ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

DIPLOMOVÁ PRÁCE

Řízení a vizualizace robotických vozidel

Autor práce: **Bc. Jan Pokorný**
Vedoucí práce: **Ing. Petr Weissar, Ph.D.**

Plzeň 2016

Originál / kopie zadání diplomové práce

- Navrhněte vhodné SW řešení pro bezdrátové řízení pojízdných robotů, které máme na KAE k dispozici.
- Předpokládejte řízení z počítače PC, ve vozidle je pouze řídicí mikropočítač.
- Uvažujte možnost nasazení kamery, preferujte WiFi přenos obrazu.
- Navrhněte vhodnou vizualizaci provozních stavů vozidla (např. stav baterie, chybovost přenosu, stav osazených senzorů typu akcelerometr apod.)
- Zvolte vhodnou formu rozhraní s uživatelem.

Abstrakt

Tato diplomová práce je zaměřena na vytvoření programu pro snadné řízení robotických vozidel a pro přehledné zobrazení osazených senzorů. ConVis Studio, jak byl program nazván, je pojato jako univerzální objektová aplikace, kde si sám uživatel navolí, co přesně chce vizualizovat a řídit. K této práci byl navržen i komunikační protokol optimalizovaný pro sběrnici CAN. Program je tedy schopen řídit a vizualizovat nejen robotická vozidla, ale cokoli, co do jisté míry implementuje tento protokol. Aplikace umožňuje komunikaci s okolními zařízeními prostřednictvím sériové linky, přenos obrazu z kamery je řešen samostatným internetovým připojením. Vytvářené projekty jsou ukládány s příponou „cvp“ ve formátu XML. ConVis Studio bylo vyvinuto s použitím .NET Frameworku v Microsoft Visual Studiu v jazyce C#.

Klíčová slova

řízení, vizualizace, program, C#, .NET Framework, komunikační protokol, CAN, robot, IP, kamera, vozidla, ConVis, Microsoft Visual Studio

Abstract

This work is focused on developing a program for an easy control of robotic vehicles and for the visualization of available sensors. The ConVis Studio, as the program was called, is designed as a universal application, where a user chooses exactly what he wants to visualize and control. For the purposes of this work, a new communication protocol has been created. The protocol is also optimized for the CAN Bus and so the application is able to control and visualize not only robots, but everything what implements this protocol within an appropriate range. The application allows a communication through the serial port and a camera stream is transmitted by the separated internet connection. The projects are saved with the "cvp" extension in the XML format. The ConVis Studio was developed in Microsoft Visual Studio by using .NET Framework and C# programming language.

Key words

control, visualization, program, C#, .NET Framework, communication protocol, CAN, robot, IP, camera, vehicles, ConVis, Microsoft Visual Studio

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením vedoucího práce, s použitím odborné literatury a pramenů uvedených ke konci této práce v seznamu citované literatury.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujícího autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků, vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Dále prohlašuji, že veškerý software použitý při řešení této diplomové práce je legálně získaný.

.....
podpis

V Plzni dne 10.5.2016 Bc. Jan Pokorný

Poděkování

Velmi rád bych zde poděkoval vedoucímu diplomové práce Ing. Petru Weissarovi, Ph.D., za cenné profesionální rady, připomínky a za metodické vedení práce, kamarádovi Ing. Ondřeji Lufinkovi za výpomoc při ladění aplikace na platformě Tank a dále mé rodině za obětavou a vytrvalou podporu v průběhu celého studia.

OBSAH

1. ÚVOD

1.1. CÍL PRÁCE

Hlavním cílem práce je navrhnout program schopný řídit robotická vozidla a přehledně vizualizovat jejich senzory, chybová hlášení, zprávy a další potřebné stavy robota. Na katedře aplikované elektroniky a telekomunikací (KAE) je k dispozici několik robotických platforem: pásová platforma (Tank), terénní čtyřkolka (Crawler), vzducholoď, model kolejiště a další, většinou stále rozestavěné, platformy, jako například pavouk či robotická ruka. Pro všechny tyto platformy je potřeba navrhnout vhodnou řídicí aplikaci a sjednotit komunikační protokol, podle kterého budou platformy schopny s programem komunikovat. Součástí programu má být i možnost zobrazit kamerový přenos přes WiFi. Požadováno je snadné ovládání, uživatelská přívětivost a případná možnost rozšíření programu o další funkce i vizuální komponenty.

1.2. ČÁSTI PRÁCE

Práce je rozdělena do pěti hlavních kapitol. První kapitola pojednává obecně o řídicích a vizualizačních nástrojích a o některých programech, které jsou na trhu k dispozici.

Druhá kapitola popisuje komunikační protokol a aplikaci z hlediska samotného kódu, rozebírá detailně implementace důležitých částí a jejich vzájemnou provázanost.

Ve třetí kapitole jsou rozebrány problémy a chyby, které nastaly během vývoje programu a během testování na několika platformách.

Předposlední kapitola je zaměřena na popis aplikace z hlediska běžného uživatele. Jde tedy hlavně o popis graficko-uživatelské části, jejího ovládání, popis založení projektu, editace projektu, popis vizuálních komponent apod.

Závěrečná kapitola pak vše shrnuje do uceleného vyhodnocení, srovnává dosažený výsledek se zadáním a popisuje další možná rozšíření programu.

2. ŘÍZENÍ A VIZUALIZACE

„Je známou skutečností, že pro člověka je nejnázornější prezentací informací grafické zobrazení v nejrůznějších podobách. Charakteristickým rysem vizualizace je poměrně velký objem dat a jejich prezentace v takové (především grafické) podobě, že je možno velmi rychle porozumět jejich obsahu a významu. Zkoumáním možností a řešením teoretických problémů se zabývá vědní disciplína: vizualizace dat (angl. scientific visualization). (…) Do počítače s nainstalovaným speciálním programovým vybavením s vysokou vypovídací schopností připojeného komunikační linkou k ŘS (řídicímu systému) se pravidelně přenáší důležité informace z řízeného děje, takže obsluha má k dispozici mohutný nástroj k názornému sledování děje a případnému zásahu do něj. Programové vybavení automaticky sleduje případné chybové nebo nezvyklé (tzv. alarmové) stavy, zaznamenává parametry z řízeného děje a celkově tak děj monitoruje a jeho průběh archivuje.“ (Citováno z [1], str. 109)

Toto byl na úvod malý teoretický základ, popsaný v publikaci Jaroslava Vlacha „Řízení a vizualizace technologických procesů“. Je v něm stručně a názorně popsán obecný princip vizualizačních a řídicích nástrojů. Samotný řídicí systém může být navržen dvěma způsoby. První způsob je vytvoření aplikace na míru v některém programovacím jazyce, přesně podle požadavků daného systému. Takové řešení je většinou drahé, protože se musí vyvinout celé od základu, případně jen s použitím frameworku. Většinou ale celá graficko-uživatelská část musí být navržena a naprogramována znova. Na druhou stranu je toto řešení schopné plnit i velmi náročné požadavky, ať už od systému samotného, nebo od zákazníka či obsluhy programu. Aplikace zpravidla bývá velmi dobře optimalizovaná, nenáročná a úsporná.

Druhý způsob je navržení univerzální aplikace, která bude mít velkou škálu možných (většinou objektových) komponent. Zákazník si tak může sestavit řídicí a vizualizační proces sám. Je patrné, že takovéto řešení bude levnější, s dobrou komunitní podporou, a aktualizace či úprava procesu bude podstatně snazší. Oproti tomu bude zákazník omezen možnostmi studia a komunikačními protokoly, které studio zvládá. Celá aplikace bude rozsáhlá a složitá a předem bude dáno, jak aplikaci ovládat.

Zmíněná studia nicméně dnes dosahují takové robustnosti, optimalizace i nabídky dostupných komponent a rozšíření, že se vyplácejí i pro vcelku složité nároky řídicího systému. S takto rostoucími možnostmi ale roste i složitost tvorby vizualizačního procesu.

2.1. NÁSTROJE PRO ŘÍZENÍ A VIZUALIZACI DAT

Potřeba řídit a vizualizovat je dnes nedílnou součástí všech průmyslových aplikací. Pomalu se však dostává i do běžné komerční sféry, čímž klade stále větší nároky jak na uživatelsky příjemné a jednoduché rozhraní, tak i na snadnou instalaci. Na trhu existuje velké množství řídicích aplikací – některé z nich jsou nazývány systémy SCADA. „SCADA je zkratka pro "Supervisory Control And Data Acquisition", tedy „dispečerské řízení a sběr dat“. Obvykle se tento pojem používá pro software, který z centrálního pracoviště monitoruje průmyslová a jiná technická zařízení a procesy a umožňuje jejich ovládání.“ (Převzato z [2])

V následujících kapitolách budou rozebrány některé dostupné systémy pro vizualizaci a řízení. Ve všech případech se jedná o univerzální aplikace s možností vlastního nastavení a zobrazení vizuálních objektů. Vlastní práce vychází z návrhových vzorů těchto aplikací.

2.1.1. PROMOTIC

„Promotic je komplexní SCADA objektový softwarový nástroj pro tvorbu aplikací, které monitorují, řídí a zobrazují technologické procesy v nejrůznějších oblastech průmyslu. Je určen pro OS Windows 10/8/7/Vista/XP/XPe/2003-12Server a novější. Umožňuje efektivně vytvářet distribuované a otevřené aplikace v nejrůznějších odvětvích průmyslu a od verze 8 je možno provozovat systém Promotic také ve freeware módu.“ (Převzato z [3])

„V systému Promotic jsou zabudovány všechny nezbytné komponenty pro tvorbu jednoduchých i rozsáhlých vizualizačních a řídicích systémů.

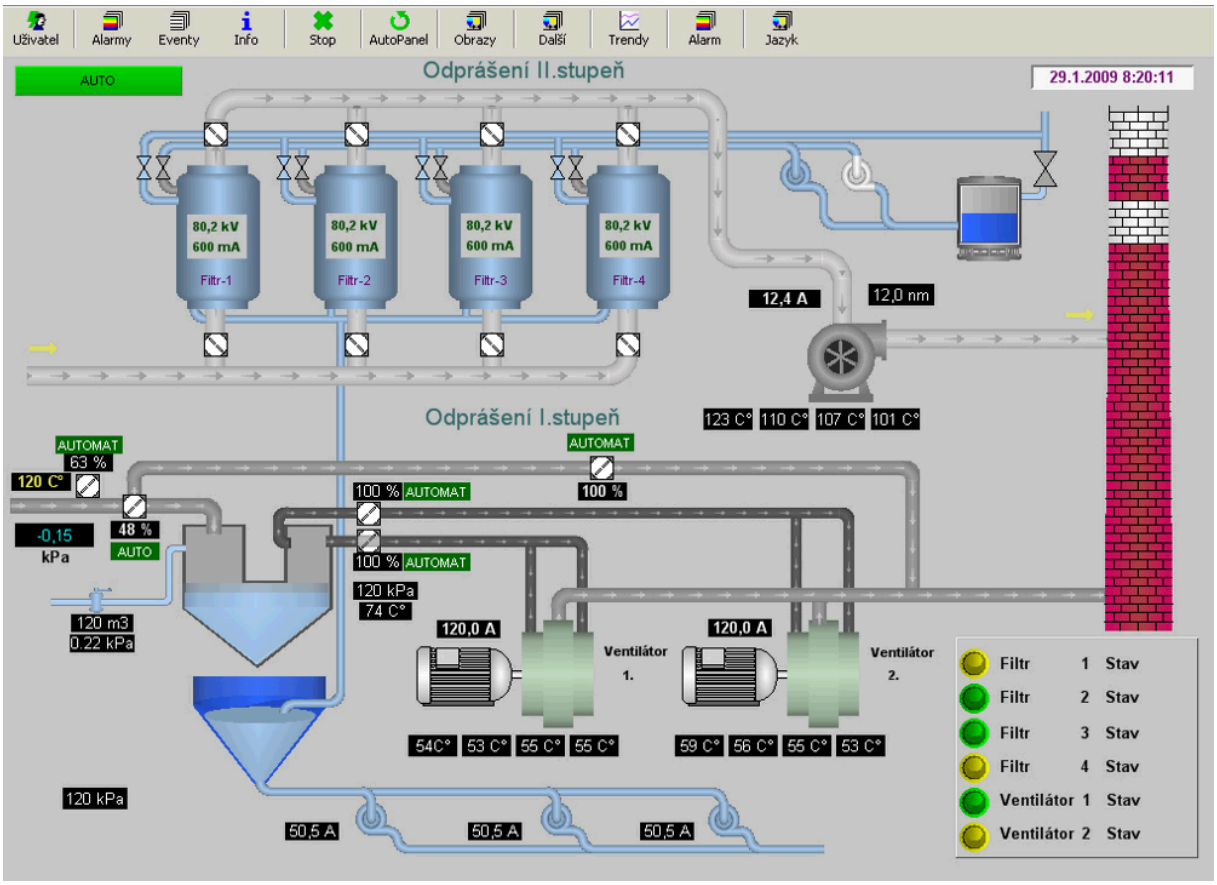
- Editor aplikace s hierarchickým stromem objektů.
- Široká nabídka objektů PROMOTIC.
- Jazyk Microsoft Basic (VBScript) pro zápis algoritmů.
- Editor obrazů.
- Bohatá paleta technologických obrázků vytvořených ve vektorové SVG grafice.
- Grafické objekty – elementární a komplexní velmi obecně konfigurovatelné prvky.
- Automatická konverze obrazů do HTML a XML[1] formátu.
- Systém trendů (tj. uchovávání hodnot s časovou známkou).
- Systém alarmů a operátorských událostí („eventů“).
- Podpora web technologií Internet/Intranet.
- SQL a ODBC rozhraní pro databáze.
- Zabudovaná rozhraní: XML, OPC[2], ActiveX, DDE[3].

[1] XML (Extensible Markup Language) – obecný značkovací jazyk.
[2] OPC (Open Platform Communications) – standardy pro komunikaci s řídicími systémy.
[3] ActiveX, DDE – rozhraní pro sdílení dat mezi aplikacemi.

- Komunikační ovladače pro přístup k PLC[4].
- Správa uživatelů, oprávnění a přihlašovací systém.
- Zabezpečení provozovaných aplikací.
- Jazykové verze PROMOTIC.
- INFO – informační a diagnostický systém.
- Elektronická i tištěná dokumentace.“ (Převzato z [3])

Systém Promotic je svým zaměřením určen především pro řízení PLC a jeho cena pro komerční využití se pohybuje od tisíců po desetitisíce Kč v závislosti na dodaných ovladačích a podpůrných nástrojích. Kompletní ceník je k dispozici zde [4].

Na obr. 1 je zobrazen ukázkový proces pro monitorování odprašovacích stupňů tepelné elektrárny. Je zde vidět funkční schéma složené z jednotlivých objektů, které slouží pro přehledné vizualizování celého procesu odprášení. Zároveň je možné vidět i styl grafického zpracování technologických obrázků i celého programu.



Obr. 1: Ukázka systému Promotic [5]

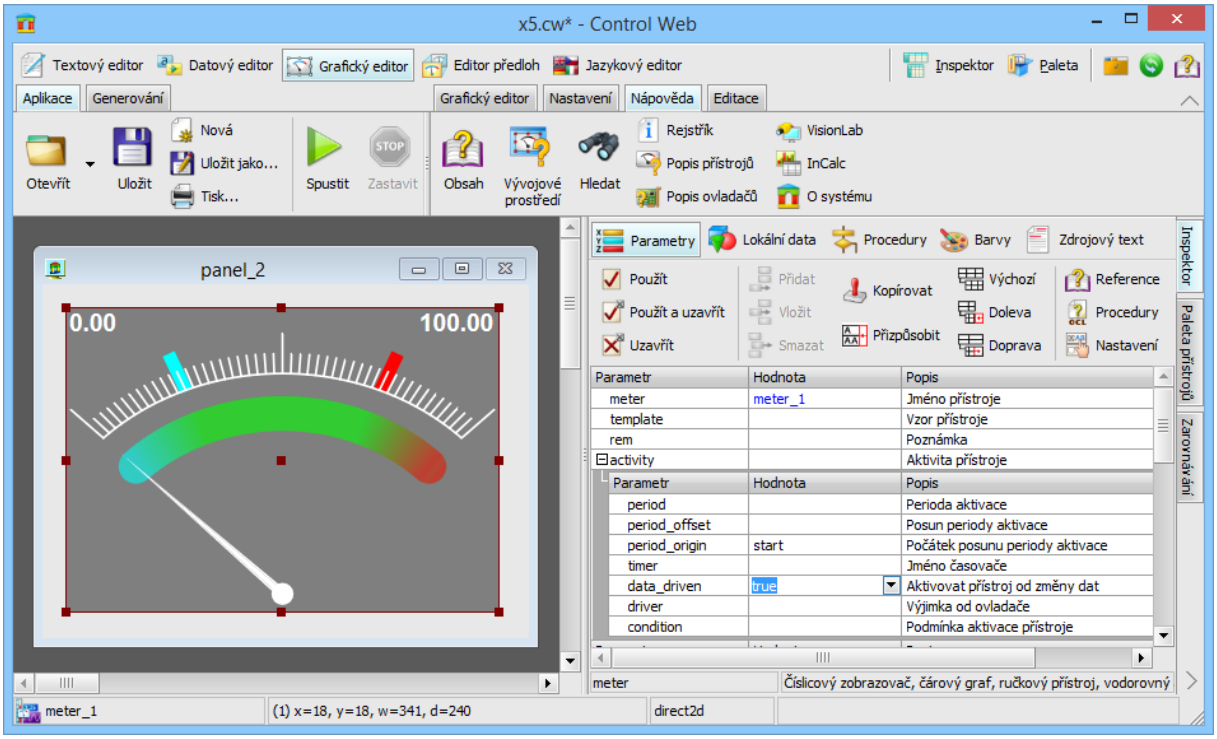
[4] PLC (Programmable Logic Controller) – programovatelný logický automat.

2.1.2. CONTROL WEB

„Control Web je programovým systémem, který dokáže vystupovat v mnoha rolích. Může pracovat v řídicích jednotkách strojů, může spojovat výrobní technologii s informačním systémem podniku, může být datovým serverem s mnoha webovými klienty, může modelovat a simulovat procesy, dokáže vytvářet náročné vizualizace a mnoho dalšího. To, že Control Web bezezbytku řeší veškerou funkčnost SCADA systémů je zřejmé, je však rovněž plně schopen řešit zadání, která klade současná epocha Internetu a globálních komunikací, jako třeba:“ (Převzato z [6])

- „Control Web v roli firemního WWW serveru.
- Control Web jako automatický registrační a aktivační server a webovým a SMS rozhraním.
- Control Web jako integrující systém laboratoře na Fakultě aplikované informatiky.“ (Převzato z [6])

Control Web je rozvinutý vývoj předchozích verzí s názvem Control Panel, doplněný o výše zmíněné internetové prvky. Cenově se řadí mezi levnější systémy, přesný ceník je dostupný zde [7]. Následuje ukázka grafického editoru komponent.



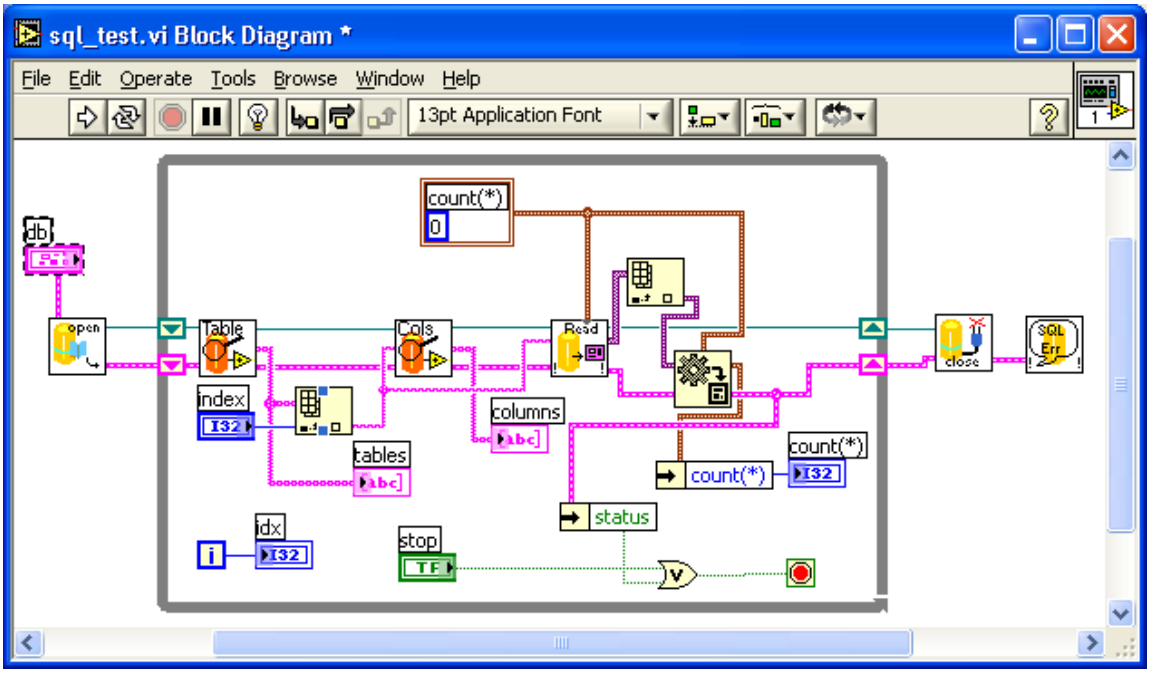
Obr. 2: Grafický editor v systému Control Web 7 [8]

2.1.3. LabVIEW

LabVIEW (angl. Laboratory Virtual Instrument Engineering Workbench) je systémová návrhová platforma a vývojové prostředí pro grafické programovací jazyky společnosti National Instruments. Grafický jazyk je pojmenován „G“ a původně vydán pro Apple Macintosh v roce 1986. LabVIEW se běžně používá pro sběr dat, řízení nástrojů a průmyslovou automatizaci na různých platformách včetně Microsoft Windows, různých verzí systému UNIX, OS Linux a OS X. Nejnovější verze LabVIEW je LabVIEW 2015, vydaný v srpnu 2015. (Přeloženo z [9])

„K dispozici jsou 4 licence (uvedená cena je pro základní verzi s roční podporou):

- ***Zkušební licence*** – zdarma
- ***Základní licence*** – 27 400 Kč
  - Vývoj grafického uživatelského rozhraní
  - Sběr dat, řízení přístrojů
  - Vytváření reportů a práce se soubory
- ***Plná licence*** – 92 200 Kč
  - Více než 950 matematicko-analytických funkcí
  - Volání externích kódů
  - Propojení s internetem
  - Vývoj pokročilého uživatelského rozhraní
- ***Profesionální licence*** – 153 500 Kč
  - Distribuce aplikací (exe, dll, …)
  - Správa zdrojových kódů
  - Síťová komunikace“ (Přeloženo z [10])



Obr. 3: Návrh blokového schématu v systému LabVIEW [11]

2.1.4. RELIANCE 4

„Reliance 4 je profesionální SCADA/HMI systém určený pro monitorování a ovládání průmyslových technologií a automatizaci budov. SCADA/HMI systém Reliance je vyvíjen na základě dlouholetých zkušeností s budováním rozsáhlých aplikací a k jeho zdokonalování přispívají i neustálé podněty ze strany zákazníků. Výsledkem je bohatě škálovatelný, bezpečný a robustní systém optimalizovaný i pro velmi rozsáhlé aplikace.“ (Převzato z [12])

„Struktura systému:

- Vývojové prostředí
- Runtime modely
- Tencí klienti
- Komunikační drivery
- Reliance OPC server“ (Převzato z [12])

Aktuální ceník je dostupný zde [13]. Na obr. 4 je možné vidět vizualizaci inteligentního rodinného domu v prostředí Reliance 4.

Obr. 4: Vizualizace rodinného domu v systému Reliance 4 [14]

3. POPIS NÁVRHU APLIKACE

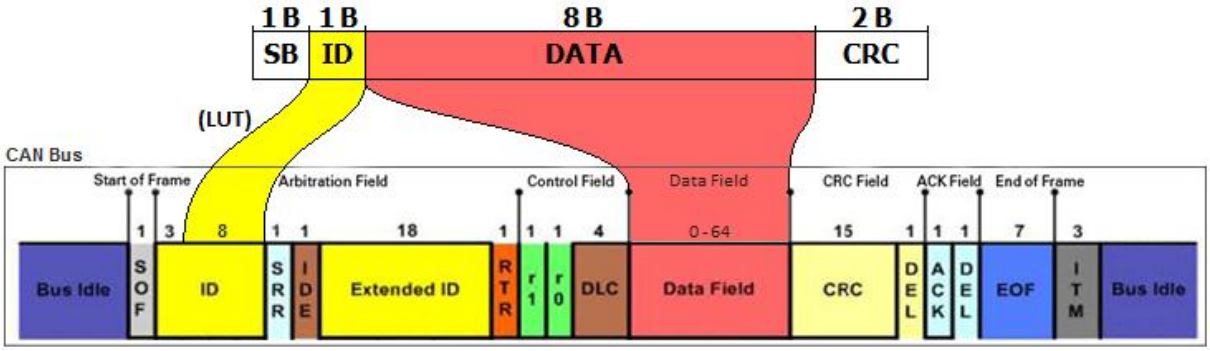
Program byl vytvořen v jazyce C# s použitím .NET Frameworku 4.5 a Microsoft Visual Studia Professional 2012, které poskytlo patřičnou robustnost návrhu i prostředky pro snadné hledání a ladění chyb (některé ladicí postupy budou popsány v kapitole 4 Testování aplikace).

Finální program byl kompilován jako „Release / Any CPU“ se zapnutým preferováním 32 bitového systému a s automatickou optimalizací kódu. Aplikace se skládá ze dvou projektů – z knihovny grafických komponent a samotného návrhu studia.

Součástí této kapitoly je i rozbor komunikačního protokolu.

3.1. KOMUNIKAČNÍ PROTOKOL

Jak již bylo zmíněno v abstraktu práce, komunikační protokol byl optimalizován pro sběrnici CAN, která umožňuje přenést až 8 bajtů dat v jednom rámci. Vlastní protokol tedy fixně definuje 8 bajtů dat a redukuje velikost identifikátoru z 11 (případně 29) bitů na 1 bajt. ID ovšem nemusí být přeložen na CAN ID přímo, ale prostřednictvím překladové tabulky (LUT[5]). Následující obrázek ukazuje celý formát protokolu i jeho transformaci na protokol CAN.



Obr. 5: Transformace komunikačního protokolu na CAN protokol

Komunikační protokol začíná vždy polem „Start-Byte“ (SB), tedy úvodní bajtovou sekvencí střídající dvě jedničky a dvě nuly (**0xCC** hexadecimálně). Následuje jednobajtový identifikátor, který určuje, o jaká data se jedná (podrobněji bude rozebrán níže). Dalšími v pořadí jsou samotná data, která jsou přenášena jako „little-endian“[6]. Nakonec je paket zakončen kontrolním 16 bitovým cyklickým součtem (CRC), který je počítán přes celý rámec

[5] LUT je zkratka pro Lookup Table.
[6] Little-endian – struktura, kdy na prvním adresovém místě leží bajt s nejnižší váhou.

včetně „start bajtu“.

CRC má doporučený polynom: $x^{16} + x^{15} + x^{13} + 1$, kterému odpovídá hodnota **0xA001** hexadecimálně – první „1“ zleva, odpovídající nejvyššímu stupni polynomu, se neuvádí. Aplikace umožňuje nastavit i polynom jiný, více v kapitole 5.4 Nastavení projektu.

Identifikátor (pole ID) dělí data do 256 možných kategorií takto:

- Id **0** (**0x00**) je zakázáno využívat.
- Id **1** (**0x01**) až **19** (**0x13**) slouží pro zprávy ze zařízení do řídicí aplikace.
- Id **20** (**0x14**) až **39** (**0x27**) slouží pro zprávy z řídicí aplikace do zařízení.
- Id **40** (**0x28**) až **69** (**0x45**) je vyhrazeno pro řízení akčních členů.
- Id **70** (**0x46**) až **199** (**0xC7**) je určeno pro data senzorů.
- Id **200** (**0xC8**) až **255** (**0xFF**) jsou volně přístupné. Uživatel si zde může navolit vlastní datové struktury, např. voltmetr (2 B), ampérmetr (2 B), rychlost otáček (2 B) a teploměr (2 B).

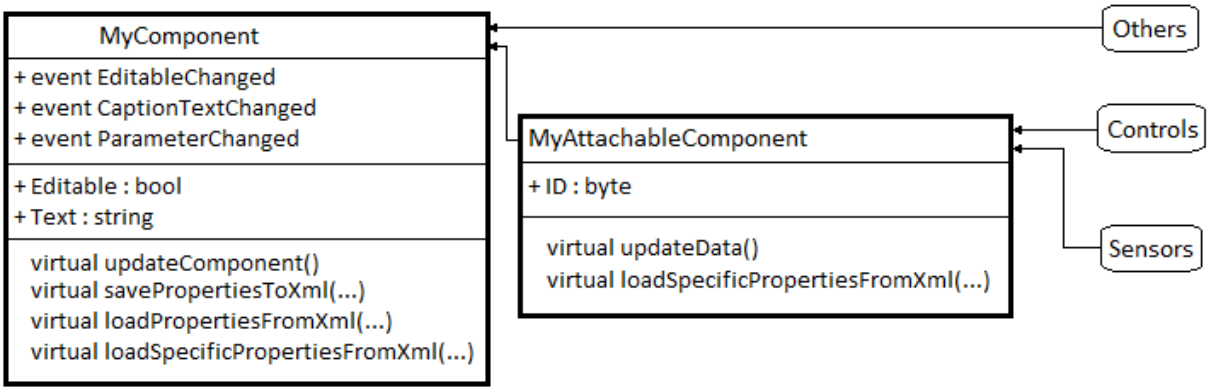
Zde uvedené rozdělení není ovšem zcela závazné, protože každá komponenta má možnost napojit se na libovolný ID. Pro zachování kompatibility s jinými projekty je však doporučeno držet se tohoto návrhu. Přesný komunikační protokol je v elektronické formě na přiloženém CD. Kvůli své rozsáhlosti je v tištěné příloze A na konci práce uveden pouze zjednodušený popis tohoto protokolu.

3.2. PROJEKT VISUALCOMPONENTS

Projekt VisualComponents vytváří knihovnu DLL obsahující vizuální komponenty, struktury datových paketů a další potřebné nástroje k popisu a výběru dat v komunikačním protokolu. Vizuální komponenty jsou jednotlivé grafické objekty, simulující senzorické či kontrolní vlastnosti nebo uživatelské prvky. Tyto komponenty jsou rozděleny na ty, které se nemohou napojit na komunikaci, a na ty, které se napojit mohou. Všechny komponenty dědí z třídy *MyComponent*, respektive z třídy *MyAttachableComponent*. Strukturu dědění ukazuje následující obrázek (obr. 6), na kterém jsou vidět i některé důležité události (angl. events), metody a vlastnosti tříd.

Rovněž je na obrázku vidět, že finální komponenty jsou rozčleněny do tří skupin: Další (angl. Others), Řídicí prvky (angl. Controls) a Senzory (angl. Sensors). Ve skupině „Řídicí prvky“ jsou komponenty, které primárně data odesílají, ve skupině „Senzory“ komponenty,

které data přijímají, a v sekci „Další“ jsou různé uživatelské prvky a komponenta kamery. Více bude popsáno v kapitole 5.3 Komponenty a jejich vlastnosti.



Obr. 6: Struktura dědění komponent

3.2.1. TŘÍDA MYCOMPONENT

Třída *MyComponent* byla zděděna z třídy *UserControl*, která je součástí .NET frameworku a která definuje rozhraní pro uživatelské ovládací prvky. Třída obsahuje mnoho nepotřebných vlastností, které bylo nutno překrýt (angl. override) a pomocí speciálního atributu vepsaného před danou vlastnost (řádek 1 v následující ukázce kódu) zakázat „prohlížitelnost“ (angl browsable) vlastnosti v prohlížeči (prvek *PropertyGrid*).

```
1       [Browsable(false)]
2       public override System.Drawing.Point AutoScrollOffset
3       {
4               get { return base.AutoScrollOffset; }
5               set { base.AutoScrollOffset = value; }
6       }
```

Třída dále obsahuje tři události: *EditableChanged*, která nastane při změně vlastnosti *Editable*, *CaptionTextChanged*, která je volána při změně vlastnosti *Text*, a *ParamaterChanged*, která indikuje třídě *ProjectForm*, že došlo ke změně důležité vlastnosti, čímž třídě sděluje nutnost uložit projekt.

Důležitou částí třídy *MyComponent* jsou metody pro ukládání a načítání vlastností z XML. Tyto metody jsou obsluhovány ze třídy projektu *ProjectForm*, která bude podrobně popsána v kapitole 3.3.1 Formulář ProjectForm. Metoda pro uložení `void savePropertiesToXml(XmlWriter writer)` je provedena triviálně. Všechny vlastnosti jsou postupně uloženy pomocí poskytované metody `void WriteElementString(string localName, string value)`, která se postará o správný XML formát.

Zde je ukázka kódu pro uložení základních vlastností:

```
1     public virtual void savePropertiesToXml(XmlWriter writer)
2     {
3           writer.WriteElementString(XmlElements.TEXT, this.Text);
4           writer.WriteElementString(XmlElements.WIDTH, this.Width.ToString());
5           writer.WriteElementString(XmlElements.HEIGHT, this.Height.ToString());
6           writer.WriteElementString(XmlElements.LEFT, this.Left.ToString());
7           writer.WriteElementString(XmlElements.TOP, this.Top.ToString());
8     }
```

Třída nové komponenty zděděná z třídy *MyComponent* tak jen překryje metodu *savePropertiesToXml*, zpětně ji zavolá na rodiči klíčovým slovem *base* (řádek 3 v kódu níže) a doimplementuje chybějící vlastnosti, příslušející dané komponentě (řádky 5 až 7).

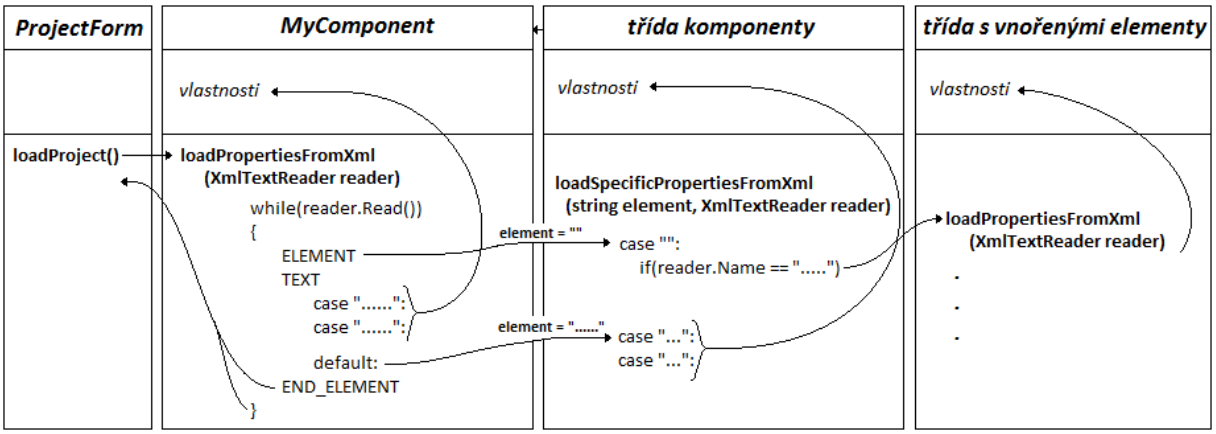
```
1     public override void savePropertiesToXml(XmlWriter writer)
2     {
3           base.savePropertiesToXml(writer);
4
5           writer.WriteElementString(XmlElements.LOGIN, this.Login);
6           writer.WriteElementString(XmlElements.PASSWORD, this.Password);
7           writer.WriteElementString(XmlElements.SOURCE_URL, this.SourceUrl);
8     }
```

Metoda pro načtení `void loadPropertiesFromXml(XmlTextReader reader)` je o poznání složitější. Metoda je zavolána opět z projektu *ProjectForm* při načtení elementu „<component …>“ (řádek 1 v ukázkovém XML níže) a obsahuje cyklus, který postupně analyzuje XML dokument. Cyklus pro analýzu skončí, pokud je čtení souboru dokončeno nebo jestliže *reader* načte `XmlNodeType.EndElement` (označen </…>) a jeho jméno (*reader.Name*) je „component“ (řádek 7 v XML níže). V případě, že *reader* načte `XmlNodeType.Element`, element je uložen a testován, zda je elementem nadřazeným (element na řádku 4 v ukázkovém XML níže). Testování se provádí voláním metody `this.loadSpecificPropertiesFromXml("", reader);` s prázdným řetězcem jako prvním parametrem. Tato metoda je implementována ve zděděné třídě podle potřeby načtení dalších vlastností, jinak nevykonává žádnou činnost.

```
1     <component type="sensors_led">
2           <text>Led</text>
3           <protocol_id>10</protocol_id>
4           <attach_to>
5                 <bit_offset>0</bit_offset>
6           </attach_to>
7     </component>
```

Poslední možností je, že *reader* načte `XmlNodeType.Text` (řádek 2, 3 a 5 mezi elementy) a metoda se pokusí rozpoznat, ve kterém elementu leží. Jestliže metoda element pozná, adekvátně zpracuje načtenou textovou hodnotu *reader.Value*. Pokud ne, je zavolána již zmíněná metoda – `this.loadSpecificPropertiesFromXml(element, reader);` – první parametr je však nyní řetězec *element*. Podrobný způsob načítání je zobrazen na následujícím obrázku

(pozn. kód na obrázku není kompletní, ale jen principiálně ukazuje strukturu načítání).



Obr. 7: Struktura načítání z XML

Poslední důležitou částí *MyComponent* je virtuální metoda *updateComponent*, která je zpravidla volána při změně některé z vlastností komponenty. Komponenta si proto v této metodě implementuje patřičnou aktualizaci vnitřních proměnných a vyžádá si překreslení vizuální části. Změny se tak projeví takřka okamžitě.

3.2.2. TŘÍDA MYATTACHABLECOMPONENT

Třída *MyAttachableComponent*, která je potomkem třídy *MyComponent*, přidává funkcionalitu pro napojení na komunikaci. Jedná se hlavně o nový konstruktor s parametrem `DataPacketDictionary dataPacketDictionary`. Dále přidává vlastnosti `byte ID` a `DataPacket Data` a metody `void updateData()` a `void newDataPacketSet()`. Parametr *dataPacketDictionary* v konstruktoru je referencí na vytvořený list datových paketů pro všechny dostupné *ID*. Při změně vlastnosti *ID* je tak do vlastnosti *Data* předán odkaz na *DataPacket*, příslušející novému *ID*. V ukázkovém kódu níže je parametr *ID* zastoupen klíčovým slovem *value*, protože se jedná o nastavovací proceduru (angl. setter).

```
1       set
2       {
3               if(this.dataPacketDictionary != null)
4               {
5                       this.Data = this.dataPacketDictionary.getDataPacketByID(value);
6                       this.updateComponent();
7               }
8       }
```

Jakmile dojde ke změně vlastnosti *Data*, je přiřazením metody *dataTarget_DataChanged* zaregistrována událost *DataChanged* (viz řádek 12 v kódu níže) pro odchycení změny těchto dat. Na řádcích 3 až 6 je doplněno případné odregistrování *DataChanged* na původních datech. Metoda *dataTarget_DataChanged* interně volá již zmíněnou metodu *updateData*,

kterou si každá ze zděděných tříd implementuje tak, aby zpracovala nová data přesně podle svého nastavení. Velmi často je v této metodě volána i metoda *updateComponent*, která se postará o překreslení grafického vzhledu komponenty. Metoda *newDataPacketSet*, volaná na řádku 13, slouží pro aktualizaci datových formátů – více v kapitole 3.2.4 Třída DataFormatProvider.

Protože se opět jedná o nastavovací proceduru, je vlastnost *Data* zastoupena klíčovým slovem *value* (pozn. pro pochopení následujícího kódu vlastnost *Data* interně pracuje s proměnnou *dataPacket*.)

```
1     set
2     {
3           if(this.dataPacket != null)
4           {
5                 this.dataPacket.DataChanged -= dataTarget_DataChanged;
6           }
7
8           this.dataPacket = value;
9
10          if(this.dataPacket != null)
11          {
12                this.dataPacket.DataChanged += dataTarget_DataChanged;
13                this.newDataPacketSet();
14          }
15    }
```

3.2.3. TŘÍDA DATAPACKET

Třída *DataPacket* v sobě obsahuje číslo identifikátoru a pole pro data o velikosti 8 bajtů. Identifikátor je předán pouze v konstruktoru a následně nemůže být upraven. Pro nastavení a vrácení dat má třída řadu metod příslušejících různým typům (*bool, byte, Int16, Int32, Int64 a String*)[7]. Všechny typy (kromě bitového) mají znaménkovou (angl. signed) i neznaménkovou (angl. unsigned) variantu. Při nastavení dat má metoda parametr `bool sendImmediately = true`, který indikuje požadavek na okamžité odeslání. Tato volba je výhodná, pokud je do stejného paketu potřeba zapsat více hodnot a odeslat je až všechny najednou – u prvních hodnot se parametr *sendImmediately* vypne (nastavením *false*), u poslední se nechá zapnutý, což způsobí zavolání událostí *DataChanged* a *DataSet* s parametrem `byte id` (v kódu níže jsou události volány prostřednictvím privátních metod *onDataChanged* a *onDataSet*, které se postarají o kontrolu validity událostí). První událost již byla popsána v předešlé kapitole a má za úkol aktualizovat všechny další komponenty, které jsou napojeny na stejná data. Druhá událost *DataSet* je zaregistrována v třídě *Communication*, která bude podrobněji popsána v kapitole 3.3.2 Třída Communication. Tato událost je vlastně požadavkem na odeslání

[7] Jedná se o datové typy v jazyce C#.

nových dat. Následuje ukázka kódu pro nastavení dat jako znaménkový *Int16*. Parametr *byteOffset* udává bajtový posun v datovém paketu a parametr *value* je hodnota, kterou je potřeba vložit do dat.

```
1       public void setDataAsSInt16(byte byteOffset, Int16 value,
                                                        bool sendImmediately = true)
2       {
3               if((byteOffset >= 0) && (byteOffset < 7))
4               {
5                       lock(this.data)
6                       {
7                               this.data[byteOffset] = (byte)(value & 0x00FF);
8                               this.data[byteOffset+1] = (byte)((value >> 8) & 0x00FF)
9                       }
10                      if(sendImmediately)
11                      {
12                              this.onDataChanged();
13                              this.onDataSet();
14                      }
15              }
16      }
```

Pro aktualizaci dat, které přišly z komunikace (z třídy *Communication*), má třída *DataPacket* metodu `void updateData(byte[] newData)`. Ta přijímá nová data, aktualizuje je a volá událost *DataChanged*. V následujícím kódu je opět událost volána prostřednictvím metody *onDataChanged*, čímž překreslí všechny registrované komponenty. Protože příjem dat běží na jiném vlákně a mohlo by dojít k neoprávněnému přístupu k datům, jsou všechny změny a čtení dat opatřeny synchronizačním zámkem (angl. lock), který zajistí správnou koordinaci vláken.

```
1       public void updateData(byte[] newData)
2       {
3               if(newData.Length < 10) return;
4               //po dobu updatu uzamkneme data
5               lock(this.data)
6               {                       //newData[0] reprezentuje SB
7                                       //newData[1] reprezentuje ID
8                       this.data[0] = newData[2];
9                       this.data[1] = newData[3];
10                      this.data[2] = newData[4];
11                      this.data[3] = newData[5];
12                      this.data[4] = newData[6];
13                      this.data[5] = newData[7];
14                      this.data[6] = newData[8];
15                      this.data[7] = newData[9];
16                                      //newData[10] reprezentuje CRC
17              }                       //newData[11] reprezentuje CRC
18              this.onDataChanged();
19      }
```

O přenesení dat do grafického vlákna se stará delegát *updateDataDelegate* (ve své podstatě ukazatel na metodu z jazyka C/C++), který ukazuje právě na metodu *updateData*.

3.2.4. TŘÍDA DATAFORMATPROVIDER

Tato třída slouží pro komponenty, konkrétně pro výběr formátu přebíraných dat. Obsahuje několik vlastností, které definují, jak přesně budou data z datového přenosu prezentována. Mezi důležité vlastnosti patří:

- *VariableSize* – určuje velikost proměnné v bitech (*1b*, *8b*, *16b*, *32b* nebo *64b*).
- *SignedVariable* – určuje, zda proměnná může být záporná (*true* / *false*).
- *ByteOffset* – určuje bajtový posun v paketu (0 – (7-*VariableSize* v bajtech)).
- *BitOffset* – určuje bitový posun v bajtu (0 – 7).

Zároveň třída obsahuje vlastnosti *VariableOffset* a *VariableMultiplier*, prostřednictvím kterých lze upravit výslednou hodnotu (angl. value) podle vztahu:

$$Value = (Value * VariableMultiplier) + VariableOffset$$

Mezi důležité funkce patří metody pro nastavení a získání proměnné – *getVariable* a *setVariable*. Tyto metody přesně podle navoleného datového formátu načtou hodnotu z objektu *dataPacket*, který je předán prostřednictvím metody *setDataPacket*. Toto předání je většinou voláno z metody *newDataPacketSet*. Následující část kódu ukazuje využití této třídy v komponentě akcelerometru (angl. accelerometer). Atribut *Category* na řádcích 4 a 11 slouží pro určení kategorie, ve které se bude daná vlastnost zobrazovat. Na řádcích 5 a 12 je speciální atribut pro typovou konverzi. Tento atribut spolu s definovaným konvertorem *ExpandableObjectConverter* vytváří možnost „rozkliknutí“ vlastností *AxisHorAttachTo* a *AxisVerAttachTo* (řádky 6 a 13), což způsobí zobrazení jejich interních vlastností – tedy vlastností třídy *DataFormatProvider*.

```
1       /// <summary>
2       /// Horizontální osa.
3       /// </summary>
4       [Category("Communication")]
5       [TypeConverter(typeof(ExpandableObjectConverter))]
6       public DataFormatProvider AxisHorAttachTo […]
7
8       /// <summary>
9       /// Vertikální osa.
10      /// </summary>
11      [Category("Communication")]
12      [TypeConverter(typeof(ExpandableObjectConverter))]
13      public DataFormatProvider AxisVerAttachTo […]
```

Akcelerometr vyžaduje dvě hodnoty – horizontální a vertikální zrychlení. U jiných komponent to může být jiné. Konkrétně budou vlastnosti komponent rozebrány v kapitole 5.3 Komponenty a jejich vlastnosti.

3.2.5. KOMPONENTY

V této kapitole budou ukázány některé implementace a programová řešení několika ukázkových komponent. Vizuální popis komponent bude až v kapitole 5.3 Komponenty a jejich vlastnosti.

CameraStream

Tato komponenta se stará o zobrazení kamerového přenosu (angl. camera stream) z IP kamer, tedy kamer dostupných přes URL adresu. Konkrétní adresa může vypadat např. takto: http://85.135.84.122:86/mjpg/video.mjpg?COUNTER. Většina těchto kamer poskytuje data ve formátu MJPEG[8], který je před zobrazením nutné dekódovat. K tomuto účelu byl použit framework AForge.NET, konkrétně knihovna AForge.Video.dll verze 2.2.5. Framework je možno zdarma stáhnout zde [15], je omezen pouze LPGL v3 licencí[9].

Nastavení přenosu je řešeno podle návodu dostupného zde [16]:

```
1     private void CameraStream_Load(object sender, EventArgs e)
2     {
3           this.EditableChanged += CameraStream_EditableChanged;
4
5           this.stream = new MJPEGStream(this.SourceUrl);
6           this.asyncSource = new AsyncVideoSource(this.stream);
7           this.asyncSource.NewFrame += asyncSource_NewFrame;
8
9           this.Login = "";
10          this.Password = "";
11    }
```

Vytvoření „streamu“ je řešeno na řádku 5 v kódu výše, na řádku 6 je vytvořen asynchronní příjem a na řádku 7 je registrována událost pro příjem nového snímku.

Následující kód ukazuje obsluhu události *asyncSource_NewFrame* pro přijetí nového snímku. Příjem snímku je asynchronní, proto je nutné opět přepnout do grafického vlákna pomocí *BeginInvoke*. Delegát *updateFrameDelegate* interně volá metodu, která překreslí nový snímek do pozadí komponenty.

```
1     if(this.InvokeRequired)
2     {
3           this.BeginInvoke(this.updateFrameDelegate,
                                    new object[] { eventArgs.Frame.Clone() });
4     }
5     else
6     {
7           this.updateFrame((Bitmap)eventArgs.Frame.Clone());
8     }
```

[8] MJPEG (Motion JPEG), někdy též MJPG, je formát videa, kde snímky jsou kódovány jako JPEG.
[9] LGPL v3 je druh veřejné licence, více na adrese http://www.gnu.org/licenses/lgpl.html

BitmapBox

Tato třída slouží pro načtení obrázku do pozadí. Protože vlastnost *Image*, která v sobě nese informaci o obrázku, neobsahuje cestu k souboru, bylo nutné při načítání předat tuto adresu do vlastnosti *Tag*. Cesty k souboru bylo zapotřebí pro celkové uložení projektu. Kód níže je součástí interní třídy *ImageLocator*, potomka třídy *UITypeEditor*.

```
1       Image image = Image.FromFile(openFileDialog.FileName);
2       image.Tag = openFileDialog.FileName;
```

NumberView

U této třídy i několika dalších tříd nejsou některé vlastnosti předávány hodnotou běžně, ale za pomoci řetězce. Pro potřeby zobrazení výběru řetězce byla vytvořena třída *MyStringConverter*, která definuje abstraktní metodu `string[] getValues()`. Z této třídy se zděděním vytvoří konkrétní konvertor, definující pole řetězcových možností právě metodou *getValues*. V komponentě *NumberView* je to takto:

```
1       private class FormatConverter : MyStringConverter
2       {
3               protected override string[] getValues()
4               {
5                       return new string[] { "DEC", "HEX", "BIN"};
6               }
7       }
```

MovementControl

Komponenta *MovementControl* vysílá dvě hodnoty zároveň – aktuální rychlost (angl. velocity) a natočení (angl. rotation). Při každém pohybu po komponentě je volána metoda *MovementControl_MouseMove* a při každém pohybu, kdy N je rovné či větší než `((this.sendingInterval / 10) + 1)`, jsou odeslána i data (viz řádek 2 v kódu níž). Na uvedeném kódu je vidět i využití parametru *sendImmediately*. U prvního nastavení proměnné (řádek 4) je parametr vypnut nastavením na *false*, při druhém nastavení je již implicitně ponechán zapnutý (řádek 5).

```
1       this.N++;
2       if(this.N >= ((this.sendingInterval / 10) + 1))
3       {
4               this.AxisVerAttachTo.setVariable(this.Velocity, false);
5               this.AxisHorAttachTo.setVariable(this.Rotation);
6               this.N = 0;
7       }
```

Tato komponenta má možnost napojit se i na joystick prostřednictvím již zmíněného frameworku AForge.NET [15]. Stav joysticku je aktualizován pomocí *Timeru* v intervalu

nastaveném vlastností *SendingInterval*. Událost *tmrUpdate_Tick* se pravidelně dotazuje na stav joysticku (řádek 1 v kódu níž) a aktualizuje tak proměnné rychlosti a natočení. *Timer* a joystick pracují pouze tehdy, jsou-li povoleny vlastností *JoystickEnabled*. Následující kód ukazuje část procedury pro aktualizaci natočení, tedy horizontální osy X (angl. x axis).

```
1       AForge.Controls.Joystick.Status status = this.joystick.GetCurrentStatus();
2
3       if((status.XAxis > 0.001) || (status.XAxis < -0.001))
4       {
5               this.rotation = (double)status.XAxis;
6               this.required.X = this.origin.X +
                        (int)(this.rotation * (this.origin.X - this.radius));
7       }
8       else
9       {
10              this.rotation = 0.0;
11              this.required.X = this.origin.X;
12      }
```

3.3. PROJEKT CONVIS

Projekt ConVis, jež je druhou částí aplikace, tvoří graficko-uživatelské prostředí. Obsahuje hlavní formulář *MainMdiForm*, formulář projektu *ProjectForm*, formulář pro nastavení projektu *SettingForm* a několik dalších méně významných formulářů. Kromě nich má projekt i několik tříd pro správu vizuálních částí studia (konzole, stromový prohlížeč komponent, prohlížeč vlastností komponent a stavový řádek) a třídy pro vytváření projektů studia a komunikaci.

3.3.1. FORMULÁŘ PROJECTFORM

Formulář projektu *ProjectForm* je tvořen třídou *ProjectFactory* prostřednictvím metod *createProject* a *loadProject*. Metoda *createProject*, vytvářející novou instanci třídy *ProjectForm* (řádek 1 v kódu níž), přidělí této instanci rodičovský formulář (řádek 2) a vytvoří novou hlavní stránku (řádek 4). Po vytvoření stránky metoda uloží projekt v patřičném formátu do souboru (řádek 6) a formulář zobrazí (řádek 8).

```
1       ProjectForm form = new ProjectForm(this.parentForm, fileName);
2       form.MdiParent = this.parentForm;
3
4       form.addNewPage("Main panel");
5
6       if(form.saveProject())
7       {
8               form.Show();
9               return true;
10      }
```

Metoda *loadProject* je velmi podobná, avšak namísto uložení projekt načte. V třídě *ProjectFactory* existuje ještě třetí metoda – *copyProject*, která vytvoří nový projekt, ovšem s parametry zadaného projektu, tedy udělá kopii na jiné adrese.

Uložení projektu do souboru probíhá totožně jako u komponent. Nejprve je uloženo nastavení projektu, poté nastavení komunikace a nakonec postupně všechny stránky a jejich komponenty. Komponenty se v již zmíněné metodě *savePropertiesToXml* starají o uložení samy. Načtení probíhá opačným postupem, kdy je analyzován XML dokument. Při načtení parametru projektu je nastavena příslušná vlastnost a při načtení komunikace je předáno načítání třídě *Communication* voláním metody *loadPropertiesFromXml*. Stejně nazvaná metoda je volána i při načtení komponenty. Komponenta i typ komunikace jsou určeny XML atributem (např. takto: `<component type="sensors_led">`, kde „component“ je element a „sensors_led“ je atribut). Načítání pak probíhá přesně podle obr. 7 na str. 20.

Komponenty jsou vytvářeny v třídě projektu *ProjectForm* voláním metody `MyComponent createNewComponent(string componentName)`, která podle jména komponenty (parametr *componentName*) vytvoří a vrátí příslušnou novou komponentu. Jmenný seznam komponent je v souboru „*VisualComponents/Utilities/XmlElements.cs*“.

Metoda `MyComponent setNewComponentProperties(MyComponent component)` má za úkol nastavit komponentě správné jméno (řádek 3 až 7 v kódu níž), aktuálně vybranou polohu (řádek 1) a rozměry (řádek 2). Poloha a rozměry jsou vzaty z atributu *selectedArea*, který je při vytváření komponent tažením po ploše aktualizován. Více v kapitole 5.3 Komponenty a jejich vlastnosti.

```
1       component.Location = this.selectedArea.Location;        //nastaví pozici
2       component.Size = this.selectedArea.Size;                //nastaví velikost
3       component.Text = component.ToString()                   //nastaví text
4               .Replace("VisualComponents.", String.Empty)
5               .Replace("Sensors.", String.Empty)
6               .Replace("Controls.", String.Empty)
7               .Replace("Others.", String.Empty);
```

Další metoda je `MyComponent addComponent(MyComponent component)`, která se stará o přidání komponenty do vizuálního prostředí projektu. Zároveň nastavuje komponentě všechny důležité události a aktualizuje adekvátně projekt. Mezi zmíněné události patří veškeré manipulace s komponentou, událost nastávající při změně důležitých vlastností s cílem nastavit projekt jako neuložený a událost změny popisku komponenty. Tato poslední událost se stará o překreslení stromového prohlížeče komponent. Metoda *addComponent* dále zobrazí komponentu na ploše, v prohlížeči komponent a v prohlížeči vlastností zobrazí její vlastnosti,

více o tom v kapitole 5.1 Prostředí ConVis Studia.

3.3.2. TŘÍDA COMMUNICATION

Třída *Communication* je abstraktní třída, která definuje obecné rozhraní pro navázání komunikace. Veškerá konkrétní rozhraní pro komunikaci, např. rozhraní pro sériovou linku (třída *SerialCom*) nebo internetové připojení, musí dědit z této třídy. Obsahuje několik důležitých vlastností a metod, jako je reference na *dataPacketDictionary*. Dále obsahuje třídu pro výpočet kontrolního součtu a vlastnosti pro zadání polynomu pro výpočet kontrolního součtu `ushort CrcPolynomial` a pro definování maximálního počtu bajtů čekajících na zpracování – `int MaxPendingBytes`.

Rovněž obsahuje abstraktní vlastnosti pro povolení příjmu (angl. enabled), vlastnosti o stavu připojení (angl. connection) a o schopnosti se připojit (angl. is connectable). Součástí jsou také abstraktní metody pro definování odeslání dat `void sendDataPacket(byte[] packet)`, připojení `void connect()`, odpojení `void disconnect()`, uložení `void savePropertiesToXml(XmlWriter writer)` a načtení komunikace `void loadPropertiesFromXml(XmlTextReader reader)`. Tyto abstraktní třídy implementuje potomek adekvátně své funkcionalitě.

V konstruktoru třídy je vytvořen pro všechny *ID* list datových paketů (*DataPacket*), kterým je zde nastavena metoda pro odchycení události *DataSet*. V této metodě je podle *id* načten správný datový paket a data z něj jsou předána metodě `void sendRawDataPacket(byte id, byte[] data)`, která data doplní do předepsaného formátu komunikačního protokolu, viz následující kód:

```
1     byte[] packet = null;
2     //vytvoříme START BYTE a ID paketu
3     packet = new byte[2];
4     packet[0] = START_BYTE;
5     packet[1] = id;
6     //připojíme DATA paketu
7     packet = packet.Concat(data).ToArray();
8     //spočítáme a připojíme CRC
9     this.addCRC(ref packet);
10    //data odešleme ve formátu: START(1B) + ID(1B) + DATA(8B) + CRC(2B)
11    this.sendDataPacket(packet);
```

Jakmile je celý paket ve správném formátu, je předán abstraktní metodě `void sendDataPacket(byte[] packet)`, která se v konkrétní implementaci postará o správné odeslání.

Naopak pro příjem dat musí zděděná třída zavolat připravenou metodu `bool updateDataPacket(byte[] packet, int bytesToRead)` a ta se následně postará o kontrolu správnosti paketu (viz 3.1 Komunikační protokol):

```
1     if(packet == null) return false;                //ověření validity
2     if(packet.Length != 12) return false;           //ověření velikosti paketu
3     //pokud data nezačínají START_BYTEM, jsou špatná
4     if(packet[0] != START_BYTE) return false;
5     if(!this.verifyCRC(packet)) return false;       //ověření CRC
6
7     //pokud nám čeká hodně bajtů k zpracování
8     if(bytesToRead >= this.maxPendingBytes)
9     {
10          //return this.parentProjectForm.saveDataPacket(packet);
11          return true;        //zahodíme je
12    }
13    //zpracujeme paket
14    return this.parentProjectForm.invokeDataPacket(packet);
```

V kódu výše si lze všimnout, že je aktuální paket zcela zahozen (řádek 11) bez zpracování, pokud na zpracování čeká více bajtů než je hodnota proměnné *maxPendingBytes* (řádek 8). Na „zakomentovaném“ řádku 10 je naznačeno uložení dat namísto prostého zahození, v této verzi ale není tato funkce implementována, tedy data jsou ztracena. Pokud vše proběhne správně a procesor zvládá zpracovávat pakety včas, je volána metoda `bool invokeDataPacket(byte[] packet)` z třídy *ProjectForm*. Tato metoda se stará o aktualizaci dat v patřičném vlákně – grafický obsah komponent totiž nelze měnit z jiného vlákna, než ve kterém byla komponenta vytvořena. K tomuto přepnutí slouží asynchronní volání *BeginInvoke*.

3.3.3. TŘÍDA SERIALCOM

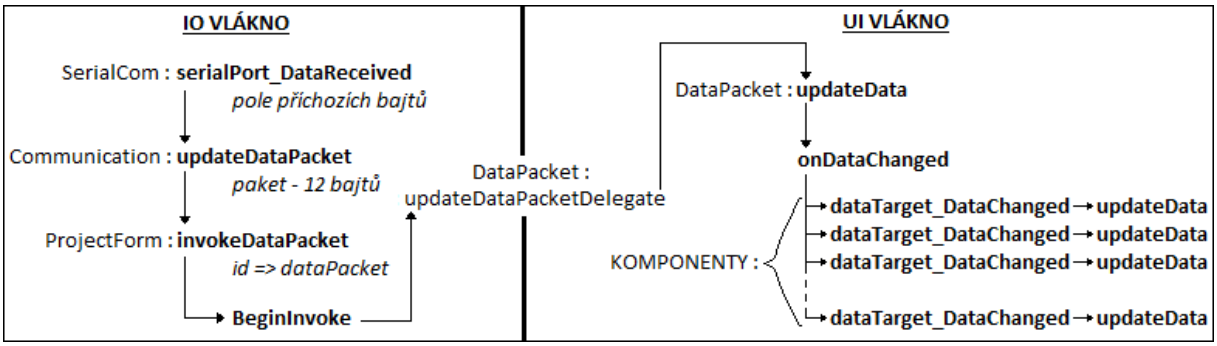
Třída *SerialCom* je potomkem třídy *Communication*, vytvářející rozhraní pro komunikaci po sériové lince. Zpracování dat je řešeno v metodě *serialPort_DataReceived*, což je obsluha události po přijetí dat ze sériové linky.

Následující ukázka kódu popisuje zpracování dat. Na řádku 1 je načteno, kolik bajtů se nachází ve frontě pro čtení. Cyklus na řádku 2 se vykonává, dokud je bajtů ke čtení více, než je aktuální potřeba bajtů k načtení. Na řádku 4 je načten požadovaný počet bajtů k vytvoření celého paketu, který je následně testován na řádku 5. Pokud je test úspěšný, může se pokračovat čtením dalšího paketu. Jestliže ale paket testem neprojde (řádek 10), je v paketu hledán nový „start-byte“ (řádek 15). Je-li „start-byte“ nalezen, předchozí bajty se označí za invalidní (řádek 17) a nastaví se požadovaný počet bajtů ke čtení tak, aby doplnil do celého paketu invalidní bajty (řádek 18). V paketu se následně validní bajty posunou na začátek

(řádek 19) a nakonec je aktualizován počet bajtů ve frontě pro čtení (řádek 24). Z následující ukázky byly odstraněny některé komentáře a části kódu, aby celkově kód zabíral v textu méně místa, avšak neztratil vypovídací schopnost.

```
1       this.bytesToRead = this.serialPort.BytesToRead; //kolik bytů můžeme přečíst
2       while(this.bytesToRead >= this.requiredBytesToRead)
3       {
4               this.serialPort.Read(this.packet, (12 - this.requiredBytesToRead),
                                                        this.requiredBytesToRead);
5               if(this.updateDataPacket(this.packet, this.bytesToRead))
6               {
7                       this.requiredBytesToRead = 12;
8                       this.invalidBytes = 0;
9               }
10              else //jsou-li data špatná
11              {
12                      this.invalidBytes = this.packet.Length;
13                      for(int i = 1; i < this.packet.Length; i++)
14                      {       //nový start byte je nalezen na pozici i
15                              if(this.packet[i] == START_BYTE)
16                              {
17                                      this.invalidBytes = i;
18                                      this.requiredBytesToRead = i;
19                                      Array.Copy(this.packet, i, this.packet, 0, (12 -
                                                        this.requiredBytesToRead));
20                                      break;
21                              }
22                      }
23              }
24              this.bytesToRead = this.serialPort.BytesToRead;
25      }
```

Proces zpracování paketu je ukázán na následujícím obrázku. IO vlákno je označení pro vlákno, které se stará o obsluhu vstupně / výstupních periferií (angl. input / output). Druhé vlákno je vlákno uživatelského rozhraní (angl. user interface), což je také hlavní vlákno programu.



Obr. 8: Proces zpracování dat ze sériové linky

4. TESTOVÁNÍ APLIKACE

Testování aplikace probíhalo trojím způsobem. Nejprve bylo testováno samotné studio pokusným zkoušením všech prvků. To mělo za cíl vyladit případné chyby v GUI, vyladit správnou funkci všech tlačítek i možností v menu a odhalit funkčnost všech módů, popisků a nastavení. Jako u tvorby většiny aplikací jsou tyto chyby velmi nepříjemné a mohou způsobit nepoužitelnost některé části studia. Součástí testů bylo i ověření uložení a ověření načtení projektu. Nejčastější chyby zde byly v překlepech, případně v chybně nastaveném parsování z XML. Všechny následující projekty jsou přiloženy v elektronické formě na CD.

4.1. TESTOVÁNÍ KOMPONENT

Dále byly testovány samotné komponenty, jejich přidávání do projektu, úpravy jejich vlastností i samotná vizualizační funkčnost komponent. K tomuto účelu byl vytvořen projekt s názvem „test_komponent.cvp“. Součástí projektu jsou interně propojené komponenty – senzorické komponenty jsou tak testovány využitím svých řídicích protějšků. Tedy např. komponenta tlačítka zapíná a vypíná LED, ovladač pohybu překresluje grafy pro rychlost a natočení, či posuvník nastavuje ručku ručkového měřicího metru. Projekt je ukázán na dalším obrázku.

Obr. 9: Testování funkčnosti komponent

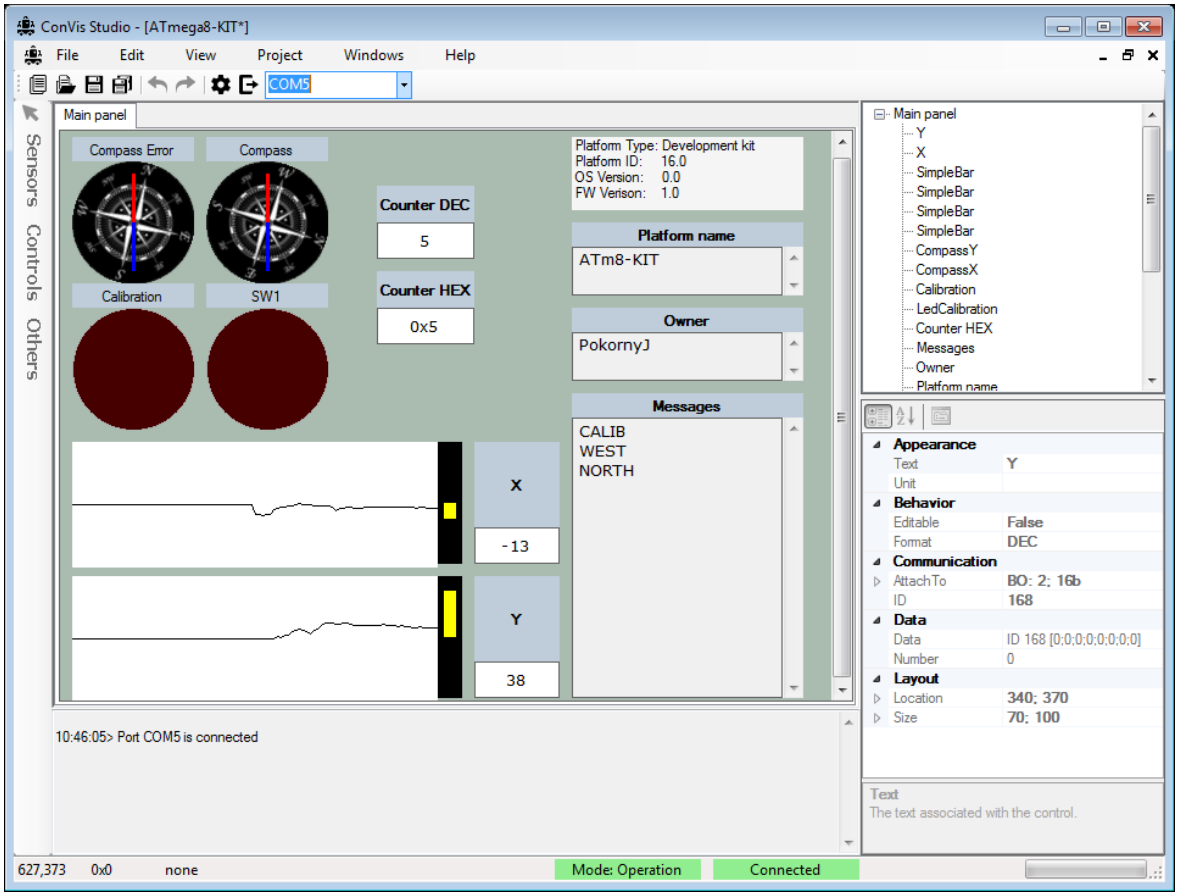
Toto testování odhalilo velmi časté špatné provázání komponent, včetně špatně

nastaveného formátování dat. Ukázalo i některé problémy s událostmi, které se občas netriviálně překrývají a způsobují špatné chování komponent. Zejména u komponenty ovladače pohybu se pohybem po ploše neustále volá událost *MouseMove*, která způsobí, že interní vzorkovací *Timer* nevolá včas svou událost *Tick*. U této komponenty byly problémy s událostmi vyřešeny tak, že v případě ovládání komponenty myší jsou data odesílána nikoli v předepsaném čase, jak je tomu při řízení joystickem, ale po určitém počtu pohybů.

Nejčastějším ladicím prostředkem byl výpis do konzole, případně ve Visual Studiu „Debug“ režim, který je možno spustit v menu „DEBUG→Start Debugging“.

4.2. TESTOVÁNÍ KOMUNIKAČNÍHO PROTOKOLU

Prvotní test spojení proběhl na vývojovém kitu ATmega8-KIT, který byl vytvořen v rámci předmětu KAE/SAC. Kit využívá mikrokontrolér ATmega8 a obsahuje kompas, dvě tlačítka a čtyři LED. Původní program, který odesílal data do PC, se doplnil pouze do předepsaného komunikačního protokolu, ostatní funkčnost změněna nebyla. V tomto případě byl kit spojen prostřednictvím USB-UART převodníku.

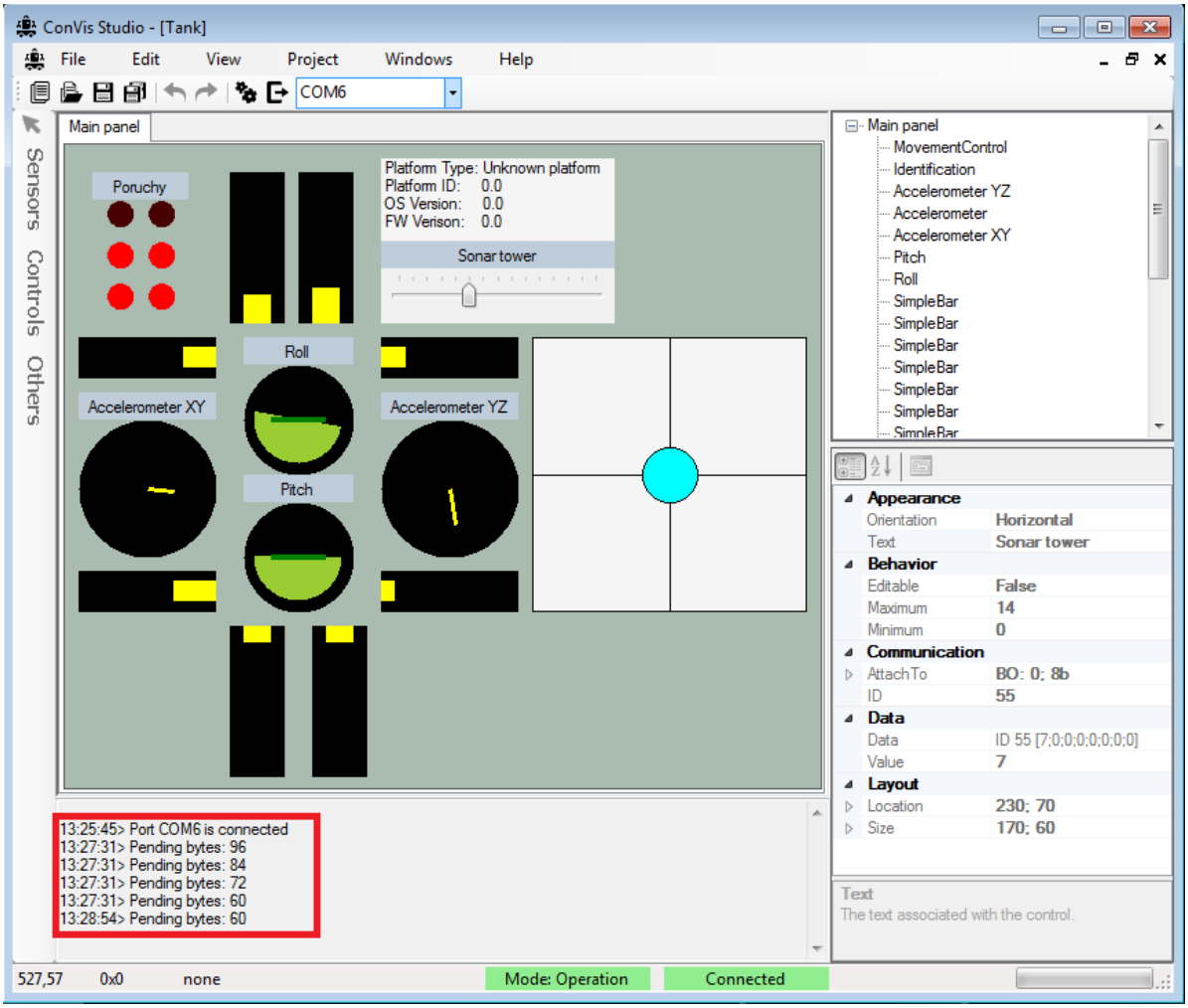


Obr. 10: Testování spojení s kitem ATmega8-KIT

Testy ukázaly problémy v návrhovém módu se zahlcováním fronty pro příchozí data. Řešením bylo, že v návrhovém módu se všechna příchozí data odstraňují. Další problémy nastaly při příchodu špatného paketu, kdy program následně nebyl schopen najít správná data. Řešením je, že se špatný paket ihned neodstraní, ale prozkoumá se, jestli v něm nezačínají jiná platná data.

4.3. TESTOVÁNÍ BEZDRÁTOVÉHO ŘÍZENÍ

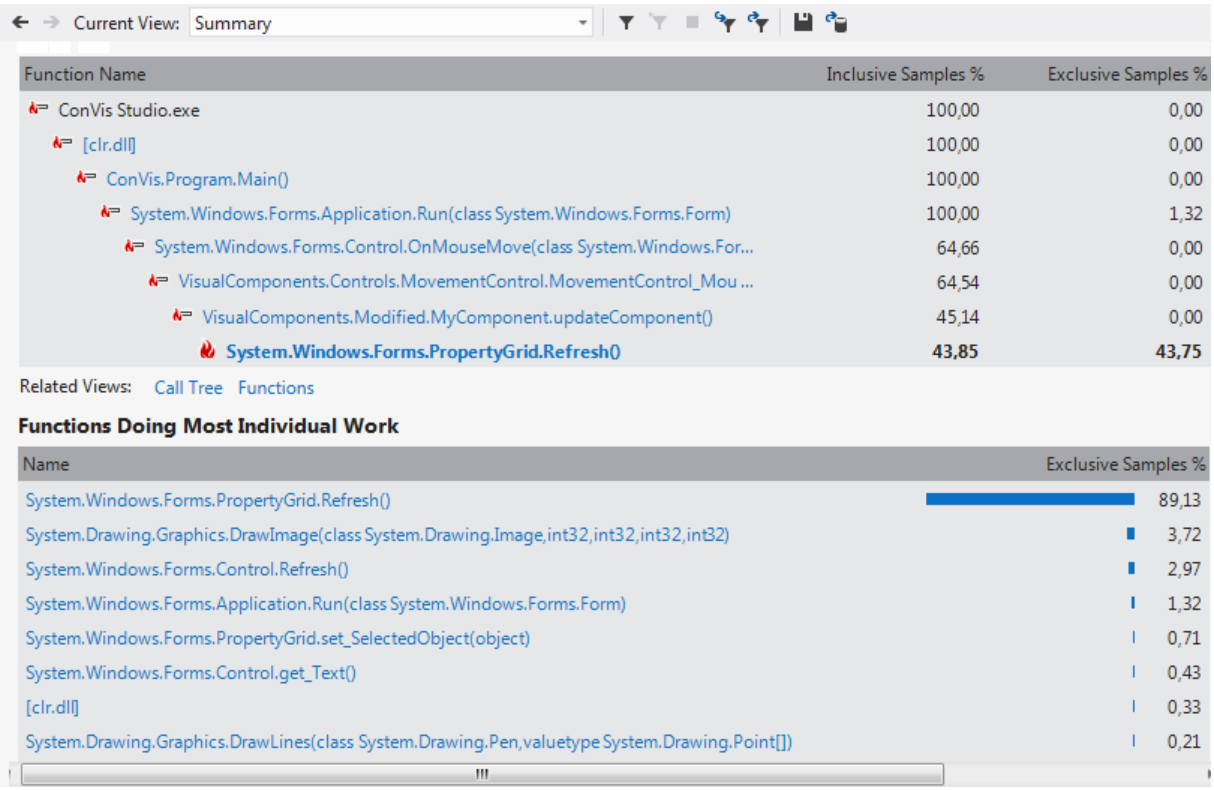
Bezdrátové spojení bylo testováno využitím stávající platformy Tank, která je dostupná na KAE. K bezdrátovému přenosu byl využit LogiLink USB Bluetooth V4.0 Dongle. Tento modul prostřednictvím Bluetooth a speciálních ovladačů vytváří virtuální sériový port v PC, což je vhodné pro ConVis Studio. Po vyzkoušení řady ovladačů, včetně těch oficiálních, se osvědčily až ovladače společnosti Toshiba – Toshiba Bluetooth Stack Driver 9.1.



Obr. 11: Testování spojení s platformou Tank

Samotné testy ale přinesly i jiné nepříjemné problémy. Platforma Tank totiž na rozdíl od

předchozího testovacího kitu odesílala mnohem více dat v kratších intervalech. To mělo za následek, že program nebyl schopen ani data včas zpracovávat, překreslovat komponenty ani data odesílat. V konzoli v červeném rámečku na obr. 11 výše jsou vidět zprávy z komunikace, informující o nadměrném počtu bajtů čekajících na zpracování (angl. pending bytes). Narůstající počet bajtů po chvíli způsobil „sekání“ programu, a program tak přestal být použitelný. První vylepšení bylo provedeno úpravou datového toku mezi vlákny (viz obr. 8 na str. 30). Předtím se totiž přepnutí vlákna provádělo až na úrovni komponenty, tedy v metodě *updateData*. Jak je vidět na obr. 8, byla metoda volána tolikrát, kolikrát byly komponenty napojeny na stejná data. To vedlo k velkému počtu přepnutí vláken, což je časově náročná operace. Řešením bylo posunutí přepnutí vlákna až na úroveň projektu. Přestože výsledkem bylo zlepšení, stále nedostačovalo potřebám náročnějších projektů.



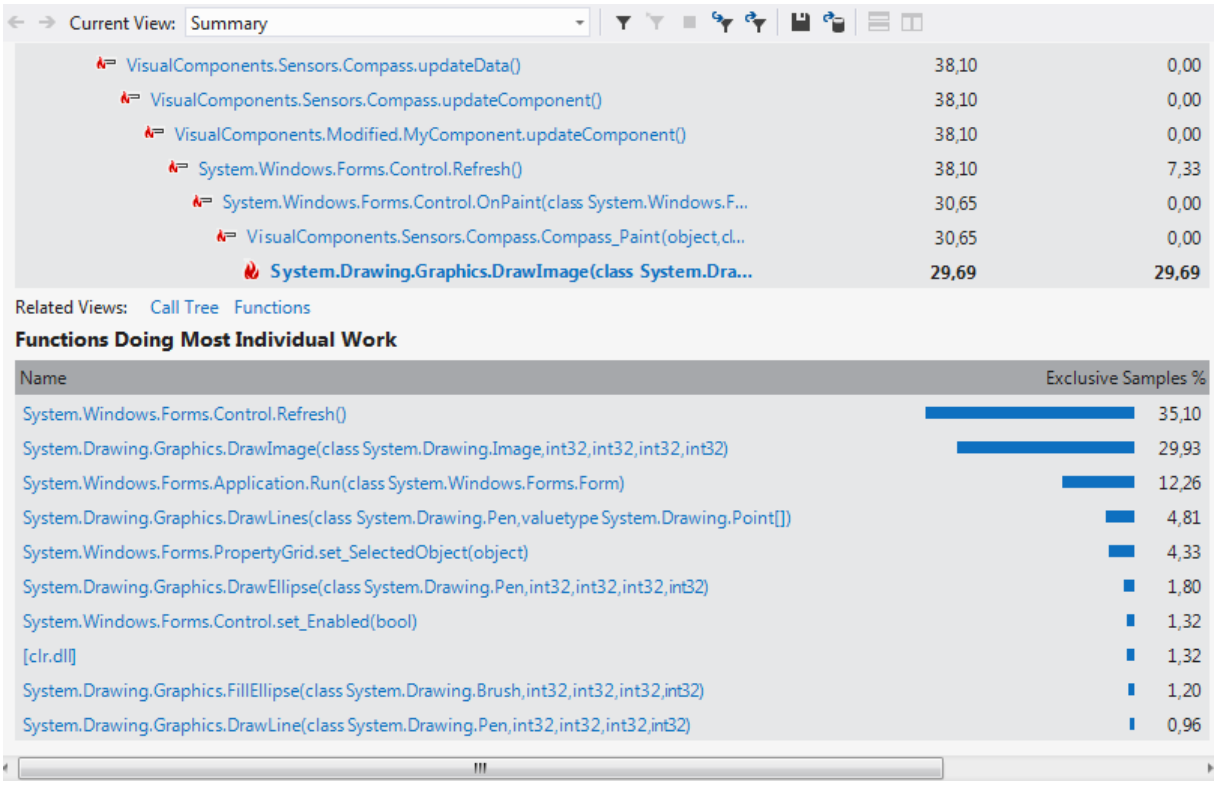
Obr. 12: „Profiling“ programu s obnovou prvku PropertyGrid

K nalezení dalšího problému se využil nástroj zvaný „profiling“, který je dostupný ve Visual Studiu v menu „DEBUG→Start Performance Analysis“. Tato funkce spustí program normálním způsobem, ale zároveň monitoruje vytížení procesoru. Výstupem je tedy zpráva, která oznamuje, kolik času procesor strávil nad provedením jednotlivých metod. Využitím této funkce se nalezl výrazný problém s obnovou (angl. refresh) prohlížeče vlastností komponent – prvku *PropertyGrid*, viz obr. 12 výše. S každou aktualizací si musel prvek znovu zjistit, jaké vlastnosti daná komponenta obsahuje, což zpráva ukázala jako velmi

časově náročnou operaci – téměř 90% času trávil procesor vykonáváním této funkce. Řešením bylo vypnutí aktualizací tohoto prvku „zakomentováním“ kódu, kde se *PropertyGrid* přiděloval komponentám, tedy v metodě *addComponent* v *ProjectForm*:

```
1       // NEPOUŽÍVAT - způsobí velké vytížení CPU!
2       //component.PropertyGrid = this.ParentMainForm.PropertyView.PropertyGrid;
```

Výsledek po „zakomentování“ je vidět na dalším obrázku. Zde je procesor nejvíce zatěžován metodou samotného překreslení komponenty – *Control.Refresh()* a metodou vykreslení obrázku komponenty – *Graphic.DrawImage(...)*. Určitě i zde by bylo vhodné najít do budoucna lepší řešení – pro nynější řídicí aplikace jsou však předešlé optimalizace již dostatečné.



Obr. 13: „Profiling“ programu bez obnovy prvku PropertyGrid

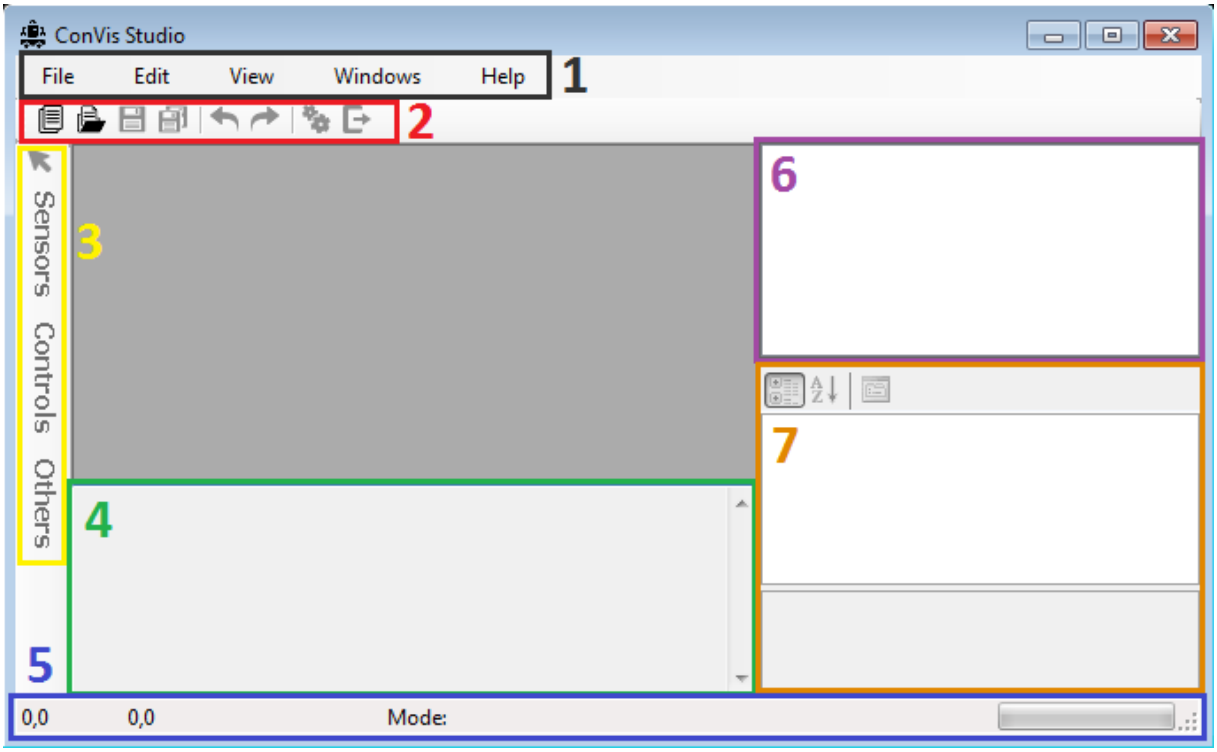
5. CONVIS STUDIO – NÁVOD

ConVis Studio (dále jen studio) bylo navrženo jako univerzální program pro řízení a vizualizaci s přehledným a jednoduchým graficko-uživatelským rozhraním. V dalších podkapitolách bude postupně ukázáno, jak s tímto studiem zacházet.

5.1. PROSTŘEDÍ CONVIS STUDIA

Po spuštění studia se otevře standardní MDI[10] aplikace, připravená spravovat projekty. Na obrázku 14 je zobrazeno několik klíčových prvků, které studio obsahuje:

1 – oblast menu,
2 – rychlé menu s často využívanými funkcemi,
3 – „ToolBox“, ze kterého je možné vybírat komponenty pro projekt,
4 – konzole,
5 – stavový řádek,
6 – větvená struktura projektu,
7 – prohlížeč vlastností komponent.



Obr. 14: Zvýrazněné důležité části hlavního formuláře

Menu nabízí klasické možnosti jako u většiny aplikací. V první položce „File“ je možné vytvořit nový projekt, otevřít projekt, uložit projekt a zavřít projekt i celé studio. Jsou zde

[10] MDI (Multiple-Document Interface) – rozhraní umožňující zobrazit více dokumentů současně.

připraveny i položky pro tisk, které však nejsou v této verzi implementovány. Rovněž položka „Edit“ pro editaci není implementována. V další položce „View“ je možné přepínat mezi zobrazením XML kódu dokumentu nebo návrhovým editorem projektu a zobrazit či skrýt většinu klíčových částí studia (položky 2 až 7 na obr. 14). V položce „Windows“ lze snadno přepínat mezi projekty a upravovat jim jejich tvar ve studiu. Poslední položka je „Help“, přes kterou je možné otevřít tuto dokumentaci a zobrazit informace o vývoji a komunikačním protokolu. Položka „Project“ se zobrazí až po vytvoření či otevření některého projektu, proto na obr. 14 chybí.

Rychlé menu obstarává často využívané funkce, jako je: nový projekt, načíst projekt, uložit projekt, uložit všechny otevřené projekty, zpět, vpřed, přepnutí módu, připojení a rychlý výběr portu sériové linky.

Konzole slouží pro výpisy zpráv z komunikace. Zobrazuje čas připojení a odpojení, případně překročení fronty pro zpracování dat (vlastnosti *MaxPendingBytes*), a také zobrazuje invalidní bajty, tj. počet bajtů v datech, které neprošly kontrolním součtem, tedy byly přeneseny chybně či neúplně.

Stavový řádek zobrazuje polohu kurzoru, velikost vybraného okna při tažení kurzoru, vybranou komponentu, mód návrhu a stav připojení.

Prohlížeč komponent zobrazuje všechny projektové komponenty, rozdělené podle stránek, kde se nalézají.

Poslední částí je prohlížeč vlastností komponent. V této části je možné vybrané komponentě číst a nastavovat její vlastnosti. Vlastnosti každé komponenty budou rozebrány v kapitole 5.3 Komponenty a jejich vlastnosti.

5.2. ZALOŽENÍ A SPRÁVA PROJEKTU

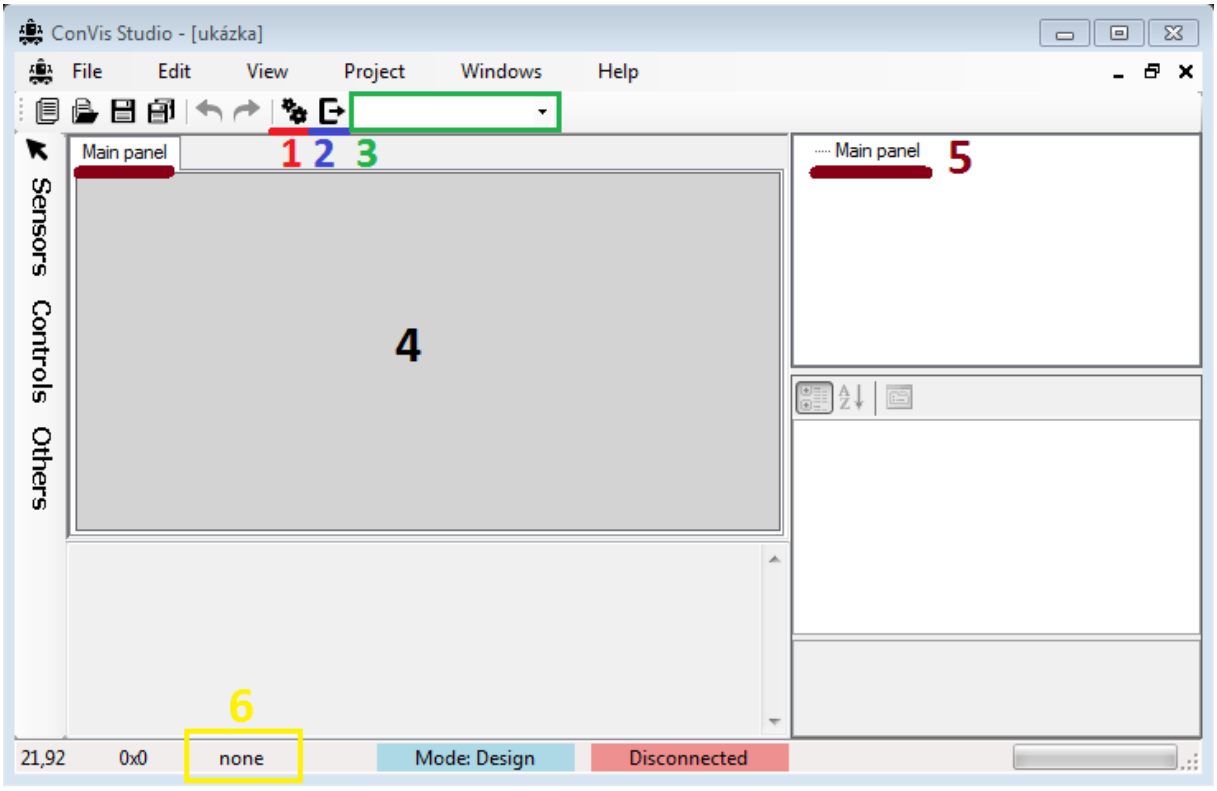
Pro založení nového projektu je třeba kliknout na „File→New→Project“ nebo na ikonu v rychlém menu. Objeví se okno „Uložit jako“, kde si uživatel navolí cestu, kam chce projekt uložit. Aplikace se následně ještě zeptá, zda se má pro projekt vytvořit vlastní složka. Po úspěšném založení se vytvoří soubor s daným názvem a koncovkou „cvp“. Soubor je možné mimo jiné otevřít i v běžném textovém editoru, kde se zobrazí zdrojový XML kód. Tento kód je sice možné editovat ručně, ale není to příliš doporučováno, protože by mohlo dojít k narušení integrity projektu. Projekt by se pak mohl otevřít s nesprávným nastavením nebo

by nešel otevřít vůbec.

Jakmile je nový projekt úspěšně otevřen, je v aplikaci odblokováno několik dalších možností (viz obr. 15 dále):

1 – přepnutí mezi módem návrhu a operačním módem,
2 – připojení či odpojení komunikace,
3 – rychlá volba sériového portu,
4 – prostor pro přidávání komponent,
5 – hlavní stránka dokumentu zobrazená ve stromovém prohlížeči,
6 – aktuálně vybraná komponenta (na obr. žádná – angl. none).

Dole ve stavovém řádku přibyla i další dvě políčka – světle modré políčko, které označuje mód (angl. mode) a světle červené s označením stavu připojení. Základní mód po vytvoření projektu je návrhový mód (angl. mode: design). V tomto módu projekt nepřijímá žádná data, je možné umisťovat komponenty na plochu, měnit jejich vlastnosti a konfigurovat celý projekt včetně komunikace. Druhý je operační mód (angl. mode: operation), v němž se projekt uzamkne a je připraven přijímat data. Přepnutím do tohoto módu se rovněž nasimuluje transparentnost objektů, odblokuje se řízení kontrolních prvků a spustí přenos kamery.



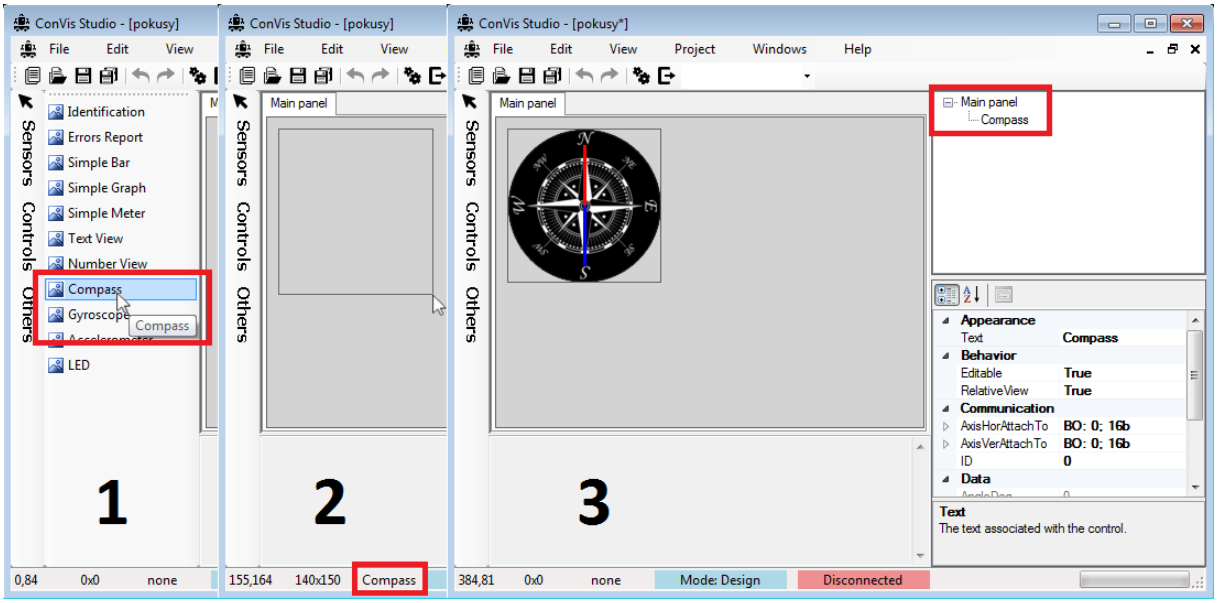
Obr. 15: Zvýrazněné důležité části po založení projektu

Pro přidávání komponent se v návrhovém módu odblokoval „ToolBox“, který po kliknutí na jednu ze tří kategorií zobrazí seznam dostupných komponent. Více v kapitole 5.3 Komponenty a jejich vlastnosti. Pokud prostor pro přidávání komponent (šedá plocha s číslem

4) nestačí, je možné kliknutím v menu na „Project→Add Page“ přidat další stránky s vlastním unikátním prostorem. Přejmenování stránky je možné dvojklikem na příslušnou záložku stránky. Nepotřebné stránky je možné odstranit v menu „Project→Remove Page“, kromě první, která odstraněna být nemůže. Strukturu stránek a komponent, které jsou na stránkách umístěny, zobrazuje stromový prohlížeč (položka 5 na obr. 15 výše).

5.3. KOMPONENTY A JEJICH VLASTNOSTI

Komponenty lze v návrhovém módu vybrat v „ToolBoxu“ kliknutím na příslušné tlačítko nebo pomocí prohlížeče komponent, dostupného v menu „Project→Add Component…“. Po výběru se typ komponenty zobrazí dole ve stavovém řádku, jak je naznačeno na následujícím obrázku 16 červeným rámečkem s číslem 2. Projekt tím označuje, že je připraven komponentu táhnutím či kliknutím přidat na plochu. Kliknutím je přidána komponenta s výchozí velikostí. Postup pro vytvoření komponenty táhnutím po ploše je na následujícím obrázku. 1. část je výběr komponenty, 2. část ukazuje tažení po ploše a v 3. části je zobrazena finální komponenta na ploše i v prohlížeči komponent. Komponenta je vytvořena v místě počátku tažení s rozměry definovanými výběrem. Všechny rozměry jsou zarovnány podle nastavení zarovnávací mřížky (angl. grid).



Obr. 16: Ukázka přidání komponenty kompasu

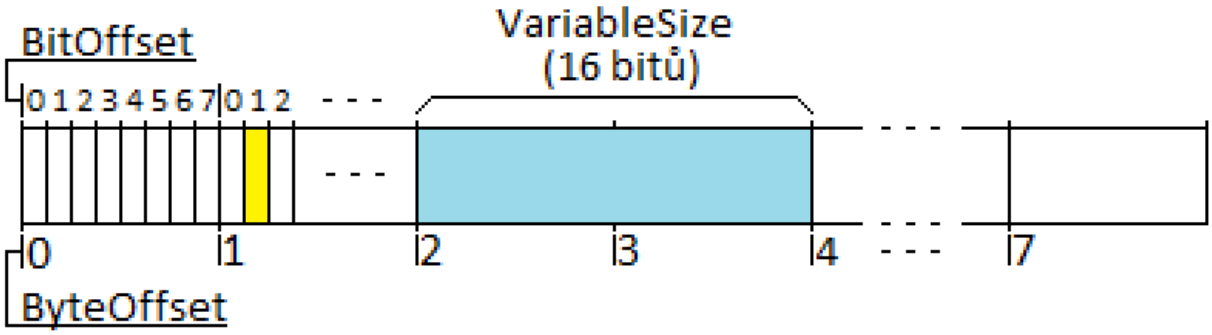
V případě potřeby je možné kliknutím na menu „Project→Remove Component“ komponentu z projektu odstranit.

Všechny komponenty mají několik vlastností.

- *Text* – popisek dané komponenty.
- *Editable* – upravitelnost. Vlastnost určuje, zda je komponentu možné editovat.
- *Location* – poloha komponenty. *X* je horizontální osa, *Y* je vertikální osa.
- *Size* – velikost komponenty. *Width* je šířka, *Height* je výška.

Připojitelné komponenty mají vždy vlastnost *ID*, která filtruje příslušná data podle jejich identifikátoru (viz kapitola 3.1 Komunikační protokol), a vlastnost *Data*, reprezentující celý datový paket. Téměř vždy mají připojitelné komponenty i vlastnost *AttachTo* (či **AttachTo*, kde * je doplňující text) pro výběr formátu dat. Tato struktura obsahuje několik dílčích vlastností, které určují přesný formát, v jakém jsou data prezentována. Mezi tyto vlastnosti patří zejména:

- *BitOffset* – určuje bitový posun v bajtu. Přípustné hodnoty jsou 0 až 7. Hodnotu lze nastavit jen v případě *VariableSize* nastavené na *1b*.
- *ByteOffset* – určuje bajtový posun v datech. Přípustné hodnoty jsou 0 až 8 *mínus velikost proměnné v bajtech* (*1b* – 1 bit je považován taky jako velikost 1 bajt). Tedy pro *VariableSize* rovno *16b* (2 bajty), je rozsah *ByteOffset* 0 až 6.
- *SignedVariable* – určuje, zda je proměnná chápána jako znaménková, tedy zda může být záporná.
- *VariableMultiplier* – násobitel hodnoty proměnné, zadáván jako desetinné číslo. Hodnota nemůže být nastavena na 0.
- *VariableOffset* – posun hodnoty proměnné, zadávaný jako desetinné číslo.
- *VariableSize* – velikost proměnné v bitech. Hodnoty na výběr jsou *1b*, *8b*, *16b*, *32b* a *64b*.



Obr. 17: Ukázka formátového připojení „AttachTo“

Například je-li požadováno napojit se na 2. bit v 2. bajtu dat – žluté políčko na

předchozím obrázku – nastaví se *BitOffset* na 1 (indexuje se od 0), *ByteOffset* na 1 (taktéž se indexuje od 0) a *VariableSize* na *1b*. Toto je příklad ukázky formátu pro napojení tlačítka (angl. button), které na tomto bitu bude měnit hodnotu. Pokud bude potřeba připojit ručkový metr (angl. meter) na 16 bitová znaménková data začínající 3. bajtem (světle modré políčko na předchozím obrázku), nastaví se *ByteOffset* na 2, *SignedVariable* na *True* a *VariableSize* na *16b*. Hodnotu (angl. value) ze získaných dat je možné upravit podle vztahu: $Value = (Value * VariableMultiplier) + VariableOffset$.

5.3.1. SENZORY

5.3.1.1. Akcelerometr

Akcelerometr (angl. accelerometer) je komponenta pro indikaci zrychlení ve dvou osách – horizontální a vertikální. Doporučený rozsah *ID* je 139 (0x8B) až 150 (0x96).

Speciální vlastnosti:

- *BarColorCritical* – barva vektoru po překročení rozsahu.
- *BarColorNormal* – normální barva vektoru.
- *Range* – rozsah vektoru.
- *AxisHorAttachTo* – napojení hodnoty zrychlení pro horizontální osu.
- *AxisVerAttachTo* – napojení hodnoty zrychlení pro vertikální osu.
- *Acceleration* – velikost celkové akcelerace.

5.3.1.2. Kompas

Kompas (angl. compass) je komponenta simulující natočení podle arkustangens hodnot vertikální a horizontální osy. Doporučený rozsah *ID* je 162 (0xA2) až 169 (0xA9).

Speciální vlastnosti:

- *RelativeView* – relativní pohled. Hodnota *False* nastaví stojící ručku a otočný kompas, hodnota *True* nastaví otočnou ručku a pevný kompas natočený na sever.
- *AxisHorAttachTo* – napojení hodnoty pro horizontální osu.
- *AxisVerAttachTo* – napojení hodnoty pro vertikální osu.
- *AngleDeg* – natočení ve stupních.
- *AngleRad* – natočení v radiánech.

5.3.1.3. Gyroskop

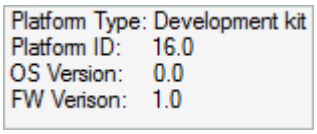
Gyroskop (angl gyroscope) je komponenta pro zobrazení náklonu. Doporučený rozsah *ID* je 155 (0x9B) až 158 (0x9E).

Speciální vlastnosti:

- *RelativeView* – relativní pohled. Hodnota *False* nastaví stojící zemskou osu a otočnou osu gyroskopu, hodnota *True* nastaví natáčení země vůči pevné ose gyroskopu.
- *AttachTo* – napojení na úhel pro gyroskop. Úhel je načítán ve stupních.
- *AngleDeg* – natočení ve stupních.
- *AngleRad* – natočení v radiánech.

5.3.1.4. Identifikace

Identifikace (angl. identification) je komponenta zobrazující informace o typu platformy, ID platformy, verzi operačního systému a verzi firmwaru. Doporučené *ID* je 1 (0x01). Přípustné hodnoty pro typ platformy jsou tyto:



- 0x00 – neznámá platforma (angl. unknown platform),
- 0x01 – satelit (angl. satellite),
- 0x02 – vzducholoď (angl. airship),
- 0x03 – letadlo (angl. aircraft),
- 0x04 – balón (angl. balloon),
- 0x05 – helikoptéra (angl. helicopter),
- 0x06 – dron (angl. dron),
- 0x07 – vznášedlo (angl. hovercraft),
- 0x08 – kolové vozidlo (angl. wheeled vehicle),
- 0x09 – pásové vozidlo (angl. tracked vehicle),
- 0x0A – obojživelník (angl. amphibian),
- 0x0B – loď (angl. ship), 0x0C – ponorka (angl. submarine),
- 0x0D – vlak (angl. train),
- 0x0E – pevná stanice (angl. stationary station),
- 0x0F – pohyblivá stanice (angl. mobile station),
- 0x10 – vývojový kit (angl. development kit).

Speciální vlastnosti:

- *FwVersion* – verze firmwaru.
- *OsVersion* – verze operačního systému.
- *PlatformID* – identifikace platformy.
- *PlatformType* – typ platformy.

5.3.1.5. Led

Led je komponenta simulující skutečnou svítivou diodu.

Speciální vlastnosti:

- *LedColor* – barva diody. Je vhodné volit tmavší barvy, aby lépe vynikl rozdíl mezi zhasnutou a rozsvícenou LED.
- *Negated* – negování stavu svítivosti.
- *AttachTo* – napojení na stav pro LED.
- *State* – stav z napojených dat, tedy před případnou negací.

5.3.1.6. Číselný zobrazovač

Číselný zobrazovač (angl. number view) je prostý zobrazovač číselné hodnoty s popiskem.

Speciální vlastnosti:

- *Unit* – jednotka (případně text) doplněný za číslo.
- *Format* – formát čísla. Možné jsou tyto formáty: dekadický (*DEC*), hexadecimální (*HEX*) ve tvaru 0x*, či binární (*BIN*) ve tvaru 0b*.
- *AttachTo* – napojení na číslo pro zobrazovač.
- *Number* – zobrazovaná hodnota.

5.3.1.7. Textový zobrazovač

Textový zobrazovač (angl. text view) slouží pro zobrazení textových zpráv. Zpráva může být v datech předčasně ukončena ASCII znakem „EOT“ (0x03 – „End Of Text“). Doporučený rozsah *ID* je 6 (0x06) až 9 (0x09).

Speciální vlastnosti:

- *Overwrite* – přepisování textu. Při hodnotě *True* je původní zpráva přepsána novou zprávou. Při hodnotě *False* jsou zprávy vkládány za sebe.
- *AttachTo* – napojení na zprávu pro zobrazovač.
- *Message* – poslední příchozí zpráva.

5.3.1.8. Jednoduchý sloupcový metr

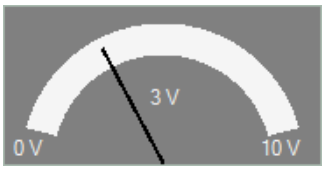
Jednoduchý sloupcový metr (angl. simple bar) je komponenta, která podle hodnoty vybudí barevný sloupec.

Speciální vlastnosti:

- *BackColor* – barva pozadí.
- *BarColorCritical* – barva sloupce při překročení rozsahu.
- *BarColorNormal* – normální barva sloupce.
- *Orientation* – orientace sloupce. Možné jsou všechny čtyři směry.
- *Inverted* – invertování sloupce. V případě *True* je sloupec vybuzen zápornou hodnotou.
- *Offset* – relativní posunutí sloupce. Povolený rozsah je 0.0 až 1.0.
- *Range* – rozsah sloupce.
- *AttachTo* – napojení na hodnotu pro sloupcový ukazatel.
- *Value* – hodnota sloupcového ukazatele.

5.3.1.9. Jednoduchý ručkový metr

Jednoduchý ručkový metr (angl. simple meter) je komponenta podobná ručkovému displeji.

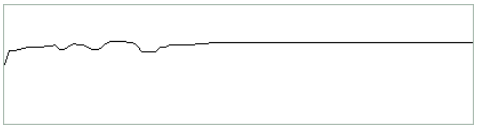


Speciální vlastnosti:

- *Unit* – jednotka (případně text), doplněná za hodnotu ručky a za hodnoty na krajích rozsahu.
- *Maximum* – maximální hodnota rozsahu.
- *Minimum* – minimální hodnota rozsahu.
- *AttachTo* – napojení na hodnotu pro ručkový ukazatel.
- *Value* – hodnota ručkového ukazatele.

5.3.1.10. Jednoduchý graf

Jednoduchý graf (angl. simple graph) je komponenta pro zobrazení posloupnosti hodnot.



Speciální vlastnosti:

- *KeepValue* – držení poslední hodnoty. Nastavení na *True* způsobí, že je vzorkována

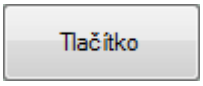
stále stejná poslední hodnota, při nastavení na *False* je po vzorkování poslední hodnoty vzorkována nula. Tato vlastnost má tedy význam jen při zapnutém vzorkování.

- *Range* – rozsah grafu. Zadaná hodnota určuje rozsah v kladné i záporné části. Nula je tedy ve středu grafu.
- *SamplerOn* – zapnutí vzorkovače. V případě *True* jsou hodnoty vzorkovány s periodou definovanou *SampleTime*.
- *Samples* – počet vzorků, které graf zobrazí.
- *SampleTime* – vzorkovací čas. Vlastnost má vliv jen při zapnutém vzorkovači.
- *AttachTo* – napojení na hodnotu pro graf.
- *Value* – poslední zapsaná hodnota.

5.3.2. ŘÍDICÍ PRVKY

5.3.2.1. Tlačítko

Tlačítko (angl. button) je komponenta pro odeslání definované hodnoty.

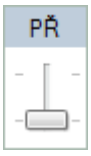


Speciální vlastnosti:

- *FlipValue* – otáčení hodnoty. Pokud je *True*, je hodnota *SendValue* po každém stisku invertována.
- *SendValue* – odesílaná hodnota.
- *AttachTo* – napojení na hodnotu v datech.

5.3.2.2. Přepínač

Přepínač (angl. switch) je komponenta, která odesílá dvě hodnoty. Odeslání proběhne jen při změně jezdce.

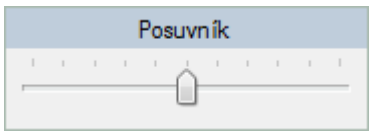


Speciální vlastnosti:

- *Orientation* – orientace přepínače.
- *AttachTo* – napojení na hodnotu v datech.
- *Value* – stav přepínače.

5.3.2.3. Posuvník

Posuvník (angl. slider) je komponenta, která odesílá hodnotu v závislosti na poloze jezdce. Odeslání proběhne jen při změně jezdce.



Speciální vlastnosti:

- *Orientation* – orientace posuvníku.
- *AttachTo* – napojení na hodnotu v datech.
- *Maximum* – hodnota v maximální poloze jezdce.
- *Minimum* – hodnota v minimální poloze jezdce.
- *Value* – aktuální hodnota jezdce.

5.3.2.4. Řízení LED

Řízení LED (angl. LED control) je komponenta pro snadné řízení jednobarevných i RGB LED. Doporučené *ID* je 27 (0x1B).

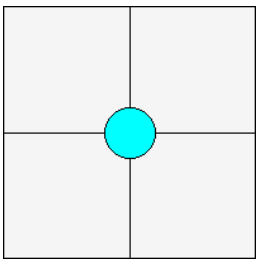
Speciální vlastnosti:

- *LedType* – typ ovládané led. Možná je volba RGB LED i jednobarevné LED.
- *LedColor* – barva LED v případě volby RGB LED.
- *LedID* – identifikátor LED.
- Value – aktuální hodnota jezdce. Tato hodnota určuje procentuelní intenzitu svítivosti LED nebo stav „zapnuto / vypnuto“.

5.3.2.5. Ovladač pohybu

Ovladač pohybu (angl. movement control) je komponenta pro ovládání pohyblivé platformy. Funguje jako jednoduchý joystick – chycením modrého kolečka uprostřed a pohybováním ve vymezeném prostoru jsou v horizontální ose odesílána data o natočení a ve vertikální ose rychlost. Rovněž je možné využít i skutečný fyzický joystick. Komponenta indikuje aktivování joysticku změnou barvy kolečka na červenou. Dvojklikem je možné joystick okamžitě vypnout a odeslat nulovou rychlost i nulové natočení – nouzové zastavení.

Speciální vlastnosti:



- *RotationRange* – rozsah natočení. Určuje rozsah pro kladné i záporné hodnoty.
- *SendingInterval* – interval odesílání dat (v případě používání

joysticku) nebo počet pohybů po ploše komponenty. Počet pohybů je upraven vztahem: $\left(\frac{SendingInterval}{10}\right) + 1$

- *VelocityRange* – rozsah rychlosti. Určuje rozsah pro kladné i záporné hodnoty.
- *AxisHorAttachTo* – napojení horizontální osy – natočení.
- *AxisVerAttachTo* – napojení vertikální osy – rychlosti.
- *Rotation* – aktuální natočení.
- *Velocity* – aktuální rychlost.
- *Joystick* – výběr dostupných fyzických joysticků.
- *JoystickEnabled* – povolení fyzického joysticku.

5.3.3. OSTATNÍ

5.3.3.1. Bitmapa

Komponenta obrázek (angl. bitmap, v aplikaci bitmap box) umožňuje nahrát do pozadí libovolný obrázek.

Speciální vlastnosti:

- *BackgroundImage* – načtený obrázek.
- *BackgroundImageLayout* – zarovnání obrázku.

5.3.3.2. Kamera

Kamera (angl. camera, v aplikaci camera stream) je komponenta pro zobrazení kamerového přenosu z IP kamer. Kamera je aktivována vždy při přechodu z návrhového módu do operačního.

Speciální vlastnosti:

- *Login* – uživatelské jméno pro přístup ke kameře.
- *Password* – heslo pro přístup ke kameře.
- *SourceUrl* – zdrojová adresa, ze které je přenášen kamerový obraz.

5.3.3.3. Popisek

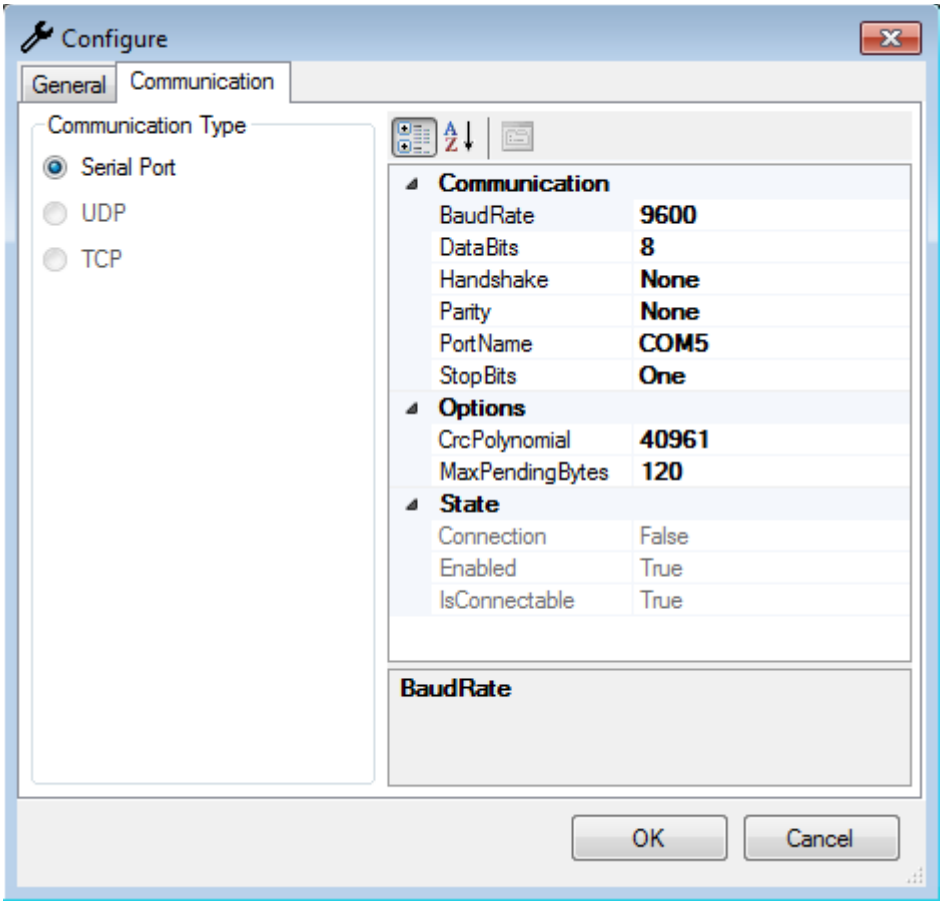
Popisek (angl. caption) slouží pro popisování ostatních komponent. Tato komponenta neobsahuje žádné speciální vlastnosti. V popisku se zobrazuje text vlastnosti *Text*.

5.4. NASTAVENÍ PROJEKTU

Do nastavení projektu se lze dostat přes menu „Project→Configure“, čímž se otevře nové okno s parametry projektu. Nastavení obsahuje v této verzi dvě záložky. První záložka *General* slouží pro obecné nastavení vlastností projektu a druhá *Communication* pro nastavení komunikace.

V první záložce je možné nastavit vertikální a horizontální velikost zarovnávací mřížky, ke které se komponenty zarovnávají. Tyto velikosti jsou dány parametry *Grid X* a *Grid Y* s implicitní hodnotou 10x10. Jiné nastavení pro projekt zatím není k dispozici.

V záložce pro komunikaci je nastavení sériové linky. Ostatní typy komunikace jsou pouze připraveny pro implementaci, v této verzi je tedy není možné vybrat. Sériová linka obsahuje řadu nastavení, viz obrázek 18 níže.



Obr. 18: Nastavení sériové linky

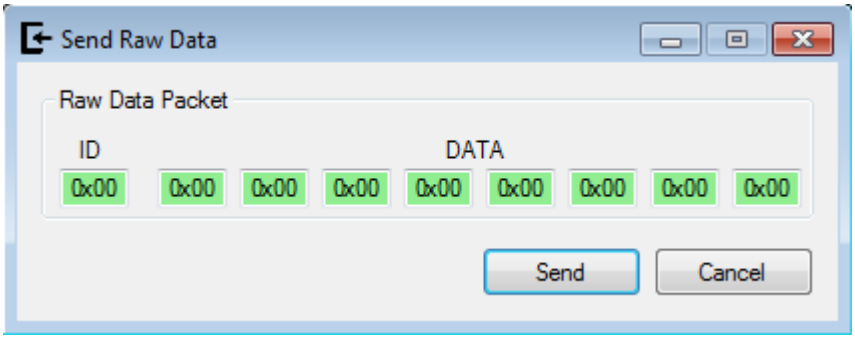
- *BaudRate* – rychlost přenosu.
- *DataBits* – počet datových bitů.
- *Handshake* – řízení datového přenosu.

- *Parity* – formát paritního bitu.
- *PortName* – seznam dostupných portů.
- *StopBits* – počet stop bitů na konci přenášených dat.
- *CrcPolynomial* – koeficienty polynomu pro výpočet kontrolního součtu nad daty. Je možné zadat hodnotu i hexadecimálně vepsáním „0x“ před číslo.
- *MaxPendingBytes* – maximální počet bajtů ve frontě pro zpracování. Pokud ve frontě čeká bajtů více, jsou bajty odstraňovány.

5.5. SPOJENÍ S KOMUNIKAČNÍ LINKOU

Pro spojení je možné využít v této verzi jen sériovou linku. Její nastavení se musí shodovat s nastavením sériové linky v daném zařízení, jinak komunikace nebude fungovat. Otevřením nastavení projektu („Project→Configure“) se aktualizuje seznam dostupných portů. V případě, že žádný z nich není k dispozici, je zobrazena zpráva: „Žádný port není k dispozici“ (angl. „No ports available!“). Port lze taktéž vybrat v oblasti rychlého menu. Po nastavení se lze připojit na daný sériový port výběrem tlačítka v menu „Project→Connect“ nebo v rychlém menu stisknutím tlačítka *Connect*. Poté je zobrazeno kompletní nastavení linky ve zprávě, která si žádá potvrdit připojení. Jakmile je studio připojeno, je dole ve stavovém řádku zobrazen stav *Connected*, funkce voleb pro připojení se změní na funkce odpojení (angl. disconnect) a do konzole je zapsán čas otevření linky s hlášením o stavu připojení.

Pokud je studio úspěšně napojeno, je možné vybrat v menu „Project→Send Raw Data“ formulář pro odeslání prostých dat. Formulář ukazuje následující obrázek.



Obr. 19: Formulář pro odeslání prostých dat

6. ZÁVĚR

Cílem práce bylo vytvořit program pro řízení a vizualizaci robotických vozidel. Výstupem této práce je univerzální program ConVis Studio, které je schopno řídit a vizualizovat elektronická zařízení s definovaným komunikačním protokolem. Jedná se o komplexní program, který v této fázi splňuje zadání v celém rozsahu.

Do finální verze studia by bylo vhodné doplnit některé další funkce, například běžné uživatelské funkce, jako je funkce zpět, kopírování, výběr více komponent najednou apod. Ty sice na funkčnost studia nemají vliv, ale podstatně zpříjemňují tvorbu vizualizačních projektů. Zároveň by si program zasloužil i přepracování grafického vzhledu komponent a sjednocení stylu. Jako další rozšíření by bylo vhodné dodělat i internetové připojení, s nímž bylo během vývoje aplikace sice počítáno, ale zatím v kódu chybí.

Projekt pro grafické komponenty je zcela oddělen od jádra studia, což umožňuje jeho vyjmutí a využití v jiných aplikacích. Současně je možné do něj doimplementovat i další potřebné komponenty. Určitě by bylo vhodné do budoucna implementovat komponentu pro zobrazení 3D modelu. Taková komponenta by ovšem pro svou složitost kosterních animací a inverzní kinematiky vydala minimálně na celou bakalářskou práci.

Další prostor pro dokončení je v samotném komunikačním protokolu. Jedná se hlavně o vytvoření definovaných povelů a chyb. Práce nyní obsahuje jen základní povely a chyby, většinou společné pro všechny platformy. Další povely a chyby bude vhodné doplnit, jakmile se přejde na kompletní řízení využitím ConVis Studia a jakmile se bude přesně vědět, jaké povely a chyby daná platforma využívá.

Samotné testování platforem se ukázalo být v pořádku, problémy nastaly většinou jen s Bluetooth ovladači pro vytvoření virtuálního sériového portu. Datový tok pro řízení platformy „Tank“ byl dostačující, „profiling“ ale ukázal, že v případě větších projektů by určitě bylo vhodné optimalizovat některé komponenty, které vykreslují složité obrazce. V tomto konkrétním případě, kdy se zobrazovalo 23 komponent, nebyla evidována ztráta dat větší, než je běžná chybovost sériové linky, tedy program dokázal vše zpracovávat, zobrazovat i odesílat bez nadměrného přetížení.

CITOVANÁ LITERATURA

1. **Vlach, Jaroslav.** *Řízení a vizualizace technologických procesů.* Praha : Nakladatelství BEN, 1999. ISBN 80-86056-66-X.

2. **Wikipedie.** SCADA. [Online] Wikipedie, 7. září 2015. [Citace: 29. březen 2016.] https://cs.wikipedia.org/wiki/SCADA.

3. **MICROSYS, s.r.o.** Co je to PROMOTIC. *PROMOTIC - SCADA visualization software.* [Online] 1. leden 2016. [Citace: 4. duben 2016.] http://www.promotic.eu/cz/pmdoc/WhatIsPromotic/WhatIsPromotic.htm.

4. —. Ceník systému PROMOTIC. *PROMOTIC - SCADA visualization software.* [Online] 1. leden 2016. [Citace: 5. duben 2016.] http://www.promotic.eu/cz/pmdoc/PriceList/PriceList.htm.

5. —. Promotic. *PROMOTIC - SCADA visualization software.* [Online] 1. leden 2016. [Citace: 6. duben 2016.] http://www.promotic.eu/cz/.

6. **Moravské přístroje, a.s.** Co je to Control Web. *Control Web.* [Online] 1. leden 2016. [Citace: 4. duben 2016.] http://www.mii.cz/art?id=380&cat=146&lang=405.

7. —. Kompletní ceník. *Control Web.* [Online] 1. leden 2016. [Citace: 7. duben 2016.] http://obchod.mii.cz/?uid=F4C1BC16-66EB-45ED-9C03-8EF746C106DF&lang=405&cat=20.

8. —. Control Web 7. *Control Web.* [Online] 16. prosinec 2014. [Citace: 8. duben 2016.] http://www.mii.cz/art?id=829&lang=405.

9. **Wikipedia.** LabVIEW. [Online] Wikipedia, 31. březen 2016. [Citace: 4. duben 2016.] https://en.wikipedia.org/wiki/LabVIEW.

10. **National Instruments, s.r.o.** LabVIEW. [Online] 1. leden 2016. [Citace: 4. duben 2016.] http://czech.ni.com/labview.

11. **SourceForge.** Free LabVIEW SQL/ODBC LLB. *SourceForge.* [Online] 1. leden 2016. [Citace: 11. duben 2016.] http://sql-lv.sourceforge.net/.

12. **GEOVAP, s.r.o.** Reliance 4. *Reliance.* [Online] 1. leden 2016. [Citace: 4. duben 2016.] https://www.reliance.cz/cs/products/reliance4-scada-hmi-system#page=overview.

13. —. Ceník systému Reliance. *Reliance.* [Online] 1. leden 2016. [Citace: 5. duben

2016.] https://www.reliance.cz/cs/pricelist/reliance4-pricelist.

14. —. Reliance 4 - Screenshots. *Reliance.* [Online] 1. leden 2016. [Citace: 12. duben 2016.] https://www.reliance-scada.com/en/products/reliance4-scada-hmi-system#page=screenshots.

15. **AForge.NET.** *AForge.NET Framework.* [Online] 31. prosinec 2012. [Citace: 8. únor 2016.] http://www.aforgenet.com/framework/downloads.html.

16. —. AsyncVideoSource Class. *AForge.NET Framework.* [Online] 31. prosinec 2013. [Citace: 8. únor 2016.] http://www.aforgenet.com/framework/docs/.

OBRÁZKY

PŘÍLOHY

Přílohy na CD-ROM:

- **Elektronická verze práce**
- **Celá aplikace ConVis Studio**
- **Zdrojové kódy aplikace**
- **Ukázkové projekty vytvořené v ConVis Studiu**
- **Komunikační protokol v XLSX formátu**

Příloha A: Zjednodušený komunikační protokol

Tab. 1: Zprávy ze zařízení a z PC

ID		Informace		ID		Informace	
0	0x00	*Zakázáno*		20	0x14	**Povel/zpráva**	**Zprávy z PC**
1	0x01	**Identifikace**	**Zprávy ze zařízení**	21	0x15	*Vyhrazeno*	
2	0x02	*Vyhrazeno*		22	0x16	**Text 1**	
3	0x03	**Chybové hlášení**		23	0x17	**Text 2**	
4	0x04	*Vyhrazeno*		24	0x18	**Text 3**	
5	0x05	*Vyhrazeno*		25	0x19	**Uživatelský text**	
6	0x06	**Jméno zařízení**		26	0x1A	*Vyhrazeno*	
7	0x07	**Kódové označení**		27	0x1B	**Řízení LED**	
8	0x08	**Firma / Majitel**		28	0x1C	*Vyhrazeno*	
9	0x09	**Uživatelský text**		29	0x1D	*Vyhrazeno*	
10	0x0A	*Vyhrazeno*		30	0x1E	**SPI**	
11	0x0B	*Vyhrazeno*		31	0x1F		
12	0x0C	**Dostupné senzory**		32	0x20	*Vyhrazeno*	
13	0x0D			33	0x21	*Vyhrazeno*	
14	0x0E			34	0x22	**I2C**	
15	0x0F			35	0x23		
16	0x10			36	0x24	*Vyhrazeno*	
17	0x11	*Vyhrazeno*		37	0x25	**Doba odezvy**	
18	0x12	*Vyhrazeno*		38	0x26	*Vyhrazeno*	
19	0x13	**Povel/zpráva**		39	0x27	*Vyhrazeno*	

Tab. 2: Akční členy a senzory

ID		Informace		ID		Informace	
40	0x28	**Motory a Serva**	**Akční členy**	80	0x50	**Ampérmetry**	**Senzory**
41	0x29			81	0x51	*Vyhrazeno*	
42	0x2A			82	0x52	*Vyhrazeno*	
43	0x2B			83	0x53	*Vyhrazeno*	
44	0x2C			84	0x54	**Wattmetry**	
45	0x2D	*Vyhrazeno*		85	0x55		
46	0x2E	*Vyhrazeno*		86	0x56		
47	0x2F	*Vyhrazeno*		87	0x57		
48	0x30	*Vyhrazeno*		88	0x58	*Vyhrazeno*	
49	0x31	*Vyhrazeno*		89	0x59	*Vyhrazeno*	
50	0x32	**Motory**		90	0x5A	**Otáčkoměry**	
51	0x33			91	0x5B		
52	0x34			92	0x5C		
53	0x35	*Vyhrazeno*		93	0x5D		
54	0x36	*Vyhrazeno*		94	0x5E	*Vyhrazeno*	
55	0x37	**Serva**		95	0x5F	*Vyhrazeno*	
56	0x38			96	0x60	*Vyhrazeno*	
57	0x39			97	0x61	**Polohoměry**	
58	0x3A	*Vyhrazeno*		98	0x62		
59	0x3B	*Vyhrazeno*		99	0x63		
60	0x3C	*Vyhrazeno*		100	0x64		
61	0x3D	*Vyhrazeno*		101	0x65	*Vyhrazeno*	
62	0x3E	*Vyhrazeno*		102	0x66	*Vyhrazeno*	
63	0x3F	*Vyhrazeno*		103	0x67	*Vyhrazeno*	
64	0x40	*Vyhrazeno*		104	0x68	**Dálkoměry**	
65	0x41	*Vyhrazeno*		105	0x69		
66	0x42	*Vyhrazeno*		106	0x6A		
67	0x43	*Vyhrazeno*		107	0x6B		
68	0x44	*Vyhrazeno*		108	0x6C	*Vyhrazeno*	
69	0x45	*Vyhrazeno*		109	0x6D	*Vyhrazeno*	
70	0x46	**Voltmetry**	**Senzory**	110	0x6E	*Vyhrazeno*	
71	0x47			111	0x6F	**Teploměry**	
72	0x48			112	0x70		
73	0x49			113	0x71		
74	0x4A	*Vyhrazeno*		114	0x72		
75	0x4B	*Vyhrazeno*		115	0x73	*Vyhrazeno*	
76	0x4C	*Vyhrazeno*		116	0x74	*Vyhrazeno*	
77	0x4D	**Ampérmetry**		117	0x75	**Tlakoměry**	
78	0x4E			118	0x76		
79	0x4F			119	0x77	*Vyhrazeno*	

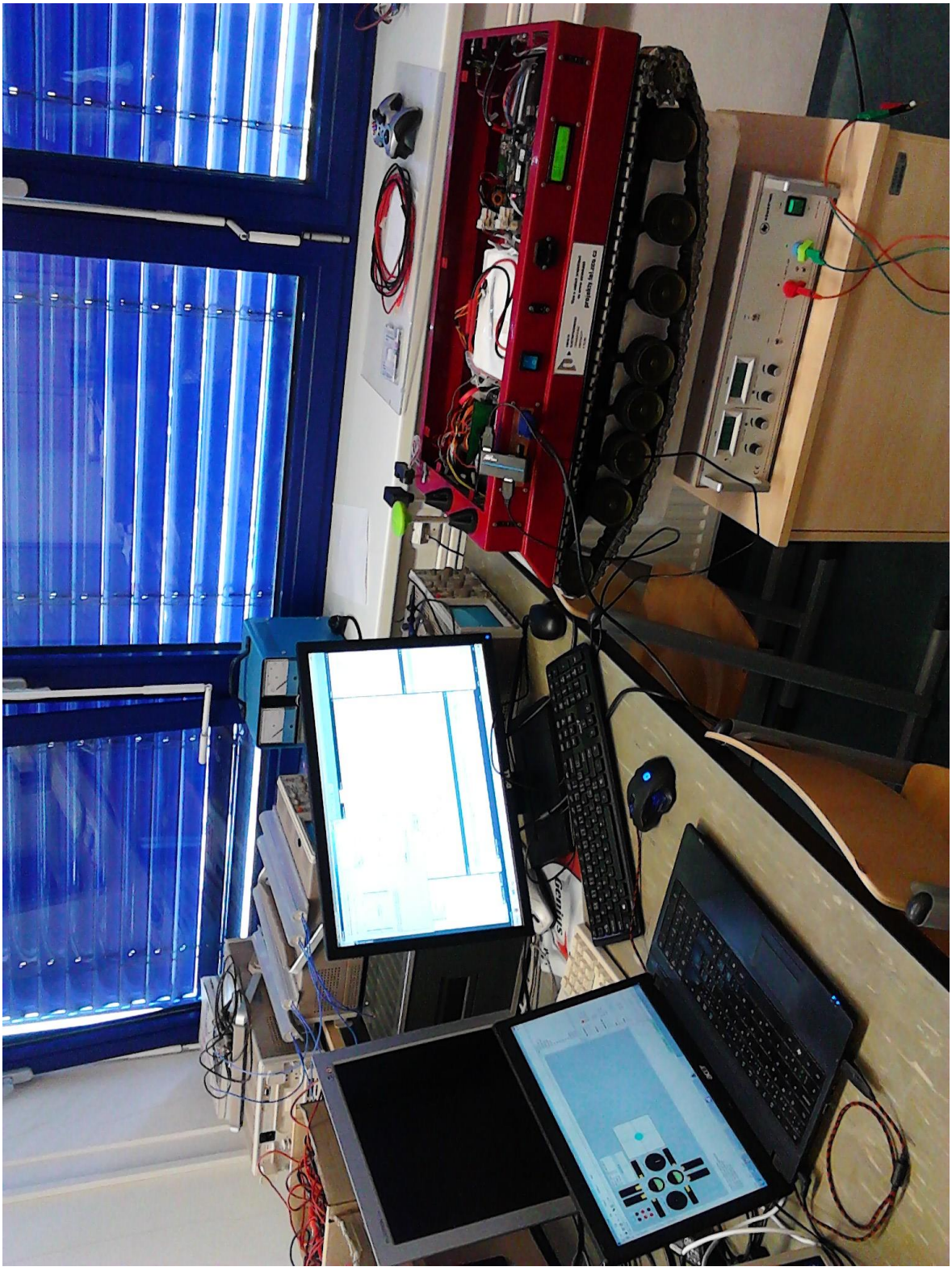
Tab. 3: Senzory

ID		Informace		ID		Informace	
120	0x78	*Vyhrazeno*	Senzory	160	0xA0	*Vyhrazeno*	Senzory
121	0x79	**Vlhkoměry**		161	0xA1	*Vyhrazeno*	
122	0x7A			162	0xA2	**Kompasy 3D**	
123	0x7B	*Vyhrazeno*		163	0xA3		
124	0x7C	**Kontakty**		164	0xA4		
125	0x7D	*Vyhrazeno*		165	0xA5		
126	0x7E	**Mikrofony**		166	0xA6	*Vyhrazeno*	
127	0x7F	*Vyhrazeno*		167	0xA7	*Vyhrazeno*	
128	0x80	**Radary**		168	0xA8	**Kompasy 2D**	
129	0x81			169	0xA9		
130	0x82			170	0xAA	*Vyhrazeno*	
131	0x83			171	0xAB	*Vyhrazeno*	
132	0x84	*Vyhrazeno*		172	0xAC	*Vyhrazeno*	
133	0x85	*Vyhrazeno*		173	0xAD	**Senzory pH**	
134	0x86	*Vyhrazeno*		174	0xAE	*Vyhrazeno*	
135	0x87	*Vyhrazeno*		175	0xAF	*Vyhrazeno*	
136	0x88	*Vyhrazeno*		176	0xB0	*Vyhrazeno*	
137	0x89	*Vyhrazeno*		177	0xB1	**Senzory plynů**	
138	0x8A	*Vyhrazeno*		178	0xB2		
139	0x8B	**Akcelerometry 3D**		179	0xB3		
140	0x8C			180	0xB4		
141	0x8D			181	0xB5		
142	0x8E			182	0xB6	*Vyhrazeno*	
143	0x8F			183	0xB7	*Vyhrazeno*	
144	0x90			184	0xB8	*Vyhrazeno*	
145	0x91	*Vyhrazeno*		185	0xB9	*Vyhrazeno*	
146	0x92	*Vyhrazeno*		186	0xBA	*Vyhrazeno*	
147	0x93	*Vyhrazeno*		187	0xBB	*Vyhrazeno*	
148	0x94	**Akcelerometry 2D**		188	0xBC	*Vyhrazeno*	
149	0x95			189	0xBD	*Vyhrazeno*	
150	0x96			190	0xBE	*Vyhrazeno*	
151	0x97	*Vyhrazeno*		191	0xBF	*Vyhrazeno*	
152	0x98	*Vyhrazeno*		192	0xC0	**GPS**	
153	0x99	*Vyhrazeno*		193	0xC1		
154	0x9A	*Vyhrazeno*		194	0xC2		
155	0x9B	**Gyroskopy**		195	0xC3		
156	0x9C			196	0xC4		
157	0x9D			197	0xC5		
158	0x9E			198	0xC6		
159	0x9F	*Vyhrazeno*		199	0xC7		

Tab. 4: Uživatelský prostor

ID		Informace	ID		Informace
200	0xC8	**Definováno uživatelem**	228	0xE4	**Definováno uživatelem**
201	0xC9		229	0xE5	
202	0xCA		230	0xE6	
203	0xCB		231	0xE7	
204	0xCC		232	0xE8	
205	0xCD		233	0xE9	
206	0xCE		234	0xEA	
207	0xCF		235	0xEB	
208	0xD0		236	0xEC	
209	0xD1		237	0xED	
210	0xD2		238	0xEE	
211	0xD3		239	0xEF	
212	0xD4		240	0xF0	
213	0xD5		241	0xF1	
214	0xD6		242	0xF2	
215	0xD7		243	0xF3	
216	0xD8		244	0xF4	
217	0xD9		245	0xF5	
218	0xDA		246	0xF6	
219	0xDB		247	0xF7	
220	0xDC		248	0xF8	
221	0xDD		249	0xF9	
222	0xDE		250	0xFA	
223	0xDF		251	0xFB	
224	0xE0		252	0xFC	
225	0xE1		253	0xFD	
226	0xE2		254	0xFE	
227	0xE3		255	0xFF	

Příloha B: Ukázka z vývoje a testování



Obr. B.1: Ukázka testování řízení tanku