**SVEUČILIŠTE U SPLITU**

**PRIRODOSLOVNO-MATEMATIČKI FAKULTET**

**Igra Križić – Kružić koristeći Akka.NET tehnologiju**

**Student**: Josip Goreta

**Predmet**: Raspodijeljeni sustavi

**Smjer**: Informatika

Sadržaj

[Uvod 1](#_Toc522125557)

[1 Što je Akka.NET i aktori? 2](#_Toc522125558)

[2 Hijerarhija aktor sustava 4](#_Toc522125559)

[3 Opis projekta 6](#_Toc522125560)

[4 Komunikacija među aktorima 8](#_Toc522125561)

[5 Zaključak 10](#_Toc522125562)

[6 Literatura 11](#_Toc522125563)

[Tablica slika 12](#_Toc522125564)

[Tablica primjera koda 12](#_Toc522125565)

# Uvod

Raspodijeljeni sustavi se sastoje od više samostalnih računala koji komuniciraju putem računalne mreže. Računala međusobno djeluju s namjerom da postignu zajednički cilj. Računalni program koji radi u raspodijeljenom sustavu se zove raspodijeljeni program, a raspodijeljeno programiranje je proces pisanja takvih programa.

To računarstvo se još naziva i distribuirano računarstvo i tu se problem dijeli na više zadataka pa se svaki dio zadatka rješava na posebnom računalu. [1]

Kroz ovaj projekt pri izradi igre Križić-Kružić koristit će se Akka.NET framework i njegove biblioteke da bi se postigla razmjena poruka koja odlikuje raspodiljene sustave.

Komunikacija u raspodijeljenom sustavu zasniva se na razmjeni podataka među računalnim procesima koji se izvode na različitim i međusobno udaljenim računalima s različitom sklopovskom i programskom opremom. Kako bi sustav izgledao kao cjelovit tada nam je potreban poseban sloj raspodijeljenog sustava čija je zadaća prikrivanje raspodiljenosti procesa i sredstava na više računala u mreži kao i izvedbene, sklopovske i programske raznorodnosti.

# Što je Akka.NET i aktori?

Akka.NET je skup „open-source“ knjižica za dizajniranje skalabilnih i elastičnih sustava koji obuhvaćaju procesorske jezgre i mreže. Akka omogućuje da se usredotoči na zadovoljenje poslovnih potreba umjesto na pisanje „low-level“ koda kako bi se osiguralo pouzdano ponašanje, tolerancija kvarova i visoka učinkovitost.

ActorSystem je hijerarhijska struktura Aktora koji imaju istu konfiguraciju. Svi aktori žive unutar aktor sustava. [2]

Primjer koda 1Kreiranje aktor sustava

|  |
| --- |
| static class Program  {  /// <summary>  /// The main entry point for the application.  /// </summary>  ///  public static ActorSystem sustav;  [STAThread]  static void Main()  {  sustav = ActorSystem.Create("mojsustav");  Application.EnableVisualStyles();  Application.SetCompatibleTextRenderingDefault(false);  Application.Run(new Form1());  }  } |

Prvi aktor kojeg kreiramo mora biti kreiran direktno iz sustava. U ovom slučaju on se naziva „postar“.

Primjer koda 2 kreiranje prvog aktora

|  |
| --- |
| IActorRef postar;  public Form1()  {  InitializeComponent();  Props prop = Props.Create(() => new FormActor(lista\_botuna)).WithDispatcher("akka.actor.synchronized-dispatcher");  postar = Program.sustav.ActorOf(prop);  } |

S aktorima nikada se ne komunicira direktno nego se to radi pomoću poruke „Tell“ kojoj šaljemo instancu poruke. U varijablu postar ne spremamo referencu na instancu aktora nego IActorRef.

Svaki aktor ima i svoj Mailbox. Tu se spremju njegove poruke koje prima. Taj Mailbox je buffer koji služi za pričuvu tih poruka, a to je važno jer je poruka u akka svijetu jedinica komunikacije.

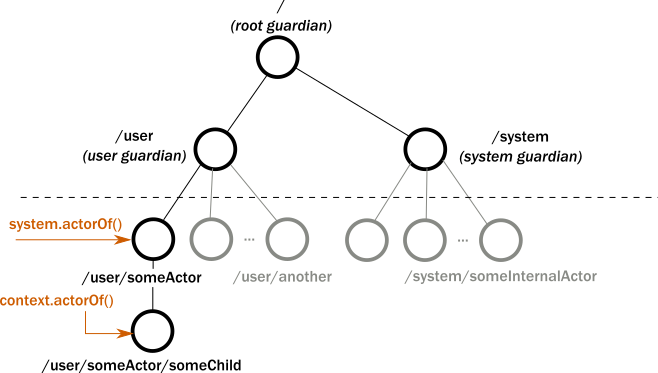
Sa aktorima ne komunicira se direktno nego preko IActorRef-a. On je handle za aktor. Kada se aktoru pošalje poruka, ona ide kroz aktor sustav, a zatim taj isti sustav brine da poruka dođe na odredište. To je prednost jer aktor sustav zapakira poruku skupa sa svim potrebnim metapodacima te ju aktor može prepoznati i raspakirati, ujedno je omogućena i transparentnost lokacije što znači da programer ne mora brinuti na kojem se stroju nalazi aktor već aktor sustav brine o tome.

Props je konfiguracijska klasa koja sadrži sve potrebne informacije da se kreira aktor određenog tipa.

# Hijerarhija aktor sustava

Na samom vrhu hijerarhije postoje tri aktora koji se zovu „Guardians“. Oni su korijeni cijelog aktor sustava. Svaki aktor ima roditelja, a neki aktori uz to imaju i djecu. Roditelji nadziru svoju djecu. Svaki aktor ima nekog tko ga nadzire i svaki aktor može postati netko tko nadzire. Unutar hijerarhije aktor sustava postoje aktori koji su direktno u nadležnosti samo aktor sustava, a postoje i djeca aktori koji su u nadležnosti drugih aktora. [3]

Hijerarhijska struktura aktor sustava izgleda ovako:



Slika 1 Hijerarhijska struktura aktor sustava [3]

Ova slika nam pokazuje kako na vrhu hijerarhije postoje tri aktora koji se nazivaju „Guardians“. Ta tri aktora su korijeni cijelog aktor sustava. Aktor s putanjom „/“ je bazni aktor sustava i možemo ga nazvati „Root Guardian Actor“. Ovaj aktor nadgleda „/user“ i „/system“ aktore. Svi aktori osim njega imaju nekoga tko ih nadgleda.

Aktora s putanjom „/system“ nazivamo „The System Guardian“. Njegov glavni posao je osigurati da se sustav pravilno izgasi. On održava taj sustav i nadzire druge aktore koji implementiraju i pružaju svojstva i pomoćne funkcionalnosti na razini frameworka.

Aktora s putanjom „/user“ nazivamo „User Guardian“. Pod njim se stvaraju svi aktori koje kreira korisnik. Djeca ovog aktora se nazivaju „top level actors“. Svaki puta kada kreiramo aktora direktno iz aktor sustava, stvoreni aktori će postati djeca ovog aktora.

„Top level“ aktore možemo kreirati pomoću sljedećeg koda:

Primjer koda 3 Kreiranje "top level" aktora

|  |
| --- |
| IActorRef aktor1 = MyActorSystem.ActorOf(Props.Create<BasicActor>(), 'aktor1');  IActorRef aktor2 = MyActorSystem.ActorOf(Props.Create<BasicActor>(), 'aktor2'); |

Ako bi sada željeli kreirati aktore koji će biti djeca aktora „aktor2“, onda unutar aktora uz pomoć njegovog Context-a kreiramo nove aktore:

Primjer koda 4 Kreiranje djece "top level"aktora

|  |
| --- |
| IActorRef dijete1 = Context.ActorOf(Props.Create<BasicActor>(), 'dijete1');  IActorRef dijete2 = Context.ActorOf(Props.Create<BasicActor>(), 'dijete2'); |

Svaki aktor ima svoju adresu i svakom aktoru se može poslati poruka preko njegove adrese. Putanja je samo dio te adrese. Primjer potpune adrese:

Primjer koda 5 Primjer potpune adrese

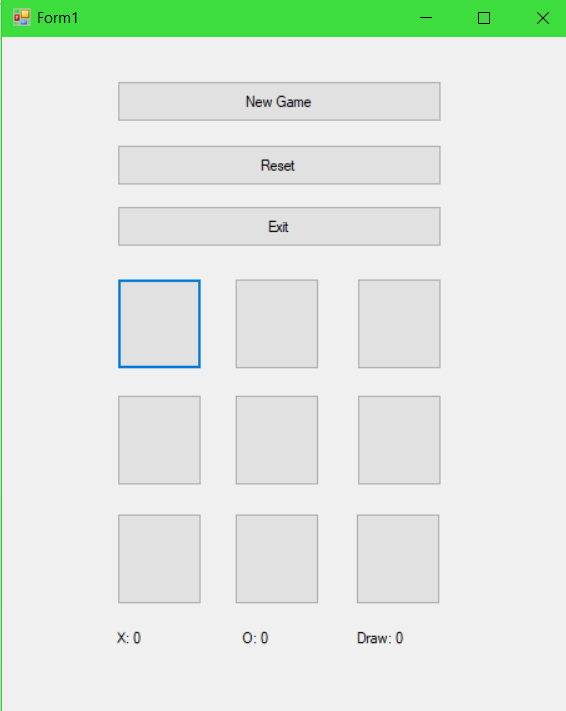
|  |
| --- |
| akka.tcp://MySystem@localhost:9001/user/aktor2/dijete1 |

Opis komponenti adrese:

* **akka.tcp** – određuje koji protokol se koristi u komunikaciji
* **MySystem** – ime aktor sustava koje je pridjeljeno prilikom kreiranja aktor sustava
* **localhost:9001** – adresa računala na mreži skupa s portom na kojem sluša
* /**user/aktor2/dijete1** – putanja koja precizno određuje gdje u hijerarhiji se nalazi traženi aktor.

# Opis projekta

Izrađeni projekt je Windows Form aplikacija napisana u Visual Studiu 2017 u jeziku C#. Simulira popularnu igricu Križić-Kružić, ali u ovom slučaju u pozadini igrice se događa razmjena poruka koja je implementirana pomoću Akka.NET-a. Igra je zamišljena tako da dva igrača moraju biti ispred istog računala da bi je mogli igrati. Prvi igrač koji ima potez će uvijek biti „X“, a drugi igrač je „O“. Sučelje aplikacije kada se pokrene izgleda ovako:

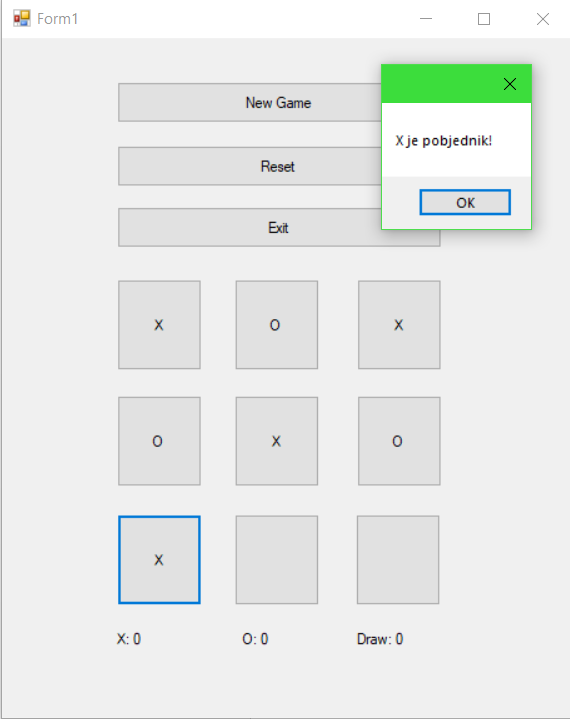


Slika 2 Početno sučelje aplikacije

Osim botuna koji simboliziraju matricu 3x3 koja služi za prikaz igre, imamo i botune „New Game“, „Reset“ i „Exit“. Pritiskom na „New Game“ izbrišu se pritisnuta polja na ploči i igra kreće ispočetka dok pritiskom na botun „Reset“ uz to izbriše se i statistika pobjeda i neriješenih rezultata. Klikom na botun „Exit“ izlazi se iz aplikacije.

Na dnu sučelja ispisuju se rezultati igre. „X“ su pobjede prvog igrača, „O“ su pobjede drugog igrača, „Draw“ su neriješene partije.

Izgled igre nakon završetka partije izgleda ovako:



Slika 3 Prikaz kraja jedne partije

Pritiskom na „OK“ prvom igraču se upiše pobjeda.

# Komunikacija među aktorima

Komunikacija između aktora se događa pomoću poruka koje su opisane u posebnoj klasi. To su dvije vrste poruka, zovu se „Prva“ i „Druga“. Da bi aktori uopće mogli primati poruke potrebno je uz pomoć metode Receive zadati koje poruke aktori mogu primiti. Obje kao parametre primaju jedan string i jedan integer.

Primjer koda 6 Opis poruka koje koriste aktori

|  |
| --- |
| public class Prva  {  public string P\_Text { get; private set; }  public int Indeks;  public Prva(string p\_text, int indeks)  {  P\_Text = p\_text;  Indeks = indeks;  }  }  public class Druga  {  public string D\_Text { get; private set; }  public int Indeks;  public Druga(string d\_text, int indeks)  {  D\_Text = d\_text;  Indeks = indeks;  }  } |

U projektu postoje dva aktora i oba nasljeđuju ReceiveActor-a. Glavni aktor je FormActor koji stvara drugog aktora koji se naziva Worker i koji provjerava tijek igre. Worker tada postaje njegovo dijete.

Primjer koda 7 Opis klase FormActora

|  |
| --- |
| class FormActor : ReceiveActor  {  private List<Button> Lista\_Botuna;  public FormActor(List<Button> lista\_botuna)  {  Lista\_Botuna = lista\_botuna;  Props props = Props.Create(() => new Worker());  IActorRef dijete = Context.ActorOf(props);  Receive<Prva>(x =>  {  dijete.Tell(new Prva(x.P\_Text, x.Indeks));  });  Receive<Druga>(x =>  {  Lista\_Botuna[x.Indeks].Text = x.D\_Text;  });  }  } |

Primjer koda 8 Opis klase Worker aktora

|  |
| --- |
| class Worker : ReceiveActor  {  public Worker()  {  Receive<Prva>(x => Provjeri(x));  }  public void Provjeri(Prva x)  {    if (x.P\_Text == "X" )  {  Sender.Tell(new Druga("X", x.Indeks));  }  if (x.P\_Text == "O" )  {  Sender.Tell(new Druga("O", x.Indeks));  }  }  } |

Pomoću klase „Prva“ FormActor šalje „znak“ igrača koji je trenutno odigrao potez. Taj znak može biti „X“ ili „O“ te indeks botuna koji je stisnut. Svaki botun ima svoj indeks koji se razlikuje od ostalih. Prvi igrač koji stisne botun automatski dobija znak „X“ i s njime igra cijelu partiju.

Nakon što FormActor pošalje poruku „Prva“, Worker aktor je pomoću metode Receive() prima i pomoću svoje metode Provjeri() je obrađuje. Pri obradi on gleda koji se znak nalazi u stringu i koji se indeks nalazi u integer varijabli koju je primio. Zatim on generira poruku „Druga“ i šalje je svom roditelju FormActoru koji pomoću nje generira stanje na igraćoj ploči.

# Zaključak

Raspodijeljeni sustavi se javljaju u mnogim područjima. Prije svega to mogu biti raspodijeljeni računalni sustavi s više ili mnogo umreženih računala organiziranih na različite načine kao što su grozd računala (engl. Cluster), splet računala (engl. Grid) ili računalni oblak (engl. Cloud). Raspodijeljeni informacijski sustavi primjenjuju se za obradu transakcija i integraciju poslovnih aplikacija te pružanje informacisjkih i komunikacijskih usluga.

# Literatura

|  |  |
| --- | --- |
| [1] | [Mrežno]. Available: https://bs.wikipedia.org/wiki/Distribuirano\_ra%C4%8Dunarstvo. |
| [2] | [Mrežno]. Available: https://getakka.net/articles/intro/what-is-akka.html. |
| [3] | [Mrežno]. Available: https://doc.akka.io/docs/akka/2.5/guide/tutorial\_1.html. |
| [4] | [Mrežno]. Available: https://doc.akka.io/docs/akka/2.5/guide/tutorial\_1.html. |

# Tablica slika

[Slika 1 Hijerarhijska struktura aktor sustava [3] 4](#_Toc522125541)

[Slika 2 Početno sučelje aplikacije 6](#_Toc522125542)

[Slika 3 Prikaz kraja jedne partije 7](#_Toc522125543)

# Tablica primjera koda

[Primjer koda 1Kreiranje aktor sustava 2](#_Toc522125548)

[Primjer koda 2 kreiranje prvog aktora 2](#_Toc522125549)

[Primjer koda 3 Kreiranje "top level" aktora 5](#_Toc522125550)

[Primjer koda 4 Kreiranje djece "top level"aktora 5](#_Toc522125551)

[Primjer koda 5 Primjer potpune adrese 5](#_Toc522125552)

[Primjer koda 6 Opis poruka koje koriste aktori 8](#_Toc522125553)

[Primjer koda 7 Opis klase FormActora 8](#_Toc522125554)

[Primjer koda 8 Opis klase Worker aktora 9](#_Toc522125555)