praksi klijent i server najčešće komuniciraju preko standardnog ulaza/izlaza, ali moguća je i komunikacija preko mrežnih soketa ili bilo kojeg drugog vida međuprocesne komunikacije.

1.2.1. Zaglavlje poruke

Zaglavlje se sastoji iz polja. Svako polje je par ključ-vrednost, razdvojeni sa **:** (dvotačka praćena razmakom). Svako polje se završava karakterima `\r\n`. Podržana polja u zaglavlju su `Content-Length` i `Content-Type`.

1.2.2. Sadržaj poruke

Sadržaj poruke koji se razmenjuje između klijenta i servera je baziran na JSON-RPC protokolu. JSON-RPC je protokol koji omogućava udaljen poziv procedura (*remote procedure call*) razmenom poruka u JSON formatu.

Sve poruke se dele na tri varijante:

1. **Zahtev** (*request*)—upućen zahtev između klijenta i servera. Svaki zahtev mora imati odgovor.
2. **Odgovor** (*response*)—odgovor na upućen zahtev.
3. **Obaveštenje** (*notification*)—slično zahtevu, ali ne zahteva odgovor.

1.2.3. Poruke za sinhronizaciju teksta

Tekst editor obaveštava jezički server o događajima, odnosno interakcijama koje korisnik vrši nad dokumentima.

Neki od mogućih događaja su:

- `didOpen`—Korisnik je otvorio datoteku.
- `didChange`—Korisnik je napravio izmene u datoteci.
- `didSave`—Korisnik je sačuvao datoteku.
- `didClose`—Korisnik je zatvorio datoteku.

Jezički server treba da "sluša" događaje i interno vodi evidenciju o svim otvorenim datotekama i njihovim sadržajem.

Kada se dogodi određeni događaj, klijent u poruci ne šalje samo URI dokumenta već njegov kompletan sadržaj u vidu stringa. Ovo je neophodno kako bi sve funkcionalnosti radile bez potrebe da korisnik prethodno sačuva datoteku na medijum za trajno čuvanje podataka.

1.2.4. Poruke za objavljivanje dijagnostike

Poruke dijagnostike imaju za cilj da korisniku prikažu greške, upozorenja ili informacije u toku samog unosa teksta. Poruke dijagnostike ne traži klijent od servera, već ih server sam isporučuje kada je to neophodno. Iz tog razloga ovakve poruke su implementirane kao obaveštenja, a ne kao par zahtev–odgovor.

2. PROGRAMSKI JEZIK MINIC

Programski jezik miniC je nastao na Fakultetu tehničkih nauka, Univerziteta u Novom Sadu, za potrebe kursa Programski prevodioci.
miniC je striktan podskup programskog jezika C. Dakle, miniC programe je moguće kompajlirati regularnim C kompajlerom, dok obrnuto ne važi. Nastao je odabirom osobina i koncepata C jezika koji su interesantni za implementaciju kompajlera. Međutim, ove osobine su uzete u određenoj meri kako bi se olakšala implementacija kompajlera. Autori su se uzdržali od mnogih karakteristika C jezika koji nepotrebno komplikuju implementaciju kompajlera, a koji, u edukativnom smislu, ne doprinose značajno.

3. OPIS SOFTVERSKOG REŠENJA

U ovom poglavlju je opisan način implementacije jezičkog servera za programski jezik miniC. Takođe, dat je opis alata i biblioteka koje su korišćene prilikom razvoja opisanog softverskog rešenja.

3.1 Opis korišćenih alata i biblioteka

3.1.1. Flex

Flex [6] [7] je program koji generiše leksički analizator, odnosno skener. Flex čita ulazna pravila i kao izlaz vraća kod koji implementira lekser u programskom jeziku C. Nastao je 1987. godine kao slobodna verzija programa Lex. Dostupan je pod modifikovanom BSD licencom, a njegov originalni autor je Vern Paxson.
Zadatak leksičkog analizatora je skeniranje sekvence tokena u ulaznom tekstu i pronalaženje određenih šablona. Kada se šablon prepozna, izvršava se zadata akcija. Šabloni se zadaju u vidu regularnih izraza. Flex koristi sebi specifičnu sintaksu za zadavanje regularnih izraza, ali je ona u velikoj meri slična uobičajenoj ERE sintaksi koja je definisana u IEEE POSIX standardu.

3.1.2. Bison

Bison [6] [8] je program koji generiše sintaksni analizator, odnosno parser. Bison čita ulazna pravila i kao izlaz vraća kod koji implementira parser u programskom jeziku C. Nastao je 1985. godine kao slobodna verzija programa Yacc. Dostupan je kao deo GNU projekta, pod GNU opštom javnom licencom, verzija 3.
Sintaksnom analizom se odgovara na pitanje "Da li je raspored tokena u skladu sa gramatikom?". Gramatika se zadaje u BNF notaciji (Bakus-Naurova forma).
Bison se često koristi u kombinaciji sa Flex-om za izradu kompajlera. Leksičkom analizom se prepoznaju šabloni u tekstu i tekst se deli na tokene, a zatim se sintaksnom analizom proverava da li se tokeni nalaze u ispravnom redosledu.

3.1.3. cJSON biblioteka za parsiranje JSON-a

cJSON [9] je biblioteka napisana u programskom jeziku C namenjana parsiranju JSON-a. S obzirom da LSP definiše razmenu poruka između klijenta i servera preko JSON

objekata, cJSON je ključna biblioteka za implementaciju jezičkog servera opisanog u nastavku. cJSON je nastao 2009. godine, a originalni autor biblioteke je Dave Gamble. Biblioteka je dostupna pod MIT Expat licencom.

3.2. Opis implementacije jezičkog servera minic-lsp za programski jezik miniC

S obzirom da jezički server mora poznavati gramatiku jezika kao i sam kompajler, za implementaciju minic-lsp jezičkog servera uzeta je kao osnova postojeći kompajler miniC jezika pod imenom MICKO. Kompajler MICKO je nastao na Fakultetu tehničkih nauka za potrebe edukacije na predmetu Programski prevodioci.

Prvu dodatnu stvar koju treba implementirati kako bi jezički server funkcionisao je sama obrada poruka i slanje odgovora. Pošto sam LSP ne definiše transportni protokol za razmenu poruka, potrebno je odlučiti koji pristup će se koristiti. Zbog jednostavnosti implementacije izabrana je komunikacija preko standardnog ulaza/izlaza.

Po startovanju programa, poziva se `lsp_event_loop` funkcija koja startuje beskonačnu petlju za obrađivanje poruka.

```
for(;;) {
        unsigned long content_length =
                lsp_parse_header();
        cJSON *request =
                lsp_parse_content(content_length);
        json_rpc(request);
        cJSON_Delete(request);
}
```

Funkcija `lsp_parse_header` preuzima zaglavlje poruke i parsira dužinu sadržaja koji sleduje. Funkcija `lsp_parse_content` preuzima sadržaj specificirane dužine `content_length` i kao rezultat vraća parsiran JSON zahtev/obaveštenje. Funkcija `json_rpc` parsira udaljeni poziv procedure po JSON-RPC protokolu i poziva odgovarajuću funkciju. Na kraju je potrebno osloboditi zauzetu memoriju pozivom `cJSON_Delete` funkcije.

4. INTEGRACIJA SERVERA SA TEKST EDITORIMA

Samim tim što je implementiran jezički server za miniC programski jezik, svi editori teksta koji poznaju zajednički LSP jezik automatski dobijaju podršku za miniC.

Da bi korisnik počeo da koristi jezički server, potrebno je da uradi sledeće:

1. Jezički server treba instalirati na neku od putanja u `$PATH` varijabli.
2. U podešavanjima tekst editora treba specificirati koji jezički serveri se koriste za koje programske jezike. Editoru je potreban samo naziv izvršne datoteke jezičkog servera, kako bi znao koji program da startuje. Nikakve dodatne informacije editoru nisu potrebne, jer svi serveri pričaju zajednički LSP jezik.

Da korisnik ne bi morao ručno da podešava tekst editor da koristi minic-lsp, u sklopu ovog rada napravljena je konfiguracija u vidu plugin-a za tekst editore Vim i Atom.

Vim [10] je konfigurabilni tekst editor namenjen efikasnom kreiranju i modifikovanju tekst dokumenata. Nastao je 1991. godine, a originalni autor je Bram Moolenaar. Vim je slobodni softver dostupan pod licencom koja ohrabruje korisnike da doniraju novac deci u Ugandi.

U vreme pisanja ovog rada, poslednja dostupna verzija Vim-a je 8.1 i ne postoji ugrađena podrška za LSP. Ipak, pošto je Vim proširiv kroz plugin-e, postoje razna proširenja pomoću kojih Vim postaje LSP klijent. Plugin koji je izabran za ovu demonstraciju se zove ALE (Asynchronous Lint Engine) [11]. ALE je takođe slobodni softver dostupan pod BSD licencom sa dve klauzule.

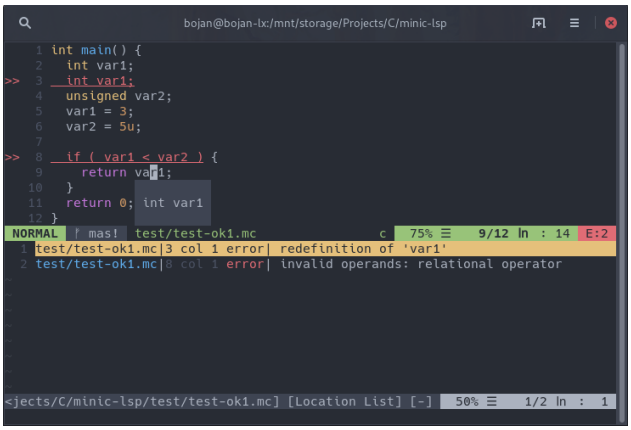
Atom [12] je "hakabilni tekst editor 21. veka" napravljen od strane Github kompanije 2014. godine. Atom je slobodni softver dostupan pod MIT Expat licencom. Baziran je na Electron platformi za razvijanje aplikacija.

U vreme pisanja ovog rada, poslednja dostupna verzija Atom-a je 1.40 i ne postoji ugrađena podrška za LSP. Međutim, Atom je takođe proširiv kroz plugin-e i dostupno je proširenje pomoću kojeg Atom postaje LSP klijent. Plugin koji Atom pretvara u LSP klijent se zove atom-ide [13] i razvijen je od strane Facebook kompanije. Dostupan je kao slobodni softver pod BSD licencom sa tri klauzule.

5. DEMONSTRACIJA RADA JEZIČKOG SERVERA MINIC-LSP

Slika 1 prikazuje Vim tekst editor sa otvorenom test datotekom. Linije 3 i 8 su podvučene crvenom linijom, a sa leve strane se vide znakovi >>, što označava grešku u navedenim linijama. U donjoj polovini editora vidljiva je lista svih dijagnostika za otvorenu datoteku.

Takođe, može se videti da je korisnik postavio kursor na liniju 9, na slovo **r** u identifikatoru `var1`, a zatim je zatražio informacije u vidu "lebdeće" poruke. Na osnovu informacija koje su vidljive, korisnik može saznati da je `var1` tipa `int`. U slučaju da identifikator na kojem se nalazi kursor predstavlja naziv funkcije, vidljivi su tipovi parametara koje funkcija prihvata kao i tip povratne vrednosti funkcije.



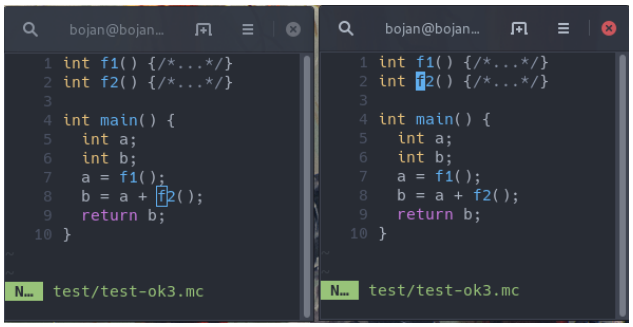
SLIKA 1. DEMONSTRACIJA RADA DIJAGNOSTIKA

Slika 2 prikazuje Vim tekst editor sa otvorenom test datotekom koja sadrži dve funkcije. U funkciji `main` su definisane dve lokalne promenljive. Korisnik u liniji 6 unosi karaktere `lang`, nakon čega tekst editor automatski otvara padajući meni sa dostupnim opcijama koje se mogu dopuniti.

U padajućem meniju se pored teksta dopune vidi i kako izgleda deklaracija simbola. Na osnovu deklaracije korisnik može saznati kojeg tipa su promenljive, koje parametre prihvata funkcija, kojeg tipa je povratna vrednost funkcije…

SLIKA 2. DEMONSTRACIJA RADA DOVRŠAVANJA

Slika 3 prikazuje funkcionalnost skoka na definiciju. Sa leve strane je prikazana inicijalna pozicija kursora. Korisnik je pozicionirao kursor na poziv `f2` funkcije. Korisnik inicira komandu skoka na definiciju i kursor se prebacuje na poziciju gde je funkcija definisana, kao što se može videti na desnoj polovini slike.



SLIKA 3. DEMONSTRACIJA RADA SKOKA NA

6. ZAKLJUČAK

Korisnici danas imaju visoka očekivanja od tekst editora. Iz tog razloga, tekst editori su veoma kompleksni programi, sa velikim brojem mogućnosti. Pritom, većina funkcionalnosti je usko povezana sa programskim jezikom u kojem je dokument napisan.

Kako programskih jezika vremenom ima sve više, tako je i razvoj tekst editora postao sve kompleksniji. Rešenje ovog problema je razvoj posebnih jezičkih servera koji pružaju "pametne" funkcionalnosti za pojedinačne programske jezike. Tekst editor više ne mora da poznaje sintaksu ili semantiku jezika, već samo treba da se integriše sa jezičkim serverom.

Međutim, razdvajanjem tekst editora od jezičkih servera, rešen je jedan problem, ali je nastao novi problem—njihova integracija. Jezički serveri su koristili različite formate poruka za komunikaciju. Tekst editori su morali na različite načine parsirati poruke koje dobijaju od jezičkih servera, a morali su poznavati i različite poruke za zadavanje upita. Kako bi se problem integracije rešio, Microsoft je razvio protokol jezičkih servera. Razvojem zajedničkog jezika kojim jezički serveri komuniciraju, postalo je moguće da tekst editori napišu integracioni kod samo jednom i da ga koriste za sve postojeće i buduće programske jezike.

Razvojem LSP-a, razlika između običnih tekst editora i integrisanih razvojnih okruženja postaje sve manja. Tekst editori podržavaju većinu funkcionalnosti i jezički su neutralni, pa nestaje potreba za specijalizovanim integrisanim razvojnim okruženjima. Korisnici mogu da koriste tekst editor sa kojim su već upoznati za sve projekte, bez obzira na programski jezik koji se koristi.

Pisanje kompletnog softverskog rešenja u okviru ovog rada je olakšano uz pomoć LSP-a. Korišćen je Vim tekst editor i Clangd jezički server za programski jezik C.

6. LITERATURA

[1] Official page for Language Server Protocol, https://microsoft.github.io/language-server-protocol/, Avgust 2019.

[2] A community-driven source of knowledge for Language Server Protocol implementations, https://langserver.org/, Avgust 2019.

[3] Language server protocol history, https://github.com/Microsoft/language-server-protocol/wiki/Protocol-History, Avgust 2019.

[4] Zorica Suvajdžin Rakić i Miroslav Hajduković, miniC, specifikacija i implementacija, zbirka zadataka, Univerzitet u Novom Sadu, FTN, Edicija tehničke nauke - udžbenici, broj ???, FTN Izdavaštvo, Novi Sad, 2014

[5] Zorica Suvajdžin Rakić, Predrag Rakić, Tara Petrić, miniC Project for Teaching Compilers Course, ICIST 2014

[6] Zorica Suvajdžin Rakić i Predrag Rakić, flex & bison, zbirka zadataka, Univerzitet u Novom Sadu, FTN, Edicija tehničke nauke - udžbenici, broj 478, FTN Izdavaštvo, Novi Sad, 2014

[7] Flex zvanična dokumentacija, https://westes.github.io/flex/manual/, Avgust 2019.

[8] Bison zvanična dokumentacija, https://www.gnu.org/software/bison/manual/, Avgust 2019.

[9] cJSON repozitorijum, https://github.com/DaveGamble/cJSON, Avgust 2019.

[10] Vim tekst editor, https://www.vim.org/, Avgust 2019.

[11] ALE plugin za Vim tekst editor, https://github.com/dense-analysis/ale, Avgust 2019.

[12] Atom tekst editor, https://atom.io/, Avgust 2019.

[13] Atom-IDE plugin za Atom tekst editor, https://ide.atom.io/, Avgust 2019.

Kratka biografija:

Bojan Stipić rođen je u Novom Sadu 1995. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Računarstvo i informatika odbranio je 2019.god. Kontakt: bojanstipic@uns.ac.rs