Fabijan Lukin, Fran Pregernik, Tomislav Sukser

Tehnička dokumentacija za obojivo računarstvo

U Zagrebu, travanj 2006.

SADRŽAJ

1 Obojivo računarstvo

1.1 Uvod

U skoroj budućnosti, tehnologija će nam omogućavati izradu procesora vrlo malih dimenzija sa istom, ili većom procesnom moći kao današnja stolna računala te vrlo malom cijenom. Pod tom pretpostavkom, biti će moguće izraditi cijeli sustav na čipu, sa procesorom, bežičnim komunikacijskim podsustavom, raznim senzorima okoline, memorijom i sustavom napajanja koji crpi energiju iz okoline. Takav sustav je osnova obojivog računarstva.

1.1.1 Zašto „obojivo“ računarstvo?

Obojivo je izraz koji je primijenio [William Joseph Butera] u radu „Paintable Computers“ Radi se o tome da, zbog svoje minijaturne veličine (zrno pijeska) i jeftine cijene, možemo veliku količinu tih računala pomiješati sa bojom i zid obojiti sa njome. Time bi dobili mrežu nasumično postavljenih računala koja su u stanju komunicirati jedni sa drugima, stvarajući pri tome neku vrstu računalne mreže.

U svom radu Buthera navodi da primjena tih računala nije isključivo vezana za boju i zid, nego ih se može umetnuti u tkanine, građevinske materijale i još puno toga, sa svrhom da obrađuju podatke koje dobivaju preko senzora o mediju u kojem se nalaze.

1.1.2 Tajna uspjeha obojivog računarstva

Buthera navodi kako će ta računala, zbog male veličine i male cijene te ponajviše zbog otpornosti na stres okoline, postajati sve prihvatljivija. Ljudi su uvijek oprezni kod kupovanja skupih stvari u malim količinama (npr. jako računalo sa dual core procesorom, i sl.), naprotiv tome istu procesnu moć bi mogli dobiti kupnjom kile i pol obojivih računala za puno manju cijenu (npr. cijenu kile praška za pranje rublja). Još jedna prednost ovog koncepta je visok stupanj paralelizma u obradi podataka. Sva ta računala imaju nezavisne jezgre te se ukupna obrada podataka može obaviti u puno manjem vremenu.

Zgodan citat: „Want a surface to become smart? Add a layer of computing. Want it to become smarter? Add a second coat! Has the computing lost its luster? Get out the belt sander!“

1.1.3 Kako „uštekati“ obojivo računalo i kako oni zapravo razgovaraju međusobno?

Svako od tih računala ima sustav napajanja koji je u stanju crpiti energiju iz okoline. Bilo to bila solarna energija, toplinska energija ili jednostavno energija iz obližnjeg magnetskog polja, ovisi o mjestu primjene. Pretpostavlja se da je takav izvor napajanja dovoljan da bi sustav mogao raditi te da bi bežični komunikacijski kanal bio u stanju komunicirati sa najbližim susjednim računalima.

Bežična veza može biti izvedena na par načina, kao na primjer: radio veza ili infracrvena veza. Ono što je bitno je da je takva veza i cijeli komunikacijski sustav asinkron, dvosmjeran i otporan.

Obojiva računala nemaju nikakvog znanja o trenutačnoj poziciji gdje se nalaze (osim ako imaju GPS senzor ugrađen), i u najgorem slučaju ne znaju da ima drugih računala okolo osim najbližih susjeda.

No i samo ti podatci su dovoljni da se razvijaju programi za razne otporne infrastrukture koje mogu davati puno veću količinu informacija drugim programima, koji onda grade još specifičnije infrastrukture.

1.2 Programiranje obojivih računala

Postoji veliki problem ovog koncepta, kao i svih drugih masovno paralelnih sustava, a to je stvaranje robusnih i općenitih metoda programiranja za takve sustave. Ovdje je to još dodatno otežano jer riječ „masovno“ je kod obojivog računarstva deminutiv.

Nesinkroniziranost – Nigdje nije garantirano da susjedna računala imaju isti takt. Sinkronizacija na neke događaje je jako otežana jer se radi o topologiji sa nepoznatim brojem jedinki i povremenim ispadima računala. Programi koji se odvijaju na jednom računalu nikad ne bi trebali eksplicitno sinkronizirati neku svoju radnju sa događajem na drugom računalu.

Velika otpornost na greške – Pretpostavka je da se može dogoditi isključenje dijela korpusa obojivih računala. U tom slučaju bi programi pisani za ta računala morali biti otporni na takve promjene u topologiji.

Nepoznato mjesto u mreži – Računala mogu samo komunicirati sa najbližim susjedima koliko im to radijus bežične komunikacije to dopušta. Čak i unutar tog prostora komunikacije, računala međusobno nemaju nikakvo znanje o međusobnoj relativnoj poziciji.

Adaptivna topologija – Topologija nije poznata u trenutku pisanja programa i programi ne bi smjeli podrazumijevati neku fiksnu topologiju.

Ograničena memorija računala – Mora se jako voditi računa o zauzeću memorije računala. Također, brzina bežične mreže je spora u odnosu na brzinu procesora.

Rješenje tog problema su samoorganizirajuće strukture koje se mijenjaju toliko dugo dok ne postignu neki lokalni minimum.

2 Simulacija obojivog računarstva

Najveći izazov je bio napraviti simulator koji je u stanju simulirati prirodno paralelne procese na sekvencijalnom računalu. Drugi problem je kako simulirati komunikacijski medij i njegova svojstva, nadalje je bilo potrebno simulirati sve greške koje mogu nastupiti u stvarnosti kao npr. slučajno (ili namjerno) umiranje računala, interferencije u komunikacijama, propagaciju signala itd.

2.1 Simulator

Cijeli projekt je rađen u jeziku c# (cis, c-sharp itd.) i koristi napredne metode .NET Frameworka kao što su Remoting, generičke liste, funkcijski delegati, reflekcija (metarazredi) i drugo. Podrazumijeva se da će čitatelj ima osnovno znanje c# jezika i .NET frameworka, no cak i ako nema, kad naiđe na nepoznati pojam, može pogledati dokumentaciju .NET Frameworka.

Cijeli simulator se bazira na plugin arhitekturi, to jest simulator pri instanicranju pregledava „Plugins“ poddirketorij za .net assembly dll-ove i u njima traži implementacije obojivih računala, senzora, objekta za bojanje i programa. Grafička sučelja onda imaju liste plugina koje mogu učitati u simulator.

Simulator koji smo izradili radi na principu ciklusa. Jedan ciklus je završen kada svako računalo obradi jednu jedinicu rada (o tome kasnije). Redoslijed izvođenja pojedinih računala je nasumičan i ne garantira da će sva računala doći na red u jednom ciklusu. Time je ostvarena činjenica da sva računala ne moraju imati isti takt.

U svakom ciklusu simulatora se također vrši obrada komunikacijskog medija, koji je na pojedinim računalima ostvaren kao FIFO red ulaznih i izlaznih paketa, a sa gledišta simulatora, „niz signala“ se preuzima i stavlja na antenu računala. Bežični paketi, zbog vrste komunikacijskog kanala, ne stižu na antene susjeda odmah, nego nakon određenog vremena ili uopće ne stignu. Time je simuliran realni komunikacijski kanal. Nadalje, pretpostavljeno je da je korekcija grešaka u mrežnim paketima besprijekorna te se neće dogoditi da se jednom programu na obradu da paket koji nije ispravan.

Simulator ima podršku i za senzore koji su realizirani kao nadogradnja na obično računalo. U kraju svakog ciklusa, simulator uz pomoću podataka o okolini daje senzorima vrijednosti za koje su oni predviđeni.

2.2 Komunikacija računala sistemom podatkovne stranice

Komunikacija je izvedena na takav način da svaki program ima pristup lokalnoj podatkovnoj stranici (homepage) i jedino tu može pisati, kao i čitati. Računala periodički razmjenjuju kopije lokalnih podatkovnih stranica i spremaju ih u područje koje se naziva lokalni UI prostor (eng. IO space). Taj prostor je isključivo za čitanje.

Princip je takav da program čita iz UI prostora i iz lokalne podatkovne stranice i na temelju tih vrijednosti izvrši obradu te rezultat zapiše u lokalnu podatkovnu stranicu. Susjedni programi nakon toga pročitaju iz UI prostora i postupak se ponavlja.

2.3 Obojivo računalo

Simulirano obojivo računalo se sastoji od dva dijela. Prvi dio je sučelje s kojim simulator komunicira. Simulator pomoću tog sučelja kontrolira pokretanje obrade kompjutera kao i čitanje podatkovne stranice kompjutera te čitanje UI prostora kojeg to računalo ima.

Drugi dio je API sučelje s kojim komuniciraju programi koji se izvršavaju na tom računalu. API sučelje ima, između ostalog, sve funkcije koje su potrebne za čitanje i pisanje na lokalnu podatkovnu stranicu, čitanje UI prostora, selidbu programa na drugo računalo i deinstalaciju istih.

Obrada jedinice rada obojivog računala je podijeljena u par dijelova. U prvom dijelu se obrađuju dolazni paketi koji mogu sadržavati kopije podatkovnih stranica susjednih računala ili preseljene programe od susjednih računala. Te kopije treba zapisati u UI prostor, a preseljene programe treba instalirati na lokalno računalo.

Drugi dio je izvršavanje svih instaliranih programa. Ovdje se provodi najveći dio procesiranja simulatora jer su ti programi ovisno o primjeni vrlo procesno zahtjevni.

Treći dio je obrada zahtjeva za prijenos programa. Naime, nakon što program preko API-a zatraži prijenos na nekog od susjeda, kompjuter to zabilježi i obradi kasnije. Taj zahtjev može biti odobren i poništen ovisno o nekim faktorima kao što je pun izlazni međuspremnik. Trenutačno u našoj implementaciji zahtjev je uvijek odobren. No implementacija nekakve restrikcija se vrlo lagano ostvaruje sa jednim if-then! Ako je zahtjev odobren onda se radi kopija programa i programu koji je zatražio prijenos se daje mogućnost postavljanja varijabli kopije samog sebe kako bi ta kopija mogla biti pokrenuta sa promijenjenim inicijalnim stanjem. Ta mogućnost se koristi u velikoj većini programa.

Četvrti dio je obrada zahtjeva za deinstalaciju programa. Programi mogu preko API-a zatražiti da budu deinstalirani iz trenutnog računala na slični način kao i u trećem dijelu obrade (obrada prijenosa programa). Jedina razlika je što je takav zahtjev uvijek odobren.

Peti i zadnji dio obrade je provjera je li je lokalna podatkovna stranica mijenjana te ako je onda se šalje njena kopija svim susjedima. Također se ovdje provjerava koliko dugo nije osvježena lokalna kopija podatkovne stranice susjeda u UI prostoru. Ako je ta vrijednost veća od neke granice onda se briše ta kopija i time se stvara mjesto za kopiju nekog drugog susjeda. Naime UI prostor je ograničene veličine u smislu da može držati npr. 16 kopija podatkovnih stranica susjeda. Ako se neki od susjeda ugasi a u susjedstvu je 20 susjeda onda onaj se kopija podatkovne stranice 17tog susjeda može kopirati u trenutno računalo.

Mrežna komunikacija je napravljena preko redova s time da je izlazni red prioritetni red i programi mogu indirektno kontrolirati prioritet nekih akcija koje stvaraju mrežne pakete, kao npr. zahtjevi za prijenos.

Programi ne mogu nikako slati vlastite pakete u mrežu. Jedini koji to može je računalo. No oni niti ne moraju slati pakete jer se sva komunikacija programa odvija preko podatkovne stranice.

2.4 Podatkovna stranica

Podatkovna stranica je organizirana tako da svaki program, prilikom izrade, izabire jedan jedinstveni identifikacijski broj te se za takav program stvara podprostor u podatkovnoj stranici. U originalnom radu je taj podatkovni prostor organiziran tako da svi programi pišu u jedan jedini prostor što povećava procesiranje koje jedan program mora napraviti. Da pojasnim, ako program funkcionira bez pomoću drugih programa onda umjesto da pretražuje i parsira što su ostali programi napisali u zajednički prostor u nadi da će naći što je njegov susjedni koprogram napisao, on može u podprostoru točno pročitati vrijednost koju je njegov koprogram napisao bez potrebe za dodatnom obradom. Nadalje, ako ipak program funkcionira uz pomoć drugih programa onda je potrebno da taj program zna jedinstveni broj programa o kojem ovisi, što i nije veliki problem te uz pomoć tog broja čita, ali i piše po potrebi, u drugi podprostor.

Naravno uvijek je moguće napisati program koji ometa druge programe pišući u njihove podprostore. No svrha takvog programa je nepotrebna i nepoželjna. Nije moguće spriječiti takvo ponašanje, ali takvo ponašanje nije kobno jer je ovo ipak samo simulacija.

Ovo vrijedi i za kopije podatkovnih stranica u UI prostoru s razlikom da je tamo moguće samo čitanje.

2.5 Programi – „Process Fragments“

Programi pisani za obojiva računala komuniciraju isključivo preko API-a sa računalom. API sučelje je proširivo, ali već sada obuhvaća sve funkcije koje su dovoljne za pisanje vrlo složenih programa (kao na primjer Shadowfax).

Obojivo računalo komunicira sa programima preko pet standardnih metoda, a te su: Install, Uninstall, TransferGranted, TransferRefused i najvažnije Update. Programi nisu ograničeni samo na te metode ali ih moraju imati što je osigurano nasljeđivanjem.

Install metoda se poziva u trenutku kada se program prvi puta instalira na računalo. U njoj se vrši inicijalizacija programa.

Uninstall metoda se izvršava nakon što je program preko API-a zatražio deinstalaciju. Obično se tu brišu vrijednosti zapisane u lokalnoj podatkovnoj stranici.

TransferGranted metoda se poziva nakon što je obrađen zahtjev za prijenos te se tu pruža mogućnost programu da promijeni vrijednosti lokalnih varijabli kako bi ta kopija prilikom instalacije već mogla imati neke podatke o okolini i prilagoditi se tome.

TransferRefused metoda se poziva ako je zahtjev za prijenos odbijen te se time signalizira programu da, pogodili ste, mu je zahtjev odbijen.

Update metoda je najvažnija metoda od svih, jer se u njoj treba nalaziti glavno procesiranje programa. Ona se poziva u svakom ciklusu kompjutera. U njoj treba vršiti čitanje UI prostora, reagiranje na pročitano, selidba, deinstalacija i pisanje u lokalnu podatkovnu stranicu.

Ovo su samo smjernice, jer, moguće je napraviti program koji, ako je to tako potrebno, glavnu procesnu funkciju ima smještenu u npr. Install metodu. Nema

nikakvih restrikcija osim da ako program u Uninstall metodi zatraži prijenos na neko računalo taj zahtjev nece biti zabilježen.

Program je zapravo inkapsuliran u apstraktni razred PocessFragmentBase. Taj razred zahtvjeva da se implementiraju gore navedene metode te da se dodatno stavi u svojstvo ProcessFragmentId jedinstveni broj koji definira program.

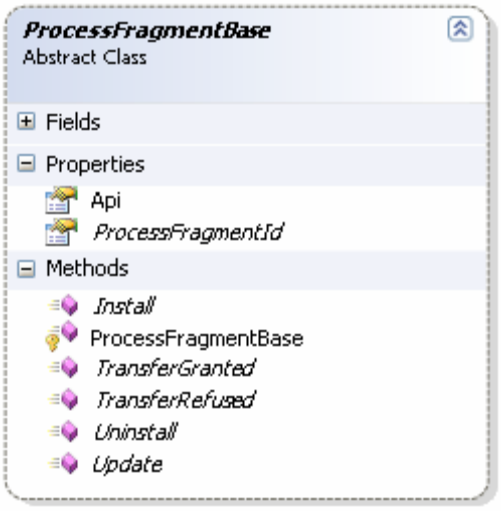


Diagram razreda ProcessFragmentBase

Važno je napomenuti da svaki program (process fragment) ima **prazni konstruktor**, kako bi se mogao učitati. Zapravo se neće ni dozvoliti programima koji to nemaju da budu u popisu valjanih programa simulatora.

Provjera strukturne valjanosti svih dostupnih programa se obavlja prilikom inicijalizacije simulatora u razredu PluginLoader u metodi CheckProcessFragmentValidity(Type pfrag). Zahtjeva se da je program direktni potomak razreda ProcessFragmentBase, da je razred programa označen kao sealed (što znači da ga se ne može naslijediti) i najvažnije da ima jedan konstruktor i to prazni konstruktor. U toj metodi se mogu pogledati sva pravila restrikcije na strukturu razreda programa. Općenito u razredu PluginLoader se učitavaju svi dodatci i tamo su također strukturna pravile ostalih dodataka provjeravana.

2.6 Okruženje simulacije – „Environment“

Kao što je već napomenuto, simulator ima podršku za računala koja imaju senzore. Princip izrade senzora je slijedeći: senzor mora imati iste funkcije kao i obično računalo, senzor je zapravo nadogradnja računala i to je najlakše izvesti tako da računalo senzor nasljeđuje od običnog računala. Prilikom izrade senzora, zna se kakve on fizičke pojave prati i koje vrijednosti prima. Računalo senzor može imati i više mjerača u sebi.

Istodobno sa izradom senzora izrađuje se i tzv. tok fizikalnih podataka „data feeder“ koji zapravo preuzimaju ulogu fizikalne pojave te ju na neki način modeliraju. Model te fizikalne pojave može biti empirijski podatci koji su se mijenjali tijekom vremena zapisani u nekoj tablici ili matematička funkcija ili nešto kompletno nasumično.

Objekt toka fizikalnih podataka u svakom ciklusu simulatora na temelju lokacije senzora i proteklog vremena izračunava neku vrijednost te ju predaje senzoru. Kako je ta vrijednost dobivena je prepušteno autoru toka podataka. Više o izradi tokova podataka biti će rečeno kasnije.

3 Interna realizacija simulatora

3.1 Obojivo računalo

Obojivo računalo je realizirano kao apstraktni razred PaintableComputerBase te ona sadrži sve funkcije koje je potrebno implementirati, kako za sučelje prema simulatoru tako i za API sučelje prema programima.

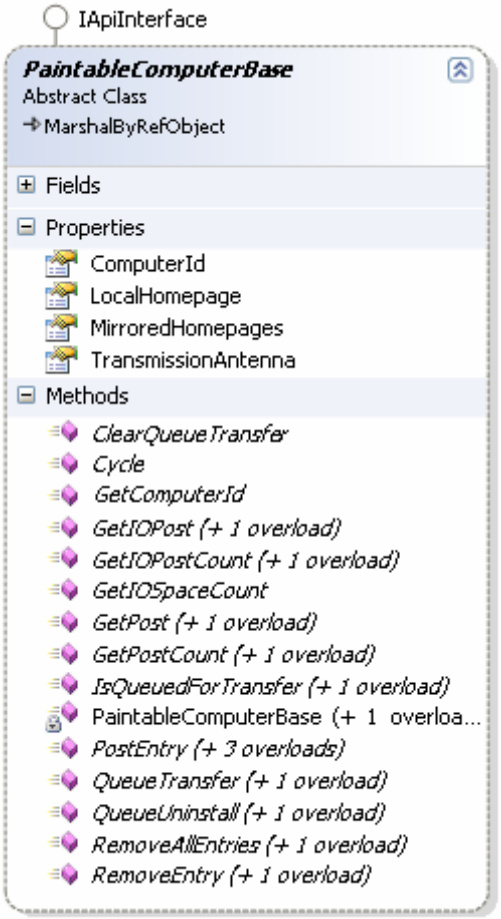


Diagram razreda PaintableComputerBase

Razred PaintableComputerBase ima dvije grupe metoda, prva služi za komunikaciju prema simulatoru a druga za komunikaciju prema programima. Slijedi opis svake od njih.

3.1.1 Definicija razreda

3.1.1.1 Svojstva razreda PaintableComputerBase

public uint ComputerId – svojstvo(property) služi simulatoru za identifikaciju kompjutera u listi. Vrijednost ovog svojstva postavlja sam kompjuter i nije garantirano

u potpunosti da će biti jedinstveno zbog prirode generiranja slučajnih brojeva. Bez obzira na to može se uzeti da je jedinstveno.

public Homepage LocalHomepage – svojstvo koje služi više grafičkim sučeljima nego simulatoru u svrhu traženja pogrešaka programa. Vraća lokalnu podatkovnu stranicu.

public List<Homepage> MirroredHomepages – svojstvo koje također služi više grafičkim sučeljima nego simulatoru u svrhu traženja pogrešaka programa. Vraća listu kopija podatkovnih stranica koje trenutno računalo ima u UI prostoru.

public NetworkPacket TransmissionAntenna – svojstvo preko kojeg simulator rasprostire pakete bežične veze. Kada simulator postavlja vrijednost ovog svojstva on zapravo stavlja paket „na“ antenu i taj paket se onda postavlja u ulazni red. Kada simulator čita vrijednost ovog svojstva, to zapravo simulira kao da je računalo uspješno poslalo paket iz svog odlaznog spremnika.

3.1.1.2 Metode razreda PaintableComputerBase

public abstract void Cycle() – metoda koja predstavlja operacijski sustav obojivog računala. Poziva ju simulator. U njoj se izvršavaju onih pet koraka obrade nabrojenih u poglavlju 2.3.

Ostale metode spadaju u API funkcije s kojima direktno komuniciraju programi. Slijedi popis istih ali detaljna upotreba API funkcija se nalazi u dokumentaciji za pisanje programa.

public abstract int PostEntry(...); – piše proizvoljnu vrijednost u podprostor lokalnog podatkovnog prostora za trenutni program.

public abstract bool RemoveEntry(...) – briše određenu vrijednost iz podprostora lokalnog podatkovne stranice za trenutni program.

public abstract int RemoveAllEntries(...) – briše sve zapisane vrijednosti iz podprostora lokalnog podatkovne stranice za trenutni program.

public abstract int GetPostCount() – vraća broj zauzetih mjesta u podprostoru lokalnog podatkovne stranice za trenutni program.

public abstract object GetPost(...) – vraća vrijednost zapisanu u podprostoru lokalnog podatkovne stranice za trenutni program.

public abstract int GetIOSpaceCount() – vraća broj kopija podatkovne stranice u UI prostoru. Odgovara broju trenutno vidljivih susjeda.

public abstract int GetIOPostCount(...) – vraća broj zauzetih zapisa u jednoj od kopija podatkovne stranice u UI prostoru za određeni program.

public abstract object GetIOPost(...) – vraća vrijednost iz podprostora programa u specifičnoj kopiji podatkovne stranice u UI prostoru.

public abstract bool QueueTransfer(...) – zapiše zahtjev za prijenos programa na susjedno računalo i stavlja ga u red za obradu.

public abstract bool IsQueuedForTransfer(..) – vraća istinu ili laž o tome je li je trenutni program još uvijek u listi za čekanje na prijenos.

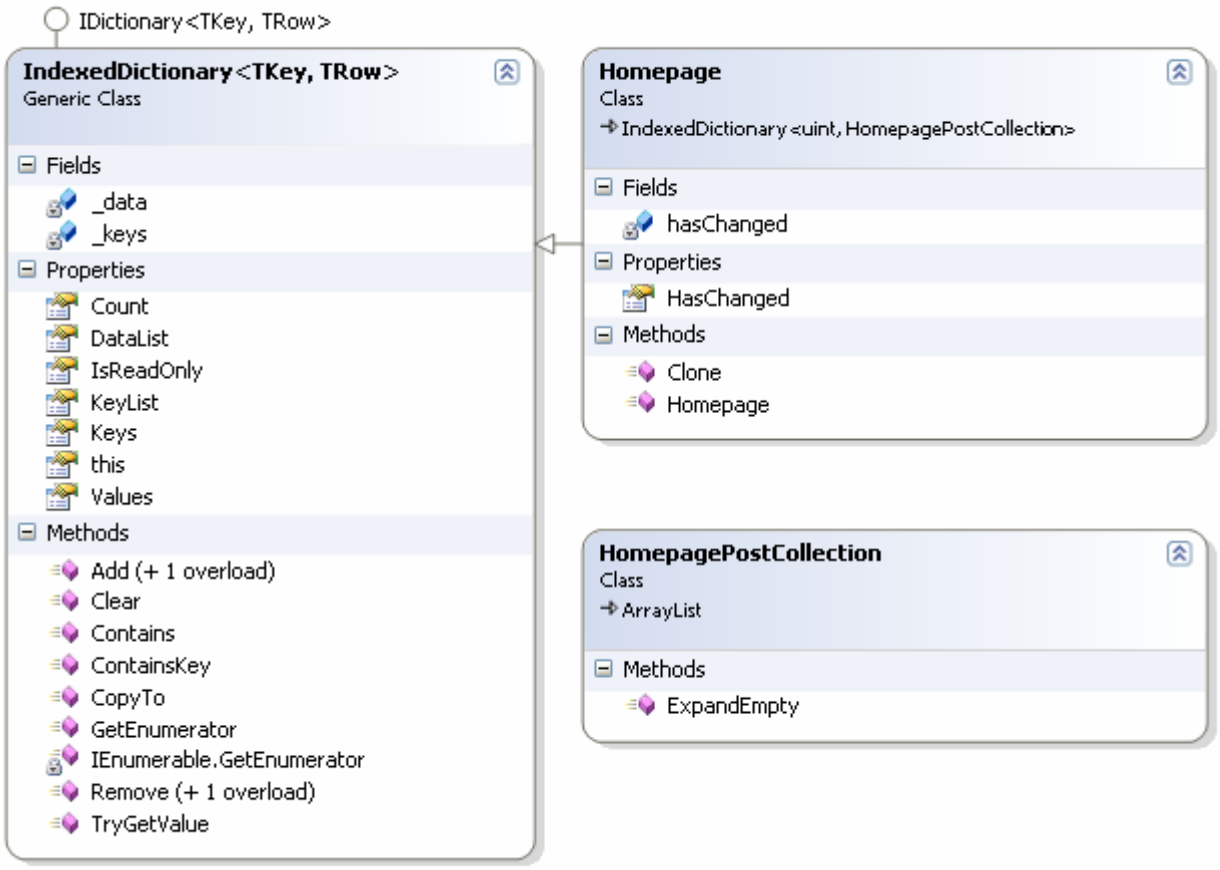
public abstract bool ClearQueueTransfer(...); - poništava zahtjev za prijenos na određenog susjeda.

public abstract bool QueueUninstall() – zapiše zahtjev za deinstalaciju trenutnog programa.

public abstract uint GetComputerId() – vraća slučajno generirani identifikacijski broj trenutnog računala koji se ne garantira da je jedinstven.

3.1.2 Podatkovna stranica – razred Homepage

Podatkovna stranica je implementirana kao rječnik čiji je ključ jedinstveni broj programa, a vrijednost indeksirana lista objekata – razred HomepagePostCollection.



Diagrami razreda Homepage i HomepagePostCollection

Ako bi željeli dobiti listu svih vrijednosti koje je jedan program zapisao, napravili bi to na slijedeći način: homepage[programID].

Ako bi željeli pristupiti točno nekoj vrijednosti, onda bi to napravili ovako: homepage[programID][indeksVrijednosti].

3.1.3 Mrežni podsustav

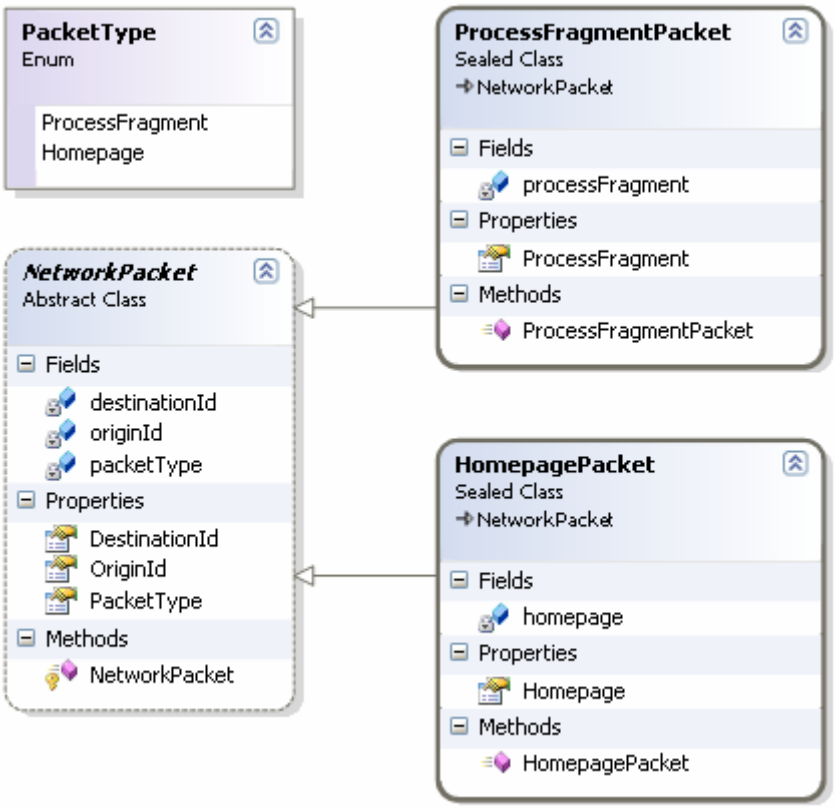


Diagram razreda mrežnog podsustava

Mrežni sustav obojivog računala je realiziran, kao što je prije rečeno, kao dva FIFO međuspremnika (reda) – ulazni i izlazni. Svi mrežni paketi su realizirani preko apstraktnog razreda NetworkPacket. Trenutačno postoje samo dvije specijalizacije za mrežne pakete: ProcessFragmentPacket i HomepagePacket koji pored osnovnih podataka o polazištu i odredištu sadržavaju redom, novi program koji će se instalirati na odredištu i kopiju podatkovne stranice.

Prilikom primanja paketa računalo ima podatke koji je susjed poslao paket i komu. Bežična veza kao takva pošalje paket svim susjedima te je na njima da obrade paket koji je namijenjen njima. Naravno, moguće je napraviti takvo računalo koje ima promiscuous mrežni podsustav, no svrha takvog računala je upitna.

3.2 Senzori i objekti tokova fizičkih podataka (DataFeeders)

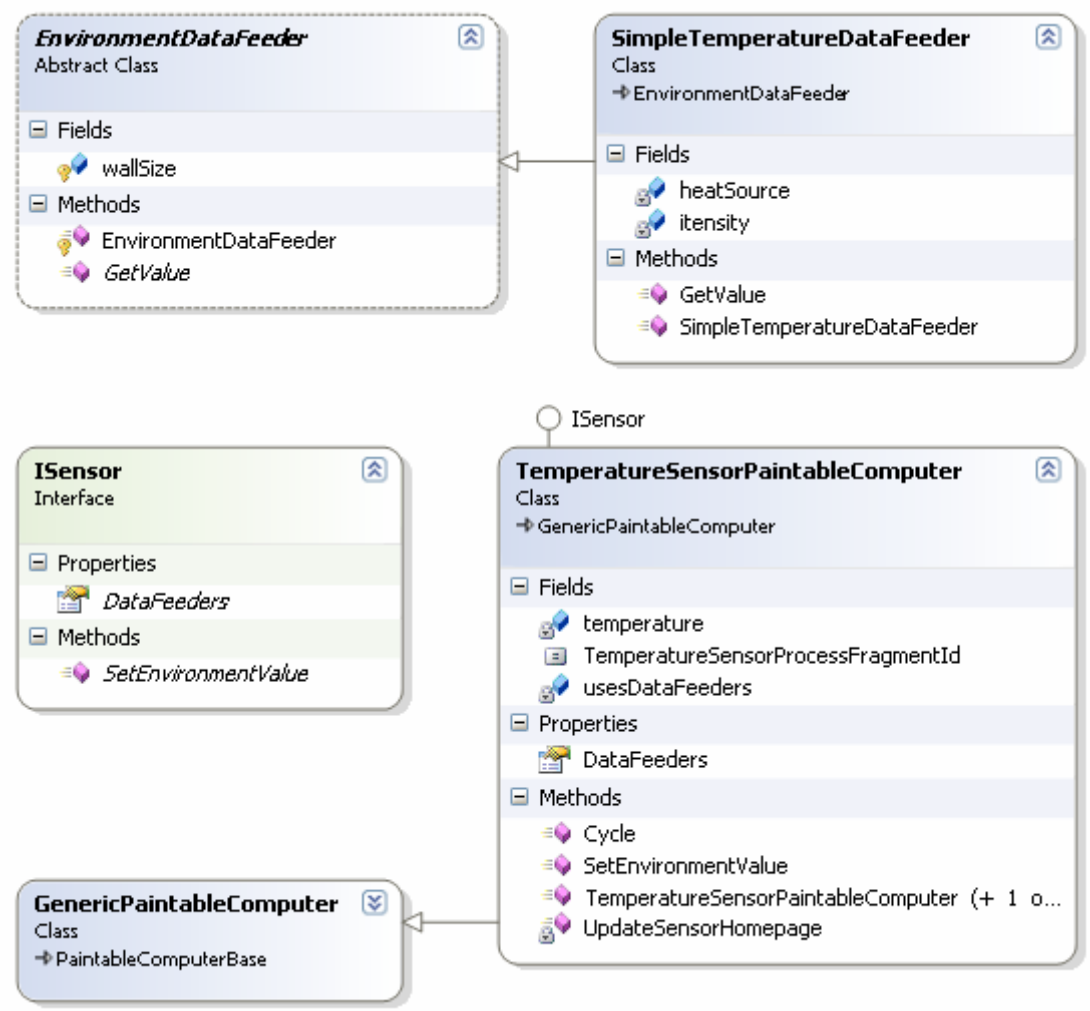


Diagram razreda tokova podataka i senzora

Prije je napomenuto da svaki senzor radi sa jednim ili više tokova podatak i samo s njima. Prilikom izrade senzora se fiksno postave u kodu tipovi tokova podataka s kojima senzor radi (svojstvo Type[] ISensor.DataFeeders). To daje simulatoru uvid o tome koji će tok podataka zatražiti kako bi tom senzoru vratio neku vrijednost.

Simulator prilikom stvaranja računala ili prilikom naknadnog dodavanja novih računala provjerava imaju li ta računala senzore i koje. Te ako ima neki registrirani tok podataka koji odgovara tom senzoru smješta ga u njegovu listu senzora. Ako nema onda se prilikom registracije tokova podataka nanovo rade iste provjere. Prilikom obrade ciklusa senzora, simulator zatraži od toka podataka da svakom senzoru u njegovoj listi izračuna i preda vrijednost.

3.2.1 Senzori

Koncept senzora je implementiran kao sučelje ISensor kojeg razredi, koje nasljeđuju od PaintableComputerBase (može biti bilo koji stupanj nasljeđivanja sve dok je vrh nasljeđivanja PaintableComputerBase), trebaju implementirati. Sučelje ISensor ne daje nikakvu funkcionalnost sa strane računala ali zato simulatoru daje mogućnost

da tom računalu daje vrijednosti, koje bi realan senzor pretvarao iz fizičkih vrijednosti u vrijednosti čitljive programima.

Rad senzora je jednostavan, nakon što primi neke vrijednosti od toka podataka, on te vrijednosti zapisuje kao virtualni program u podprostor podatkovne stranice sa nekim fiksnim identifikacijskim brojem. Taj identifikacijski broj može poslužiti onda programima da nađu i pročitaju vrijednosti senzora te da promjene svoje ponašanje na temelju toga.

3.2.1.1 Opis sučelja ISensor

Sučelje ISensor ima samo jedno svojstvo DataFeeders i jednu metodu SetEnvironmentValue!

Type[] DataFeeders – svojstvo vraća listu koja treba biti nepromjenjiva i fiksno postavljena u izvorni kôd. U toj listi se nalaze svi tipovi tokova podataka s kojima senzor zna razgovarati.

Void SetEnvironmentValue(Type, double) – metoda koja postavlja vrijednosti na senzor. Njeni parametri su tip toka podataka koji daje trenutnu vrijednost i sama vrijednost.

3.2.1.2 Primjer senzora – Temperaturni senzor

Slijedi primjer koda senzora koji prati temperaturu okoline. Radi samo sa tokom podataka temperature opisanog kasnije.

```
/// <summary>

/// This is a PaintableComputer with a temperature sensor on
board

/// </summary>

[DefaultHomepageColorizer(typeof(TemperatureSensorHomepageColor
izer))]

public class TemperatureSensorPaintableComputer :
GenericPaintableComputer, ISensor

{

    /// <summary>

    /// This sensor only works with the
simpleTemperatureDataFeeder

    /// </summary>

    private readonly Type[] usesDataFeeders = new Type[]
{typeof(SimpleTemperatureDataFeeder)};

    /// <summary>

    /// Initializes a new instance of the <see
cref="T:TemperatureSensorPaintableComputer"/> class.
```

```
    /// </summary>
    public TemperatureSensorPaintableComputer()
        : base(DEFAULT_MAXIMUMNEIGHBOURCOUNT)
    { ... }



    /// <summary>
    /// Initializes a new instance of the <see cref="T:TemperatureSensorPaintableComputer"/> class.
    /// </summary>
    /// <param name="maximumNeighbourCount">The maximum neighbour count.</param>
    public TemperatureSensorPaintableComputer(uint maximumNeighbourCount)
        : base(maximumNeighbourCount)
    { ... }


    /// <summary>
    /// Sets the environment value.
    /// </summary>
    /// <param name="dataFeederType">Type of the data feeder.</param>
    /// <param name="sensorValue">The sensor value.</param>
    public void SetEnvironmentValue(Type dataFeederType, double sensorValue)
    { ... }


    /// <summary>
    /// Gets the data feeders this sensor works with.
    /// </summary>
    /// <value>The data feeders.</value>
```

```
        public Type[] DataFeeders
        { get { return usesDataFeeders; } }

        /// <summary>
        /// Cycles this instance.
        /// </summary>
        public override void Cycle()
        { ... }

        /// <summary>
        /// Updates the sensor homepage.
        /// </summary>
        private void UpdateSensorHomepage()
        { ... }

    }
```

3.2.1.3 Primjer senzora –Senzor pritiska

Slijedi primjer koda za senzor koji registrira pritisak (npr. hod mrava). Taj senzor radi samo sa tokom podataka hodajućeg mrava opisanog kasnije.

```
    /// <summary>
    ///
    /// </summary>
    [DefaultHomepageColorizer(typeof(PressureSensorColorizer))]
    public class PressureSensorComputer : GenericPaintableComputer,
    ISensor
    {
        public const uint ProcessFragmentId = 0x12121212;

        /// <summary>
```

```
    ///     This     sensor     only     works     with     the
simpleTemperatureDataFeeder

    /// </summary>

    private  readonly  Type[]  usesDataFeeders  =  new  Type[]  {
typeof(AntWalkerDataFeeder) };



    private double pressure;



    /// <summary>

    ///    Initializes    a    new    instance    of    the    <see
cref="T:PressureSensorComputer"/> class.

    /// </summary>

    public PressureSensorComputer()

        : this(DEFAULT_MAXIMUMNEIGHBOURCOUNT)

    {}




    /// <summary>

    ///    Initializes    a    new    instance    of    the    <see
cref="T:PressureSensorComputer"/> class.

    /// </summary>

    ///    <param    name="maximumNeighbourCount">The    maximum
neighbour count.</param>

    public PressureSensorComputer(uint maximumNeighbourCount)

        : base(maximumNeighbourCount)

    { ... }



    /// <summary>

    /// Sets the environment value.

    /// </summary>

    ///   <param   name="dataFeederType">Type   of   the   data
feeder.</param>
```

```
        /// <param name="sensorValue">The sensor value.</param>

        public void SetEnvironmentValue(Type dataFeederType, double
    sensorValue)

        { ... }


        /// <summary>

        /// Gets the data feeders this sensor works with.

        /// </summary>

        /// <value>The data feeders.</value>

        public Type[] DataFeeders

        { get { return usesDataFeeders; }}




        /// <summary>

        /// Cycles this instance.

        /// </summary>

        public override void Cycle()

        { ... }


        /// <summary>

        /// Updates the sensor homepage.

        /// </summary>

        private void UpdateSensorHomepage()

        { ... }


    }
```

3.2.2 Tokovi podataka

Objekti toka podataka (engl. Datafeeders) su objekti koji nasljeđuju od razreda EnvironmentDataFeeder. Važno je napomenuti da kod stvaranja objekta toka podataka njegov konstruktor zahtjeva parametar veličine zida kako bi se mogla

dodatno prilagoditi fizička pojava koju tok podataka modelira. Razred EnvironmentDataFeeder ima samo jednu jedinu metodu koju simulator koristi, a ta je GetValue koja zahtjeva dva parametra, lokaciju senzora i vrijeme simulatora.

3.2.2.1 Opis razreda EnvironmentDataFeeder

Razred EnvironmentDataFeeder je jako jednostavna, ima samo jednu metodu GetValue koja je zadužena da na temelju lokacije senzora i vremena izračuna vrijednost koju senzor registrira,

double GetValue(Point location, uint time) – metoda koja na temelju lokacije i vremena vraća vrijednost koju simulator daje senzoru.

3.2.2.2 Primjer toka podataka – tok podataka o temperaturi

```
public class SimpleTemperatureDataFeeder : EnvironmentDataFeeder

{



    private Point heatSource;

    private double itensity;



    /// <summary>

    /// Initializes a new instance of the <see cref="T:SimpleTemperatureDataFeeder"/> class.

    /// </summary>

    /// <param name="wallSize">Size of the wall.</param>

    public SimpleTemperatureDataFeeder(Size wallSize)

        : base(wallSize)

    { ... }



    /// <summary>

    /// Gets the environmental value.

    /// </summary>

    /// <param name="location">The location.</param>

    /// <param name="ticks">The ticks.</param>

    /// <returns></returns>
```

```
    public override double GetValue(System.Drawing.Point
location, uint ticks)

    { ... }

}
```

3.2.2.3 *Primjer toka podataka – „hodajući mrav“*

```
/// <summary>

///

/// </summary>

public class AntWalkerDataFeeder : EnvironmentDataFeeder

{

    private Point centerPoint;

    private int minimalRadius;

    public const double MAXIMUMPRESSURE = 100.0;



    /// <summary>

    /// Initializes a new instance of the <see
cref="T:AntWalkerDataFeeder"/> class.

    /// </summary>

    /// <param name="wallSize">Size of the wall.</param>

    public AntWalkerDataFeeder(Size wallSize)

        : base(wallSize)

    { ... }



    /// <summary>

    /// Gets the environmental value.

    /// </summary>

    /// <param name="location">The location.</param>

    /// <param name="ticks">The ticks.</param>

    /// <returns></returns>
```

```
    public override double GetValue(System.Drawing.Point
location, uint ticks)

    { ... }


}
```

3.3 Jezgra simulatora

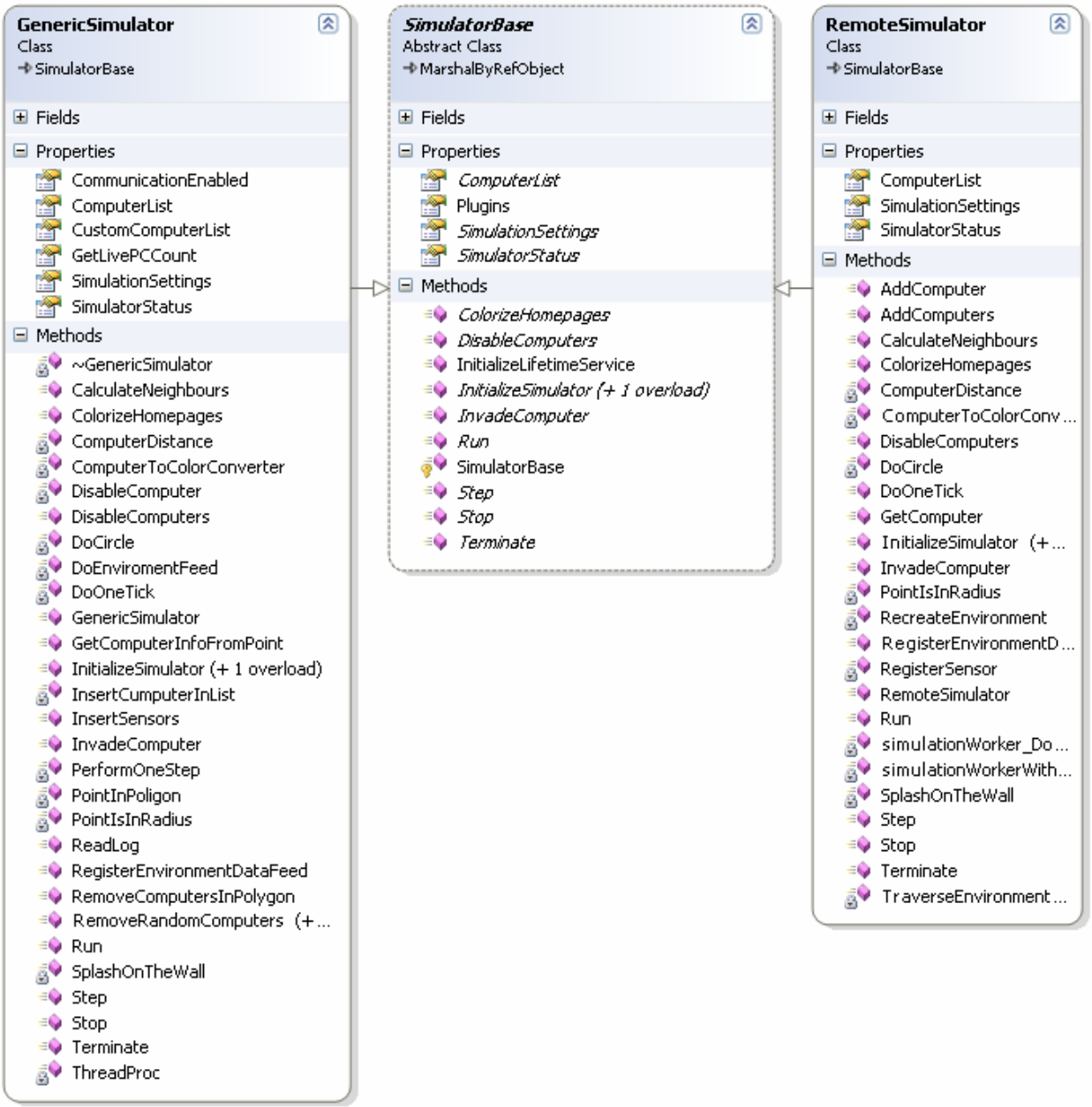


Diagram jezgre simulatora i dvije izvedbe simulatora

Svaki simulator prilikom inicijalizacije traži da mu se postave neke postavke (SimulationSettings) koje kontroliraju njegovo ponašanje. Te postavke su predstavljene razredom EngineSettings.

3.3.1 Postavke simulacije

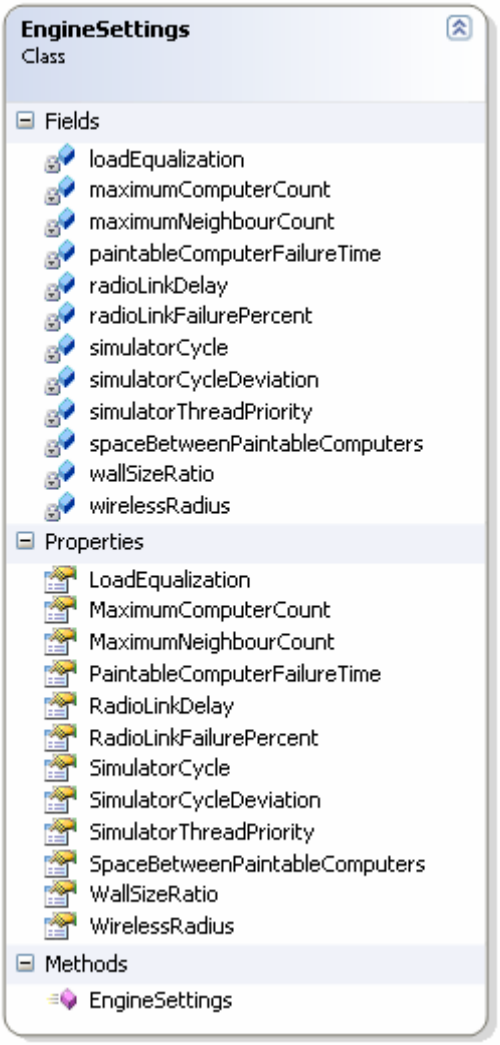


Diagram razreda postvaka simulatora EngineSettings

Razred EngineSettings ima slijedeće postavke:

bool LoadEquilization – Postavka koja uključuje mjerenje snage procesora računala na kojem se vrši simulacija u svrhu da se izbjegne početan ubrzan rad simulatora. Naime, ako simulirana računala nemaju u sebi programe onda je simulacija brža. Čim više računala instaliraju programe, simulacija se usporava. Prilikom dodavanja prvog programa u prvo računalo, simulacija je još uvijek podosta brza i grafičko sučelje ne stigne pratiti rad tog programa, nego promjene znaju biti skokovite. Sa LoadEquilization se može simulacija usporiti na početku u svrhu lakšeg praćenja simulacije na početku.

ThreadPriority SimulatorThreadPriority – prioritet dretve simulatora. Najveća postavka je ThreadPriority.Normal. Ima utjecaj na praćenje simulacije u smislu da

ako je prioritet simulatorske dretve manji od prioriteta grafičkog sučelja (ThreadPriority.Normal), grafičko sučelje stigne sve iscrtati prije nego se nešto znatno promijeni.

uint MaximumNeighbourCount – postavka koja ograničava broj susjeda koje računalo „vidi“. Direktno je povezano sa veličinom UI prostora obojivog računala.

uint MaximumComputerCount – postavka koja ograniči broj računala na zidu.

int SimulatorCycleDeviation – Odstupanje od prosječnog trajanja jednog ciklusa simulatora, isto u taktovima obojivog računala. Razlog uvođenja jest nejednakost brzine takta svih obojivih računala.

uint PaintableComputerFailureTime – odgovara srednjem vremenu do greške računala (Mean time before failure – MTBF). Broj koji simulator koristi kako bi povremeno mogao simulirati ispadane obojivog računala.

uint RadioLinkDelay – vrijeme koje je potrebno da signal propagira od antene do antene računala.

double RadioLinkFailurePercent – postotak gubljenja bežičnog paketa podataka.

uint SimulatorCycle – broj simulatorskih ciklusa izvršenih jedan za drugim, prije nego se provjeravaju uvjeti zaustavlajnja.

uint SpaceBetweenPaintableComputers – srednja udaljenost od računala do računala.

uint WirelessRadius – radijus bežične veze, odgovara jačini odašiljača na računalu.

Size WallSizeRatio – odnos širine i visine zida.

3.3.2 Tok simulacije

Nakon postavljanja vrijednosti postavki simulatora i odabira vrste računala koje želimo nanijeti na zid, poziva se metoda simulatora InitializeSimulator. Ona stvara računala, raspoređuje ih po zidu i pronalazi susjede svakog računala unutar radijusa bežične veze.

Sada kada je simulator inicijaliziran moguće ga je pokrenuti sa Run(), zaustaviti sa Stop() i izvršavati ciklus po ciklu sa Step().

Svaki simulator mora podržavati uklanjanje (isključivanje) računala na zidu. Ta funkcionalnost se implementira u metodi DisableComputers. Ona uklanja računala koja želimo izbrisati te nanovo izvršava pretragu susjeda.

Simulacija se obavlja u četiri koraka.

Prvi korak je provjera je li prošlo više vremena od vremena navedenog pod PaintableComputerFailureTime te se nasumično ugasi jedno računalo.

Drugi korak je dostava svih bežičnih paketa na antene susjeda računala koje je odaslalo pakete. Tu se također uzima u obzir vrijeme propagacije signala i postotak gubljenja paketa. Time je simuliran realni komunikacijski medij.

Drugi korak je izvršavanje jedinice rada svakog računala, ako je na njemu red. Točnije svako računalo se zabilježi u nekom trenutku vremena za obradu te ako je taj trenutak došao računalo se pokrene (PaintableComputerBase.Cycle() metoda)

3.3.3 Vizualizacija obojivog računala

Glavni dio grafičkog sučelja je prikaz obojivih računala i promjena koje su se događale na njima. Javila se potreba da se na temelju stanja računala, točnije na temelju sadržaja podatkovne stranice računala, računalima pridruži neka značajka – boja.

Za to smo predvidjeli objekte za bojanje, koji pogledaju cijeli sadržaj podatkovne stranice (ili samo njen mali dio) i na temelju zapisanih vrijednosti vrate boju koja će predstavljati računalo na zidu u grafičkom sučelju. Trenutačno samo dvije vrste razreda mogu imati za sebe vezane objekte za bojanje, to su programi i senzori. To je zbog toga jer si želimo predočiti kako program radi i npr. koje vrijednosti senzor registrira od okoline.

Također je moguće napraviti i objekt za bojanje koji nije isključivo vezan uz jedan program ili senzor. Objekti za bojanje koji su vezani za programe i senzore se automatski izabiru prilikom učitavanja istih. Oni koji nisu vezani se obično izrađuju kako bi se lakše moglo pratiti neke složene interakcije između raznih programa i senzora. Vezani objekti za bojanje bi trebali služiti kako bi se prikazalo napredovanje programa. Svi oni vide što se događa na podatkovnoj stranici, tako da sintaktički nema razlike među njima. To su samo smjernice kako bi se trebalo programirati.

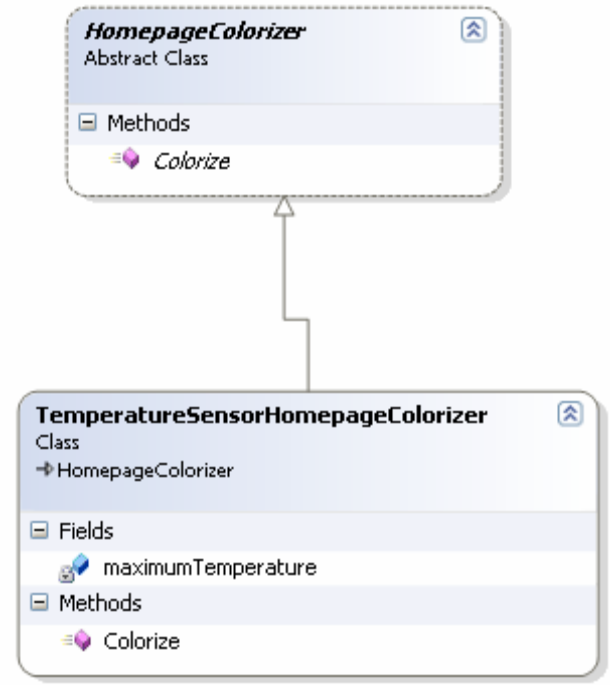


Diagram razreda objekta za bojanje i jedne njegove implementacije

Objekti za bojanje su instance razreda koji nasljeđuju od razreda HomepageColorizer. Taj razred zahtjeva da se implementira metoda Colorize kojoj je jedini parametar podatkovna stranica a rezultat boja (struktura System.Drawing.Color)

Grafičko sučelje je zaduženo za dohvat liste boja i njezin prikaz na temelju izabranog objekta za bojanje.

3.3.3.1 Primjer objekta za bojanje

Objekti za bojanje se vežu uz programe i senzore pomoću DefaultHomepageColorizerAttribute razreda (pogledati razred Attribute u .NET Frameworku). Slijedi primjer gdje se veže PressureSensorColorizer na PressureSensorComputer:

```
[DefaultHomepageColorizer(typeof(PressureSensorColorizer))]

public class PressureSensorComputer : GenericPaintableComputer,
ISensor

{ ... }
```

Sustav za učitavanje dodataka i grafičko sučelje mogu pročitati te atribute pomoću .NET metarazreda (Reflection). Grafička sučelja su napravljena tako da prilikom učitavanja programa se automatski izabrati vezani objekt za bojanje.

Pogledajmo primjer jednog objekta za bojanje temperaturnog senzora:

```
/// <summary>

/// Colorizes the specified homepage.

/// </summary>

/// <param name="homepage">The homepage.</param>

/// <returns></returns>

public override Color Colorize(Homepage homepage)

{

    Color retval = Color.WhiteSmoke;



    for (int i = 0; i < homepage.Count; i++)

        if                (homepage.KeyList[i]                ==
TemperatureSensorPaintableComputer.TemperatureSensorProcessFrag
mentId)

        {

            if (homepage.DataList[i].Count == 0)

                break;



            // post at index 0 is the temperature

            double              temperatureValue              =
(double)homepage.DataList[i][0];
```

```
            if (maximumTemperature < temperatureValue)
                maximumTemperature = temperatureValue;

            int color = (int)(255 * temperatureValue / maximumTemperature);

            retval = Color.FromArgb(color, 0, 0);

            // stop processing the rest of the homepage
            break;
        }

    return retval;
}
```

Kao što je vidljivo, on zapravo pretražuje podatkovnu stranicu i kad naiđe na ono što ga zanima on čita vrijednost i na neki način izračunava boju koju će prikazati.

Na ovaj način programer programa i senzora, odmah piše i vizualizaciju istih u simulatoru, što se pokazalo vrlo fleksibilnim i daje jako velike mogućnosti za prikaz akcija i interakcija programa i senzora.

3.3.4 Opis razreda GenericSimulator

Rad na lokalnom računalu nije ograničen brzinom veze između simulatora i korisničkog sičelja i zahtjeva simulator koji je višenitni (kako bi iskoristio potencijal dvojezgrenih i HT procesora) i ima brzi odziv na naredbe. Za takve potrebe napisan je GenericSimulator.

Sastoji se od jednog razreda koji obrađuje naredbe i vraća informacije korisniku, dok je sama simulacija rada obojivih računala, komunikacije među njima i simulacije okoliša odvojena u posebnu nit, kako bi se mogla izvoditi paralelno sa ostalim korisnikovim akcijama te vizualizacijom stanja simulatora.

3.3.5 Opis razreda RemoteSimulator

Remote simulator je simulator optimiran za rad na udaljenost u SmartClient grafičkom sučelju. Pogotovo su optimirane metode dohvata podataka o računalima. Tu se radi o velikom broju računala koja treba prikazivati i pobojat što rezultira

velikom količinom podataka koju treba poslati preko mreže. RemoteSimulator to uspješno smanjuje na minimum te omogućava praćenje simulacije bez velikih zaostataka.

RemoteSimulator se ne razlikuje od GenericSimulator-a u strukturi već samo u optimiranju količine podataka preko mreže.

Nema nikakvih ograničenja na korištenje RemoteSimulatora lokalno, štoviše to radi bez ikakvih dodatnih postavaka.

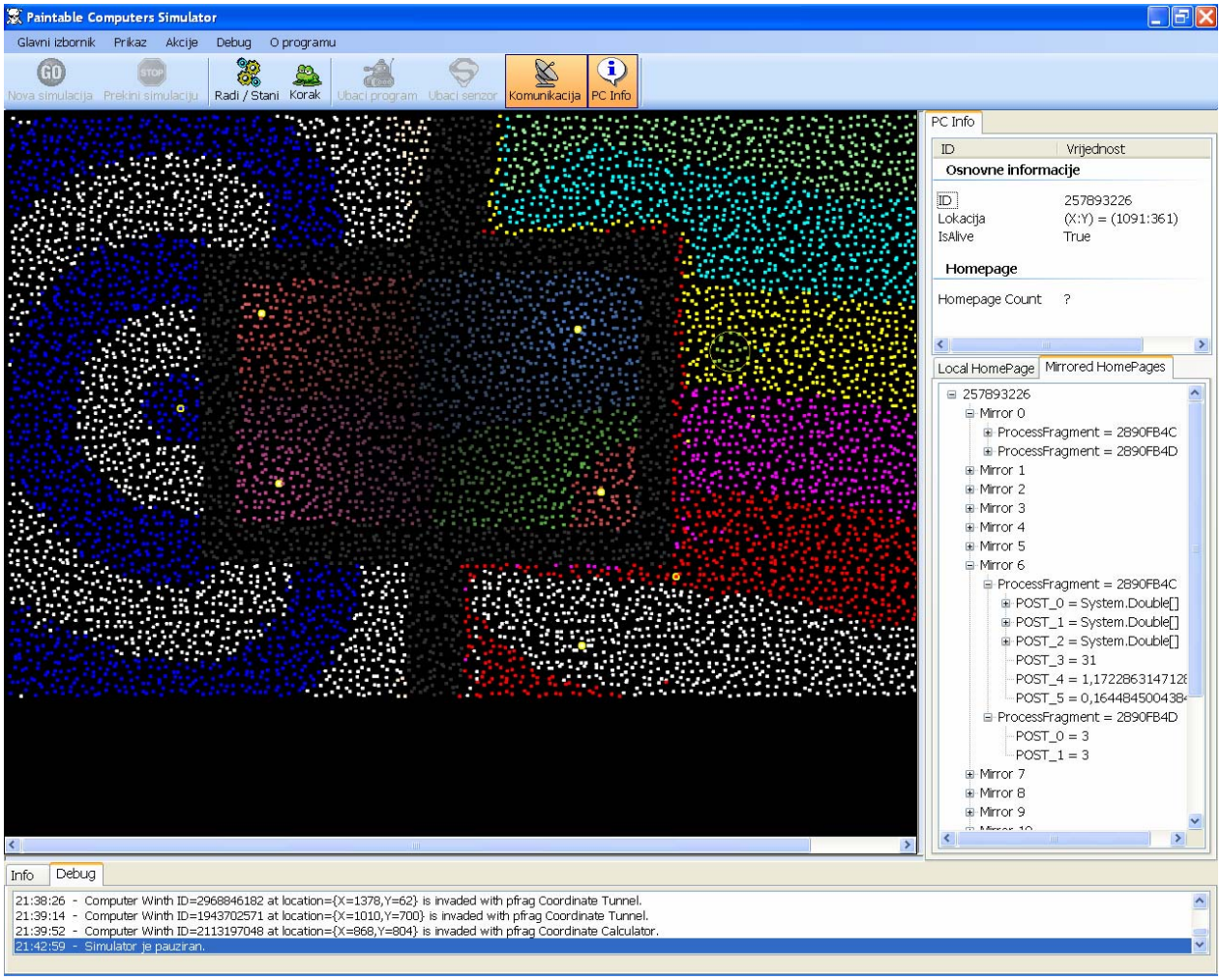
Ono što omogućuje rad na daljinu sa RemoteSimulator-om je .NET Remoting sustav koji nam omogućuje da koristimo objekt na isti način kao i da ne koristimo rad na daljinu. To omogućava da napravimo grafičko sučelje za rad lokalno i uz par dodatnih naredbi omogućimo rad na daljinu. Više o Remoting sustavu treba pogledati .NET dokumentaciju.

Dodatna razlika je što RemoteSimulator pokreće pozadinskog radnika prilikom pozivanja Run() koji je jako efikasan način za obavljanje nekog posla, u ovom slučaju simulacije, i kontrolu iste.

4 Grafička sučelja simulatora

4.1 Grafičko sučelje: PaintableSimulator GUI

PaintableSimulator.GUI je korisničko sučelje za rad na jednom računalu. Objedinjuje razrede GenericSimulator i Visualization i tjera ih da rade zajedno. Program se pokreće izvršavanjem datoteke PaintableComputerSimulator.GUI.exe. Programi moraju biti u podmapi „Plugins“.



PaintableSimulator.Gui grafičko sučelje

Ovo korisničko sučelje podržava spremanje simulacija u obliku slika i filmova u avi formatu. Filmovi mogu biti nekompresirani, ili kompresirani s Intel Indeo 5 ili HuffYUV koderima.

4.1.1 Različita iscrtavanja

Za spremanje slika i filmova, te za samo iscrtavanje simulatora, nužna je biblioteka ProcessSimulator.Visualizations.dll. U njoj se nalaze sljedeći razredi :

4.1.1.1 Razred Visualize

Razred Visualize je apstraktni razred koja sadrži funkcije koje svaki način crtanja mora imati. Sve funkcije i metode koja pruža ovaj razred, konkretni razred koji obavlja crtanje mora naslijediti i prekriti. Na taj način moguće je proširivati načine prikaza. Trenutno se nudi prikaz putem standardnog GDI i novog GDI+ načina iscrtavanja. GDI je posebno učinkovit ukoliko se program pokreće preko RemoteDesktopa (iscrtavanje je brže za 50% u odnosu na GDI+ prikaz).

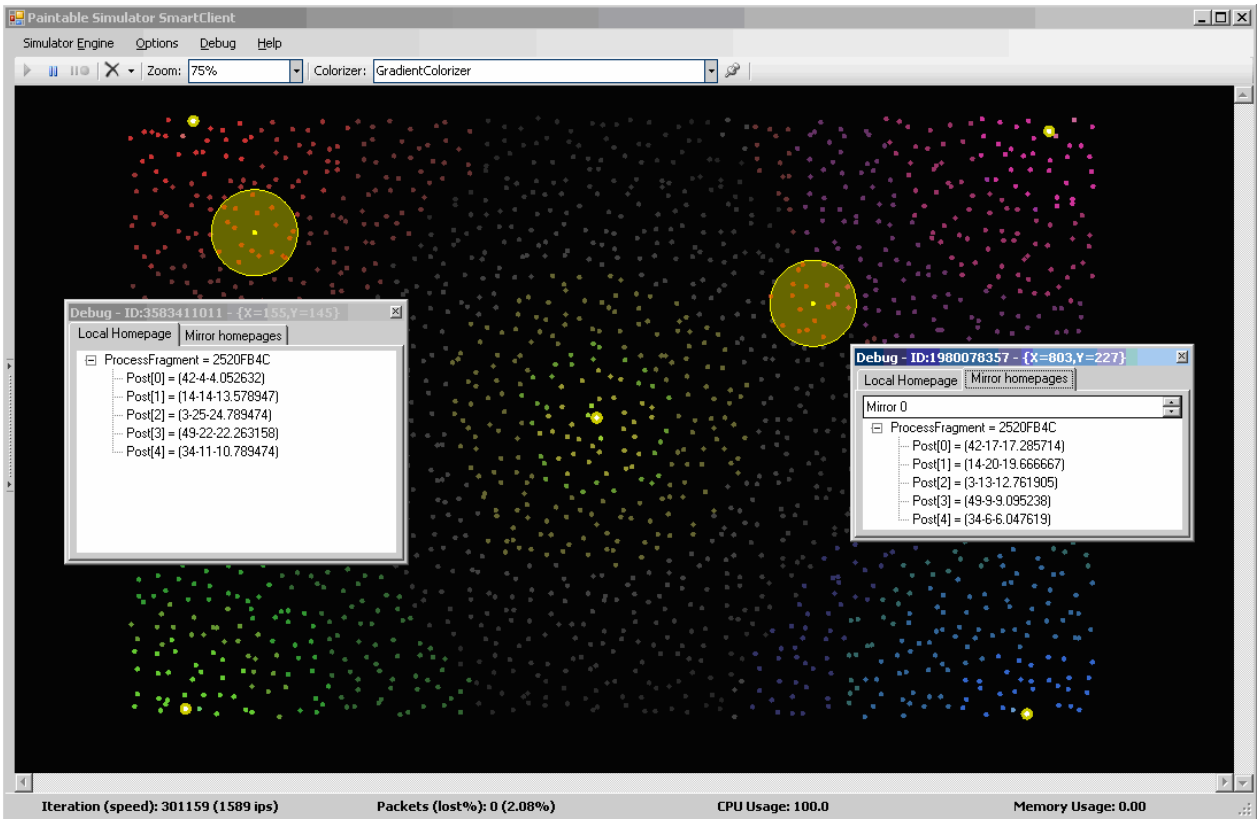
U budućnosti je moguće ugraditi DirectX ili OpenGL iscrtavanje i tako još više ubrzati crtanje i efikasno napraviti 3D prikaz obojivih računala.

4.1.1.2 Razred AVIExport

Pomoću ovog razreda moguće je izrađivati filmove simulacija u svim kompresijskim formatima koji su instalirani na računalo, kao i u nekompresiranom avi formatu. Trenutno su, uz spremanje bez kompresije, podržana dva formata: Intel Indeo 5 (kompresija s gubitkom kvalitete) i HuffYUV (kompresija bez gubitka kvalitette).

4.2 Grafičko sučelje - PaintableSimulator.SmartClient

Grafička sučelja, pored simulatora, moraju obavljati rad iscrtavana stanja na ekran. Taj rad je također procesno intenzivan kao i rad samog simulatora. Tako se rodila ideja da se ta dva posla odvoje na različita računala i odmah nakon toga, SmartClient grafičko sučelje. Važno je napomenuti da je SmartClient također moguće izvoditi na jednom računalu, zapravo tada radi najbrže, no također ga je moguće pokrenuti u modu za rad na daljinu.



PaintableSimulator.SmratClient u akciji s potpunim prikazom zida

Dodatna posebnost ovog grafičkog sučelja je u tome da je moguć izbor objekata za bojanje koji nisu isključivo vezani uz programe i senzore i to pomoću alatne trake iznad prikaza zida (Colorizer).

SmartClient ne razlikuje dodavanje senzora od običnih računala te se ispod postavaka simulacije mogu izabrati koja računala želimo dodati na zid. Posebno treba napomenuti da za razliku GenericSimulatora i PaintableSimulator.GUI grafičkog sučelja SmartClient i RemoteSimulator ne registriraju automatski tokove podataka nego to korisnik treba sam učiniti pomoću izbornika ispod postavaka simulacije.

4.2.1 Samostalni način rada

Samostalni način rada je uključen po defaultu. U tom načinu rada simulator radi kao i PaintableSimulator.GUI grafički klijent, možda malo sporije zbog zaštita protiv grešaka koje se mogu dogoditi u načinu rada na daljinu.

Ovaj način rada se pokreće direktno izvršavanjem datoteke PaintableSimulator.SmartClient.exe.

4.2.2 Klijent-server način simulacije

Rad na daljinu se odvija pomoću serverske aplikacije koja se pokreće na jačem računalu kako bi se simulacija mogla odvijati brže. **Treba napomenuti da Plugins direktorij sa dodatcima mora biti na svakom od računala kao poddirektorij direktorija gdje se nalaze PaintableSimulator.Server.exe i PaintableSimulator.SmartClient.exe, također je važno da su ti direktoriji po sadržaju identični kako bi se mogli udaljeno pokrenuti programi.**

Ovaj način rada se pokreće tako da se prilikom poziva PaintableSimulator.SmartClient.exe datoteke u nastavku navede „/remote“ (bez navodnika).

Za ostvarenje komunikacije je potrebno da server radi na računalu sa poznatom IP adresom i da je klijent „vidi“. Nažalost, zasada je moguće da se klijent i server izvršavaju na Windows XP operacijskim sustavima. Windows Server operacijski sustavi imaju jake sigurnosne mjere i ne omogućavaju izvršavanje bez problema.

5 Sigurnost u simulatoru

Riječ sigurnost ovdje ima dvojako značenje. Prvo sigurnost u smislu obojivog računala a drugo sigurnost računala na kojem se to simulira.

Ako pričamo o sigurnosti obojivog računala onda mislimo na slučajeve kada je program napisan za računalo sa greškom. Oporavak od grešaka traje i usporava simulator, tako da je napravljen mehanizam u svakom obojivom računalu da ako program uzrokuje više od npr. 3 greške za redom onda se prisilno deinstalira sa računala za kaznu.

Zaštita računala na kojem se vrti simulator je druga stvar. Pošto su svi programi za obojiva računala pisani također u c# jeziku radi jednostavnosti, imaju pristup svemu što nudi .NET Framework. To može imati ozbiljne posljedice (čitaj virusi). Tako da smo morali to zaštiti pomoću Code Access Security (abbrev. CAS), koji nudi .NET Framework, čime se može zabraniti kodu da izvršava pojedine operacije. Naravno mi smo za programe zabranili bilo kakav pristup resursima računala na kojem se odvija simulacija. Programi koji pokušavaju takve stvari biti će spriječeni i odmah izbačeni. Naravno, ova zaštita je zapravo nedovoljna za potpunu sigurnost, ali dovoljna je zasad. **Sigurnost obuhvaća sve hijerarhijski ispod simulatora što uključuje obojiva računala, senzore, tokove podataka i programe.**

Napominjemo još jednom ovo je samo osnovna zaštita sustava, u nadi da će se unaprijediti i da će to omogućiti puštanje u pogon jednog server računala koje bi pokretalo simulatore na sebi.

Slijedeće dozvole su zabranjene:

FileIOPermission

EnvironmentPermission

FileDialogPermission

FileIOPermission

IsolatedStorageFilePermission

KeyContainerPermission

PublisherIdentityPermission

ReflectionPermission

RegistryPermission

StorePermission

UIPermission

UrlIdentityPermission

Preporuča se čitatelju da pogleda u dokumentaciji .NET Frameworka za objašnjenje svake dozvole.

6 Literatura

- William Joseph Butera: Programming a Paintable Computer
- J.R.R. Tolkien: The Lord of the Rings
- Microsoft Developer Network: http://msdn.microsoft.com
- Wikipedia (Trilateration): http://en.wikipedia.org/wiki/Trilateration