Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/16BE44Kolosnjaji

GRAFIČKO OKRUŽENJE ZA POMOĆ PRI OTKRIVANJU IDIOMA U PROGRAMSKOM KODU

A GRAPHICAL ENVIRONMENT FOR ASSISTANCE WITH MINING CODE IDIOMS FROM SOURCE CODE

Marijana Kološnjaji, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Generisanje koda u cilju automatizacije razvoja softvera predstavlja najčešći slučaj primene MDSE pristupa. Jedan od alata za generisanje koda je i RoseLib biblioteka, koja nudi API za generisanje C# koda. U ovom radu je predstavljeno rešenje implementirano sa ciljem proširenja RoseLib biblioteke. Implementirano je grafičko okruženje koje omogućava otkrivanje idioma na osnovu skupa idiomatskih softverskih projekata i njihovu integraciju u RoseLib biblioteku. Integracijom pronađenih idioma, API koji nudi RoseLib biblioteka se proširuje na način da omogućava generisanje pronađenih idioma. Time se omogućava lakše generisanje koda koji se ponavlja kroz softverske projekte, čime se ubrzava razvoj softverskog proizvoda.*

Ključne reči: *MDSE, generisanje koda, idiomi*

Abstract – *Code generation with the goal of software development automation is the most common case of applying the MDSE approach. One of the tools for code generation is the RoseLib library, which offers an API for generating C# code. This paper presents a solution implemented with the goal to extend the RoseLib library. A graphical environment which enables mining code idioms based on a set of idiomatic software projects and their integration in the RoseLib library was implemented. By integrating found idioms, the API offered by the RoseLib library is extended to enable generating found idioms. This makes generation of code that recurs over software projects easier, which speeds up the development of a software product.*

Keywords: *MDSE, code generation, code idioms*

1. UVOD

Softversko inženjerstvo upravljano modelima (*Model Driven Software Engineering* – MDSE) predstavlja metodologiju za primenu prednosti modelovanja na aktivnosti softverskog inženjerstva [1]. Automatizacija razvoja softvera generisanjem koda na osnovu modela predstavlja jednu od najčešćih primena MDSE principa. Upotreba generatora koda u procesu razvoja softverskog proizvoda može znatno uticati na smanjenje vremena razvoja i poboljšanje kvaliteta proizvoda. Međutim, generisanje kompletnog softverskog rešenja često nije izvodljivo zbog kompleksnosti njegove implementacije. Tada postoji potreba za dodavanjem određenih delova koda ručno, od strane programera. Iz tog razloga je potrebna upotreba tehnika za integraciju generisanog i ručno pisanog koda.

Postoje razne tehnike za integraciju generisanog i ručno pisanog koda. Međutim, većina njih zahteva određenu strukturu rešenja. U situacijama kada se MDSE tehnike koriste za unapređenje postojećih softverskih proizvoda, promena strukture rešenja često nije moguća. U tom slučaju je potrebno koristiti alate koji omogućavaju integraciju bez potrebe za uvođenjem promena u okviru strukture postojećeg rešenja.

Jedan od takvih alata je i RoseLib biblioteka za generisanje C# koda. RoseLib biblioteka je osmišljena kao poboljšanje Roslyn-a, alata koji nudi mogućnost generisanja koda manipulacijom sintaksnim stablom izvornog koda. RoseLib nudi API (*Application Programming Interface*) koji omogućava generisanje koda upotrebom osnovnih koncepata C# jezika i time nudi intuitivniji način generisanja koda u odnosu na Roslyn.

U ovom radu je predstavljeno rešenje implementirano u cilju proširenja RoseLib biblioteke. Implementirano je grafičko okruženje koje omogućava integraciju idioma pronađenih na osnovu skupa idiomatskih softverskih projekata u RoseLib biblioteku. Pod integracijom idioma u RoseLib se podrazumeva proširenje API-ja biblioteke kako bi se omogućilo generisanje idioma.

2. TEORIJSKE OSNOVE

U ovom poglavlju su opisane tehnologije i alati na kojima se zasniva rad i čije razumevanje je potrebno za razumevanje implementacije rešenja.

2.2. Roslyn

.NET Compiler Platform SDK [2], poznat i pod nazivom Roslyn, predstavlja set API-ja razvijen od strane Microsoft-a. Omogućava pristup modelu koda koji kompajler gradi u toku validacije sintakse i semantike koda. Na taj način pruža mogućnost direktnog pristupa mnoštvu informacija o izvornom kodu koje su dostupne samo kompajleru. Može se primeniti u razvoju alata za analizu, generisanje i transformaciju koda.

Za kontekst rada su iz seta API-ja Roslyn-a najznačniji API kompajlera i API radnog prostora. API kompajlera

---

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila prof. dr Gordana Milosavljević.

preslikava redosled faza kompajlera. Za svaku fazu kompajlera postoji model objekata koji omogućava pristup informacijama dostupnim u toj fazi. Osnovna struktura podataka koja se koristi iz API-ja kompajlera je sintaksno stablo. Koristi se za analizu strukture izvornog koda, kao i za kreiranje novih i izmenu postojećih delova koda. API radnog prostora spaja informacije o svim projektima u rešenju u jedan model objekata i omogućava im direktan pristup. To doprinosi omogućavanju analize i generisanja koda u okviru čitavih rešenja.

2.3. RoseLib

RoseLib [3] predstavlja biblioteku za generisanje C# koda koja apstrahuje detalje implementacije Roslyn-a. Implementirana je u cilju rešavanja nedostataka Roslyn-a, koji ga čine kompleksnim za upotrebu. Nudi API kojim omogućava generisanje koda upotrebom osnovnih koncepata C# jezika, bez potrebe za fokusiranjem na izgradnju sintaksnih stabala. To čini generisanje koda pomoću RoseLib-a intuitivnijim u poređenju sa generisanjem koda upotrebom Roslyn-a.

RoseLib API se sastoji od dva tipa klasa – selektora i kompozera. Selektori predstavljaju klase koje omogućavaju pretragu sintaksnih stabla. Kompozeri su klase koje omogućavaju izmenu sintaksnih stabla. Organizovani su u vidu hijerarhije koja prati strukturu C# dokumenta.

2.3 Otkrivanje idioma

Idiomi predstavljaju sintaksne fragmente koda koji se ponavljaju kroz softverske projekte i imaju jednoznačnu semantičku ulogu [4]. Mogu sadržati parametre, koji se zovu još i metavarijable. Metavarijable obuhvataju imena identifikatora i sintaksne strukture poput izraza ili blokova koda. Kao jednostavan primer idioma za iteriranje kroz niz u C#-u, može se navesti for petlja. Iako postoji više različitih načina na koji se može iterirati kroz niz u C#-u, for petlja se najčešće koristi od strane programera i zbog toga se smatra idiomom.

RoseLibML [5] predstavlja rešenje za automatsko otkrivanje idioma na osnovu datoteka iz postojećeg skupa softverskih projekata u C# programskom jeziku. Njegova implementacija je zasnovana na Haggis [4] sistemu za automatsko otkrivanje idioma. Korišćene metode su primarno statističke prirode, i kao takve nezavisne od konkretnog jezika. Konkretno, koristi se Markov Chain Monte Carlo (MCMC) metoda za aproksimaciju na osnovu probabilističke kontekstno slobodne gramatike (pCFG) bazirane na dostupnom kodu postojećih softverskih projekata, čime se pronalaze idiomi.

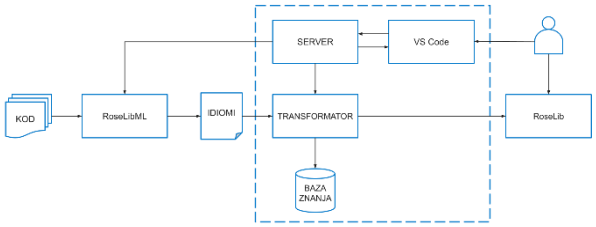
2.4 Language Server Protocol

Language Server Protocol (LSP) [6] definiše standard za komunikaciju između klijenta, koji može biti IDE ili neki od alata za razvoj koda, i jezičkog servera. Kreiran je sa ciljem da olakša obezbeđivanje podrške za različite programske jezike u okviru različitih alata za uređivanje koda.

LSP specifikacija definiše komunikaciju između klijenta i servera preko JSON–RPC protokola. Omogućeno je slanje zahteva, odgovora i obaveštenja. LSP takođe definiše i mogućnost otkazivanja zahteva, kao i mogućnost izveštavanja o napretku i parcijalnim rezultatima u toku obrade zahteva. Definisane su i komande koje omogućavaju izvršavanje proizvoljnih operacija registrovanih od strane servera.

3. IMPLEMENTACIJA

Fokus ovog rada je na implementiranom okruženju za pomoć pri otkrivanju idioma na osnovu skupa softverskih projekata i njihovu integraciju u RoseLib biblioteku. Arhitektura implementiranog rešenja i njegova komunikacija sa ostalim delovima sistema u procesu otkrivanja i integracije idioma su prikazani u okviru dijagrama na slici 3.1. Isprekidanim linijama su obuhvaćeni delovi sistema koji su implementirani u okviru rada.



Slika 3.1 Arhitektura implementiranog rešenja

Rešenje je implementirano u vidu ekstenzije za Visual Studio Code (VS Code) uređivač koda koja preko LSP-a komunicira sa serverom, implementiranim u vidu jezičkog servera. Implementacija servera obuhvata dva glavna zadatka:

1. Otkrivanje idioma na osnovu datoteka iz skupa softverskih projekata
2. Integracija idioma u RoseLib biblioteku

Za otkrivanje idioma, server koristi rešenje implementirano u RoseLibML projektu. Integracija idioma u RoseLib se vrši pomoću posebne komponente koja je nazvana transformator. Za konfiguraciju transformatora se koristi baza znanja.

3.1 Server

Server je implementiran u vidu jezičkog servera, koristeći implementaciju LSP-a napravljenu od strane OmniSharp-a [7]. U najvećoj meri se zasniva na izvršavanju komandi koje su registrovane u okviru servera. Da bi se pokrenulo izvršavanje neke komande, potrebno je sa klijenta poslati *workspace/executeCommand* zahtev iz LSP specifikacije. Kao parametri zahteva se šalju naziv komande i argumenti. Za svaku komandu podržanu od strane servera postoji registrovan rukovalac (eng. *handler*). U toku obrade zahteva, u okviru svakog rukovaoca se prvo vrši validacija argumenata poslatih u zahtevu. Nakon uspešne validacije, nastavlja se sa obradom zahteva i šalje odgovor klijentu.

3.2 Transformator

Transformator je komponenta sistema čiji je primarni zadatak integracija pronađenih idioma u RoseLib biblioteku. Pod integracijom idioma u RoseLib se podrazumeva proširenje biblioteke kako bi omogućila generisanje idioma. To znači da je potrebno proširiti kompozere, komponente RoseLib-a koje omogućavaju generisanje novog koda.

Za svaki idiom pronađen upotrebom RoseLibML projekta, proširuje se odgovarajući kompozer. Informacija o tome koji kompozer se može proširiti nalazi se u bazi znanja. Za proširenje je potrebno u parcijalnu klasu koja predstavlja kompozer generisati novu metodu čija namena je generisanje odabranog idioma. Za generisanje metode koriste se mogućnosti API-ja kompajlera iz Roslyn-a. S obzirom na to da se telo metode razlikuje u zavisnosti od kompozera, za svaki kompozer postoji T4 šablon na osnovu koga se gradi telo metode.

S obzirom na to da je omogućena integracija više idioma u okviru jednog poziva transformera, potrebno ih je najpre grupisati po kompozerima koje proširuju u cilju optimizacije procesa. Ukoliko je datoteka sa parcijalnom klasom koja sadrži odabrane idiome već kreirana, nove metode se integrišu u klasu u toj datoteci. U suprotnom se datoteka sa parcijalnom klasom kreira. Za uključivanje novih datoteka u RoseLib ili integraciju novih metoda u postojeće datoteke, koriste se mogućnosti API-ja radnog prostora iz Roslyn-a.

3.4 Baza znanja

Baza znanja predstavlja komponentu sistema čija uloga je u konfiguraciji transformatora. U implementiranom rešenju, baza znanja je predstavljena u vidu JSON datoteke. U njoj se nalaze podaci o tome u koji kompozer je moguće integrisati koji idiom na osnovu tipa korenskog čvora. Takođe, za svaki kompozer iz RoseLib-a se u okviru baze znanja nalaze pomoćne informacije koje se koriste u toku generisanja koda za njegovo proširivanje. U bazi znanja se takođe nalazi i informacija o putanji na kojoj se nalazi RoseLib biblioteka.

3.5 Klijent

Klijentski deo rešenja je implementiran u vidu ekstenzije za Visual Studio Code uređivač koda. Ekstenzija proširuje korisnički interfejs VS Code-a upotrebom *Webview API*-ja [8]. *Webview API* omogućava kreiranje panela – komponenti koje u okviru postojećeg korisničkog interfejsa mogu prikazati proizvoljne HTML stranice. Za potrebe ekstenzije je kreirano tri panela, od kojih svaki prikazuje jednu fazu u procesu otkrivanja i integracije idioma. Za svaki panel je kreirana HTML stranica u kojoj je definisan izgled, kao i JavaScript datoteka za komunikaciju panela sa ekstenzijom. U svaku stranicu je uključena i CSS datoteka u kojoj je definisan stil izgleda panela. Paneli se prikazuju u obliku kartica u delu za uređivanje koda VS Code alata.

Jedan od zadataka ekstenzije je i komunikacija sa serverom, za koju se koristi *Language Client* modul [9]. Zadužena je za inicijalizaciju i pokretanje servera, koji se dešavaju prilikom aktivacije ekstenzije. Za svaku instancu VS Code-a se pokreće zasebna instanca servera.

Komunikacija ekstenzije sa serverom u toku rada se u najvećoj meri oslanja na slanje zahteva za izvršavanje komandi i obrade odgovora od strane servera. Ekstenzija je takođe zadužena i za zaustavljanje servera, koje se dešava prilikom deaktivacije ekstenzije.

4. PRIKAZ REŠENJA

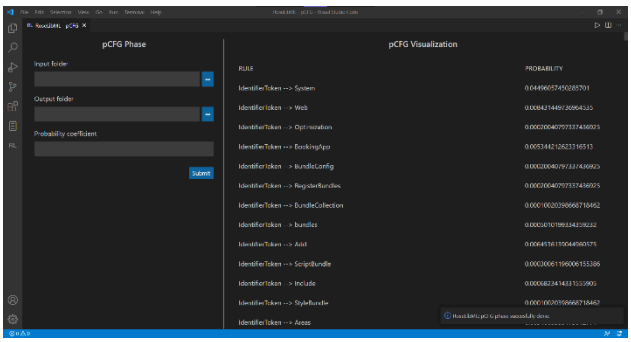
Grafičko okruženje za otkrivanje i integraciju idioma u RoseLib biblioteku, koje je tema rada, implementirano je u vidu ekstenzije za Visual Studio Code uređivač koda. Obuhvata tri glavne funkcionalnosti, koje su ujedno i tri faze u procesu otkrivanja idioma i njihove integracije u RoseLib:

1. Kreiranje pCFG na osnovu skupa datoteka
2. Pokretanje MCMC metode u cilju otkrivanja idioma
3. Podešavanje idioma i njihova integracija u RoseLib biblioteku

Za svaku fazu postoji posebna kartica u okviru VS Code-a, koja se otvara odabirom odgovarajuće opcije iz bočne trake. Predviđeno je da se izvršavaju u redosledu u kome su navedene, s obzirom na to da je izlaz iz jedne faze neophodan za pokretanje sledeće faze.

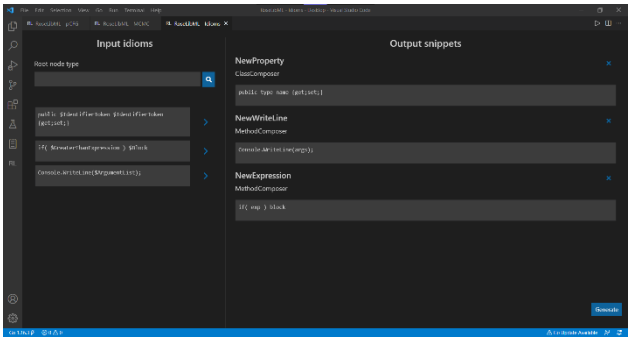
4.1. Upotreba ekstenzije

U okviru kartica za pCFG fazu i MCMC fazu, popunjavaju se forme za unos podataka potrebnih za izvršavanje odgovarajuće komande na jezičkom serveru. U svakoj od njih takođe postoji i mesto za prikaz vizualizacije rezultata. Kao primer se na slici 4.1 može videti prikaz kartice za pCFG fazu.



Slika 4.1 Prikaz izgleda kartice za pCFG

U okviru kartice za podešavanje idioma i njihovu integraciju u RoseLib, prikazuje se lista idioma. Odabirom nekog idioma iz liste, prelazi se na prikaz forme za njegovo podešavanje. U okviru forme se bira jedan od kompozera u koji se idiom može integrisati, unosi se naziv idioma, kao i nazivi za sve metavarijable u okviru idioma. Podešeni idiomi se prikazuju u posebnom delu kartice rezervisanom za njihov prikaz. Prikaz ekstenzije za VS Code sa otvorenom karticom u okviru koje je podešeno tri idioma se može videti na slici 4.2.



Slika 4.2 Prikaz kartice sa podešenim idiomima

Nakon završenog podešavanja, pokreće se opcija generisanja koda koja, na osnovu odabranih idioma i informacija popunjenih u procesu podešavanja, proširuje kompozere iz RoseLib biblioteke.

4.2 Proširenje i upotreba RoseLib biblioteke

Nakon uspešno završene faze integracije, API RoseLib biblioteke je proširen metodama koje omogućavaju generisanje koda sa odabranim idiomima. Za svaki idiom je, u datoteku koja odgovara odabranom kompozeru, dodata metoda čiji zadatak je generisanje tog idioma. Na slici 4.3 se može videti sadržaj generisane datoteke sa parcijalnom klasom za proširivanje *ClassComposer*-a.

```
1  // ------------------------------------------------------------------------------
2  //     This file was generated on 10/13/2021 01:14:28.
3  //
4  //     Changes to this file may cause incorrect behavior
5  //     and will be lost if the code is regenerated.
6  // ------------------------------------------------------------------------------
7  using Microsoft.CodeAnalysis.CSharp;
8  using Microsoft.CodeAnalysis.CSharp.Syntax;
9  using System;
10 using System.Linq;
11
12 namespace RoseLibApp.RoseLib.Composers
13 {
14     public partial class ClassComposer
15     {
16         public ClassComposer AddNewProperty(string type, string name)
17         {
18             if (!IsAtRoot())
19             {
20                 throw new Exception("A class must be selected (which is also a root to
                       the composer)");
21             }
22
23             string fragment = $"public {type} {name} {{get;set;}}";
24             var compilationUnit = CSharpSyntaxTree.ParseText(fragment).GetRoot();
25             var members = (compilationUnit as CompilationUnitSyntax).Members.ToArray();
26             var newNode = (CurrentNode as ClassDeclarationSyntax).AddMembers(members);
27             Replace(CurrentNode, newNode, null);
28             return this;
29         }
30     }
31 }
```

Slika 4.3 Generisana datoteka za *ClassComposer*

Generisana metoda koja se integriše u *ClassComposer* sadrži naziv idioma (*NewProperty*) u svom nazivu i metavarijable idioma (*type* i *name)* kao parametre. U okviru metode se metavarijable iz idioma zamenjuju vrednostima prosleđenim putem argumenata, što se može videti u liniji 23. Zatim se na osnovu idioma pravi sintaksno podstablo koje se dodaje na odabrano mesto u okviru sintaksnog stabla izvornog koda koji se menja.

Na slici 4.4 može se videti primer upotrebe RoseLib biblioteke, nakon integracije idioma, za kreiranje nove C# datoteke. U liniji 17 se vidi upotreba nove generisane metode *AddNewProperty*, čija je definicija prethodno prikazana u kodu na slici 4.3. U ovom primeru se, u liniji 29, takođe koristi i metoda *AddNewWriteLine* koja služi za generisanje još jednog od idioma prethodno prikazanih na slici 4.2.

```
1  CompilationUnitComposer cuComposer = new CompilationUnitComposer();
2  cuComposer.AddUsing("System").AddNamespace("RoseLibTest");
3
4  cuComposer.SelectNamespace();
5  var nsComposer = cuComposer.ToNamespaceComposer();
6
7  nsComposer.AddClass(new ClassOptions()
8      {
9          ClassName = "TestClass",
10         AccessModifier = RoseLib.Enums.AccessModifierTypes.PUBLIC
11     }
12 );
13
14 nsComposer.SelectClassDeclaration("TestClass");
15 var clComposer = nsComposer.ToClassComposer();
16
17 clComposer.AddNewProperty("string", "TestProperty");
18 clComposer.AddMethod(new MethodOptions()
19     {
20         MethodName = "TestMethod",
21         ReturnType = "void",
22         Parameters = new List<RLParameter>()
23     }
24 );
25
26 clComposer.SelectMethodDeclaration("TestMethod");
27 var mComposer = clComposer.ToMethodComposer();
28
29 mComposer.AddNewWriteLine("TestProperty");
```

Slika 4.4 Upotreba RoseLib biblioteke

5. ZAKLJUČAK

U radu je opisano rešenje implementirano u cilju proširenja RoseLib biblioteke. Implementirano rešenje predstavlja grafičko okruženje koje omogućava otkrivanje idioma na osnovu skupa idiomatskih softverskih projekata i njihovu integraciju u RoseLib.

Kao rezultat upotrebe implementiranog okruženja, RoseLib API se proširuje i sadrži nove metode od kojih je svaka namenjena generisanju određenog idioma. Na taj način se korisniku biblioteke pruža mogućnost lakšeg generisanja koda koji se ponavlja kroz softverske projekte. To može znatno da utiče na ubrzanje razvoja softvera upotrebom MDSE principa.

Dalji razvoj okruženja bi se mogao usmeriti na unapređenje interpretacije idioma koji se dobiju upotrebom RoseLibML projekta. Pronađeni idiomi bi se mogli grupisati na osnovu njihove namene, čime bi se umesto proširivanja postojećih, mogli praviti namenski kompozeri. Time bi se smanjla kompleksnost postojećih kompozera koja može biti rezultat integracije velikog broja idioma. Takođe, mogla bi se obezbediti i funkcionalnost automatskog predlaganja naziva idioma i njegovih metavarijabli na osnovu konteksta. Time bi se uklonila potreba za popunjavanjem tih informacija od strane korisnika, čime bi se ubrzao sam proces integracije.

6. LITERATURA

[1] M. Brambilla, J. Cabot and M. Wimmer, *Model-Driven Software Engineering in Practice: Second Edition*. Morgan & Claypool, 2017.

[2] The .NET Compiler Platform SDK, dostupno na: https://docs.microsoft.com/en-us/dotnet/csharp/roslyn-sdk/ (pristupano u oktobru 2021.)

[3] N. Todorović, A. Lukić, B. Zoranović, R. Vaderna, Ž. Vuković and S. Stoja, „RoseLib: A Library for Simplifying .NET Compiler Platform Usage“. in: *ICIST 2018 Proceedings*. 2018, pp. 216–221

[4] M. Allamanis and C. Sutton, „Mining idioms from source code“, in: *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*. 2014, pp. 472–483

[5] N. Todorović and A. Lukić, RoseLibML, dostupno na: https://github.com/lukic-aleksandar/RoseLibML, (pristupano u oktobru 2021.)

[6] Language Server Protocol, dostupno na: https://microsoft.github.io/language-server-protocol/ (pristupano u oktobru 2021.)

[7] OmniSharp Language Server Protocol Implementation, dostupno na: https://github.com/OmniSharp/csharp-language-server-protocol (pristupano u oktobru 2021.)

[8] Webview API, dostupno na: https://code.visualstudio.com/api/extension-guides/webview (pristupano u oktobru 2021.)

[9] VSCode Language Client – Server Module, dostupno na: https://www.npmjs.com/package/vscode-languageclient (pristupano u oktobru 2021.)

Kratka biografija:

Marijana Kološnjaji rođena je u Vrbasu 1996. godine. Osnovne akademske studije na Fakultetu tehničkih nauka u Novom Sadu, smer Računarstvo i automatika, upisala je 2015. godine. Diplomirala je 2019. godine i iste godine je upisala Master akademske studije na Fakultetu tehničkih nauka u Novom Sadu, smer Računarstvo i automatika.