TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: Informatika a logistika

Model virtuálního stroje s možností připojení do sítě EPSNET

Model of virtual machine with the possibility of connecting to the network EPSNET

Bakalářská práce

Autor: **Michal Kosek**
Vedoucí práce: Ing. Přemysl Svoboda
Konzultant: Ing. Miloš Hernych

V Liberci 28. 5. 2009

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum ……

Podpis …….

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu bakalářské práce Ing. Přemyslu Svobodovi za trpělivost, cenné připomínky a veškerou pomoc. Dále bych rád poděkoval všem, kteří se svými radami a připomínkami podíleli na zlepšení kvality této práce. Poděkování patří také všem mým blízkým za jejich podporu.

Abstrakt

Tato bakalářská práce se zabývá možností vytvoření virtuálního stroje, který bude možné řídit pomocí PLC (Programmable Logic Controller). Toto PLC komunikuje s počítačem po síti EPSNET.

V první části je popsána volba programovacího jazyka a jeho vazby na grafické API. Také jsou zde detailněji probrány vlastnosti vybrané vazby (JOGL).

Druhá část popisuje realizaci 3D modelu v Javě pomocí JOGL. Je to stručný návod jak tvořit objekty, pohybovat jimi a nanášet na ně textury. Dále je zde čtenář seznámen s vlastnostmi protokolu ESPNET a jeho příkazy, které jsou pro realizaci důležité. Je zde také popsána vlastní implementace aplikace.

Na konci bakalářské práce je vysvětleno, jak tuto aplikaci nastavit a propojit, aby pomocí EPSNET správně komunikovala s PLC.

Klíčová slova:

EPSNET, virtuální stroj, PLC, Java, JOGL

Abstract

This bachelor thesis deals with possibility of development of virtual machine, which can be controlled by means of PLC (Programmable Logic Controller). This PLC communicates with computer via EPSNET network.

First part describes selection of programming language and it's bindings with graphic API. Further I describe in detail characteristics of selected bindings (JOGL).

Next part describes realization of 3D model in Java through the use of JOGL. It is a brief manual for objects generating and movement, and application of texture on objects. Further I make reader acquainted with properties of ESPNET protocol and I describe the very implementation of application.

At the end of thesis, there is explained how to set the application and link it in order to communicate correctly with PLC through EPSNET.

Keywords:

EPSNET, virtual machine, PLC, Java, JOGL

Obsah

Seznam Obrázků

Seznam Tabulek

1 Úvod

V současné době se na TUL v předmětu MTI/ZLR při výuce programování automatů využívají modely vyvinuté ve vývojovém prostředí Delphi, které pro komunikaci s PLC využívají rozhraní COM. Pro provoz modelu a zároveň i virtuálního PLC na jednom počítači je nutné nainstalovat aplikaci, která v počítači vytváří virtuální sériové porty u nichž zajistí propojení (null-modem). Nakonfigurování virtuálních portů může být problematické a někdy i zcela nemožné.

Cílem práce bylo nalézt vhodný programovací jazyk a technologii pro realizaci 3D modelu s podporou hardwarově akcelerované grafiky. Seznámit se s možnostmi sítě EPSNET a sestavit model virtuálního stroje, který bude schopný komunikovat s PLC přes tuto síť pomocí UDP protokolu. Tento způsob komunikace je oproti rozhraní COM elegantnější, zvláště při provozování aplikace i virtuálního PLC na jednom počítači. V takovém případě není nutné instalovat program, který vytvoří virtuální sériový port ani nastavovat další parametry komunikace.

2 Výběr programovacího jazyka a grafického rozhraní

2.1 Java

Tato bakalářská práce byla realizována v jazyce Java. K tomu vedlo hned několik důvodů. Jedná se o programovací jazyk, který se snadno učí a je objektově orientovaný. Jeho výhodou je také přenositelnost na různé platformy, protože jeho výstup po zkompilování není spustitelní soubor, ale soubor s příponou *.jar. Tyto soubory je potom možné spouštět na každém počítači, kde na nainstalovaný Java Virtual Machine (JVM). JVM je dostupný na mnoha systémech, např. Microsoft Windows, Solaris OS, Linux a Mac OS. Pro všechny tyto systémy má Java připravené již ověřené třídy pro komunikaci prostřednictvím UDP.

Pro volbu programovacího jazyka bylo také důležité, že Java je snadná na učení. Navíc se jedná o jazyk, který je v dnešní době hodně používaný a proto je i perspektivní.

S mnoha výhodami Javy existují i její nevýhody. Největší nevýhoda plyne z výše uvedený faktů. Java je naproti např. C nebo C++ pomalejší při spouštění aplikací, protože v jiných jazycích jsou aplikace již zkompilovány pro danou platformu a pak se již jen spouští. JVM musí nejdříve aplikaci zkompilovat a pak teprve spustit. Dá se říci, že je to vlastně „daň“ za přenositelnost. První verze Javy (uvedena v roce 1996) byly dvacetkrát až čtyřicetkrát pomalejší než stejný algoritmus naprogramovaný v C/C++. Díky postupným vylepšením mezi jednotlivými verzemi se rychlost postupně zvyšuje, a tento rozdíl už není oproti jiným jazykům tak patrný jako dříve.

Java je oproti jiným jazykům o něco více náročná na operační paměť, protože kromě aplikace musí být zároveň spuštěno i celé prostředí. I tento fakt již v dnešní době není tak výrazný jako dříve.

Java není příliš často využívána pro grafické aplikace, protože u nich je většinou rychlost vykreslování na prvním místě. Navíc bez přidaných knihoven toho s grafikou příliš neumí. V následujících částech jsou popsány jednotlivé rozhraní, která jsou k dispozici, a také důvody výběru jednoho z nich.

2.2 OpenGL

Knihovna OpenGL je standardní aplikační rozhraní pro 2D a 3D grafiku. Používá se při tvorbě počítačových her, CAD programů, aplikací virtuální reality či vědeckotechnické vizualizace. OpenGL je navrženo s důrazem na to, aby bylo použitelné na různých typech grafických akcelerátorů. V případě, že na určitém systému není žádný grafický akcelerátor nainstalován, lze využívat i na hardwaru, který ho sám o sobě nepodporuje. V tomto případě se používá softwarová simulace, která využívá CPU a nabízí nižší výkon. Programátorské rozhraní OpenGL je navrženo tak, aby knihovna byla použitelná v mnoha programovacích jazycích.

Tato knihovna je velice populární, a díky tomu je možné na internetu najít velké množství různých tutoriálů, hotových příkladů a řešení pro různé jazyky. Zvláště zajímavé jsou NeHe toturials [3], kde je jejich část dokonce přeložena do češtiny [4]. Menší nevýhodou může být fakt, že jsou napsány v jazyce C++. Většinu z nich je možné snadno přepsat do JOGL (vybraná vazba, viz. níže) a pro pochopení základních souvislostí to postačuje.

Pomocí funkcí OpenGL lze vytvářet tělesa a obrazce složené z elementárních geometrických prvků, jako jsou bod, úsečka, trojúhelník, čtverec a bitmapa. Tyto grafické prvky se nazývají grafická primitiva. Tyto objekty se skládají z jednotlivých vrcholů. Na tyto vrcholy lze aplikovat různé transformace, např. otočení, změna velikosti, posun. Tímto vznikají animace. Primitiva lze také různými způsoby osvětlit nebo pokrýt texturami. Tímto se budu detailněji zabývat v kapitole 4.

2.3 Grafické rozhraní pro OpenGL

Pro propojení Javy a OpenGL je možné vybírat z různých řešení (frameworků). Některá řešení poskytují skutečně jen propojení Javy a OpenGL, jiná jsou komplexnější a poskytují i podporu pro práci s různými druhy modelů, zvuky a grafického hardware.

2.3.1 GL4Java

Starší propojení Javy a OpenGL bylo v počátcích Javy nejpoužívanější. Poslední verze vyšla v roce 2001. Přímo na stránkách tohoto projektu odkazují na novější JOGL. Podporuje OpenGL 1.3 API a GLU 1.2 API mapování. Je velice podobné OpenGL z C i když v Javě musí být přístup

objektový. Pro programátora, který umí pracovat s OpenGL v C není problém přejít na GL4Java. Tento projekt je distribuován pod GNU licencí.

2.3.2 Lightweight Java Game Library

LWJGL je zaměřená na profesionální i amatérské programátory, kteří chtějí psát hry v Javě. Vývojářům poskytuje přístup k OpenGL a OpenAL (Open Audio Library). Poskytuje také snadný přístup k hardwaru jako jsou např. herní ovladače.

Není určen k tomu, aby psaní her v něm bylo nějak obzvláště snadné, spíše poskytuje technologie pro přístup ke zdrojům, které by jinak ve stávající Javě nebyly přístupné vůbec nebo jen obtížně. Tento projekt je open source, a lze ho využívat zdarma pro jakékoliv účely.

Toto propojení bylo pro účely této práce zhodnoceno jako hůře použitelné, složité a zbytečně robustní.

2.3.3 Java 3D

Java 3D je objektově orientované API, které obsahuje velkou škálu různých tříd pro práci s obrazem a zvukem. Lze v něm také do scén nahrávat již hotové modely. Je vyvíjeno komunitou a v budoucnu bude pravděpodobně zařazeno přímo do JDK. Výhodou může být, že pro něj existuje český překlad oficiálního tutoriálu.

Ukázka jednoduchého příkladu pro Javu 3D. Tento program ve vlastním okně vykreslí krychli:

```
public class Hello3D {
	public Hello3D() {
		// vytvoříme scénu
		SimpleUniverse universe = new SimpleUniverse();
		// před tím než do scény dáme objekt musíme pro něj
		// vytvoří skupinu do které bude náležet
		BranchGroup group = new BranchGroup();
		// do skupiny group přidáme kostku velikosti 0.3
		group.addChild(new ColorCube(0.3));
		// nastavíme pohled jakým do scény vidíme
		universe.getViewingPlatform().setNominalViewingTransform();
		// skupinu vložíme do scény
		universe.addBranchGraph(group);
	}

	public static void main(String args[]) {
		// nyní vytvoříme instanci třídy Hello3D
		new Hello3D();
	}
}
```

Z příkladu je patrný objektový přístup. Základem každé scény v Javě 3D je objekt typu VirtualUniverse. V tomto případě jeho podtřída SimpleUniverse, kterou je vhodné používat ve všech jednodušších programech, kde bohatě postačuje a využívá se sadněji než VirtualUniverse, který je komplexnější a práce s ním musí být ošetřena několika dalšími objekty.

Všechny objekty dávají dohromady stromovou strukturu, na jejíž počátku je právě VirtualUniverse. Ten odkazuje na objekt typu Locale, který tvoří počátek souřadné soustavy. Objekt typu Locale dále odkazuje na objekty typu BranchGroup, což už jsou jednotlivé části scény.

Při rozhodování o tom, jaké propojení bude v této práci použito, byly předchozí dvě zamítnuty již na základě obecných informací o nich. Java 3D byla nainstalována a bylo v ní napsáno několik jednoduchých programů. I když nakonec bylo použito jiné propojení z důvodů, uvedených níže, dá se říci, že Java 3D by byla také vhodná pro realizaci zadání této bakalářské práce.

2.3.4 Java Bindings for OpenGL (JOGL)

Je to jedna z novějších vazeb Javy na OpenGL. JOGL je podobný LWJGL, podporuje OpenGL verze 2.0, a také umožňuje přístup k většině OpenGL funkcí dostupných v jazyce C , s výjimkou GLUT (Java obsahuje svůj vlastní systém pro práci s okny).

Tento Projekt je vyvíjen skupinou Game Technology Group ve společnosti Sun Microsystems. Je šířen pod licencí BSD a jeho použití je tak pro soukromé i komerční účely zdarma.

JOGL byl vybrán pro realizaci projektu z důvodu, že je svojí syntaxí velmi podobný programování OpenGL v jazyku C. Pro OpenGL v C je na internetu k dispozici velké množství návodů. Většinu z nich není těžké pro JOGL přepsat.

2.4 Rozšiřující rozhraní pro OpenGL

Po nainstalování některého z rozhraní pro OpenGL je možné na toto rozhraní použít některou z již hotových nadstaveb, která mu přidá nové vlastnosti a funkce. Tyto rozhraní rozšiřují JOGL nebo LWLGL, popřípadě oboje. Většinou přidávají objektově orientované API a podporu pro nahrávání již hotových modelů z různých aplikací. Pro Javu 3D žádné významnější rozšíření nebylo nalezeno.

Java is Doomed (JID)

JID je open source 3D engine napsaný v Javě. Jako vazbu na OpenGL používá JOGL. Cílem tohoto projektu je vytvořit engine, který bude snadno použitelný i pro neprofesionální vývojáře. Balík obsahuje ModelLoader, díky kterému lze načítat modely ve formátu md2(Quake II) a 3ds(3D Studio Max). V balíku je přiložena hra ESCAPE, která je postavena na tomto enginu. Hra je inspirována známou hrou DOOM.

jMonkey Engine (JME)

JME je vysoce výkonné grafické API. Mnoho řešení použitých v JME pochází z knihy Davida Eberlyho 3D Game Engine Design. Je postaveno tak, aby plnilo požadavky, které Java i s pomocí rozhraní pro OpenGL neumožňuje. JME dříve pracovalo pouze s LWJGL, od nedávné

doby ho lze provozovat i nad JOGL. Je šířeno pod licencí BSD, a lze jej zdarma využít jak pro komerční, tak pro soukromé projekty.

Podporuje nahrávání modelů ve formátech ase, 3ds, md2, milkshape a collada.

Xith3D

Je velmi podobné výše zmíněnému Java 3D a lze si vybrat, zda poběží nad LWJGL nebo nad JOGL. Na rozdíl od Javy 3D v něm lze přímo volat funkce OpenGL. Navíc obsahuje systém pro řešení kolizí. I v něm je možné načítat mnoho různých typů modelů. Je také šířeno pod licencí BSD.

Ani jedno z těchto rozšíření nakonec nebylo použito. Jejich funkce jsou spíše vhodné pro vývojáře her, kteří potřebují pro svoji práci načítat již hotové modely. Vzhledem k zadání práce není potřeba žádné z nadstandardních funkcí. Tyto rozšíření jsou zde uvedeny spíše pro úplnost, a dále v této práci již nebudu zmiňovány.

3 JOGL

3.1 Systémové požadavky

Na počítači by měl být nainstalovaný systém Linux, MacOS, Solaris OS nebo Microsoft Windows. Pro provoz JOGLu je třeba mít nainstalovanou Javu minimálně verze 1.4.2 nebo vyšší. Konkrétní nároky na hardware počítače jsou závislé na konkrétní aplikaci. Pro málo komplikované aplikace, kde se nepoužívají žádné speciální efekty, postačí i starší grafická karta s podporou OpenGL verze 1.2. Pro vykreslování složitých scén, kde se používají nejnovější funkce rozhraní, je pak potřeba mít grafickou kartu s podporou OpenGL verze 2.0

Pro bezproblémový provoz by měla být nainstalována nejnovější verze JOGL, Javy a ovladačů grafické karty.

3.2 Instalace

Nejnovější verze je vždy k dispozici na stránkách projektu JOGL [7]. Pro úspěšnou instalaci je třeba stáhnout verzi pro systém na kterém má výsledná aplikace pracovat. Ze staženého archivu jsou především důležité soubory uložené ve složce lib. Ten obsahuje potřebné knihovny pro spouštění aplikací. Tyto knihovny je nutné nakopírovat do několika adresářů:

c:\Program Files\Java\jdk1.6.0_05\jre\lib\ext\

c:\Program Files\Java\jdk1.6.0_05\lib\ext\

c:\Program Files\Java\jre1.6.0_05\lib\ext\

Tyto cesty platí ve Microsoft Windows pro případ, kdy máte JDK i JRE nainstalováno ve standardním umístění. Pro jiné systémy je třeba si tyto cesty najít v souboru Userguide.html, který je součástí archivu.

Po provedení výše popsaných kroků je možno využívat JOGL stejně, jako běžné součásti Javy. Pokud je při vývoji použito některé z moderních vývojových prostředí pro Javu (NetBeans a Eclipse), budou v kontextové nápovědě zobrazeny funkce JOGL a automaticky doplňovány importy tak, jak to dělají pro standardní Javu.

3.3 Provoz aplikace bez nainstalovaného JOGL

V některých případech je potřeba spustit aplikaci na počítači, kde JOGL není nebo nejde nainstalovat. Tato situace může nastávat především na počítačích, kde uživatel nemá práva administrátora a není mu povolen zápis do adresáře, kde je nainstalována Java. V takovém případě lze aplikaci spustit pomocí příkazu:

```
java -Djava.library.path=./win_lib/ -jar Aplikace.jar
```

Kdy knihovny JOGL budou umístěny do podsložky win_lib dané aplikace.

Další možností je připsat do souboru manifest.mf, který se po zkompilování stává součástí Jar souboru aplikace, následující řádek:

```
Class-Path: jogl.jar gluegen-rt.jar
```

Poté nakopírujete všechny knihovny JOGL do složky s Jar souborem. Díky tomuto příkazu jsou načteny všechny třídy JOGL a aplikaci je možné spustit.

V případech, kdy je potřeba aplikaci provozovat na více operačních systémech, je vhodné v kořenovém adresáři vytvořit několik podadresářů, do kterých se umístí knihovny pro jednotlivé systémy. Do kořenového adresáře se pak pro tyto systémy umístí jejich dávkové soubory, ve kterých jsou příkazy pro JAVAC k importu knihoven.

4 Realizace modelu virtuálního stroje

4.1 JOGL – OpenGL

Grafické rozhraní JOGL poskytuje pro umístění grafiky třídu GLCanvas. Na instanci této třídy se pak veškerá grafika vykreslená pomocí OpenGL zobrazuje. Tato třída se dědí z třídy Canvas, proto je její umístění v GUI stejné.

Třída GLCanvas je pasivní, to znamená, že samotné vykreslování probíhá pouze při zpracovávání událostí. K tomu se využívá interface GLEventListener. V konstruktoru se přiřadí reference na příslušnou třídu GLCanvas, na kterou chceme vykreslovat. Samotné vykreslování se pak realizuje v metodách display() a initialize().

Třída Animator zajišťuje opakované překreslování scény třídy GLCanvas. Překreslování je zajištěno pomocí opakovaného generování událostí. Probíhá neustále, po vykonání překreslení Animator znovu zavolá metodu display(), která je naimplementována ve třídě GLEventListener, a scéna se znovu překreslí.

Při spuštění se zavolají metody reshape() a init(). V těchto metodách se poté nastaví základní vlastnosti scény (pozice pozorovatele, barva pozadí, perspektiva, osvětlení…).

4.2 Třída Scene

Třída Scene vytváří JFrame (API JavaSwing) a instance tříd GLCanvas a Animator. V metodě *main* se volá konstruktor Scene(). V konstruktoru Scene() se instance třídy GLCanvas přiřadí interface GLEventListener. Animátoru se nastaví, s jakou instancí třídy GLCanvas pracuje. Oknu JFrame je třeba nastavit velikost, titulek, umístění a standardní operaci pro ukončení. Oknu se nastaví viditelnost a spustí se cyklické překreslování scény (viz. příklad níže).

```
public class Scene extends JFrame
{
GLCanvas canvas = new GLCanvas();
Animator animator = new Animator();
        public static void main(String[] args)
{
                new Scene();
        }

         public  Scene()
        {
                canvas.addGLEventListener(new Renderer());
                animator.add(canvas);
                this.getContentPane().add(canvas);
                this.setTitle(" EPSNET Robot");
                this.setSize(1024, 768);
                this.setLocationRelativeTo(null);
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setVisible(true);
                animator.start();
                canvas.requestFocus();
        }
}
```

4.3 Třída Renderer

Třída Renderer implementuje rozhraní GLEventListener, které řeší události. Ve třídě Renderer je definována celá scéna, která se vykresluje.

```
public class Renderer implements GLEventListener
{
private GLU glu = new GLU();
        private GL gl;
//… (další kód)
}
```

Metoda displayChange() zůstane prázdná, je nutné ji ale implementovat, protože je definována v rozhraní GLEvent Listener.

```
public void displayChanged(GLAutoDrawable gLDrawable, boolean modeChanged, boolean deviceChanged)
    {
    }
```

V metodě Init() se nastavují základní parametry scény, protože je volána jako první po vytvoření okna. Nastavuje se zde přístup k OpenGL. Je vhodné právě zde nastavit např. barvu pozadí. Tu v průběhu vykreslování již většinou není potřeba měnit. Barva má čtyři parametry, první tři jsou jednotlivé složky RBG, poslední nastavuje průhlednost, která v případě barvy pozadí nemá význam.

Při práci s prostorovými objekty je nutné nastavit testování hloubky. To zajišťuje, aby objekty bližší překrývali objekty vzdálenější a zapne se korekce perspektivy.

```
public void init(GLAutoDrawable gLDrawable)
    {
        this.gl = gLDrawable.getGL();
        gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        gl.glEnable(GL.GL_DEPTH_TEST);
        gl.glDepthFunc(GL.GL_LEQUAL);
        gl.glHint(GL.GL_PERSPECTIVE_CORRECTION_HINT, GL.GL_NICEST);
    }
```

Metoda Reshape() je volána vždy při změně velikosti okna, ve kterém je scéna vykreslována, a při jejím prvním vykreslení. V této metodě se nastavuje perspektiva dle aktuální velikosti okna. GL_PROJECTION nastavuje, jak moc se budou vzdálenější objekty zmenšovat vůči těm blízkým. Metodou gluPerspective se nastavují ořezávající roviny. Pokud nějaký objekt překročí tuto minimální nebo maximální vzdálenost od pozorovatele, jednoduše nebude zobrazen.

```
public void reshape(GLAutoDrawable gLDrawable, int x, int y, int width, int height)
   {
      if (height <=0)
            height=1;
      final float h = (float)width / (float)height;
      gl.glViewport(0, 0, width, height);

      gl.glMatrixMode(GL.GL_PROJECTION);
      gl.glLoadIdentity();
      glu.gluPerspective(45.0f, h, 1.0, 30.0);
      gl.glMatrixMode(GL.GL_MODELVIEW);
      gl.glLoadIdentity();

   }
```

Metodu Display() cyklicky volá pro zobrazení každého snímku (framu) Animator. Je zde definována celá scéna. Nejdříve je nutné vymazat hloubkový zásobník (buffer) a resetovat matici. Dále je třeba posunout scénu o určitou vzdálenost dozadu. Důvodem je ořezávací matice, která nezobrazí objekty kameře bližší, než jaká je její hodnota. Pokud bychom měli nastavenou ořezávací matici na 1, pak je třeba celou scénu posunout minimálně o více jak -1 dozadu. V uvedeném příkladě se posouvá o -14.

```
public void display(GLAutoDrawable gLDrawable)
{

gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

         gl.glLoadIdentity();
         gl.glTranslatef(1f, 1f, -14f);
```

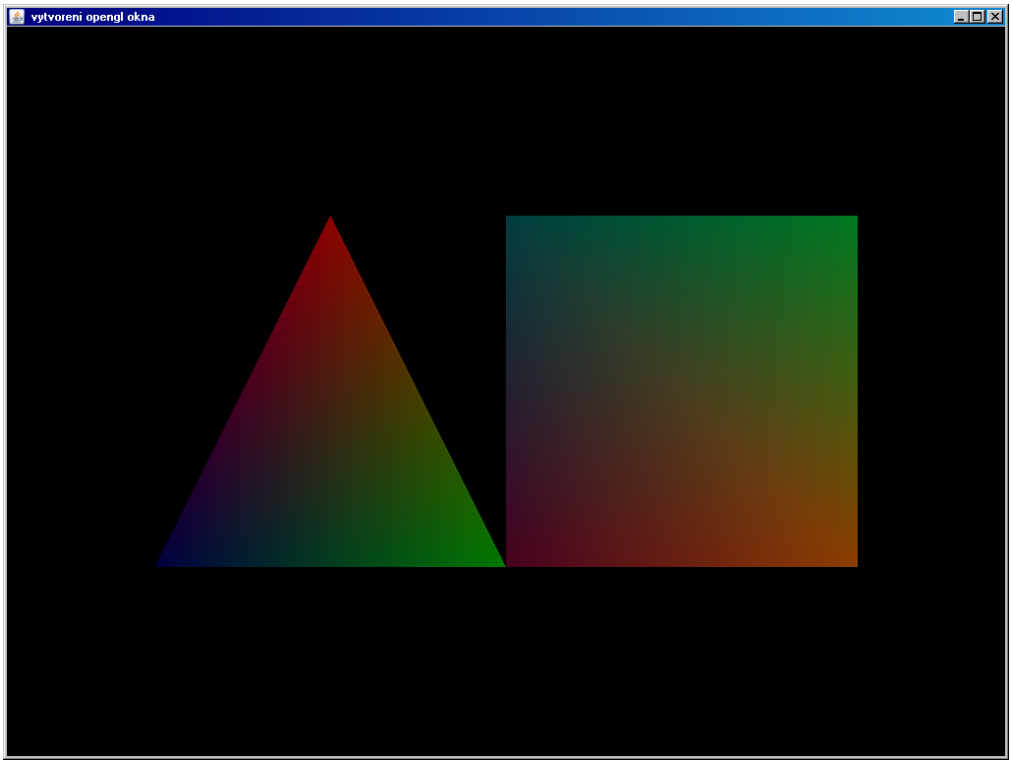
Před samotným kreslením objektů je třeba zvolit druh objektu který bude vytvářen. V OpenGL je na výběr několik základních prvků, které popisuje tabulka č. 1.

GL_POINTS	Nejjednodušší, tvoří pouze body. K zadání jednoho bodu je třeba jeden příkaz. Bod je zobrazen ve velikosti jednoho pixelu. Velikost lze změnit.
GL_LINES	Vznikají úsečky. K zadání jedné úsečky je potřeba dvou příkazů glVertex
GL_LINE_STRIP	Lomená úsečka, počáteční dva body definují první úsečku. Každý další bod přidává další segment.
GL_LINE_LOOP	Úzce souvisí s předchozím, pouze s tím rozdílem, že se poslední bod navíc propojí s prvním.
GL_TRIANGLES	Nejjednodušší typ plochy. Ze tří bodů se vykreslí trojúhelník.
GL_TRIANGLE_FAN	První bod je vrchol, každé další dva body vykreslí trojúhelník s tímto společným vrcholem. Používá se např. pro vykreslení vrchlíku koule.
GL_TRIANGLE_STRIP	První tři vrcholy definují trojúhelník, každý další bod vykreslí další trojúhelník. Použije pro to předchozí dva body. Často se využívá pro kreslení složitějších těles.
GL_QUADS	Slouží k vykreslování čtyřúhelníků.
GL_QUAD_STRIP	Jedná se o pás čtyřúhelníků. První čtyři body vykreslí čtyřúhelník, každé další dva pak také. Využijí pro to předchozí dva body.
GL_POLYGON	Vykreslí plochu složenou z libovolného množství bodů.

Tab. 1: Možnosti kreslení v OpenGL

Nejdříve je nutné aktivovat vykreslování a zvolit druh objektu, který se bude kreslit. Poté se nastavuje barva vrcholu. Barva má tři parametry. Jedná se o jednotlivé složky RBG, které nabývají hodnot <0;1>. Dále se definuje vrchol, jehož parametry jsou souřadnice (X,Y,Z). Kreslení je třeba také vždy ukončit.

```
gl.glBegin(gl.GL_QUADS);
        gl.glColor3d(0.0, 0.5, 1.0);// Barva levého horního bodu
        gl.glVertex3f(-1.0f+1, 1.0f, 0.0f);// Levý horní bod
        gl.glColor3d(0.0, 1.0, 0.5);//Barva pravého horního bod
        gl.glVertex3f( 1.0f+1, 1.0f, 0.0f);// Pravý horní bod
        gl.glColor3d(1.0, 0.5, 0.0);//Barva Pravého dolního bodu
        gl.glVertex3f( 1.0f+1,-1.0f, 0.0f);// Pravý dolní bod
        gl.glColor3d(0.5, 0.0, 0.5);//Barva levého dolního bodu
        gl.glVertex3f(-1.0f+1,-1.0f, 0.0f);// Levý dolní bod
gl.glEnd();

gl.glBegin(gl.GL_TRIANGLES);
        gl.glColor3d(1.0, 0.0, 0.0);
        gl.glVertex3f( 0.0f-1, 1.0f , 0.0f);
        gl.glColor3d(0.0, 0.0, 1.0);
        gl.glVertex3f(-1.0f-1,-1.0f, 0.0f);
        gl.glColor3d(.0, 1.0, 0.0);
        gl.glVertex3f( 1.0f-1,-1.0f, 0.0f);
gl.glEnd();
```



Obr. 1: Okno, které se vykreslí

4.4 Pohyby a rotace objektů

V předchozí části bylo popsáno, jak vykreslit nějaký objekt. Při vytváření jakékoliv scény je ale neméně důležité, jak takto vykreslené objekty rozpohybovat. Ve stručnosti je zde popsáno, jak na pohyby a rotaci vykreslených objektů.

Pro přemístění objektů, popřípadě pro změnu jejich velikosti nebo tvaru, stačí přidat atribut, který bude vyjadřovat, o kolik se má určitý vrchol objektu přemístit. Při volání metody glVertex3f, pak v parametrech tento atribut přičítá k požadované souřadnici. Pokud je toto uděláno pro všechny vrcholy objektu, objekt se přemístí. V případě pouze některých vrcholů objekt změní tvar. Při použití součinu místo součtu, objekt změní svoji velikost.

Pokud je potřeba, aby se nějaký objekt scény otáčel, stačí přidat jeden atribut třídy, nejlépe typu float nebo double. V tomto atributu bude uloženo, o kolik stupňů se má objekt oproti základnímu zobrazení natočit.

```
float otoceni = 0;
```

Samotná rotace se pak provede pomocí metody glRotatef (otoceni, x, y, z). Souřadnice x, y, z udávají směr osy, podle které se bude objekt otáčet, nabývají hodnot 0 – 1. Volání metody glRotatef je nutné umístit před počátek samotného vykreslování, tzn. před volání metody glBegin. Po ukončení vykreslování jednoho objektu je možné metodu glRotatef opět zavolat a nastavit pro další objekt rotaci jinou, popřípadě žádnou. Pokud do metody display() přidáme inkrementaci proměnné, která určuje úhel otočení, bude vykreslený objekt rotovat.

```
otoceni += 1;
```

4.5 Textury

Proto, aby model vypadal realisticky, je vhodné pro velké jednobarevné plochy použít textury. Texturování je technika, která umožňuje dodat realistický vzhled virtuálnímu trojrozměrnému modelu. Z hlediska způsobu vytváření lze textury rozdělit do dvou skupin.

Procedurální textury jsou vytvářeny pomocí nějaké matematické funkce. Jejich výhodou je, že u nich nezáleží na tom jak velkou plochu pokrývají, vypadají stále stejně. Nevýhodou je, že ne každý druh povrchu, který chceme vytvořit lze napodobit nějakou matematickou funkcí.

Rastrovou texturu tvoří předem připravený rastrový obrázek. Proto, aby tyto textury vypadaly dobře, je důležité, aby měly obrázky dostatečné rozlišení. Zároveň ale příliš detailní textury budou neúměrně zatěžovat CPU, a počet snímků vykreslených za sekundu bude malý, a model pak nebude působit příliš plynule. Rastrové textury se používají mnohem častěji než procedurální. V této práci byly využity pouze textury rastrové.

Proto, aby bylo možné v JOGL otexturovat nějakou plochu, je třeba dopsat do třídy Renderer několik metod. Samotné použití textur při vykreslování v metodě Display() je celkem triviální. Nejdříve je nutné tyto textury načíst z předpřipravených obrázků a zpracovat je do požadovaného formátu.

Do atributů třídy Renderer se přidají dvě konstanty, ve kterých je uložen počet textur, které budou načítány, a cesty s názvy souboru, ve kterých jsou uloženy. Dále pole, ve kterém budou uloženy indexy textur.

```
private final String textureFile = "epsnetrobot/tex.png";
private final int texturesCount = 1;
private int[] texture = new int[texturesCount];
```

Do metody init (třídy Renderer) se doplní volání metody pro generování textur a metody glEnable s parametrem(GL_TEXTURE2D), která v OpenGL aktivuje použití 2D textur.

```
generateTextures();
gl.glEnable(GL.GL_TEXTURE_2D);
```

V metodě generateTextures je volána metoda glGenTextures, které se jako parametr předává kolik textur bude načítáno, dále proměná do které se textury budou ukládat a offset textury. Voláním metody glBindTexture jsou jako parametr předány GL_TEXTURE2D, čímž se nastaví práce s 2D texturou, a jako další parametr index textury, která bude načítána. Poté je textura načtena, a dále pomocí volání metod glTexParametri se v jejích parametrech nastaví jaké filtry budou pro danou texturu použity. Parametr GL_TEXTURE_MAG_FILTER nastavuje, jaký filtr bude použit pro plochu, která je větší než originální bitmapa. Parametr

GL_TEXTURE_MIN_FILTER dělá to samé pro plochy menší. V této práci je použit filtr GL_LINEAR, jehož efektem je že textura vypadá dobře v obou případech. Tento filtr je ale náročný na výkon CPU a grafické karty. Při problémech s výkonem je vhodné použít filtr GL_NEAREST, který není tak náročný. Při roztažení textury na velkou plochu budou na textuře patrné jednotlivé body originální bitmapy.

```
private void generateTextures()
   {
       gl.glGenTextures(texturesCount, texture, 0);
       gl.glBindTexture(GL.GL_TEXTURE_2D, texture[0]);
       Texture newTexture = null;
      try
      {
         newTexture = readTexture(textureFile);
      }
       catch (IOException e)
       {
          e.printStackTrace();
          throw  new RuntimeException(e);
       }
      makeRGBTexture(gl, glu, newTexture1, GL.GL_TEXTURE_2D);
            gl.glTexParameterf(GL.GL_TEXTURE_2D,GL.GL_TEXTURE_MIN_FILTER,
      GL.GL_LINEAR);
      gl.glTexParameterf(GL.GL_TEXTURE_2D,GL.GL_TEXTURE_MAG_FILTER,
      GL.GL_LINEAR);
       }
```

Metoda makeRBGTexture() vytváří z textury, která je uložená v instanci třídy Texture texturu ve formátu, který je nutný pro správné načtení v OpenGL.

```
private void makeRGBTexture(GL gl, GLU glu, Texture img, int target)
   {
          gl.glTexImage2D(target, 0, GL.GL_RGB, img.getWidth(), img.getHeight(), 0, GL.GL_RGB,
          GL.GL_UNSIGNED_BYTE, img.getPixels());
   }
```

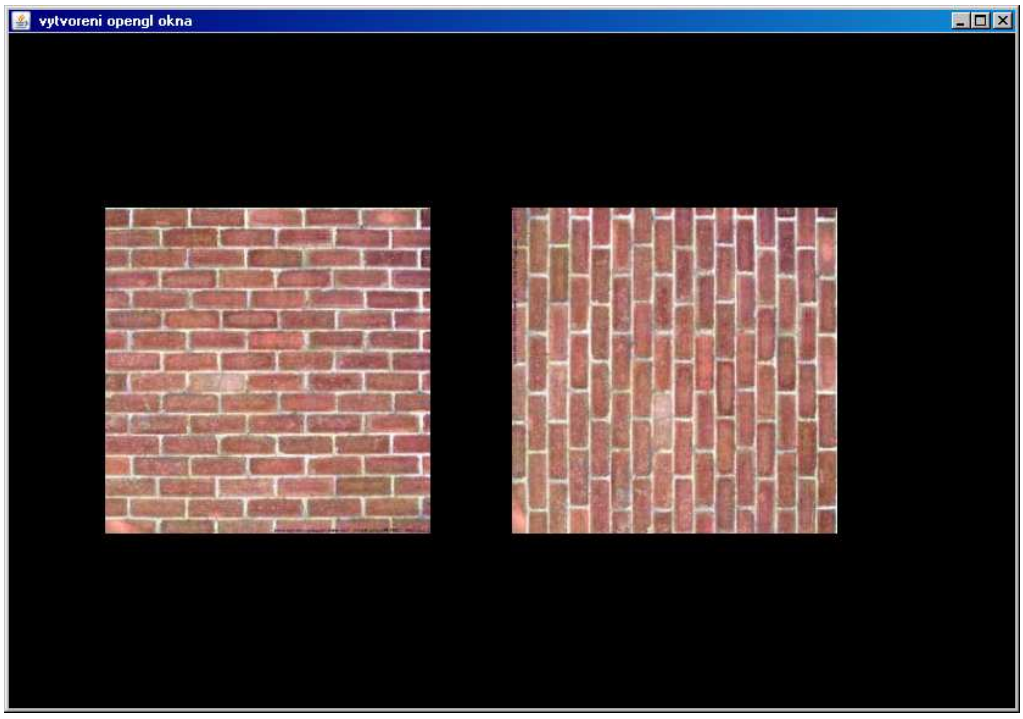
Třída Texture se používá pro dočasné uchování textury od doby jejího načtení do doby, kdy je v metodě makeRGBTexture skutečně načtena pomocí metody glTexImage2D().

```
private class Texture
  {
     private ByteBuffer pixels;
     private int width;
     private int height;

     public Texture(ByteBuffer pixels, int width, int height)
     {
        this.height = height;
        this.pixels = pixels;
        this.width = width;
     }
     public int getHeight()
     {return height;}

     public ByteBuffer getPixels()
     {return pixels;}

     public int getWidth()
     {return width;}
  }
```

V metodě display() pak už lze s texturami pracovat. Na jednoduchém příkladu je demonstrováno použití textury, na dvou objektech stejného tvaru, s rozdílným namapováním.

Před začátkem kreslení je nejdříve potřeba nastavit texturu, která bude používána. To nastaví metoda glBindTexture. Před vykreslením každého vrcholu je nyní nutné uvést texturovací souřadnici. Tímto se určuje, jak bude textura na objekt nanesena. Následující příklad názorně ukazuje použití textur, včetně různého namapování.

```
gl.glBindTexture(GL.GL_TEXTURE_2D, texture[0]);
gl.glBegin(gl.GL_QUADS);
        gl.glTexCoord2f(0.0f, 0.0f);
        gl.glVertex3f(-1.0f+1, 1.0f, 0.0f);
        gl.glTexCoord2f(0.0f, 1.0f);
        gl.glVertex3f( 1.0f+1, 1.0f, 0.0f);
        gl.glTexCoord2f(1.0f, 1.0f);
        gl.glVertex3f( 1.0f+1,-1.0f, 0.0f);
        gl.glTexCoord2f(1.0f, 0.0f);
        gl.glVertex3f(-1.0f+1,-1.0f, 0.0f);
gl.glEnd();

gl.glBegin(gl.GL_QUADS);
        gl.glTexCoord2f(1.0f, 1.0f);
        gl.glVertex3f(-1.0f -1.5f, 1.0f, 0.0f);
        gl.glTexCoord2f(0.0f, 1.0f);
        gl.glVertex3f( 1.0f -1.5f, 1.0f, 0.0f);
        gl.glTexCoord2f(0.0f, 0.0f);
        gl.glVertex3f( 1.0f -1.5f,-1.0f, 0.0f);
        gl.glTexCoord2f(1.0f, 0.0f);
        gl.glVertex3f(-1.0f -1.5f,-1.0f, 0.0f);
gl.glEnd();
```



Obr. 2: Různé namapování textur

5 EPSNET

EPSNET je protokol určený pro komunikaci mezi PLC a okolím. Jako okolí jsou myšleny další PLC, různé senzory nebo také počítače. Tento protokol ve svých PLC využívá Kolínská firma Teco a.s.

EPSNET je otevřený, veřejně přístupný pro každého, bez nutnosti platit za jeho využívání nějaké poplatky. Tyto vlastnosti ho odlišují od mnohé konkurence.

5.1 Možnosti komunikace

V síti ESPNET existují dva druhy stanic. Jedná se o nadřízené stanice, které vysílají požadavky (to jsou aktivní účastníci) a podřízené stanice, které naslouchají, a pokud je požadavek cílen na ně tak odpovídají (pasivní účastníci).

Existují dva základní modely rozdělení nadřízených a podřízených. Ten častější je, že nadřízený účastník komunikace je pouze jeden a má pod sebou více podřízených tzv. monomaster. Počet podřízených účastníků je teoreticky 126, v praxi je to ale omezeno použitým přenosovým médiem. Komunikace probíhá pouze mezi nadřízeným a podřízeným účastníkem. V případě, že je třeba předat informace mezi dvěma podřízenými stanicemi, musí tato výměna projít přes nadřízenou stanici.

Tato varianta propojení se často používá pro komunikaci mezi PLC a počítačem, na kterém běží nějaký vizualizační nebo dispečerský software.

Druhá, komplikovanější, ale také komplexnější možnost je, že je více jak nadřízených účastníků, tak i podřízených tzv. multimaster. Počet nadřízených stanic není omezen, ale v této konfiguraci spolu může být propojeno maximálně 127 nadřízeních i podřízeních zařízení dohromady. Většinou je skutečným omezením opět přenosové medium. Komunikaci zde po určitý čas vždy řídí jedna nadřízená stanice, a všechny ostatní stanice, včetně nadřízených, se chovají jako podřízené. Po vyřízení všech jejích požadavků předá řízení další stanici, která je nadřízená. Díky tomu lze snadno realizovat výměnu informací i mezi nadřízenými stanicemi navzájem.

S protokolem je možné pracovat pomocí několika komunikačních rozhraní. Při výběru které bude v této práci použito, bylo především zváženo, jak je dané rozhraní dostupné v Javě.

USB

Rozhraní USB lze použít pouze pro propojení dvou zařízení, které slouží spíše k servisu a naprogramování PLC. Skrze toto rozhraní se komunikuje vždy v režimu PC, kdy logická adresa zařízení je 0. Není určeno pro trvalé propojení, proto o jeho využití nebylo ani uvažováno.

Sériová linka

Sériová linka je základní a často používané rozhraní pro komunikaci s PLC. Pro účely této práce jí nebylo nepoužito, protože při vývoji bude využíván software SoftPLC for Windows, který bude simulovat přítomnost skutečného PLC. Tento program bude provozován na stejném počítači jako vlastní aplikace. Z tohoto důvodu by musel být nainstalován ještě speciální software, který v počítači vytváří virtuální sériový port. Tento druh propojení by bylo možné vytvořit, ale realizovat přístup k sériovému portu v Javě je složitější, než níže uvedená třetí možnost komunikace.

Ethernet

Pomocí sítě Ethernet lze snadno propojit počítač s PLC. Lze ho využít pouze u zařízení, která ho podporují.

Toto rozhraní bylo vybráno pro realizaci této bakalářské práce, protože má hned několik výhod. V Javě se poměrně snadno realizuje komunikace s okolím pomocí IP paketů, a také je snadné provozovat obě aplikace na jednom počítači. Nastaví se adresa 127.0.0.1 (localhost) a oba programy spolu bez problémů komunikují.

Název režimu	Provoz na portu	Protokol	Aktivace
PC	61682	EPSNET UDP	Trvale
PLC	61681	EPSNET UDP	Inicializace uživatelským programem
UNI	Volitelně	Obecný UDP a TCP	Inicializace uživatelským programem
MDB	502	MODBUD UDP MODBUD TCP	Trvale

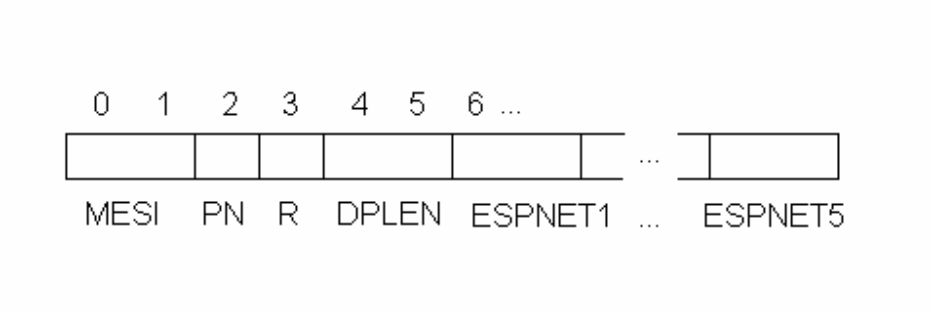
Tab. 2: Dostupné režimy komunikace pomocí Ethernetu

Pro komunikaci byl vybrán režim PC z toho důvodu, že není nutné jej aktivovat uživatelským programem. Navíc v režimu PC lze provádět některé operace, které jsou v jiných režimech zakázané.

5.2 Hlavička paketu pro EPSNET UPD

Využívá se pro přenos po Ethernetu pomocí protokolu UDP. Tento protokol je někdy nazýván jako nespolehlivý. Je totiž na rozdíl od TCP nespojovaný. UDP paket se odešle na cílovou IP adresu, a už není zjištěno zda daný paket skutečně došel. Při komunikaci pomocí ESPNET ale PLC vždy po obdržení paketu odpoví, ať už potvrzením nebo hlášením o chybě. Tento problém lze díky tomu řešit tak, že pokud nepřijde žádná odpověď do stanovené doby, paket se jednoduše odešle znovu. UDP pakety pro protokol ESPNET v režimu PC se vždy odesílají na port 61682, a pro režim PLC na port 61681.

Datová oblast každého UDP paketu, který se odesílá, musí obsahovat 6 bitů, které tvoří hlavičku protokolu ESPNET. Za hlavičkou už následují jednotlivé příkazy protokolu. Do jednoho paketu lze umístit až 5 příkazů ESPNET, čímž lze zrychlit vzájemnou komunikaci. Pokud je v paketu více příkazů, vykonají se podle pořadí, ve kterém přišli.



Obr. 3: Pořadí bitů v hlavičce ESPNET UDP

Pořadí bitů	Název bitu	Význam bitu
0 a 1	MESI	Slouží k jednoznačné identifikaci jednotlivých paketů. Odpověď od podřízeného účastníka má tyto bity stejné. Odpověď tak není možné zaměnit s odpovědí účastníka jiného.
2	PN	Určuje režim komunikace. Hodnota 2 znamená režim PC, hodnota 3 pak režim PLC.
3	R	Tento bit slouží jako rezerva. Nemá žádné využití. Jeho hodnotu lze ponechat rovnou 0
4 a 5	DPLEN	Určuje délku dat následujících za hlavičkou. Slouží jako kontrola, že paket došel celý a žádná část se neztratila. Bit 4 je více významný, a bit 5 méně významný(tzv. konvence Motorola).

Tab. 3: Význam jednotlivých bitů hlavičky ESPNET UDP

Příklad hlavičky, kde MESI = 255, režim komunikace je PC a délka dat následujících za hlavičkou je rovna 184. Znak $ se využívá pro vyjádření hodnoty v hexadecimální soustavě, tak jak je to časté u programovacích jazyků typu asembler.

$0 $EE $2 $0 $0 $B8

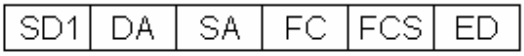
Celková délka dat v paketu musí být vždy sudá. Pokud DPLEN vyjde jako liché číslo, musí být délka paketu prodloužena o jeden nulový bit. Hodnotu DPLEN není nutné měnit, může být ponechána lichá.

5.3 Obecná struktura příkazů EPSNET

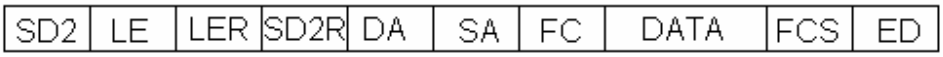
ESPNET má dvě skupiny příkazů. Veřejné komunikační služby sloužící k výměně dat a systémové služby, které slouží s ladění a programování. Pro tuto práci jsou důležité pouze některé komunikační služby.

Druhy možných příkazů ve směru od nadřízené stanice k podřízené:

a) zpráva bez datového pole



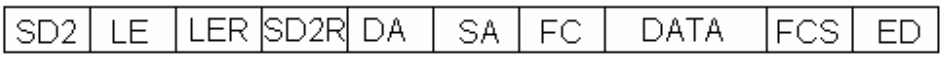
b) zpráva s datovým polem



Druhy možných odpovědí ve směru od podřízené stanice k nadřízené:

a) krátká pozitivní odpověď

b) pozitivní odpověď s datovým polem



Žádná odpověď znamená buď přerušené spojení nebo špatně sestavenou hlavičku EPSNET UDP paketu.

Vysvětlivky názvů, významu a hodnot jednotlivých bitů:

SD1 - úvodní znak u zpráv bez datových polí, pevná hodnota $10

SD2 – úvodní znak pro zprávy které obsahují nějaká data, pevná hodnota $68

LE – počet bitů položek DA, SA, FC a DATA

LER – opakování hodnoty LE

SD2R – opakování hodnoty SD2

DA – adresa stanice, pro kterou je zpráva určena

SA – adresa stanice, od které zpráva pochází

FC – pro každý příkaz rozdílná hodnota

DATA – datové tělo zprávy, obsahuje přenášené informace

FCS – kontrolní součet. Suma položek DA, SA, FC a DATA. Pokud je větší než 256, jeho výsledek je zbytek po celočíselném dělení.

ED – koncový znak, pevná hodnota $16

SAC – krátké pozitivní potvrzení, pevná hodnota $E5

5.4 Příkaz CONNECT

Tento příkaz slouží k navázání spojení

Od nadřízeného účastníka k podřízenému:

SD1	DA	SA	FC	FCS	ED

FC= $69

Odpověď podřízeného:

SD1	DA	SA	FC	FCS	ED

FC= $0

Příklad: nadřízený s adresou $10 komunikuje s podřízeným s adresou $1

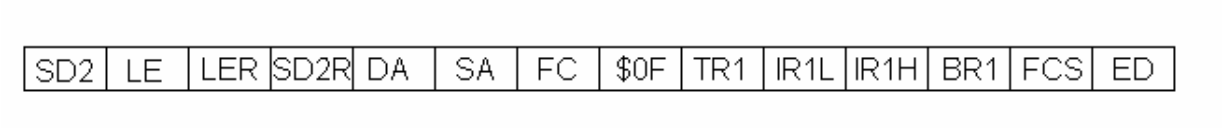
Nadřízený - $10 $1 $10 $69 $50 $16

Podřízený - $10 $10 $1 $0 $11 $16

5.5 Příkaz READB

Příkaz sloužící k čtení jednotlivých bitů z registru PLC

Od nadřízeného účastníka k podřízenému:



SD2	LE	LER	SD2R	DA	SA	FC	$0F	TR1	IR1L	IR1H	BR1	FCS	ED

FC= $6C

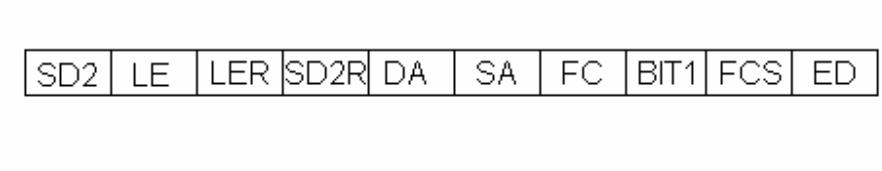
TR1 - oblast ze které je čteno 0- registr X , 1 – Y , 2 – S , 3 - R

IR1L – dolní byte čteného registru

IR1H – horní byte čteného registru

BR1- index čteného bitu

Odpověď podřízeného:



SD2	LE	LER	SD2R	DA	SA	FC	BIT1	FCS	ED

Příklad: čtení bitů Y3.0 a Y 3.1 , nadřízená stanice má adresu 10 a podřízená 1

Nadřízený - $68 $C $C $68 $1 $10 $6C $0F $1 $3 $0 $0 $1 $3 $0 $1 $8F $16

Podřízený - $68 $5 $5 $68 $10 $1 $8 $FF $0 $16

Odpověď podřízeného znamená, že v bitech Y3.0 je uložena log. 1 a v Y3.1 je uložena 0

5.6 Příkaz WRITEB

Tento příkaz slouží k přepisování jednotlivých bitů v registru PLC

Od nadřízeného účastníka k podřízenému:

SD2	LE	LER	SD2R	DA	SA	FC	$10	TW1	IW1L	IW1H	BW1	FCS	ED

FC=$63

TW1 – oblast, do které je zapisováno 0- registr X , 1 – Y , 2 – S , 3 –R

IW1L – dolní byte přepisovaného registru

IW1H – horní byte přepisovaného registru

BW1 – v první pozici hexadecimální hodnoty uložena zapisovaná hodnota, na druhé pozici pak index bitu, do kterého se zapisuje. Zápis 0 – $00 až $07, zápis 1 - $80 až $87.

Odpověď podřízeného:

SAC

Příklad: Do bitu X3.0 nastavíme do log. 1 a bit X3.1 vynulujeme. Nadřízená stanice má adresu 10 a podřízená 1

Nadřízený - $68 $C $C $68 $1 $10 $63 $10 $0 $3 $0 $80 $0 $3 $0 $1 $5 $16

Podřízený - $E5

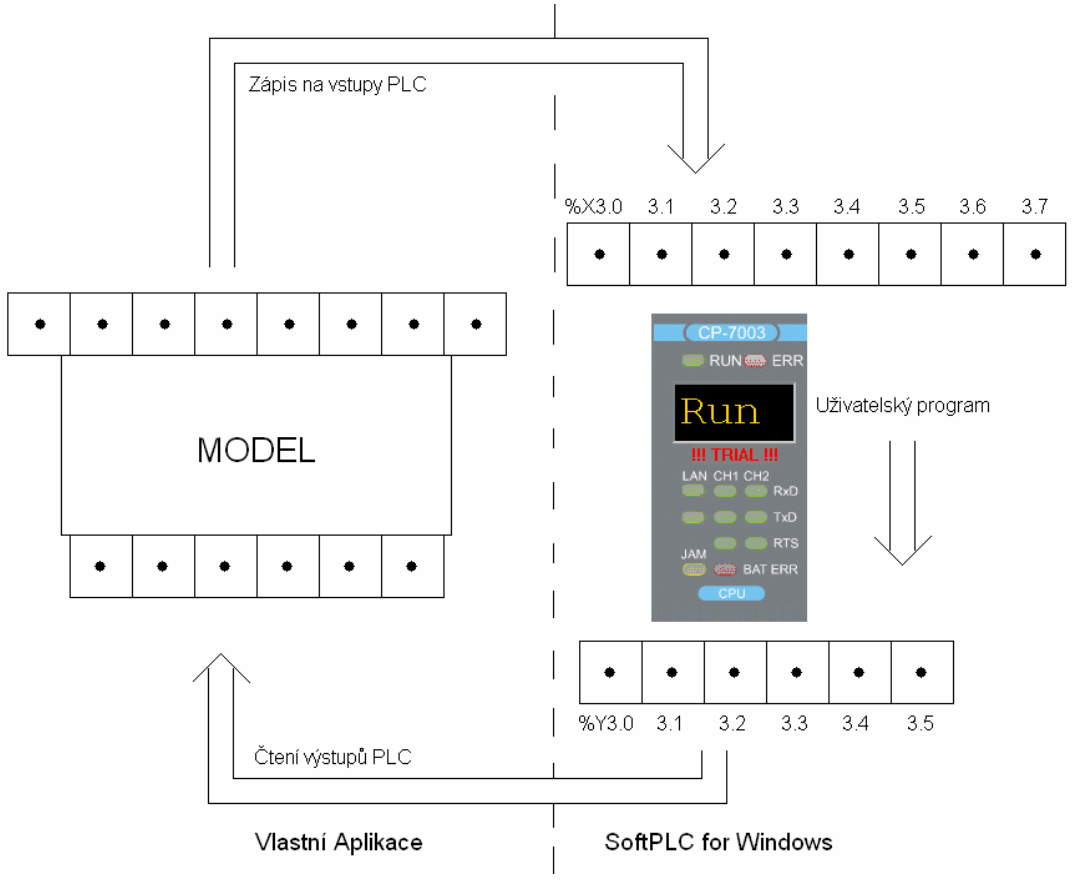
5.7 Další možnosti EPSNET

Protokol EPSNET kromě příkazů, které jsou zde podrobně popsány z důvodu, že se s nimi v práci pracuje, obsahuje ještě několik dalších. Umožňuje čtení a zápis do paměti po celých bytech, podporuje čtení i zápis do datové paměti pomocí jednoho příkazu. Tento příkaz má ale složitější syntaxi. Dále umí destruktivně číst jednotlivé bity nebo i datovou paměť (po přečtení dochází k vynulování). Lze také pomocí příkazu na dálku změnit čas v PLC. Všechny tyto příkazy jsou zde zmiňovány pouze ve zkratce, protože jsem jejich možností nebylo využito.

6 Implementace

6.1 Koncepce řešení

Model má komunikovat s virtuálním PLC prostřednictvím sítě EPSNET přes protokol UDP. Cyklicky zapisuje stavy svých výstupů do vstupního registru PLC (%X3.x). Z informací zapsaných ve vstupním registru a dle uživatelského programu, který je v PLC nahraný, se nastaví výstupní registr. Aplikace poté tyto informace přečte, a do doby dalšího čtení je model podle nich řízen.



Obr. 4: Komunikace programu s PLC

PLC má registry X a Y, ve kterých jsou uloženy stavy jeho vstupů a výstupů. Při připojení aplikace k reálnému přístroji s osazenými moduly se vstupy a výstupy mají tyto moduly část paměti vyhrazenou pro sebe. Paměť bývá přidělována od začátku (X0.0, Y0.0). Aby se předešlo

případnému konfliktu a program nepřepisoval vstupy již využité pro jinou aplikaci, je zaveden offset. Aplikace proto pracuje s registry až na adrese X3.0 a Y3.0.

6.2 Model

Vlastní model je realizován podle popisu v kapitole 4. Většina jeho viditelných ploch je z objektů typu GL_QUAD. Pro realistický vzhled jsou viditelné plochy otexturovány. Je navržen tak, aby jeho vzhled připomínal průmyslový manipulátor. Má tři osy volnosti.

Před začátkem vlastního vykreslování se převádějí údaje, které uživatel zadá při startu programu jako referenční body z hodnot, které jsou snadno srozumitelné(0 až 100) na skutečné souřadnice které jsou v modelu použity pro vykreslování v 3D prostoru, např.:

```
lLimit[i]= ((Scene.points[i])/(50f))-(2.0f)-(0.1f);
```

Při realizaci pohybů po osách je nutné ošetřit koncové body na kterých se pohyb po osách 2 a 3 zastaví, např.:

```
if (deltaY > 2.5 ) deltaY=2.5f ;
if (deltaY < 0 ) deltaY=0.0f ;
```

Osa 1 nijak limitována není, model jde otáčet o 360°. Pro správné vyhodnocení polohy robota na ose 1 je nutné ošetřit interval, na kterém se robot pohybuje. Parametr, který řídí úhel natočení robota, může z důvodu porovnávání s referenčními body nabývat pouze hodnot od 0 do 359.

```
if (rotateY>360 ) {rotateY=rotateY%360;}
if (rotateY<0) {rotateY=360 - Math.abs(rotateY);}
```

Model je možné ovládat nejen pomocí sítě EPSNET. Lze jej také ovládat ručně pomocí klávesnice. Tato funkce může být pomocná při ladění uživatelského programu PLC.

Ovládání pomocí klávesnice je realizováno pomocí rozhraní KeyListener, které implementuje třída Renderer. Klávesy „vlevo“ a „vpravo“ pohybují modelem po ose 1, klávesy „nahoru“ a „dolů“ po ose 3. Klávesy Page Up a Page Down ovládají osu 2 (osy robota popisuje obr. 9). Program lze ukončit klávesou Escape.

Metody keyReleased a keyTyped jsou prázdné. Je nutné je ošetřit, protože jsou implementovány rozhraním KeyListener.

```
public class Renderer implements GLEventListener, KeyListener
{
…
public void keyPressed(KeyEvent e)
{
if (e.getKeyCode() == KeyEvent.VK_ESCAPE)
System.exit(0);

if (e.getKeyCode() == KeyEvent.VK_LEFT)
rotateY-= 0.8;

if (e.getKeyCode() == KeyEvent.VK_RIGHT)
rotateY+= 0.8;

if (e.getKeyCode() == KeyEvent.VK_DOWN)
        deltaY-= 0.05;

if (e.getKeyCode() == KeyEvent.VK_UP)
        deltaY+= 0.05;

if (e.getKeyCode() == KeyEvent.VK_PAGE_DOWN)
        deltaZ+= 0.05;

if (e.getKeyCode() == KeyEvent.VK_PAGE_UP)
        deltaZ-= 0.05;
}
  public void keyReleased(KeyEvent e) {}
   public void keyTyped(KeyEvent e) {}
}
```

6.3 Třída UDP

Model má komunikovat s PLC pomocí sítě EPSNET. Pro tuto komunikaci byl v kapitole 5 zvolen protokol UDP před sériovou linkou a USB rozhraním. Protokol UDP je na rozdíl od TCP nespojovaný, ale také nespolehlivý.

Díky tomu, že je tento protokol nespojovaný, je o něco snadnější jej naimplementovat proti TCP. Tato vlastnost má i své stinné stránky. Nelze zaručit, že odeslaný paket skutečně

k příjemci dorazil. Může se také stát, že pakety přijdou v jiném pořadí než v jakém byly odeslány, popřípadě může jeden přijít vícekrát.

EPSNET funguje na principu dotaz – odpověď. Po odeslání dotazu se čeká na odpověď. Pokud odpověď nedorazí po uplynutí určité doby (uživatel v programu nastavuje dobu prodlení - timeout), považuje se paket za ztracený, a program už na něj neočekává odpověď. V tomto případě program pokračuje dále v běhu a předpokládá se, že v příštím cyklu se již paket dotazu neztratí. Z tohoto vyplívá, že pro provoz EPSNET je nutná kvalitní síť.

V Javě mají na starost práci s protokolem UDP třídy DatagramSocket a DatagramPacket. Třída DatagramPacket představuje UDP paket. Poskytuje přístup k IP adrese, portu, délce dat a k samotným datům. Konstruktor DatagramPacket(packetBytu, packetDelka) vytvoří paket nad polem bytů a nastaví délku paketu. Pokud bude pole kratší, doplní se nulami. Pokud bude delší , bude oříznuto.

Třída DatagramSocket odesílá a přijímá pakety. K vytvoření socketu je využit jeho bezparametrový konstruktor. K odesílání slouží metody *send*(DatagramPacket) a k naslouchání, zda nějaký paket nepřijde slouží metoda *receive*(DatagramPacket). Kam se má paket odeslat (na jakou IP adresu a port) je uloženo ve vlastnostech paketu. Pokud už daný socket nebude potřeba, lze ho uzavřít metodou *close*().

V následujícím příkladu je pole packetWrite naplněno pomocí metody packetWrite(zapis). Nad oběma datovými poli jsou postaveny pakety. Vytvoří se *socket* a nastaví se doba jeho doby prodlení (timeoutu). Paketu *write* se nastaví jeho IP adresa a port. Packet *write* se odešle, a odpověď na něj se uloží do paketu *answer*. Nakonec se socket uzavře.

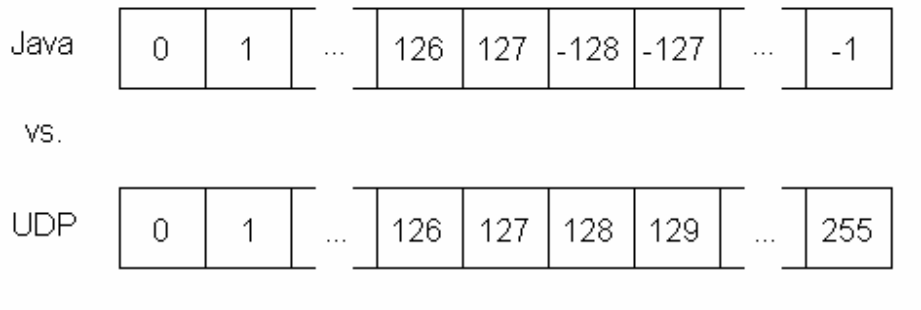
```
DatagramSocket udpSocket = null;
  byte[] packetAnswer = new byte[250];
  byte[] packetWrite = new byte[250];

  packetWrite= packetWrite(zapis);
  DatagramPacket answer = new DatagramPacket(packetAnswer, 250);
  DatagramPacket write = new DatagramPacket(packetWrite, 48);

  try
  {
        udpSocket = new DatagramSocket(); //vytvoří soket
        udpSocket.setSoTimeout(Scene.timeOut);
        write.setSocketAddress(new InetSocketAddress(Scene.ipAdress, Scene.port));
        udpSocket.send(write);
        udpSocket.receive(answer);
   }
   catch(IOException e)
  { e.printStackTrace(System.err);
  } finally { if(udpSocket != null) udpSocket.close();
```

V případě vlastní aplikace se při vykreslování scény vždy po určité době zavolá metoda *communicate*, které se předá jako parametr pole se stavy jednotlivých výstupů modelu. Metoda zavolá další metody třídy UDP, které ji vrátí již sestavené pole bytů. Tyto pole obsahují příkazy CONNECT, READB a WRITEB sítě ESPNET (podrobně popsáno v kapitole 5). Postupně se odešlou jednotlivé příkazy. Vždy po odeslání se čeká na odpověď. Z odpovědi na příkaz READB se přečtou stavy vstupů modelu, které metoda *communicate* vrací.

Při vytváření pole bytů s příkazy sítě EPSNET je důležité uvědomit si, že byte v Javě nabývá hodnot -128 až 127. Naproti tomu v EPSNET je přijatý byte vyhodnocován klasicky jako 0 až 255. Při programování tříd, které vytvářejí pole bytů ze kterého bude složen paket, se osvědčilo pracovat s datovými typy integer na intervalu 0 až 255. Třída DatagramPacket ale vyžaduje pole bytů. Proto se tyto proměnné přetypují na byte, čímž vznikne požadovaný, a zároveň i srozumitelný tvar (dokumentace pro EPSNET je psaná pro klasický byte v hexadecimální podobě).

Příklad takového přetypování:

```
packetWrite[45]=(byte)0x87;
```



Obr. 5: Rozdíl datového typu byte pro Javu a pro UDP

6.4 Třída Dialog

Před vytvořením okna do kterého se vykresluje, se v metodě *main,* volá konstruktor třídy Dialog. Tato třída je děděna z JDialog (API JavaSwing) a implementuje rozhraní ActionListener. Je modální, tzn., že dokud uživatel toto okno neukončí stiskem tlačítka Start, program nepokračuje v běhu v metodě *main,* ale vyčkává na ukončení tohoto okna. Této vlastnosti je využito k nastavení některých nezbytných parametrů modelu.

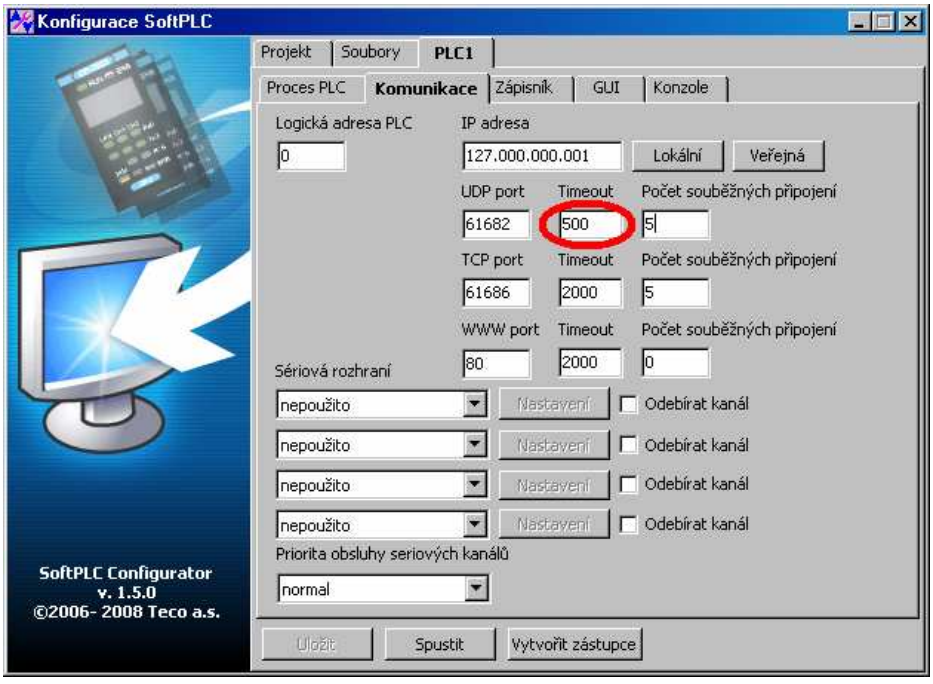
Jelikož máme třídu, která implementuje rozhraní ActionListener můžeme objekt této třídy zaregistrovat jako posluchače tlačítka. Rozhraní ActionListener má pouze jednu metodu s významem „tlačítko stisknuto“, která se jmenuje actionPerformed(). V metodě actionPerformed() se informace vložené uživatelem do objektů JSpinner a JTextField převedou ze stringů na integery a uloží se do připraveného pole, které je nastaveno jako globální. Nakonec se nastaví viditelnost na false, čímž se dialog ukončí.

7 Nastavení programu

7.1 SoftPLC for Windows

Vlastní aplikace představuje model virtuálního stroje, který na zvolenou IP adresu cyklicky odesílá pakety přes UDP protokol, které obsahují příkazy sítě EPSNET pro čtení a zápis do datové paměti PLC. Předpokládá se, že na dané IP adrese je do sítě připojeno skutečné PLC, které síť ESPNET podporuje nebo počítač na kterém je spuštěno virtuální PLC. Software, který jsem používal pro vývoj aplikace, se jmenuje SoftPLC for Windows, je možné ho stáhnout z webových stránek firmy Teco a.s. [1]. Program je po zaregistrování možné ve verzi TRIAL provozovat zdarma. Jediným omezením je, že takto simulované PLC pracuje pouze po dobu 4 hodin, což je pro testování modelu zcela dostatečné.

Na stránkách [1] je také dostupný návod, jak virtuální PLC správně nastavit. Pro spolehlivou funkčnost modelu se doporučuje změnit hodnotu Timeout pro UDP komunikaci z 2000 ms na 500 ms.



Obr. 6: Spolehlivé nastavení komunikace

Do spuštěného SoftPLC je třeba nahrát uživatelský program, podle kterého bude virtuální model ovládán. Tento uživatelský program je nutné do PLC poslat pomocí programu Mosaic.

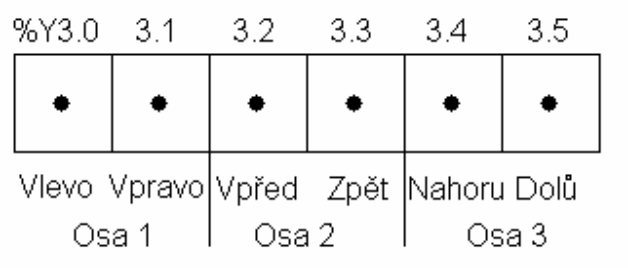
Mosaic je integrované vývojové prostření firmy Teco a.s. pro její produkty. Podrobný návod pro práci v tomto prostředí je nad rámec této práce. Uživatelská dokumentace pro tento program je dostupná na stránkách firmy [1]. Pro správnou komunikaci se SoftPLC je nutné v programu ručně nastavit typ spojení a IP adresu. U virtuálního PLC program automaticky nerozpozná typ jednotky, proto je nutno nastavit CP-7003.

Krátká ukázka jednoduchého programu pro PLC v jazyku memokódu, který ovládá model:

```
P 0
    ld   %X3.1
    set  %Y3.0
E 0
P 1
    ld   %X3.2
    res  %Y3.0
    set %Y3.5
E 1
P 2
    ld %X3.7
    res %Y3.5
E 2
```

Tento program načte vstup PLC na adrese %X3.1 (osa 1 – bod 1) a nastaví výstup %Y3.0 (rotace modelu doleva). Poté se vyčkává do doby kdy se model dostane do bodu 2 (%X3.2 v log. 1) zastaví se rotace doleva a nastaví se výstup %Y3.5 (osa 3 – pohyb dolů). Až se vstup %X3.7 dostane do log.1 se tento pohyb zastaví.

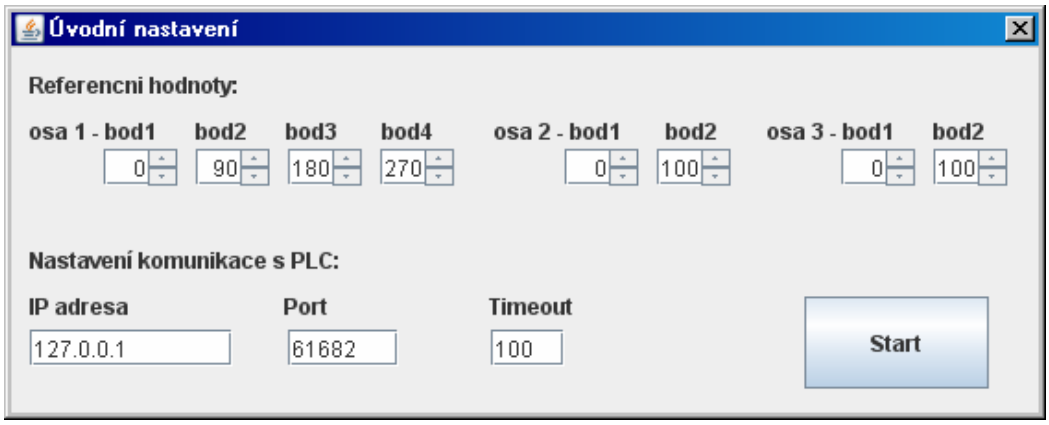
Na následujícím obrázku jsou znázorněny virtuální vstupy modelu (současně je lze označit i za výstupy PLC), včetně jejich funkce.



Obr. 7: Vstupy modelu – výstupy PLC

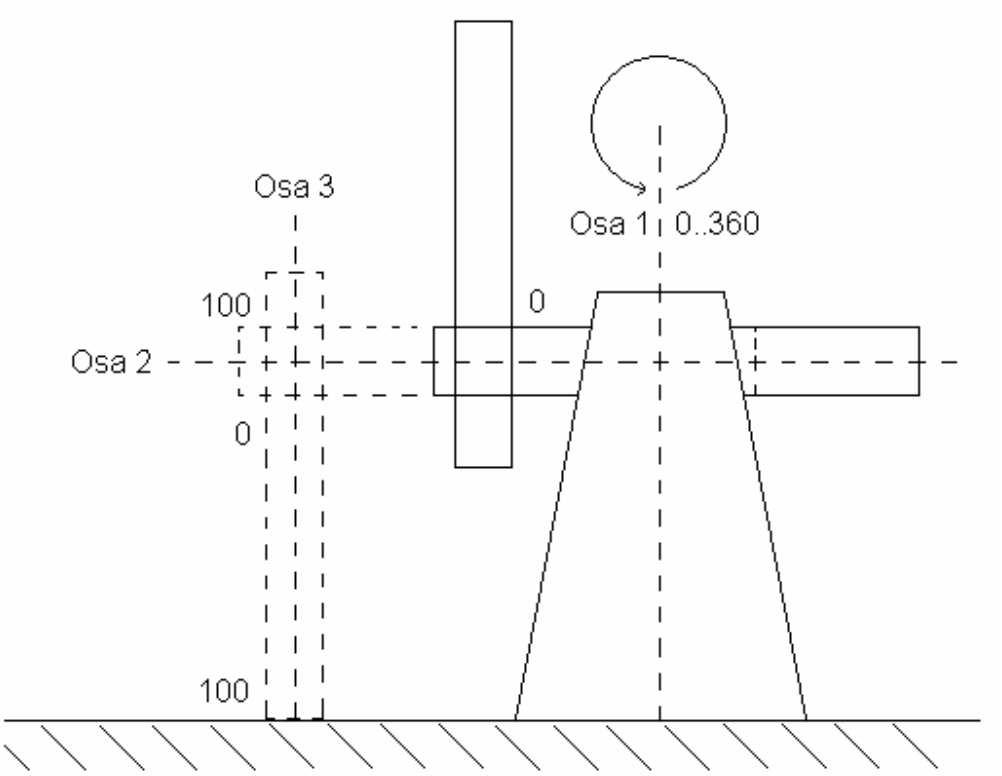
7.2 Vlastní aplikace

Před spuštěním vlastního modelu se nejprve nastavuje několik parametrů aplikace.



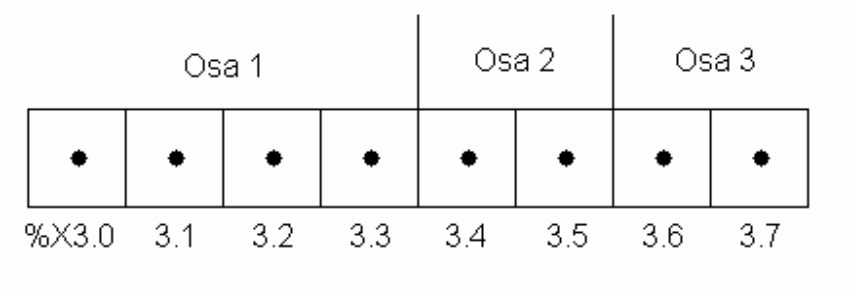
Obr. 8: Úvodní nastavení vlastního programu

V parametrech Referenční hodnoty se nastavují pro jednotlivé osy body, ve kterých budou výstupy modelu nastaveny na log. 1. Osa 1 je rotace modelu okolo své vlastní osy, hodnota se uvádí ve stupních, a lze ji nastavit v intervalu 0 - 359. Osa 2 je pohyb ramene dopředu a zpět. Nastavuje se v rozmezí 0 – 100, kde 0 je nejblíže základně modelu. Osa 3 vyjadřuje pohyb dolů a zpět, a opět se nastavuje v rozmezí 0 – 100. Toto rozložení názorně ukazuje obrázek č.9.



Obr. 9: Znázornění jednotlivých os modelu

Stavy těchto výstupů z robota se pomocí sítě EPSNET později cyklicky přenášejí do paměti PLC na adresy s absolutní hodnotou %X3.0 až %X3.7, kde jsou dostupné pro jeho uživatelský program.



Obr. 10: Výstupy modelu – vstupy PLC

V nastavení komunikace se nastavuje IP adresa, na kterou se budou pakety odesílat. Standardně je zde nastavena adresa počítače, na kterém je aplikace spuštěná (localhost). Port, na kterém bude komunikace probíhat je port 61682, standardní pro síť EPSNET. Timeout nastavuje dobu (v milisekundách), po které přestává aplikace čekat na odpověď za odeslaný UDP paket a považuje ho za ztracený. Nastavená příliš krátká doba může znemožnit komunikaci.

Model vždy nastartuje v poloze <osa 1 = 0, osa 2 = 0, osa 3 = 0>. Po stisku tlačítka Start se spustí model, který ovládá PLC dle uživatelského programu a nastavení referenčních bodů.

Fakt, že programy spolu opravdu komunikují, lze ověřit v okně SoftPLC, kde by se měli rozblikat stavové kontrolky provozu po síti LAN (viz obrázek 11).

Obr. 11: SoftPLC, které právě komunikuje po síti LAN

Obr. 12: Ukázka spuštěného virtuálního stroje

8 Závěr

Cílem této práce bylo vytvoření modelu virtuálního stroje, který bude komunikovat se sítí EPSNET. První bod zadání dává za úkol najít vhodný programovací jazyk pro realizaci modelu a jeho vazby na akcelerovanou 3D grafiku, což popisuje druhá kapitola. Třetí a čtvrtá kapitola popisují instalaci tohoto rozhraní a detaily práce v něm.

Kapitola 5 shrnuje vlastnosti sítě EPSNET. Kapitoly šest a sedm pak popisují, jak je vlastní aplikace naimplementována a jak ji správně používat.

Výsledkem práce je model virtuálního stroje, který umí komunikovat s PLC přes síť EPSNET pomocí protokolu UDP. Model byl navržen tak, aby vzhledem připomínal průmyslový manipulátor se třemi osami volnosti. Použití akcelerované grafiky umožnilo při vykreslování modelu využít funkce, které jsou jinak v jazyce Java nedostupné, a tím urychlit zobrazování.

8.1 Možné další pokračování projektu

V současné aplikaci nelze bez zásahu do kódu nic měnit. Proto by možným pokračování projektu mohla být aplikace, která v sobě nemá pevně zabudovaný model. Uměla by ale virtuální stroj sestavit z již hotového modelu, který by byl vytvořený v nějakém programu pro vytváření 3D grafiky, např. 3D Studio MAX. V takovém případě by bylo vhodné využít nějakou z nadstaveb pro JOGL, které již mají integrovány funkce pro načítání hotových modelů. Aplikace by pak zajišťovala vykreslování, změnu pohledů pozorovatele a samotnou komunikaci s PLC.

Seznam použitých zkratek

API - Application Programming Interface

Jedná se o sbírku procedur, funkcí či tříd nějaké knihovny, které může využívat programátor, který danou knihovnu využívá.

GLUT - The OpenGL Utility Toolkit

Doplněk ke grafické knihovně OpenGL. Základem této nadstavbové knihovny je podpora pro práci s okny (včetně zpracování událostí), vyskakovací menu a písmem.

GUI - Graphical User Interface

Grafické uživatelské rozhraní je uživatelské rozhraní, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků.

IDE - Integrated Development Environment

Vývojové prostředí, je software usnadňující práci programátorů, většinou zaměřený na jeden konkrétní programovací jazyk.

IP - Internet Protocol

Datový protokol používaný pro přenos dat přes paketové sítě. Tvoří základní protokol dnešního Internetu.

JDK - Java Development Kit

Soubor základních nástrojů pro vývoj aplikací pro platformu Java.

JID - JAVA is DOOMED

Rozšiřující knihovna pro OpenGL.

jME – jMonkeyEngine

Rozšiřující knihovna pro OpenGL.

JOGL - Java Binding for the OpenGL API

Vazba mezi Javou a OpenGL.

JRE - Java Runtime Environment

JVM a API společně tvoří celek, který je poskytován jako Java Runtime Environment.

JVM - Java Virtual Machine

Java Virtual Machine je sada počítačových programů a datových struktur, která využívá modul virtuálního stroje ke spuštění dalších počítačových programů a skriptů vytvořených v jazyce Java.

LAN - Local Area Network

Počítačová síť, která pokrývá malé geografické území

LWJGL - The Lightweight Java Game Library

Vazba mezi Javou a OpenGL.

OpenGL - Open Graphics Library

Standard specifikující rozhraní pro tvorbu aplikací počítačové grafiky.

PLC - Programmable Logic Controller

Průmyslový počítač používaný pro automatizaci procesů, řízení strojů nebo výrobních linek v továrnách.

TCP - Transmission Control Protocol

Protokol internetu představující transportní vrstvu. Protokol garantuje spolehlivé doručování a doručování ve správném pořadí.

UDP - User Datagram Protocol

Protokol internetu představující transportní vrstvu. Nezaručuje, zda se přenášený datagram neztratí, a zda se nezmění pořadí doručených datagramů nebo se některý datagram nedoručí vícekrát.

Literatura

[1] Teco a.s.
URL: <http://www.tecomat.cz>
[citováno 28. dubna 2009]

[2] Sériová komunikace programovatelných automatů Tecomat - model 32 bitů
URL: <http://www.tecomat.cz/docs/cze/General/txv00403.pdf>
[citováno 28. dubna 2009]

[3] NeHe Production
URL: < http://nehe.gamedev.net/>
[citováno 4. května 2009]

[4] CZ NeHe OpenGL
URL: < http://nehe.ceske-hry.cz/tut_obsah.php>
[citováno 4. května 2009]

[5] Seriál Grafická knihovna OpenGL
URL: < http://www.root.cz/serialy/graficka-knihovna-opengl/ >
[citováno 28. dubna 2009]

[6] Java Platform Standard Edition 6 API Specification
URL: < http://java.sun.com/javase/6/docs/api/index.html>
[citováno 15. května 2009]

[7] JOGL
URL: < https://jogl.dev.java.net/> [citováno 3. dubna 2009]

[8] JOGL-demos
URL: < https://jogl-demos.dev.java.net/> [citováno 3. dubna 2009]

[9] Java 3D
URL: < http://www.java3d.org/> [citováno 20. března 2009]

[10] Java 3D API
URL: < http://java.sun.com/javase/technologies/desktop/java3d/>
[citováno 20. března 2009]

[11] OpenGL for Java
URL: < http://gl4java.sourceforge.net/> [citováno 6. dubna 2009]

[12] Lightweight Java Game Library
URL: < http://lwjgl.org/> [citováno 6. dubna 2009]

[13] JAVA is DOOMED
http://javaisdoomed.sourceforge.net/> [citováno 6. dubna 2009]

[14] jMonkeyEngine
URL: < http://www.jmonkeyengine.com/> [citováno 6. dubna 2009]

[15] Xith3D project
URL: < http://xith.org/> [citováno 6. dubna 2009]

[16] Síťování v Javě: UDP
URL: < http://www.root.cz/clanky/sitovani-v-jave-udp/ >
[citováno 25. dubna 2009]

[17] OpenGL v Javě
URL: < http://jogl.sislik.net/> [citováno 15. dubna 2009]

[18] Wikipedia
URL: < http://www.wikipedia.cz > [citováno 24. května 2009]

[19] BSD

URL: < http://www.opensource.org/licenses/bsd-license.php >

[citováno 24. května 2009]

[20] Zakhour, Sharon. Java 6 - Výukový kurz. 1. vydání Brno: Computer Press, 2007. ISBN 978-80-251-1575-6

[21] Herout Pavel. Java - grafické uživatelské prostředí a čeština. 2. vydání České Budějovice: KOPP 2007 ISBN 978-80-7232-328-9

[22] GNU General Public License

URL: < http://www.gnu.org/licenses/gpl.html> [citováno 24. května 2009]