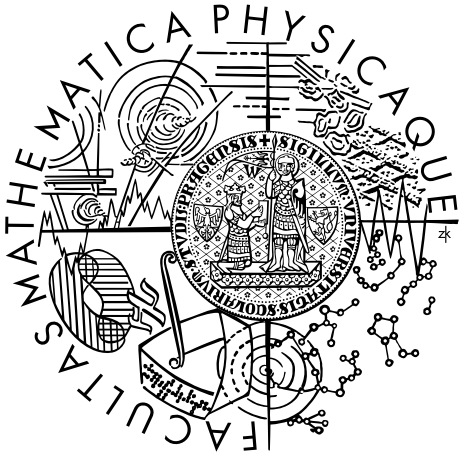
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Josef Janoušek

CPX Device Integrator

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2015

Děkuji RNDr. Tomáši Holanovi, Ph.D., za dlouhodobé odborné vedení práce i za čas věnovaný průběžným konzultacím, při kterých mi poskytl mnoho cenných připomínek a rad.

Dále bych chtěl poděkovat MUDr. Kryštofovi Slabému za návrh zadání práce, trpělivost i čas věnovaný průběžnému testování a návrhům na postupné zlepšování programu.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 17.5.2015 Podpis autora

Název práce: CPX Device Integrator

Autor: Josef Janoušek

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Tomáš Holan, Ph.D., Katedra softwaru a výuky informatiky

Abstrakt: Práce se zabývá možnostmi komunikace mezi programy zpracovávajícími naměřené veličiny a měřicími zařízeními sloužícími při vyšetřování pacientů v zátěžové laboratoři. Dosavadní postup spočívá v přímém propojení konkrétní dvojice zařízení – vyhodnocovací program, což znemožňuje paralelní připojení více zařízení a programů. Cílem tedy bylo vytvoření integračního nástroje, na který se mohou všechna zařízení napojit. Jeho úkolem je řídit a zpracovávat komunikaci mezi zařízeními při měření. Obsluha programu bude moci snadno nastavovat, jak má propojení mezi zařízeními vypadat, což umožní vytvářet různé vyšetřovací scénáře. Program bude připraven na integraci zařízení různých výrobců. Ve výsledku tak přispěje ke zjednodušení a zefektivnění vyšetřovacího postupu při diagnostice pacientů v zátěžové laboratoři.

Klíčová slova: zátěžové testování, bicyklový ergometr, EKG, integrační nástroj

Title: CPX Device Integrator

Author: Josef Janoušek

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Tomáš Holan, Ph.D., Department of Software and Computer Science Education

Abstract: The thesis studies possibilities of communication among programs (that process measured quantities) and measuring devices used for testing of patients in an exercise lab. The existing technique is based on direct interconnection of concrete pair of device and program, which makes parallel connection of more devices and programs impossible. So the goal was to create the integration tool which all devices can connect to. His job is to manage and process a communication among devices during the measurement. The program attendant can easily set the form of interconnection which allows creating various examination scenarios. The program will be prepared for integration of devices from various manufacturers. As a result it contributes to simplification and efficiency improvement of examination technique of diagnostics of patients in an exercise lab.

Keywords: exercise testing, ergometer, ECG, integration tool

Obsah

Úvod

Zátěžové testování je důležitou součástí sportovní medicíny. V oboru rehabilitace a tělovýchovného lékařství je jedním ze základních nástrojů užívaných k diagnostice stavu pacienta, na jejímž základě dochází k volbě dalších léčebných postupů a metod. Stejně jako v mnoha oborech lidské činnosti, i zde nabývá na důležitosti využití výpočetní techniky. Používané měřicí přístroje jsou tedy již napojeny na počítač, což přináší bohaté možnosti dalšího zpracování naměřených dat.

Jedním z předních pracovišť tohoto oboru je i zátěžová laboratoř na Klinice rehabilitace a tělovýchovného lékařství UK 2. LF a Fakultní nemocnice v Motole. V laboratoři se při vyšetřování pacientů používá mnoho druhů zařízení, např. bicyklový ergometr (tzv. kolo), pohyblivý pás či pulsní oxymetr. U těchto zařízení můžeme nastavovat požadované parametry (např. výkon, otáčky, rychlost, sklon) a zároveň měřit různé veličiny (např. tepová frekvence, systolický a diastolický tlak, saturace). Veličiny a parametry se dále zpracovávají a vyhodnocují ve specializovaných programech, např. Oxycon a EKG. Tyto programy na základě vyhodnocení mohou rovněž jednotlivá zařízení ovládat.

Podstatou problému je ovšem způsob komunikace zařízení s programy na PC. V dosavadním stavu se zařízení připojují přímo k vyhodnocovacímu programu (Oxycon) s tím, že komunikace je omezena vždy jen na konkrétní dvojici přístrojů (resp. dvojici zařízení – program). Připojení více zařízení najednou, aby mohly sdílet komunikaci (např. hodnoty veličin), není možné. Stejně tak připojení druhého programu (EKG) je obtížné a řeší se připojením k prvnímu programu (Oxycon) jako virtuální zařízení. Vlastní komunikace probíhá přes sériový port pomocí textových protokolů, které se u jednotlivých zařízení často velmi liší.

Cílem práce je tedy vytvořit integrační nástroj, přes který by veškerá komunikace probíhala. Tento program by ji dále řídil, tedy přijímal a odesílal jednotlivé příkazy připojeným zařízením a vyhodnocovacím programům. Dle zadavatele MUDr. Kryštofa Slabého by tedy aplikace měla sloužit jako určitá sběrnice či broker, který by transparentně emuloval peer-to-peer komunikaci s logickou topologií typu hvězda. Všechna zařízení tedy budou moci pracovat paralelně a být připojena k integračnímu programu. V něm se vytvoří pro konkrétní zařízení jakýsi virtuální obraz, který bude se zařízením průběžně komunikovat dle jeho komunikačního protokolu. Vůči existujícím vyhodnocovacím programům se bude program CPX DI tvářit jako virtuální zařízení.

V první kapitole podrobněji analyzujeme problematiku způsobu propojení měřicích přístrojů s počítačem. Zaměříme se především na podobu komunikace se zařízeními a možnosti jejího řízení. Dále představíme návrh řešení v podobě integračního programu, u kterého rozebereme strukturu, požadované funkcionality i další aspekty návrhu, které bylo potřeba při vývoji řešit. Také jej srovnáme s již existujícími řešeními v této oblasti.

Druhá kapitola přináší uživatelskou dokumentaci. Prezentuje především uživatelské rozhraní, které je řešeno formou aplikace pro systém Windows (WinForms).

Třetí kapitolu věnujeme programátorské dokumentaci, ve které popíšeme fungování jednotlivých objektů programu i komunikaci mezi nimi.

Ve čtvrté kapitole ukážeme možné postupy při využití jednotlivých programů.

1. Analýza

V této kapitole budeme analyzovat problémy, se kterými se setkáváme při využívání přístrojové techniky v oboru tělovýchovného lékařství. Podíváme se blíže na problematiku zátěžového testování, kde rozebereme měřicí zařízení a programy, které se při vyšetřování pacientů používají. Na základě analýzy současného stavu specifikujeme požadavky na program, který chceme v rámci tohoto projektu vytvořit.

V další části již popíšeme návrh nového řešení. Probereme celkovou strukturu programu i jeho jednotlivé části. Vysvětlíme jednotlivé problémy, kterým bylo třeba při tvorbě programu věnovat pozornost a zhodnotíme možné způsoby jejich řešení. Nakonec výsledek srovnáme s existujícími nástroji, které řeší podobné záležitosti.

1.1 Popis problematiky

Podstatnou součástí oboru fyziologie tělesné zátěže [1] je problematika zátěžového testování, které se tato práce věnuje. Představme si tedy, jak je v této oblasti používána výpočetní technika, konkrétně vyhodnocovací a ovládací programy, ve spojení s měřicími přístroji. Dále se podíváme, jakými způsoby spolu jednotlivá zařízení komunikují.

1.1.1 Zařízení v laboratoři

Při kardiopulmonální zátěžové funkční diagnostice jsou využívány různé druhy přístrojů, např. ergometry, EKG, metabolické analyzátory či pulsní oxymetry. Od jednoho druhu zařízení se používají různé modely, často pocházející od různých výrobců.

Obrázek 1.1: Různé varianty ergometru [2]

Bicyklový ergometr

Tento přístroj slouží při vyšetření k simulaci jízdy na kole. Žádanou obtížnost jízdy definujeme pomocí požadovaného výkonu, který zařízení přijímá. Následně umí odpovídat na dotazy, kterými lze zjišťovat aktuální výkon a aktuální otáčky kola. Kromě toho zná i ovládací příkazy jako Start, Start se současným měřením krevního tlaku (využije se připojitelný měřič) či Stop. Různé varianty přístroje vidíme na obrázku 1.1.

Pulsní oxymetr

Obrázek 1.2: Pulsní oxymetr v činnosti [3]

Jedná se o přenosné zařízení se senzorem, který je vyšetřované osobě nasazován na prst. Z něj pak dokáže měřit charakteristiky krve a jejího oběhu. Konkrétně jde o nasycení krve kyslíkem (tzv. saturaci, označení SpO2) a tepovou frekvenci. Naměřené hodnoty přístroj odesílá v pravidelných intervalech po celou dobu své aktivity. Žádné ovládací příkazy již nepřijímá. Činnost přístroje ukazuje obrázek 1.2.

Běžecký trenažér

Dalším zařízením je běžecký pás, jinak také nazývaný „běhátko“ (viz obrázek 1.3). Pro nastavení žádané obtížnosti běhu na pásu přijímá hodnoty požadované rychlosti a požadovaného sklonu. K těmto hodnotám se však při měření přibližuje postupně, proto na dotaz poskytuje hodnoty aktuální rychlosti a sklonu. Kromě toho má také celou řadu jiných ovládacích příkazů, kupř. start a stop pásu, nastavení jednotek (km/h nebo míle za hodinu), inkrementální změna rychlosti

Obrázek 1.3: Podoba běžeckého pásu [4]

či sklonu, reset, atd. Velký důraz je zde kladen na bezpečnost, proto zařízení vyžaduje od ovládacího programu pravidelné zadávání příkazů a má i možnost okamžitého nouzového zastavení.

1.1.2 Vyhodnocovací programy

Tyto specializované aplikace, které jsou nainstalované na počítači v zátěžové laboratoři, mají dvě základní funkce. Jednak vyhodnocují naměřená data, z čehož mohou např. generovat průběh elektrokardiogramu či počítat další charakteristické hodnoty potřebné pro správnou diagnostiku. Dále mohou na základě zjištěných údajů vysílat příkazy pro ovládání připojených zařízení, čímž řídí další průběh měření.

Oxycon

Oxycon je komplexním programem pro řízení průběhu měření na ergometru, s nímž využívá stejný protokol Ergoline. Umí přijímat hodnoty aktuálního výkonu, otáček kola, tepové frekvence a saturace. Na základě toho pak generuje hodnoty požadovaného výkonu, které odesílá zpět ergometru.

EKG

Jak již název napovídá, tento program slouží především pro tvorbu elektrokardiogramu. Opět využívá protokolu Ergoline, dle kterého přijímá hodnoty aktuálního výkonu a otáček. Vypočítat umí také požadovaný výkon a rovněž tepovou frekvenci.

1.1.3 Komunikace se zařízeními

Měřicí přístroje komunikují s počítačem přes sériový port RS232. Jedná se o online komunikaci, jejíž latence se může pohybovat až v řádu sekund. Probíhá prostřednictvím textového protokolu rychlostí řádově desítky až stovky B/s. Podoba textových příkazů, stejně jako požadavky na vlastnosti portu, jsou v protokolech přesně definovány. Vyhodnocovací programy v řídícím počítači jsou také napojeny na virtuální sériové porty.

1.2 Co je třeba zlepšit

Fyziologie tělesné zátěže, což je obor, kterého se naše práce týká, se neobejde bez přístrojové techniky včetně elektrokardiografických metod. Pracoviště Kliniky rehabilitace a tělovýchovného lékařství UK 2.LF a FN Motol, které vzneslo požadavek na vytvoření našeho programu, je už historicky zaměřené i na lékařskou informatiku a biostatistiku. Proto jeho pracovníci na základě dlouhodobých zkušeností a zjištěných potřeb při vyšetřování pacientů na měřicích přístrojích dospěli k tomu, že současný způsob připojování přístrojů není zcela vhodný a neumožňuje realizovat měření v potřebném rozsahu.

1.2.1 Jak probíhá měření

Jádro problému je v tom, že mezi jednotlivými přístroji jsou použita obecně taková rozhraní, u nichž známe specifikaci, ale nelze je nijak upravovat. Ve většině případů, ale zdaleka ne výlučně, se jedná o jednoduchý textový protokol, přičemž komunikační linku představuje sériový port RS232. Výsledkem je omezení zapojení na konkrétní dvojici přístrojů. Schéma se stávající situací vidíme na obrázku 1.4.

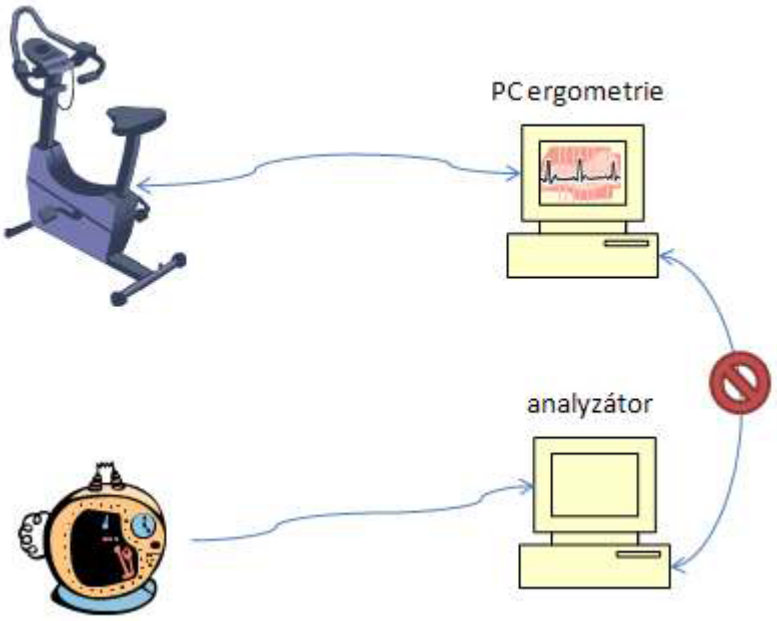
Jediná použitelná cesta, jak připojit nové (nepodporované) zařízení nebo signál posílat více zařízením naráz (to je důležitější), je vložit do komunikace prostředníka, který bude emulovat koncové zařízení a implementuje potřebné nové funkce.

MUDr. Kryštof Slabý, který je hlavním zadavatelem projektu za pracoviště Fakultní nemocnice v Motole, uvádí konkrétní příklad. Pacient při vyšetření „šlape“ na rotopedu, který je řízený počítačovým programem, jenž současně zajišťuje nahrávání EKG. Takže z PC do rotopedu běží požadavek na nastavení určité zátěže a zpět jde informace o aktuálně nastavené hodnotě. Chceme-li ale tutéž informaci (požadovanou nebo aktuální zátěž) dostat ještě do třetího zařízení, není to možné.

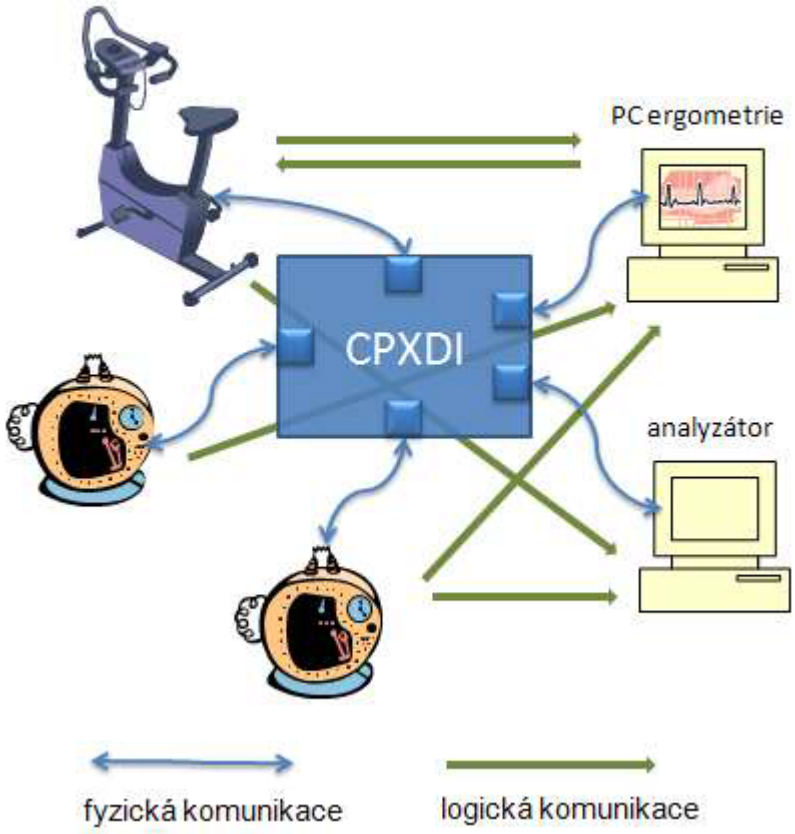
Zamýšlená aplikace by měla sloužit jako určitá sběrnice nebo spíš broker, transparentně emulující peer-to-peer komunikaci, přičemž celkově by se jednalo o topologii typu hvězda. Podobu cílového stavu s vyznačením fyzické i logické formy komunikace znázorňuje obrázek 1.5.

1.2.2 Požadavky na program

V rámci specifikace programu jsme stanovili soubor následujících funkčních požadavků:



Obrázek 1.4: Stávající situace zapojení přístrojů



Obrázek 1.5: Cílový stav propojení přístrojů

- konfigurace zařízení, která zahrnuje přidávání, mazání a editaci
  - název a popis zařízení
  - výběr komunikačního protokolu a jeho konfigurace (např. výběr portu)
  - výběr parametrů pro ovládání zařízení (např. výkon, otáčky, rychlost, sklon) a měření (např. saturace, puls, systolický a diastolický tlak)
- konfigurace parametrů pro měření a ovládání zařízení
  - zkratka, název a popis parametru
  - jednotky
- konfigurace scénářů (zahrnuje přidávání, mazání a editaci)
  - název a popis scénáře
  - výběr zařazených zařízení
  - u jednotlivých zařízení nastavení vstupních a výstupních parametrů
- zajištění průběhu měření
  - možnost řízení událostmi (start, stop)
  - manuální ovládání – přímo nastavovat hodnoty některých parametrů zařízení
- komunikace se zařízením
  - inicializace v závislosti na protokolu (např. otevření portu, navázání spojení, popř. zaslání inicializačního příkazu, apod.)
  - zpracovávat příkazy dle protokolu
  - přenášet přes sériový port

1.3 Návrh řešení

Dle požadavků zadavatele jsme tedy navrhli vytvoření nového programu, který bude sloužit jako integrační nástroj. Bude mít na starosti kompletní řízení a zpracovávání komunikace mezi připojenými zařízeními. Velký důraz je při návrhu kladen na modularitu a rozšiřitelnost, aby byl program dlouhodobě využitelný i v případech, kdy bude třeba zapojit nové zařízení od jiného výrobce či s jiným komunikačním protokolem.

1.3.1 Základní struktura objektů

Nejprve bylo třeba určit, jak bude vypadat struktura objektů a základní objekty v programu. Ty se v podstatě přímo odvíjejí od výše uvedených funkčních požadavků, ze kterých vyplývají i jejich požadované vlastnosti. Důležité bylo i rozmyšlení hierarchie, protože některé objekty se vzájemně potřebují. Na vrcholu se tedy ocitl objekt `Veličiny`. Během vývoje přibyl na jeho úroveň ještě

objekt `Události` (s podobnými vlastnostmi). S veličinami a událostmi dále pracuje `Protokol`. Jednotlivé protokoly pak jsou užívány `Zařízeními`. Nakonec jsou zařízení zařazována do `Scénářů`. Vlastnosti základních objektů musí být uživatelsky editovatelné, takže mají k sobě okna pro správu. Uživatelem nastavené objekty si musí program pamatovat, proto jsou přítomny jejich seznamy určené k ukládání do konfiguračního souboru.

Později, když byly rozšířeny možnosti komunikace (popsány níže v samostatné sekci), dostaly se `Protokoly` spolu s objekty `Komunikace` do samostatné skupiny servisních tříd. Ty už nejsou editovány uživatelsky, tedy ani není potřeba jejich instance ukládat. Nové potomky těchto tříd lze naopak snadno přidávat programátorsky (opět popsáno níže).

Výslednou podobu základní struktury objektů a jejich hierarchie ukazuje obrázek 3.1.

1.3.2 Monitoring měření

Jednou ze stěžejních funkcí hlavního okna programu má být monitorování naměřených hodnot od jednotlivých zařízení. Pro ukládání veličin a hodnot byl zvolen slovník (konkrétně `Dictionary<Veličina, string>`). Hodnota je tedy uchovávána jako řetězec, aby byla zajištěna obecná funkčnost pro všechny protokoly. Typicky jsou sice hodnotami čísla, nicméně jejich formát je v různých protokolech natolik rozdílný, že nebylo v možnostech obecného monitoringu jej kontrolovat. Někdy se ale může vyskytnout i nečíselná hodnota, takže jsme na to připraveni.

Vizuální prezentace pro uživatele je zajištěna pomocí prvku `DataGridView`, který je součástí prostředí WinForms. Při přijetí nové hodnoty je jeho zobrazení vždy obnoveno.

Přijaté události není potřeba ukládat, ale rovnou se rozesílají ostatním zařízením.

1.3.3 Ukládání hodnot

Příkazy přijaté od zařízení nejdříve zpracuje jejich protokol, který po rozpoznání dané veličiny či události předává její hodnotu na monitoring. Při reálném měření, když je připojeno více zařízení, můžeme dostávat ke zpracování ve velmi malých intervalech i mnoho hodnot najednou. Potřebovali jsme tedy zajistit frontu, do které by se hodnoty čekající na zpracování shromažďovaly. Byla tak využita fronta typu *producent – konzument*. Ta má více možných způsobů implementace. V nejnovější verzi jazyka C# a platformy .NET jsou dokonce některé struktury tohoto typu již vestavěny (např. `BlockingCollection<T>`). Nakonec jsme ale zůstali u starší verze a vlastní třídy `TaskQueue<T>`. Na jedné straně tedy zajišťuje producentskou metodu pro vložení úkolu, která je volána při příjmu hodnoty. Na druhé straně obsahuje konzumentské vlákno, které hodnoty z fronty vybírá a volá zpracovatelskou metodu pro uložení veličiny na monitoring, resp. rozeslání události.

Stejnou frontu (ale samozřejmě vždy vlastní instanci) používá i každé zařízení pro příjem příkazů od své komunikační komponenty. Z tohoto důvodu je tato třída generická, protože zařízení do ní ukládá pouze řetězcové hodnoty příkazů, zatímco

monitorovací okno ukládá trojice `Tuple<Velicina, string, Zarizeni>`, resp. `Tuple<Udalost, string, Zarizeni>`.

1.3.4 Vazby mezi zařízeními

Zajistit propojení mezi připojenými zařízeními, včetně možnosti nastavit, jak budou konkrétní vazby v rámci scénáře měření vypadat, bylo od začátku jedním z klíčových úkolů programu CPX DI. Přičemž původně bylo pod pojmem vazby myšleno přímé nastavení toho, odkud kam se budou hodnoty veličin posílat. Proto byl v první verzi programu u každého scénáře obsažen seznam nastavených vazeb. Vazba obsahovala tyto údaje: zdrojové zařízení, veličina a seznam cílových zařízení. Vlastní funkčnost pak zajišťovalo monitorovací okno tak, že v momentě, kdy přišla od zdrojového zařízení daná veličina, byla automaticky odeslána do všech cílových zařízení.

Při následném testování programu v laboratoři jsme však došli k tomu, že takto pojatý koncept vazeb je příliš specifický a neumožňuje práci se zařízeními požadovaným způsobem. Potřebovali jsme totiž spíše větší nezávislost jednotlivých zařízení. Bylo vyžadováno, aby program uměl emulovat komunikaci s každým zařízením nezávisle na ostatních. Přitom se musel chovat přesně dle pravidel, která dané zařízení vyžaduje (např. odpovídat na dotazy ohledně hodnot veličin). Původní koncept vazeb byl tedy opuštěn.

Emulace zařízení

Místo toho byl důraz kladen na zajištění toho, aby program prováděl komunikaci s každým zařízením přesně dle jeho pravidel. Konkrétně např. u vyhodnocovacího programu Oxycon bylo třeba na jeho dotazy posílat odpovědi s hodnotami veličin. Tím pádem jsme se museli starat o to, aby již byly naměřené hodnoty známé. Proto jsme k zařízení přidali časovač, pomocí kterého se (tam, kde je to potřeba) zařízení dotazujeme na jeho naměřené hodnoty. Z tohoto nového uspořádání pak vlastně vazby vyplynuly implicitně. Místo vazeb totiž stačí, aby uživatel ve scénáři nastavil vstupní a výstupní veličiny u každého zařízení. Na výstupní veličiny se pak pomocí časovače můžeme periodicky ptát a tím si udržovat jejich aktuální hodnotu. Pokud se nás naopak např. vyhodnocovací program na nějakou veličinu zeptá, lze mu hodnotu okamžitě poslat v odpovědi.

1.3.5 Řešení komunikace

Většina fyzických zařízení v laboratoři komunikuje přes sériový port. Stejně umí komunikovat i vyhodnocovací programy. Proto byl v první verzi programu implementován právě tento způsob. Dalším cílem bylo samozřejmě připravit program na další způsoby komunikace, jejichž implementace mohou být do programu přidány i v budoucnu. Museli jsme na ně tedy stanovit přesné požadavky, aby mohly být vhodně a správně doprogramovány.

Sériový port

Podpora pro práci se sériovým portem je poskytována přímo v jazyce C# na platformě .NET. Zajišťuje ji třída `SerialPort`. Pro odesílání příkazů využíváme

její metodu `WriteLine`, která sama doplňuje nastavený koncový znak, jenž je nastaven ve vlastnostech portu. Mezi další vlastnosti patří BaudRate, DataBits, StopBits, Parity a Handshake. Jejich nastavení má možnost provést protokol při inicializaci.

Na přijímací straně nás zajímá událost portu `DataReceived`, která je automaticky vyvolána, když vnější zařízení zapíše nějaký příkaz na port. My této události přiřazujeme EventHandler, v jehož obsluze data čteme metodou `ReadLine()` a posíláme k dalšímu zpracování.

Při inicializaci je potřeba port se zadaným názvem otevřít a nastavit jeho vlastnosti. V deinicializaci port zavřeme a dáme k dispozici jeho systémové prostředky.

TCP klient

V souvislosti s požadavkem na vytvoření samostatných programů pro manuální ovládání a grafickou vizualizaci bylo třeba také vyřešit, jak budou tyto programy komunikovat s hlavním integrátorem. Jistě by i mezi programy bylo možné komunikovat přes sériové porty, nicméně to by nebylo příliš efektivní z hlediska spotřeby zdrojů, tedy fyzických portů řídícího počítače. Zde by sice stačilo použít virtuální porty, ale pro jejich emulaci by zase byl potřeba další program. Navíc jsme měli další požadavek, aby komunikující programy mohly eventuálně pracovat na různých počítačích, dokonce i na různých místech. V takovém případě už kabelové spojení vůbec nepřipadalo v úvahu, proto jsme se rozhodli použít síťové spojení.

Co se týče síťového spojení, jazyk C# na platformě .NET poskytuje podporu pro oba hlavní užívané protokoly, tedy Transmission Control Protocol (TCP) i User Datagram Protocol (UDP). Tyto protokoly se zásadně liší. UDP je nespolehlivý (tedy negarantuje doručení) a nespojovaný, tedy nepotřebuje inicializaci a obě dvě strany komunikace jsou ve stejném postavení, takže i implementace je o něco jednodušší. Oproti tomu je TCP spolehlivý a spojovaný, vyžaduje tedy inicializaci a rozlišení stran, kdy jedna vystupuje jako server, druhá pak představuje klienta.

Pro naše programy jsme nakonec zvolili TCP. Upřednostnili jsme tedy spolehlivost i přes možnou nižší rychlost (v případě chyby se paket posílá znovu). Navíc model klient – server přesně odpovídá našemu uspořádání integrátor – zařízení. Nejdříve totiž musí být zapnuté zařízení (server), pak se k němu připojí integrátor (klient). V hlavním CPX DI byla tedy implementována komunikace z pohledu TCP klienta.

Na straně odesílání pracujeme se síťovým proudem, který poskytuje pouze metodu `Write`, která navíc přijímá jen pole bytů. Sami tedy musíme přidat koncový znak, který máme nastavený ve vlastnostech komunikace. Pak z odesílaného řetězce získáme bytové pole pomocí zadaného kódování. Další vlastností je velikost přijímacího i odesílacího bufferu, kterým je určena maximální délka příkazu. Opět je může nastavit protokol při inicializaci.

Pro přijímání využíváme naslouchací vlákno, v jehož obsluze čteme ze streamu metodou `ReadByte()` (umí tzv. blokovací čekání, právě proto musí být v samostatném vlákně), sami naplňujeme přijímací buffer a po nalezení koncového znaku posíláme získaný text k dalšímu zpracování. Musíme ještě hlídat přetečení přijímacího bufferu.

V inicializaci je třeba založit TCP klienta se zadanou IP adresou a číslem portu, získat z něj síťový proud a nastavit potřebné vlastnosti. Při deinicializaci se stream i klient uzavřou.

Univerzalizace

Důležitým cílem, který jsme při vývoji programu sledovali, bylo zachování modularity a rozšiřitelnosti. To se zásadně projevilo i při návrhu komunikace programu. Musí totiž být připraven, aby v budoucnu bylo možné snadné přidání dalších způsobů, a to pokud možno bez zásahů do zbytku programu.

K tomu se velice hodí návrhový vzor *Factory Method pattern* [5], který přesně tuto situaci řeší. Jde totiž o způsob vytváření objektů, který dovoluje se rozhodnout, od jaké konkrétní podtřídy chceme vytvořit instanci. Je možno mít mnoho různých podtříd implementujících dané rozhraní. Tzv. Factory metoda pak vrátí instanci příslušné třídy na základě informací dodaných klientem nebo zjištěných z aktuálního stavu.

V programu tedy máme abstraktní třídu `Komunikace`, která plní roli rozhraní definujícího produkt. Konkrétně definuje metody pro inicializaci (na základě zadaných vlastností) a deinicializaci komunikační komponenty, metody pro nastartování a ukončení procesu čtení dat a metodu pro zápis textu do výstupního bufferu. Naopak metoda pro zpracování přijatých dat je definována jednotně ve třídě `Zařízení`. Komunikační komponenta ale neví, ke kterému zařízení je připojena. Proto ještě definuje delegáta na metodu ke zpracování dat. Zařízení pak pomocí tohoto delegáta dá referenci na svou přijímací metodu Tvůrci komunikace a ten ji dále předá v konstruktoru příslušné komunikační třídě. Ta už ji pak může volat vždy při přečtení dat. Od abstraktní třídy `Komunikace` se dále dědí podtřídy implementující konkrétní způsoby komunikace.

Roli Creatoru pak plní třída `TvurceKomunikace` se svou kreační metodou `VratKomunikaci`, která vrací na základě parametru obsahujícího název portu nebo IP adresu a číslo portu instanci příslušné komunikační třídy. Navíc umí ještě vracet slovník všech dostupných komunikačních tříd.

Podstatné je, že zařízení pracuje vždy s obecnou třídou Komunikace a neví, jakou konkrétní podtřídu představuje instance, kterou obdrželo od tvůrce. Proto při programování nového způsobu komunikace stačí dodržet pouze tyto kroky:

- implementovat novou třídu odvozenou od třídy Komunikace
- upravit třídu TvurceKomunikace takto:
  - přidat položku (dvojici název a nápověda pro tvar komunikačního parametru) do slovníku komunikačních komponent
  - upravit metodu VratSlovnik pro vracení správného indexu pro naši třídu dle parametru
  - upravit metodu VratKomunikaci pro vracení nové instance naší třídy dle parametru
- případně ještě do třídy VlastnostiKomunikace přidat položky vyžadované naší třídou

Žádnou další třídu ve zbytku programu pak již není nutné měnit.

1.3.6 Protokoly

Komunikační protokol především u každého zařízení určuje, jak vypadají příkazy, které umí zařízení přijímat a odesílat. Dále definuje požadované vlastnosti pro komunikační komponentu, případně ostatní potřebná nastavení konkrétního zařízení. Protokol je určen výrobcem zařízení, což znamená, že zařízení od stejného výrobce většinou užívají tentýž protokol. Ale i tak se mohou objevit rozdíly mezi verzemi či v podporovaných funkcích. Abychom mohli komunikaci se zařízením správně emulovat, musíme korektně implementovat veškeré náležitosti daného protokolu.

Podrobnější popis jednotlivých protokolů, které již bylo třeba do programu implementovat, uvádíme včetně přehledu jejich příkazů a požadavků na vlastnosti komunikační komponenty v příloze A – Implementované protokoly. Konkrétně se jedná o `Nonin` (pro pulzní oxymetr), `ManualControl` (slouží v aplikaci manuálního ovládání), `ChartPresent` (ke grafické vizualizaci naměřených hodnot), `Ergoline` (pro bicyklový ergometr a vyhodnocovací programy) a `T2000` (využívá běžecký pás).

Dvojí typ chování

Při návrhu práce s protokoly bylo třeba ještě zohlednit dvojí úlohu našeho programu ohledně chování k zařízením. Zadavatel projektu MUDr. Kryštof Slabý uvádí příklad pulzního oxymetru, který posílá každou vteřinu jeden textový řádek. Když bude CPX DI v komunikaci se zařízením emulovat přijímající stranu (počítačový program), tak bude jen poslouchat a jakmile přijde koncový znak, vezme celý buffer, pokusí se z něj získat hodnoty tepové frekvence a saturace a vrátí se zpět k poslouchání. Když ale bude CPX DI emulovat vysílající stranu, tak naopak bude periodicky posílat po vteřinách aktuální hodnoty těchto dvou veličin. Lze se na to dívat tak, že pro tyto dvě situace implementujeme dva odlišné protokoly. Druhou možností je vnímat obě jako jeden protokol a pak se musí zohlednit, kterou stranu zrovna integrátor emuluje – buď „master“ (PC), nebo „slave“ (device).

Nakonec se při implementaci ukázala jako schůdnější první varianta s oddělenými protokoly, protože chování v těchto dvou situacích bylo skutečně velmi odlišné a nenašlo se nic, kvůli čemu by bylo výhodné je spojovat do jednoho protokolu. Pro komunikaci s fyzickými zařízeními, pro která emulujeme počítačový program, tedy používáme protokol typu Slave. Naopak při komunikaci s vyhodnocovacími programy, kterým emulujeme měřicí zařízení, nastavujeme protokol typu Master.

Univerzalizace

Podobně jako u Komunikace jsme i třídu `Protokol` postavili jako abstraktní a implementace konkrétních protokolů jako její potomky. Původně jsme však protokoly řadili do základní struktury tříd, takže jsme jejich instance nastavené u jednotlivých zařízení přímo ukládali do konfigurace.

S přidáním samostatné třídy Komunikace jsme ale i u Protokolu přešli na uspořádání dle návrhového vzoru *Factory Method pattern* a tyto třídy označujeme

jako „servisní“. V konfiguraci je u zařízení tedy uložen jen název protokolu a jeho instance se vytvoří až při konstrukci zařízení.

Abstraktní třída `Protokol` má tedy opět roli rozhraní definujícího produkt. Po potomcích požaduje metody pro výpis jména, nastavení vlastností komunikace a zařízení, obsluhu zařízení (provedení periodických úkonů) a událostí, zpracování přijatých příkazů a definici slovníku ovládacích příkazů.

Úlohu Creatoru v tomto případě zajišťuje třída `TvurceProtokolu` se svou tvůrčí metodou `VratProtokol`, která vrací dle zadaného názvu protokolu jeho konkrétní instanci. Navíc umí poskytovat ještě seznam názvů všech dostupných protokolů.

Zařízení opět pracuje jen s obecnou třídou Protokol a nezná konkrétní podtřídu, kterou představuje instance přijatá od tvůrce. Takže i při programování nového protokolu stačí dodržet tyto kroky:

- implementovat novou třídu odvozenou od třídy Protokol
- upravit třídu TvurceProtokolu takto:
  - přidat položku (název) do slovníku názvů protokolů
  - upravit metodu VratSeznam pro vracení správného indexu pro naši třídu dle parametru
  - upravit metodu VratProtokol pro vracení nové instance naší třídy dle parametru

Žádné další třídy ve zbytku programu už není třeba měnit.

1.3.7 Způsob ukládání dat

Při návrhu programu zaujímalo důležitou část rozhodnutí, jakým způsobem budeme ukládat konfigurační data. Je potřeba mít uloženy informace o vytvořených objektech od všech čtyř tříd základní struktury, které jsou zároveň uživatelsky editovatelné, a to v rozsahu prakticky všech jejich datových položek. Vnitřně ukládáme vzniklé objekty do příslušných seznamů, konkrétně se jedná o List<Veličina>, List<Událost>, List<Zařízení> a List<Scénář>.

Vzhledem k tomu se pro externí ukládání nabízelo jako vhodné řešení využít koncept serializace. To je mechanismus, pomocí něhož mohou být objekty reprezentovány v textové nebo binární formě. Dokáže tak vzít objekt v paměti či graf objektů (tedy množinu objektů, které jsou vzájemně referencovány) a převést je do bytového proudu či XML uzlů, což může být dále uloženo či přenášeno. Reverzně k tomu funguje deserializace, která vezme proud dat a rekonstruuje z nich objekt v paměti či graf objektů. Platforma .NET podporuje několik serializačních mechanismů [6].

Pro náš program jsme preferovali serializaci do XML souboru, aby bylo možno konfigurační soubor případně upravovat i ručně. Proto jsme jako první zvažovali použití XMLSerializeru. Ten nabízel poměrně jednoduché rozhraní a také celkem dobře čitelný výstup. Bohužel jeho zásadní nevýhodou bylo, že nezachovává reference mezi objekty. Protože v naší struktuře objektů jsou reference velmi často používané, nebylo možno XMLSerializer využít.

V první verzi programu jsme tedy pracovali s Binary Serializerem, který už zachování referencí mezi uloženými objekty umožňoval. Serializované třídy bylo potřeba označit atributem [Serializable]. Na druhou stranu, binární výstup samozřejmě nebyl uživatelsky čitelný, takže tuto variantu jsme vnímali jen jako dočasnou.

DataContractSerializer

Posléze jsme tedy již našli vyhovující mechanismus, kterým je DataContractSerializer. U něj už je možno zvolit, aby zachovával reference, navíc umí produkovat poměrně dobře čitelné a standardům vyhovující XML. Serializovanou třídu je třeba označit atributem [DataContract]. Datové položky, které chceme zahrnout, pak musíme označit atributem [DataMember], k němuž jsme rovnou přidali příznak (Order=*číslo*) pro dosažení požadovaného pořadí položek v souboru (jinak jsou řazeny abecedně). Dalšími příznaky se dá nastavit např. speciální jméno pro každou položku, my jsme ale ponechali výchozí názvy (stejné jaké jsou zapsány v programu). Pouze jména označující položky slovníků ve scénáři jsme pro lepší čitelnost přejmenovali, kvůli čemuž jsme museli vytvořit nové třídy odvozené od daných slovníků a označit je atributem [CollectionDataContract] s příznaky ItemName, KeyName a ValueName.

Čtyři základní seznamy pak pro serializaci obalujeme do třídy ObjektKSerializaci. Ta představuje výchozí typ, který musíme serializeru předat v konstruktoru, spolu s parametry určujícími např. maximální počet objektů či právě volbu zachovávání referencí. Při samotné serializaci pak využíváme XMLWriter, kterému nastavujeme příznak pro indentaci při zápisu. Ten vytvoří XML soubor, do kterého následně zapíše text vytvořený serializerem.

Deserializace potom probíhá analogicky. Za pomoci Streamu si otevřeme konfigurační soubor, z něhož serializer načte výchozí ObjektKSerializaci obsahující čtyři základní seznamy, které už si zapíšeme do paměti. Pokud by během deserializace došlo k chybě, naplníme seznamy připravenou metodou dle Modelového zapojení.

Ještě je třeba poznamenat, že při deserializaci neprobíhá vytváření objektů konstruktorem. To znamená, že deserializace zajistí pouze naplnění serializovaných datových položek. Pokud chceme v rámci konstrukce objektu provádět ještě další činnosti, musíme je vyvolat jiným způsobem. K tomu slouží atributy [OnDeserialized] pro označení metody, která se má provést po deserializaci, příp. [OnDeserializing] pro metodu volanou před deserializací (analogické atributy existují i pro serializaci). V našem případě se to týká pouze třídy Zařízení, kde potřebujeme po naplnění jeho serializovaných položek ještě vytvořit instance jeho protokolu, komunikace, časovače a okna, takže jsme metodu zajišťující tyto činnosti označili právě atributem [OnDeserialized].

1.4 Srovnání s existujícími řešeními

Jak uvádí zadavatel projektu MUDr. Kryštof Slabý, aplikace fungující stejným způsobem jako naše řešení doposud k dispozici není. Ovládací programy, které k jednotlivým zařízením existují, vždy řeší komunikaci pouze na přímo v uspořádání jeden na jednoho (jedno zařízení s jedním programem). Tedy není možné

připojit více příjemců jedné informace nebo smíchat více informací od různých producentů a servírovat je jednomu příjemci jedním kanálem (CPX DI oba požadavky úspěšně zvládne).

Na druhou stranu, výrobci jednotlivých ovládacích programů k zařízením jsou dál v konfigurovatelnosti. Stačí jim konfigurační program nebo soubor a zvládnou tím definovat většinu nových zařízení. Zato CPX DI v současné době potřebuje každý nový typ komunikačního protokolu implementovat v kódu. Takže i toto může být jedním z možných námětů na budoucí rozšiřování a vylepšování programu.

2. Uživatelská dokumentace

Projekt se skládá ze tří základních a několika pomocných aplikací:

- 2.1 Integrační program CPX DI – hlavní nástroj
- 2.2 Manuální ovládání – samostatná aplikace
- 2.3 Grafická vizualizace naměřených dat – doplňkový program
- 2.4 Simulační programy – pomocné testování chování různých zařízení

2.1 Integrační program CPX DI

2.1.1 Základní pojmy

Program se používá v zátěžové laboratoři při vyšetřování pacientů pomocí různých přístrojů, např. Kolo, Běhátko či Pulsní oxymetr. Tyto přístroje jsou připojeny k počítači nejčastěji pomocí sériového portu. Naměřená data pak zpracovávají různé vyhodnocovací programy, např. Oxycon. I ty jsou s naším programem propojeny a program se na ně dívá stejně jako na fyzické přístroje. Proto připojené přístroje i programy společně označujeme pojmem `Zařízení`. Program CPX DI pak především sleduje a řídí komunikaci, která mezi nimi probíhá.

Zařízení dokážou měřit hodnoty různých sledovaných parametrů, kterým souhrnně říkáme `Veličiny`. Příkladem je třeba výkon, rychlost nebo saturace. Dále dokážou generovat příkazy pro různé (předem definované) akce, kterým říkáme `Události`.

Když chceme provádět měření, musíme především určit, jaká zařízení se budou měření účastnit. Následně také to, jaké veličiny a události budeme od těchto zařízení přijímat a jaké jim budeme naopak odesílat. Tomuto nastavení souhrnně říkáme `Scénář` měření.

Tomu, jakým způsobem jsou zařízení k programu připojena, říkáme souhrnně `Komunikace` (např. přes sériový port nebo přes TCP spojení). To, jakým způsobem se zařízení ovládá a jak vypadají data, která od něj přijímáme a která mu odesíláme, označujeme souhrnně jako `Protokol` (např. `Ergoline_Slave`). Fyzické přístroje většinou využívají protokol typu *Slave* (tím je ovládáme), vyhodnocovací programy naopak typ *Master* (tím ovládají náš program, potažmo jiná zařízení). Komunikaci i protokol nastavujeme pro každé zařízení v jeho konfiguraci.

Tabulku aktuálně naměřených hodnot veličin nazýváme `Monitoring`. Hodnoty veličin můžeme do monitoringu posílat i manuálně přímo z hlavního okna, případně přes samostatnou aplikaci Manuální ovládání (viz kapitola 2.2).

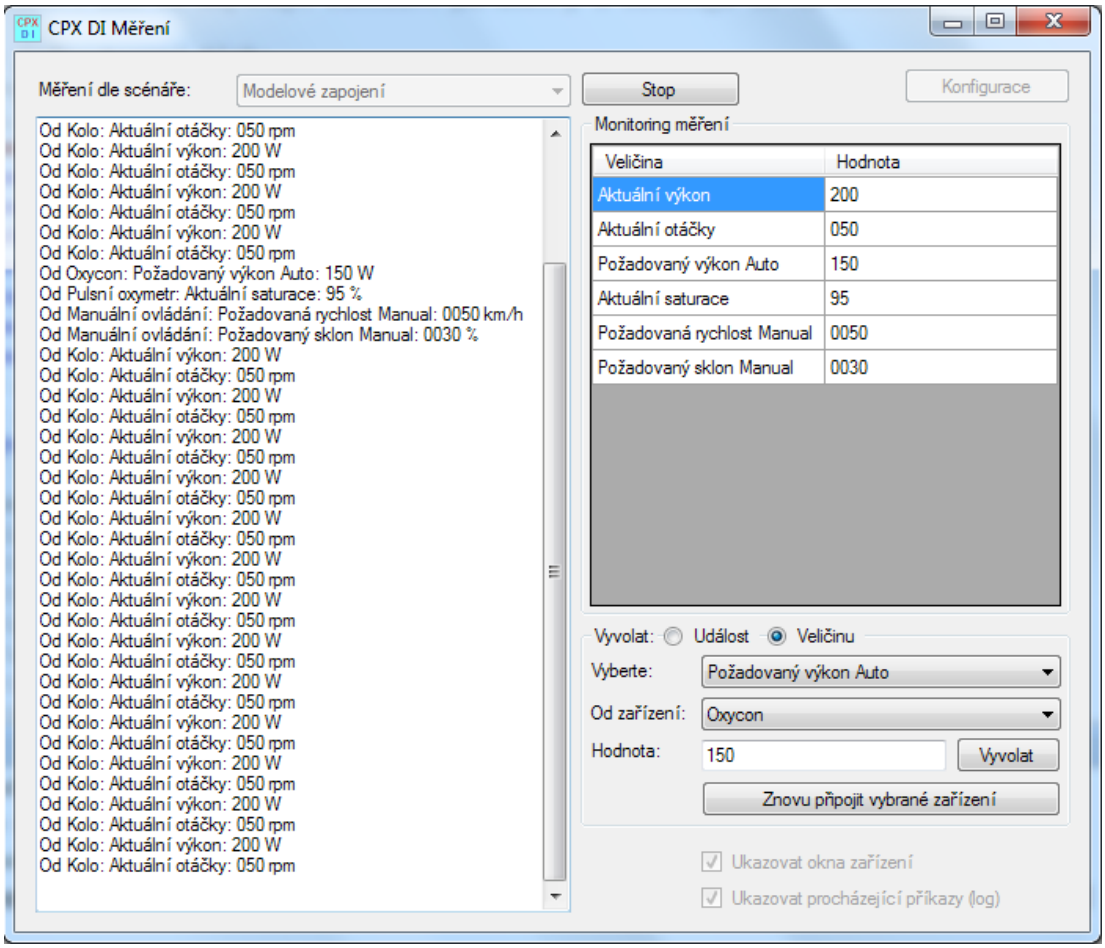
2.1.2 Spuštění programu

Instalátor programu `CPX Device Integrator` najdeme na přiloženém CD ve složce projektu `CPXDI_model`. Po nainstalování můžeme program spustit z nabídky Start. V systémové složce programu je umístěn i konfigurační soubor nazvaný souborSeznamy.xml.

Další možností je spuštění programu z příkazového řádku, v tom případě mu můžeme předat ještě některé další parametry:

- Pokud požadujeme načtení jiného konfiguračního souboru, než je výchozí souborSeznamy.xml, použijeme parametr `-c`, za kterým následuje jméno požadovaného konfiguračního souboru ve formátu XML.
- Následujícím parametrem je `-o`, za kterým následuje `Id` scénáře, který si přejeme předvybrat v hlavním okně z rozbalovacího seznamu scénářů.
- Také je k dispozici parametr `-s`, který způsobí automatický start měření dle scénáře vybraného ve výše uvedeném rozbalovacím seznamu.
- Pokud jsme parametr `-s` využili a měření automaticky spustili, můžeme ještě automaticky vyvolat nějakou událost pomocí zadaného parametru `-e`, za kterým následuje `Id` události, kterou chceme provést.

2.1.3 Okno měření



Obrázek 2.1: Okno měření

Okno měření představuje vstupní bod do programu a stará se především o zobrazování informací vztahujících se k vlastnímu průběhu vyšetřovacího měření. Vidíme jej na obrázku 2.1. Nahoře máme možnost vybrat scénář, podle kterého

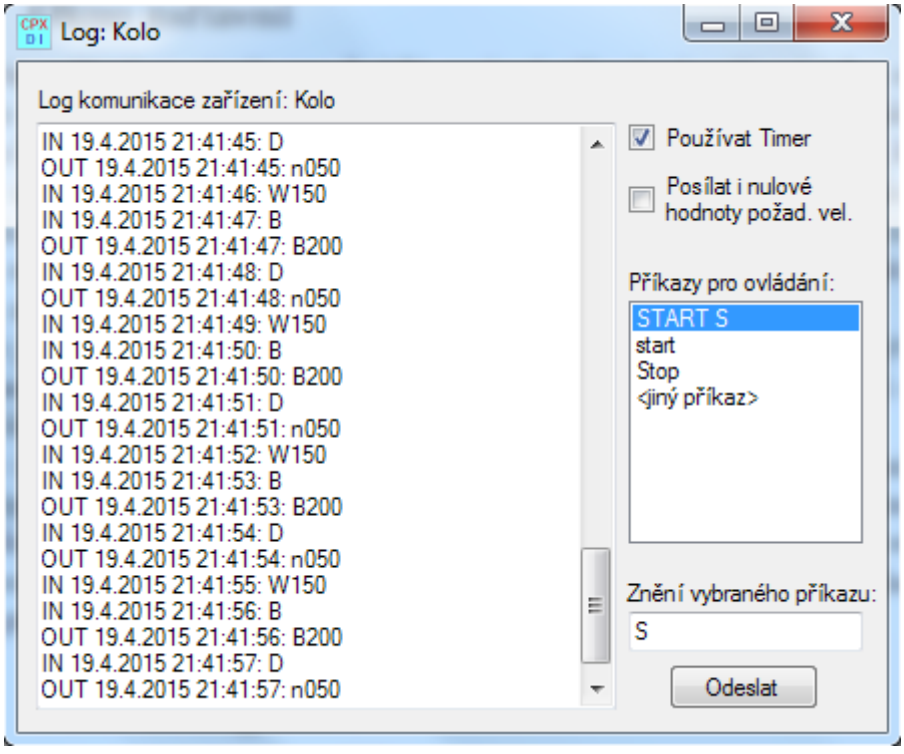
chceme měřit, spolu s ovládacím tlačítkem pro zahájení, resp. ukončení měření. Vedlejším tlačítkem otevřeme okno konfigurace (kapitola 2.1.5).

Dominantním prvkem okna je pole určené pro tzv. log. V něm se zobrazují všechny přicházející příkazy, které měřicí okno zpracovává. Patří sem tedy především informace o veličinách, které se zapisují na monitoring. Přitom se ale mohou vyskytnout různé nepovolené stavy, např. když dané zařízení nemá danou veličinu v aktuálním scénáři nastavenou jako výstupní, tak ji od něj nemůžeme akceptovat. O všech takových stavech je uživatel informován v tomto boxu, což mu umožňuje udržovat přehled o průběhu měření a chování připojených zařízení. Zobrazování logovacích informací je ovšem možno vypnout odškrtnutím vedlejšího zaškrtávacího políčka. Dalším zaškrtávátkem zase určíme, jestli se mají zobrazit ovládací okna zařízení.

Ještě důležitější je patrně oblast nazvaná Monitoring měření. Ve formě tabulky jsou zde zobrazovány všechny veličiny a jejich hodnoty, které právě sledujeme. Tabulka je průběžně aktualizována tak, jak dostáváme naměřené hodnoty od jednotlivých zařízení.

Pod tabulkou máme ještě možnost manuálně vyvolat zápis nějaké veličiny či události na monitoring. Abychom dodrželi nastavená pravidla pro příjem na monitoring, musíme ještě vybrat, od kterého zařízení předstíraná hodnota mohla přijít. Aby byla veličina či událost akceptována, musí být od zařízení, které ji má v aktuálním scénáři zařazenou mezi výstupními.

Níže ještě můžeme tlačítkem znovu připojit vybrané zařízení. To se hodí, když se zařízení z nějakého důvodu odpojí, protože takto jej lze připojit bez přerušení běhu scénáře. Stejně tak můžeme připojit i zařízení, které na začátku scénáře ještě neběželo. Takhle jdou připojit samozřejmě jen ta zařízení, která jsou zařazena v aktuálním scénáři.



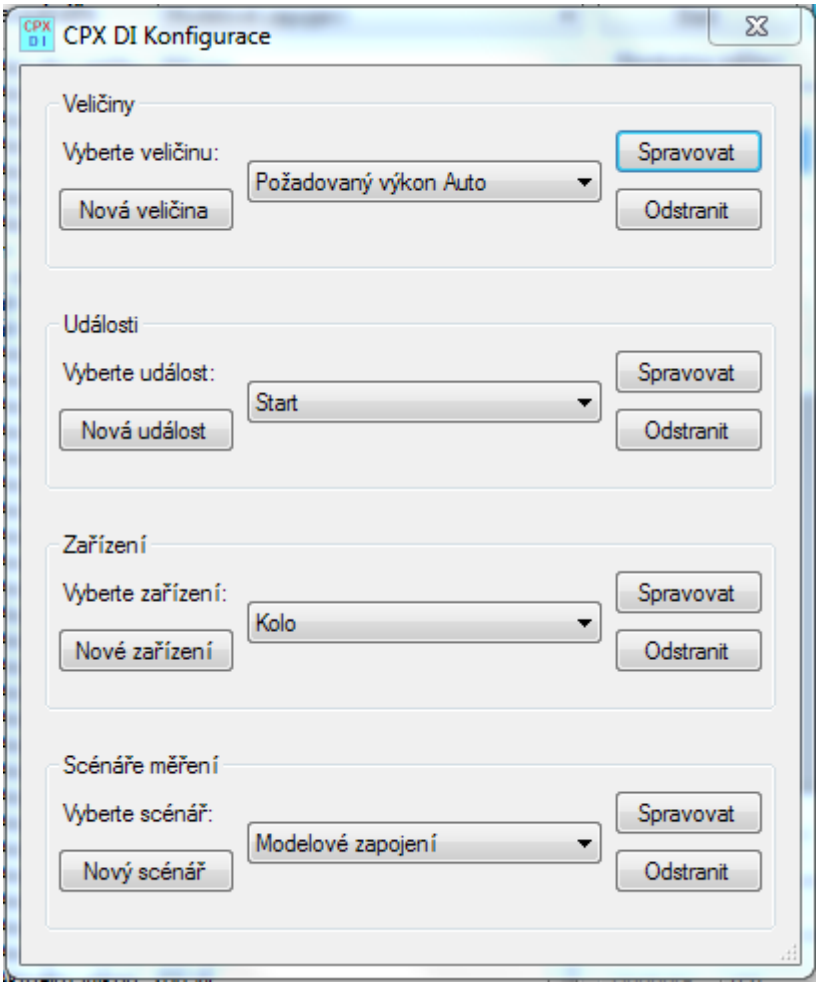
Obrázek 2.2: Okno zařízení

2.1.4 Okno zařízení

Po startu měření se otevře pro každé správně připojené zařízení jeho ovládací okno (pokud si předem zaškrtneme příslušné zaškrtávací políčko v hlavním okně měření). Je vidět na obrázku 2.2. Dominantní prvek zde tvoří pole pro sledování průběhu komunikace se zařízením. Zobrazují se zde tedy všechny příkazy, které do zařízení odesíláme a které z něj přijímáme, a to včetně přesného časového údaje, kdy příkaz proběhl. Rovněž zde vidíme zprávy o chybách, ke kterým došlo při příjmu, odesílání či rozpoznávání příkazů. Můžeme si tak průběžně kontrolovat, jestli se zařízení chová očekávaným způsobem.

Dále jsou zde zaškrtávací políčka, kterými můžeme řídit, jestli má být v provozu časovač (např. pro periodické posílání dotazů do zařízení) a jestli se mají posílat do zařízení i nulové hodnoty veličin.

Níže máme k dispozici seznam ovládacích příkazů pro dané zařízení, které připravil jeho protokol. Po vybrání příkazu dle jeho popisu ze seznamu se v dolním poli zobrazí jeho přesné znění. Tím je příkaz připraven k odeslání, které provedeme daným tlačítkem. Takto lze tedy do zařízení posílat požadované povely, případně i libovolné jiné příkazy dle momentálních potřeb.

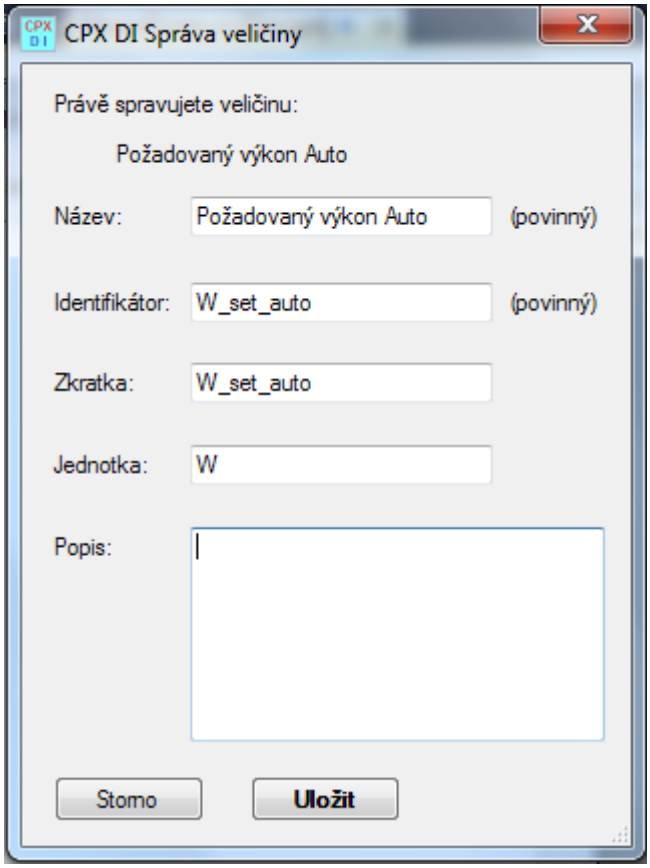


Obrázek 2.3: Okno konfigurace

2.1.5 Okno konfigurace

Konfigurační okno (na obrázku 2.3), otevírané příslušným tlačítkem z hlavního okna měření, umožňuje spravovat všechny základní objekty, se kterými v programu pracujeme, což jsou veličiny, události, zařízení a scénáře. Lze přidávat nové i upravovat a odstraňovat existující položky. Mazání stávajících položek probíhá přímo v konfiguračním okně, kde vybereme požadovanou položku z příslušného seznamu a použijeme tlačítko `Odstranit`. Při odstraňování nás program upozorní na kolize, aby nedošlo ke smazání něčeho, co je ještě jinde použito (např. pokud je veličina nastavena u nějakého zařízení, nebo zařízení v nějakém scénáři). Pro správu existujících objektů a vytváření nových se již pomocí odpovídajících tlačítek otevírají samostatná okna, která popisujeme níže.

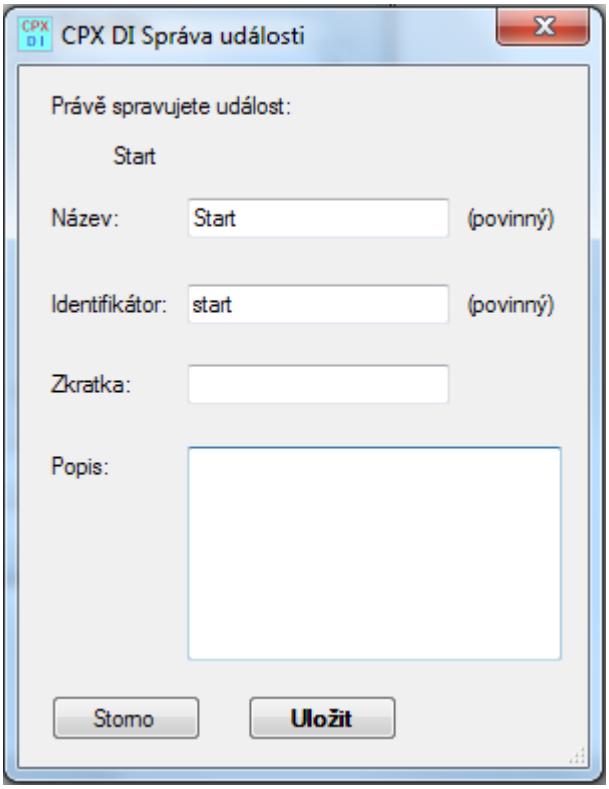
2.1.6 Správa veličin



Obrázek 2.4: Správa veličiny

Při správě veličiny máme možnost nastavit její základní údaje, kterými jsou název, identifikátor, zkratka, jednotka a popis (viz obrázek 2.4). Název a identifikátor musíme vyplnit, jinak si program doplní výchozí hodnoty automaticky. U těchto údajů se navíc kontroluje unikátnost, v případě shody s údajem u jiné veličiny program doplní rozlišovací znak. Identifikátor je důležitý pro správné rozpoznávání příkazů od zařízení, proto je potřeba při případných změnách postupovat opatrně. Tlačítkem `Uložit` údaje uložíme, tlačítkem `Storno` opustíme okno bez uložení. Při zavírání okna se program zeptá, jestli chceme údaje uložit.

2.1.7 Správa událostí



Obrázek 2.5: Správa události

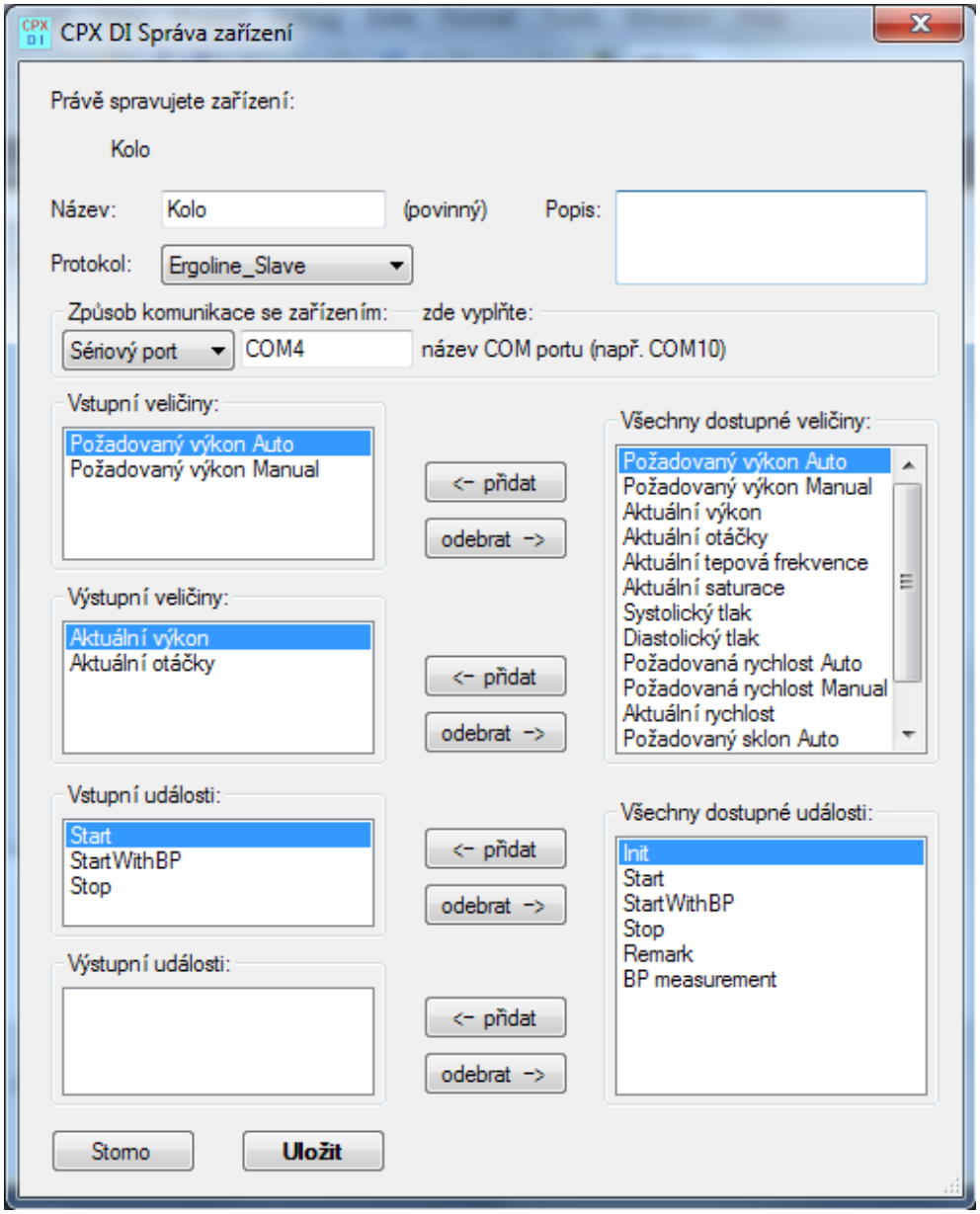
Události spravujeme velice podobně jako veličiny, nastavujeme jim tedy název, identifikátor, zkratku a popis (viz obrázek 2.5). Název a identifikátor jsou opět povinné položky, které musí být zároveň unikátní. Opět můžeme pomocí tlačítek údaje uložit nebo stornovat a při zavření okna dostaneme dotaz na uložení údajů.

2.1.8 Správa zařízení

Jak ilustruje obrázek 2.6, u zařízení je již potřebných údajů více. Základem je název, který má být povinný a unikátní, nechybí zde ani popis. Dále určíme použitý protokol, který vybíráme z rozbalovacího seznamu. Správně zvolit musíme také způsob komunikace se zařízením. Do vedlejšího políčka pak vyplníme konkrétní hodnotu parametru pro komunikaci daného zařízení dle zobrazené nápovědy. Typicky se jedná o název sériového portu nebo IP adresu a číslo portu síťového spojení.

Do následujících seznamů vybíráme vstupní a výstupní veličiny a události, které dané zařízení podporuje. Vstupními položkami jsou ty, které do zařízení posíláme. Výstupní hodnoty naopak zařízení naměří a vydává ven. Požadovanou veličinu či událost si vždy nejdříve označíme v seznamu všech dostupných, následně pomocí příslušného specializovaného tlačítka přidáme do odpovídajícího seznamu. Položku označenou v některém seznamu na levé části z něj můžeme zase odebrat druhým ze skupiny tlačítek označených šipkami.

K uložení či stornování změn opět slouží tlačítka umístěná v dolní části. Ani zde nezapomínáme na dotaz při přímém zavírání okna.



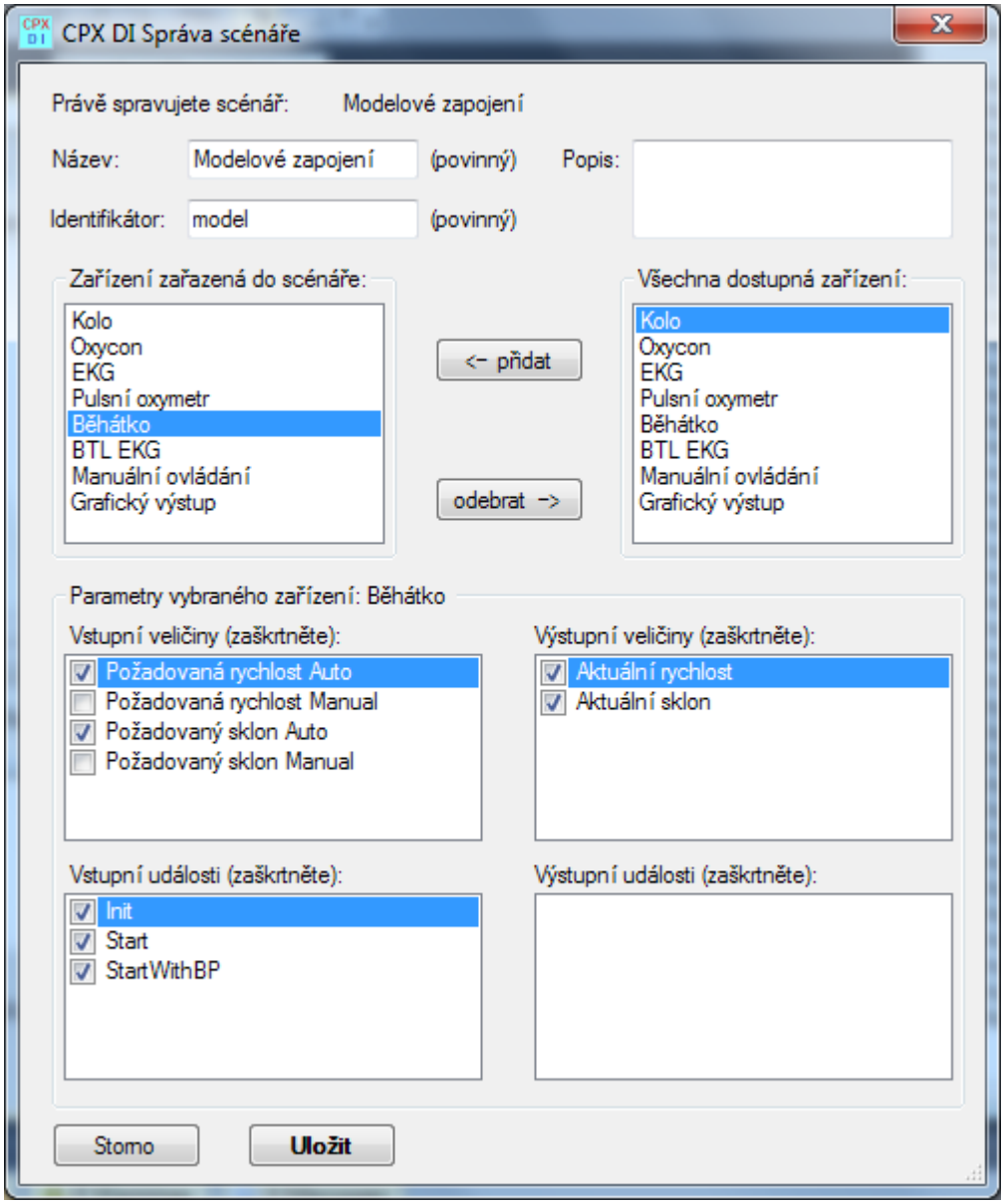
Obrázek 2.6: Správa zařízení

2.1.9 Správa scénářů měření

Základními spravovanými údaji u scénáře měření jsou opět název a identifikátor, které musí být povinné a unikátní. Identifikátor zde slouží pro možnost automatizovaného vybrání scénáře pro start měření při spouštění programu z příkazového řádku. Přidat můžeme také textový popis. Podobu okna vidíme na obrázku 2.7.

Zásadní je však výběr zařízení, které se budou v daném scénáři účastnit měření. Při výběru si požadované zařízení nejdříve označíme v seznamu všech dostupných, poté jej zařadíme do scénáře tlačítkem `přidat`. Naopak označené zařízení v seznamu již zařazených z něj vyřadíme tlačítkem `odebrat`.

Každému zařazenému zařízení ještě musíme určit vstupní a výstupní veličiny a události platné pro daný scénář. Při označení zařízení v seznamu zařazených se do boxů níže vypíšou všechny položky, které podporuje (tedy ty, co jsme mu nastavili ve správě zařízení). Z nich pak pomocí zaškrtávacích políček stanovíme,



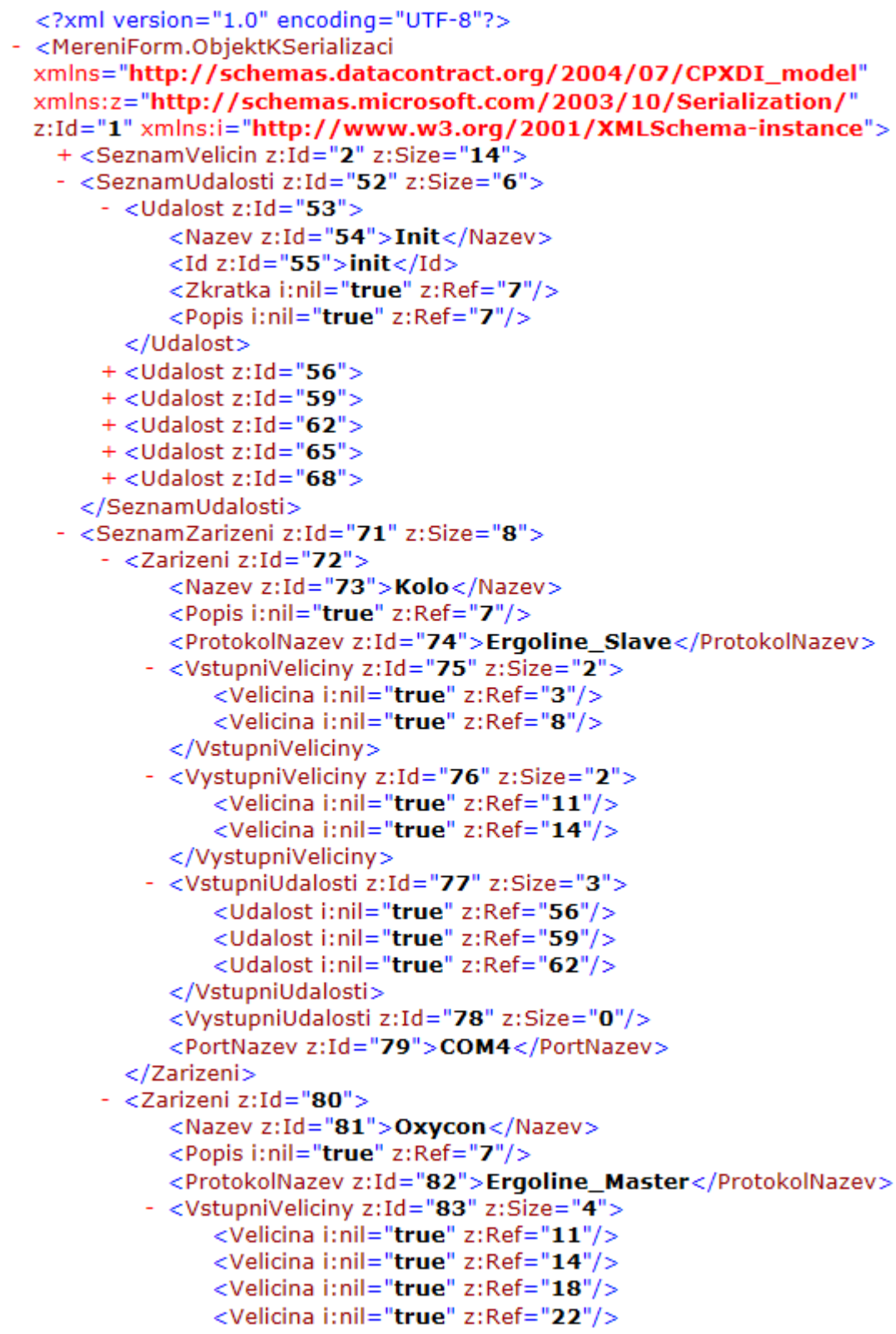
Obrázek 2.7: Správa scénáře

které mají být uplatněny v daném scénáři. U výstupních veličin to například může znamenat, že se na jejich hodnoty budeme zařízení periodicky dotazovat. Rovněž na monitoring měření v hlavním okně jsou přijímány pouze takto zaškrtnuté výstupní veličiny.

Možnosti uložení nebo stornování zůstávají stejné jako v předchozích oknech správy objektů.

2.1.10 Konfigurační soubor

Drobnější úpravy parametrů uložených objektů se dají provádět i přímou editací konfiguračního souboru. Ten je uložen v systémové složce nainstalovaného programu. Výchozí název souboru je souborSeznamy.xml, ale můžeme si zvolit i jiný, resp. vytvořit více variant a ty pak načítat při spouštění programu přes příkazový řádek.



Obrázek 2.8: Ukázka části konfiguračního souboru

Formát souboru je XML a jednotlivé tagy jsou celkem dobře čitelné, jelikož svými názvy poměrně přesně odpovídají uloženým objektům a jejich parametrům. Textový obsah tagů lze tedy upravit docela snadno. Je ovšem potřeba dávat pozor, aby se neporušily reference mezi objekty. Ty jsou vyjádřeny tak, že ve svém prvním výskytu dostane tag daného objektu atribut `z:Id` s číselnou hodnotou. Při dalším výskytu tohoto objektu má pak jeho tag atribut `z:Ref` se stejnou hodnotou a jinak je prázdný. Dalším specfikem schématu, do kterého program ukládá, je to, že každý prázdný tag musí mít atribut `i:nil` s hodnotou `true`. Část souboru můžeme vidět na obrázku 2.8.

2.2 Manuální ovládání

2.2.1 Základní pojmy

Některá zařízení v laboratoři (konkrétně Kolo a Běhátko) mají tu vlastnost, že nastavení hodnot vstupních veličin (konkrétně výkon, rychlost a sklon) probíhá postupně po menších krocích. Proto u nich musíme rozlišovat tzv. Aktuální veličiny (konkrétně Aktuální výkon, Aktuální rychlost, Aktuální sklon) a Požadované veličiny (Požadovaný výkon, rychlost, sklon). Hodnoty těchto Požadovaných veličin dostáváme při měření často vypočteny od připojených vyhodnocovacích programů (např. Oxycon).

Vedle toho ale chceme zachovat i manuální nastavování Požadovaných veličin. Proto Požadované veličiny dále rozlišujeme na tzv. veličiny typu Auto (např. Požadovaný výkon Auto, což dostáváme od vyhodnocovacích programů) a veličiny typu Manual (např. Požadovaný výkon Manual), pro jejichž generování byla vytvořena samostatná aplikace ManualController. Z pohledu integračního programu CPX DI je ManualController jen dalším zařízením, jeho konfigurace tedy probíhá stejně jako u jiných zařízení.

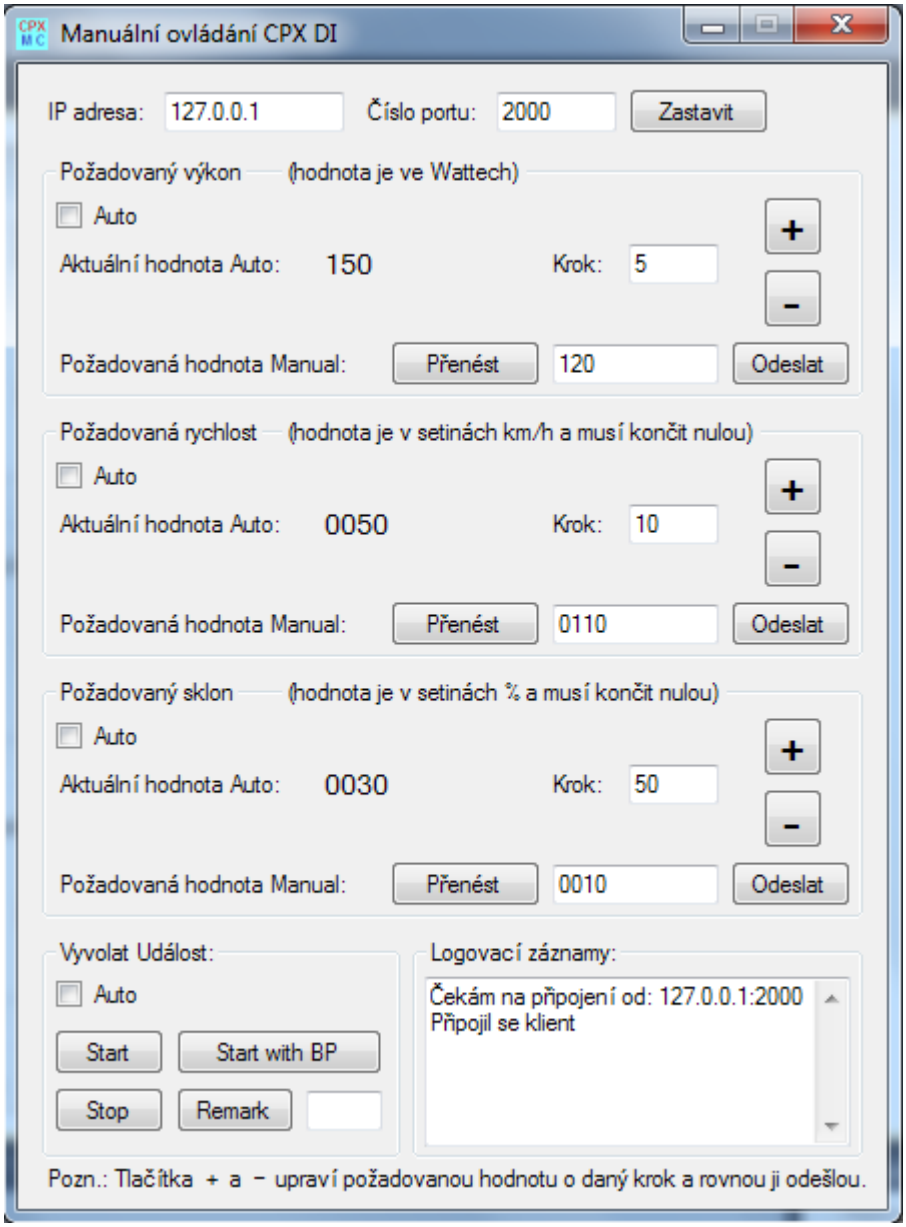
2.2.2 Spuštění programu

Program `ManualController` je samostatná aplikace, jejíž instalátor najdeme na přiloženém CD ve složce projektu `CPXDI_ManualController`. Po nainstalování program spustíme z nabídky Start.

2.2.3 Okno ovládání

Veškerá funkcionalita je soustředěna do hlavního ovládacího okna, které ukazujeme na obrázku 2.9. Program komunikuje přes síťové spojení, takže je třeba nejdříve spustit jeho komunikaci s potřebnými parametry. Vyplníme tedy IP adresu (předvyplněna je lokální adresa 127.0.0.1) a číslo portu (předvyplněno č. 2000) síťového spojení a stiskneme tlačítko `Spustit`. Teprve pak se k němu může připojit integrátor CPX DI.

Program přijímá z integrátoru hodnoty Požadovaných veličin typu Auto (konkrétně výkonu, rychlosti a sklonu) a zobrazuje je na popsaných místech. Tlačítkem `Přenést` máme možnost vyplnit tuto hodnotu i do políčka pro požadovanou hodnotu typu Manual. Můžeme tam samozřejmě zapsat i zcela libovolnou hodnotu,



Obrázek 2.9: Okno manuálního ovládání

jen by měla dodržet formát popsaný v nápovědě nad políčkem, jinak ji nebude možno zpracovat. Tuto hodnotu pak lze vedlejším tlačítkem odeslat zpět do integrátoru.

Speciální funkcí disponují tlačítka + a −. Ta totiž umí zapsanou hodnotu zvětšit nebo zmenšit o daný krok, který máme možnost předem zvolit v příslušném políčku. Zároveň změněnou hodnotu rovnou odešlou do integrátoru. Vhodná jsou tedy v případech, kdy potřebujeme hodnotu požadované veličiny plynule navyšovat či naopak snižovat.

Níže lze ještě vyvolávat Události start_with_BP, start, stop a remark (ta může mít i textovou hodnotu). Vedlejší pole slouží pro informační záznamy, např. o chybách či nerozpoznaných příkazech.

Automatický režim

Pomocí zaškrtávacího políčka označeného `Auto` máme možnost (odděleně pro každou veličinu a události) přejít do automatického režimu. Pokud je některá veličina v tomto režimu, tak pro ni nezapisujeme požadované hodnoty ručně (tlačítka jsou deaktivována), ale program pracuje tak, že přijatou hodnotu typu Auto okamžitě sám odešle zpět do integrátoru (jako tzv. loopback).

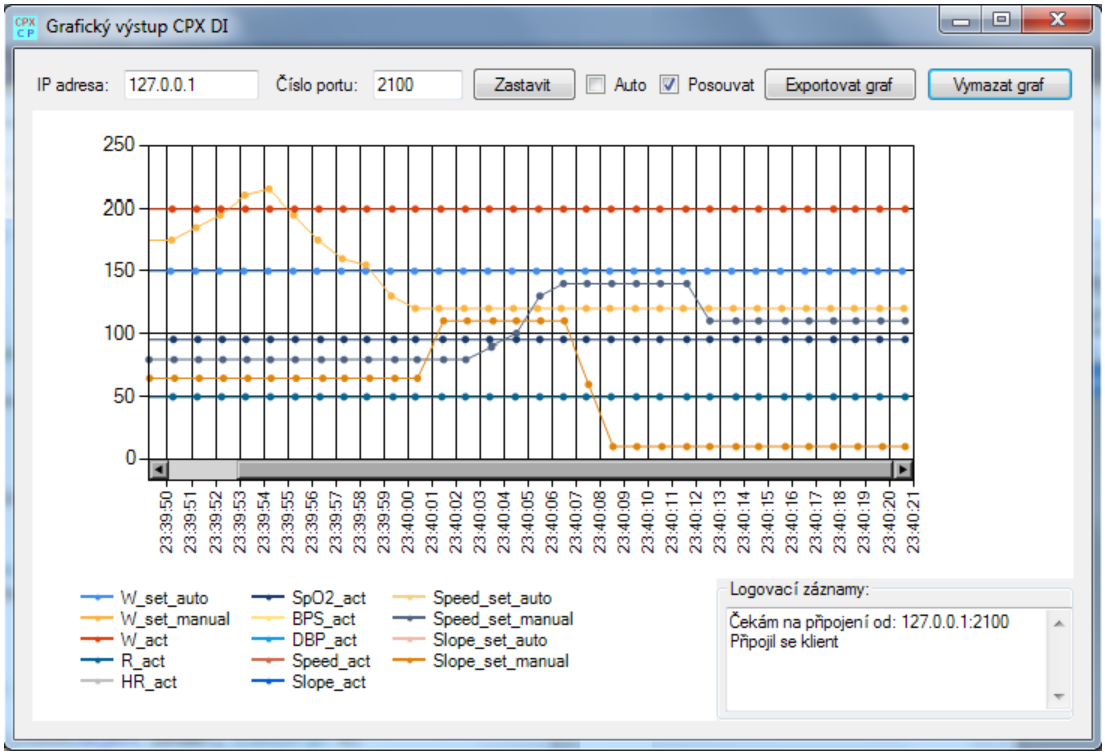
Pro události znamená automatický režim to, že si pamatujeme stav poslání událostí start a stop. Výsledkem je, že po odeslání startovní události se její tlačítko deaktivuje do doby, než je poslána událost stop. Stejně tak odesláním stop události je její tlačítko deaktivováno až do dalšího poslání startu.

2.3 Grafická vizualizace

2.3.1 Spuštění programu

Aplikace `ChartPresenter` je samostatným programem, jehož instalátor najdeme na přiloženém CD ve složce projektu `CPXDI_ChartPresenter`. Po nainstalování lze program spustit z nabídky Start.

2.3.2 Okno grafu



Obrázek 2.10: Okno grafu

Přenos dat do programu probíhá přes TCP spojení, musíme tedy nejdříve zapnout činnost TCP serveru. Zadáme tedy IP adresu (předvyplněna lokální 127.0.0.1) a číslo portu (výchozí zvoleno č. 2100) síťového spojení a klepneme na tlačítko `Spustit`. Pak teprve se k němu přípojí integrátor CPX DI.

Jak ukazuje obrázek 2.10, zbytek okna už zabírá převážně plocha s grafem. Na ose X jsou zobrazeny časové údaje, popisky mají tvar „HH:mm:ss“ a jsou v intervalu 1 sekunda. Ve viditelné části grafu je standardně obsaženo posledních 30 sekund, pohybovat se v celém grafu můžeme pomocí horizontálního posuvníku. S přibývajícími daty se osa X automaticky posouvá, aby bylo stále vidět nejnovějších 30 sekund. Automatický posuv je ale možno vypnout odškrtnutím příslušného zaškrtávacího políčka. Další zaškrtávací políčko zapíná automatické řízení grafu dle přicházejících událostí. Vedlejším tlačítkem se také dá celý graf vymazat.

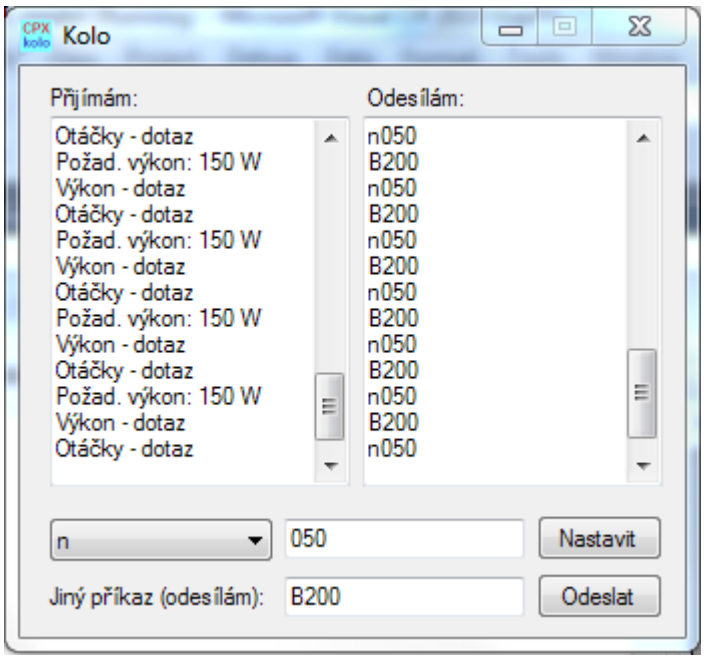
Osa Y zobrazuje číselné hodnoty veličin a automaticky se přizpůsobuje jejich aktuálnímu rozsahu. V grafu má každá veličina svoji linii a jednotlivé body jsou zvýrazněny kulatými značkami. Po najetí na danou značku se zobrazí titulek s hodnotami tohoto bodu. Pod grafem je umístěna legenda, která je připravena na všechny podporované veličiny. Vedle je ještě pole pro logovací záznamy (např. chyby či nerozpoznané příkazy).

Soubor všech hodnot zanesených do grafu lze také exportovat do formátu XML. Po stisknutí tlačítka `Exportovat graf` se ukáže dialog, ve kterém vybereme, kam si chceme soubor uložit.

2.4 Simulační programy

2.4.1 Spuštění programu

Samostatné simulační programy najdeme na přiloženém CD v příslušných složkách projektů, např. `Oxycon_model`. Jelikož jsou určeny jen k testování, příslušné spustitelné exe soubory se nacházejí v podsložkách `bin/Debug`.



Obrázek 2.11: Okno simulačního programu pro Kolo

2.4.2 Ovládací prvky

Tyto programy jsou určeny k simulaci chování různých přístrojů nebo programů používaných v laboratoři, především pro testování funkčnosti jejich komunikace s integrátorem. Proto obvykle obsahují několik textových polí, do kterých program vypisuje údaje z příkazů, které rozpoznal (tedy typicky hodnoty vstupních veličin). Další políčka jsou určena k tomu, abychom do nich zapisovali hodnoty veličin a příkazy, které chceme ze zařízení odeslat do integrátoru. Komunikace programů startuje hned po jejich spuštění a probíhá přes sériový port, jehož parametry jsou předem nastaveny v souladu se skutečným stavem zařízení v laboratoři. Příklad okna programu simulujícího Kolo je na obrázku 2.11.

3. Programátorská dokumentace

Celý projekt je rozdělen do čtyř částí:

- 3.1 Integrační program CPX DI – hlavní nástroj
- 3.2 Manuální ovládání – samostatná aplikace
- 3.3 Grafická vizualizace naměřených dat – doplňkový program
- 3.4 Simulační programy – pomocná simulace chování různých zařízení

Ve všech případech se jedná o aplikace typu Windows Forms. Aplikace jsou napsané v jazyce C# na platformě .NET. Ze zdrojových kódů jsme rovněž vygenerovali dokumentaci, která se nachází na přiloženém CD ve složce `dokumentace`.

3.1 Integrační program CPX DI

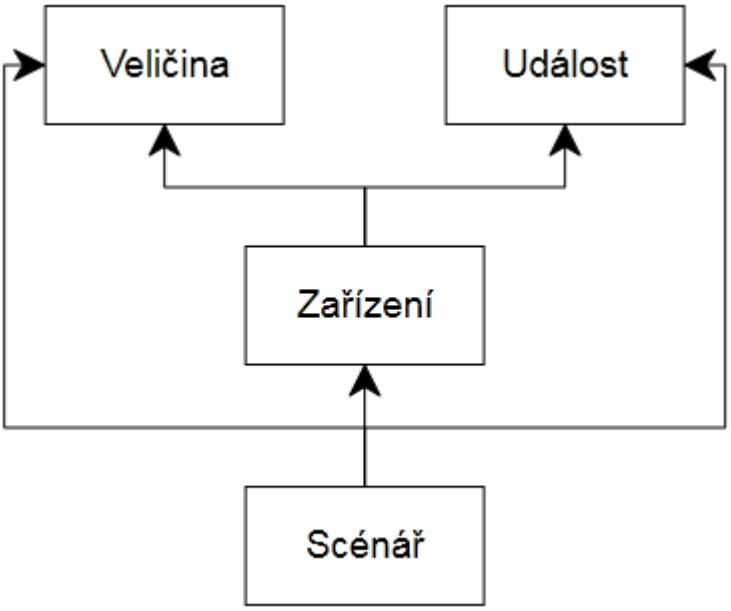
CPX Device Integrator (CPX = cardiopulmonary exercise testing) je hlavním nástrojem zajišťujícím propojení všech připojených zařízení i vyhodnocovacích programů včetně řízení komunikace mezi nimi.

3.1.1 Struktura programu

Jelikož jde o aplikaci typu WinForms, vstupním bodem do programu je formulář `Měření` (viz kapitola 3.1.11). Základní strukturu objektů, se kterými program pracuje (a které si také ukládá do konfiguračního souboru), tvoří třídy `Veličina` (v kapitole 3.1.2), `Událost` (kapitola 3.1.3), `Zařízení` (kapitola 3.1.4) a `Scénář` (v kapitole 3.1.5). K nim jsou vytvořeny i formuláře pro správu každé z nich, spolu se zastřešujícím formulářem `Konfigurace` (viz kapitola 3.1.10). K zařízení přísluší ještě další formulář `Okno zařízení` (v kapitole 3.1.9) pro ovládání a monitorování (logování) jeho aktivity při měření.

Třídy v základní struktuře objektů nejsou hierarchizovány ve smyslu dědičnosti, přesto mezi nimi určitý hierarchický vztah platí, a to spíše v tom smyslu, že jedna potřebuje druhou. `Veličina` a `Událost` ještě nic dalšího nepotřebují, proto jsou „na vrcholu“. Zato `Zařízení` už potřebuje znát seznam dostupných `Veličin`, aby se z něj mohly vybrat podporované veličiny daného zařízení. Proto je tedy `Zařízení` „níže“. `Scénář` pak pro sebe potřebuje seznam zařazených `Zařízení` (a pro každé z nich také vstupní a výstupní `Veličiny`), a proto je v tomto smyslu „nejníže“. Schematicky tuto hierarchii znázorňuje obrázek 3.1.

Dále jsou v programu „servisní“ třídy `Komunikace` (viz kapitola 3.1.6, zajišťuje fyzickou stránku komunikace se zařízením, konkrétně např. propojení na sériový port) a `Protokol` (v kapitole 3.1.7, zajišťuje obsahovou stránku, tedy především rozpoznávání a sestavování příkazů pro zařízení). Tyto třídy jsou abstraktní a definují metody, které musí jejich konkrétní potomci implementovat. Pro přiřazení správné konkrétní `Komunikace` a `Protokolu` v rámci konstrukce `Zařízení` pak slouží třídy `TvurceKomunikace` a `TvurceProtokolu`. Mezi „servisní“ třídy patří ještě `TaskQueue` (kapitola 3.1.8), která představuje frontu pro zpracování došlých příkazů do zařízení.



Obrázek 3.1: Hierarchie základních objektů

3.1.2 Veličina

Na vrcholu základní struktury objektů (resp. paralelně se třídou `Událost`) stojí třída `Veličina`. Ta představuje programovou reprezentaci jednotlivých veličin, se kterými pracují připojená zařízení. Obsahuje tyto datové položky (všechny jsou řetězce): `Název`, `Id` (pomocí toho s ní pracuje `Protokol`), `Zkratka`, `Jednotka` a `Popis`.

Dále je součástí statický `SeznamVeličin`, ve kterém jsou uloženy všechny `Veličiny`, které jsou v programu nakonfigurovány. Jako určitá podpora pro práci `Protokolu` slouží ještě statická metoda `Velicina vyhledejVelicinu(string id)`, která na základě přijatého `id` vydá příslušnou `Veličinu` ze seznamu (nebo `null`).

3.1.3 Událost

Třídou `Událost` reprezentujeme předdefinovanou akci, kterou může nějaké zařízení vyvolat (resp. požadavek na ni), nebo na ni naopak reagovat. Jinak s ní pracujeme analogicky, jako s `Veličinou`. Má opět položky `Název`, `Id`, `Zkratka` a `Popis`. Součástí je rovněž statický `SeznamUdálostí` i podpůrná metoda `Udalost vyhledejUdalost(string id)`.

3.1.4 Zařízení

Níže v základní struktuře je třída `Zařízení`. Ta představuje programovou reprezentaci různých zařízení, která můžeme mít připojena. Kromě datových položek už definuje i metody pro práci se zařízením (např. odesílání příkazů nebo inicializaci).

Základními (a zároveň konfigurovatelnými) datovými položkami jsou řetězce `Název`, `Popis`, `ProtokolNázev` a `PortNázev` (neboli komunikační parametr, ty-

picky se jedná o název portu), dále pak seznamy vstupních a výstupních `Veličin` a `Událostí`.

Další položky se vytvářejí automaticky hned po konstrukci zařízení (resp. po deserializaci), a to jsou: `VýchozíProtokol` (vznikne pomocí `TvůrceProtokolů` na základě `ProtokolNázvu`), `Komunikace` (ta je zase vytvořena prostřednictvím `TvůrceKomunikace` na základě komunikačního parametru `PortNázev`, navíc je při tom přijímán delegát na metodu pro příjem dat), `Časovač` (včetně obsluhy jeho události `Tick`) a `Okno` (pro ovládání a logování, tedy instance třídy `ZarizeniForm`).

Třetí skupina položek se vytváří až při inicializaci zařízení a tvoří ji `Fronta` (pro zpracování došlých příkazů, tedy instance `TaskQueue<string>`) a boolovské `pouzivatTimer`, `ukazovat` (pro zápis do logu) a `posilatNeexistujiciHodnoty` (pro posílání nulových hodnot veličin do zařízení) – v defaultním nastavení jsou `false`.

Součástí je i zde statický `SeznamZařízení`.

Inicializace zařízení

Pro inicializaci zařízení (tedy před začátkem měření) je potřeba nejprve zavolat metodu `InicializujKomunikaci()`, která si nejdříve od `Protokolu` vyžádá nastavení `VlastnostíKomunikace` a následně zavolá inicializaci své `Komunikace`. Teprve poté by se na zařízení měla zavolat metoda `InicializujSe()`, která dokončí přípravu na měření, tedy vyžádá si nastavení zařízení od `Protokolu`, vytvoří `Frontu` pro přijímání příkazů, nastartuje `Komunikaci`, případě i `Časovač`, dále pak nastaví a zobrazí logovací `Okno`. Shrnuje to swimlane diagram na obrázku 3.2.

Obdobně při ukončení měření by se na zařízení měla zavolat nejdříve metoda `DeInicializujSe()`, která zastaví `Časovač`, `Komunikaci` a `Frontu`, potom metoda `DeInicializujKomunikaci()`.

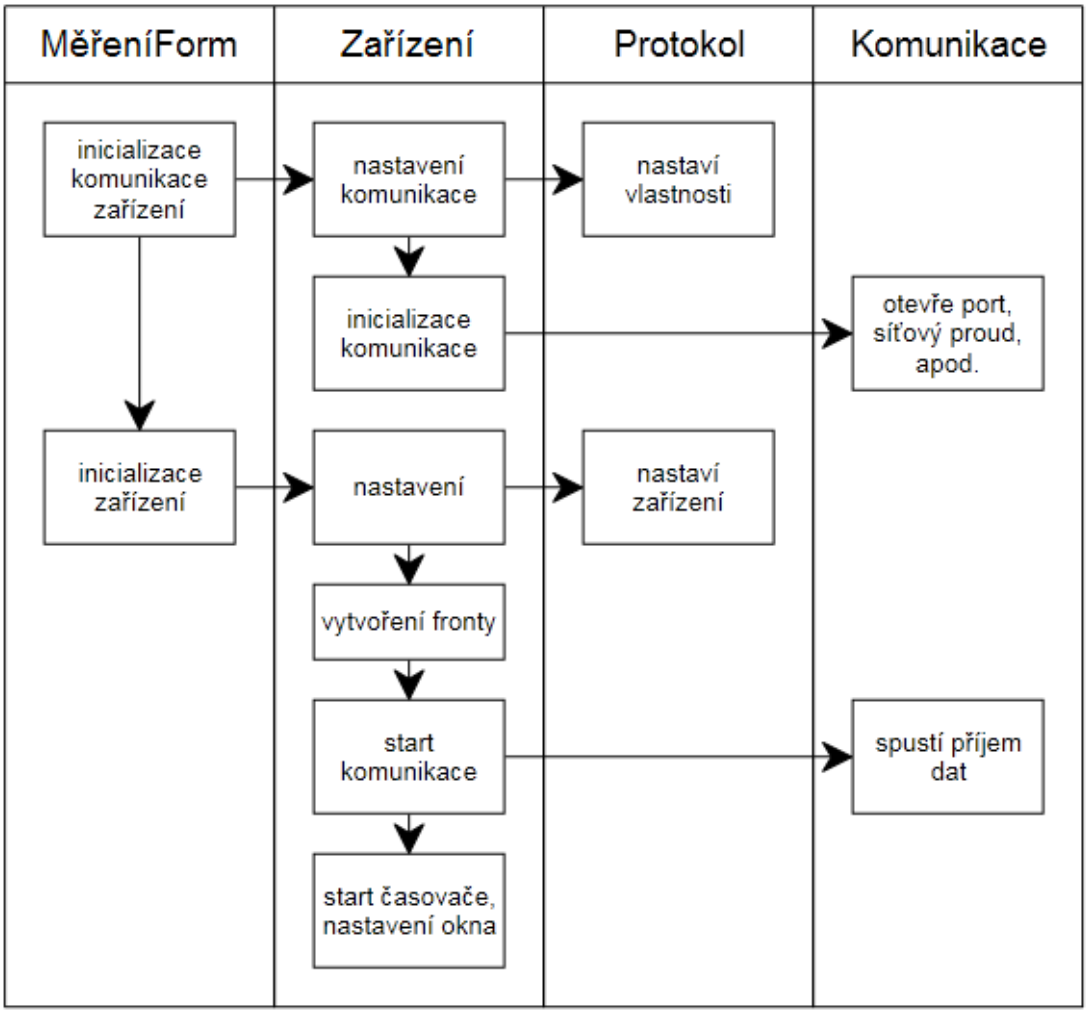
Obsluha časovače

Obsluha tikání časovače probíhá tak, že pokud se má `Časovač` použít (to udává `Protokol` při inicializačním nastavení zařízení, dále to v probíhajícím měření můžeme řídit zaškrtávátkem v `Okně zařízení`), tak zařízení na svém `Protokolu` zavolá metodu `ObsluzZarizeni` s parametrem `this`, ve které je obsaženo, jaké akce `Protokol` požaduje provést.

Příjem dat ze zařízení

Na metodu `void PrijmiDataKomunikace(string text, bool ok)` předáváme `Komunikaci` delegáta. Ta jej pak volá vždy, když se jí podaří přečíst data. Jediným úkolem metody je buď vložit přijatý text do fronty pro zpracování příkazů, nebo vypsat chybovou hlášku z parametru `text` do logu svého `Okna` (rozhodne se dle druhého parametru).

Zpracování příkazů z `Fronty` zajišťuje metoda `void ZpracujData(string data)`, která je právě z oné fronty vyvolána, přijatý příkaz zapíše na log `Okna` (pokud chceme) a dále předá příkaz ke zpracování `Protokolu`. Ten může, po-



Obrázek 3.2: Diagram postupu inicializace zařízení

kud došlo k chybě při zpracování (např. nerozpoznání příkazu), vrátit textovou zprávu, která se opět zobrazí v logu `Okna zařízení`.

Metoda `void ProvedUdalost(Udalost udalost, string hodnota)` potom zajišťuje reakci zařízení na obdrženou `Událost` (akci). Pracuje obdobně jako výše uvedené zpracování příkazů, tedy zapíše na log (volitelně), zavolá obsluhu `Události` svého `Protokolu` a případně zapíše chybovou zprávu.

Odesílání dat do zařízení

Pokud chceme do zařízení nějaký příkaz poslat (toto volání typicky využívá `Protokol`), použijeme metodu `void OdesliPrikaz(string prikaz)`. Ta opět nejdříve volitelně zapíše na log, dále pak zavolá metodu `void WriteLine(string text)` své `Komunikace`. Pokud v tomto volání nastane chyba (výjimka), odchytí ji a zobrazí chybovou zprávu opět do logu `Okna`.

Protože některá používaná zařízení potřebují mezi přijatými příkazy nechávat prodlevy, je tato metoda přetížená. Druhou variantou je tedy metoda `void OdesliPrikaz(string prikaz, int prodleva)`, která jen pozastaví provádění na danou dobu (číslo je v milisekundách, očekává se kladná hodnota maximálně 1000 ms) a poté zavolá svou první variantu.

3.1.5 Scénář

Poslední ze základních tříd je `Scénář` měření. Představuje programovou reprezentaci toho, jak bude měření uspořádáno (tedy která `Zařízení` jsou zapojena a jaké `Veličiny` měříme). Obsahuje řetězcové položky `Název`, `Id` a `Popis`, dále seznam zařazených `Zařízení` a slovníky přiřazující zařazeným zařízením seznamy jejich vstupních a výstupních `Veličin` a `Událostí` pro použití v daném `Scénáři`. I zde je obsažen statický `SeznamScénářů`.

3.1.6 Komunikace

`Komunikace` je abstraktní třída představující programovou reprezentaci komunikační komponenty, která má na starosti fyzickou stránku komunikace se zařízením. Třída definuje metody, které musí konkrétní komunikační komponenta implementovat. Jsou to:

- `void InicializujSe(VlastnostiKomunikace vlastnosti);` – argumentem je objekt s `Vlastnostmi Komunikace` (viz kapitola 3.1.6, nastaveny od `Protokolu`), provede inicializaci komunikační komponenty (např. vytvoření a otevření sériového portu a jeho nastavení)
- `void DeInicializujSe();` – obdobná metoda pro deinicializaci (zahrnuje např. uzavření portu)
- `void Start();` – pro nastartování komponenty, především pak procesu čtení dat (tedy např. obsluhy události `DataReceived` sériového portu)
- `void Stop();` – obdobná metoda pro zastavení
- `void WriteLine(string text);` – zapsání textu do výstupního bufferu (např. na sériový port) pro odeslání příkazu do zařízení, příkaz má být bez koncového znaku, metoda sama koncový znak (nastavený `Protokolem` ve `Vlastnostech Komunikace`) přidá

Dále je zde definován delegát pro přijímací metodu, který je komunikační komponentě předáván v konstruktoru. Příslušnou metodu pak komponenta volá vždy, když se jí podaří přečíst data. Čtení dat komponenta provádí na principu metody `ReadLine`, tedy přečtený příkaz sama očistí o koncový znak.

Tvůrce Komunikace

Pro vytváření nových instancí komunikačních komponent slouží statická třída `TvurceKomunikace` (fungující na principu návrhového vzoru *Factory Method*). Ta obsahuje především metodu `static Komunikace VratKomunikaci(string komunikacniParametr, Komunikace.Prijimaci metoda)`, kterou při své konstrukci zavolá `Zařízení` a na základě zadaného komunikačního parametru a delegátu na přijímací metodu dostane příslušnou instanci komunikační komponenty odpovídající danému parametru. Další metodou je `static Dictionary<string, string> VratSlovnik(string komunikacniParametr, out int index)`, ta je zase volána formulářem `SprávaZařízeníForm`, který z ní dostane slovník všech dostupných komunikačních komponent (klíčem je název a hodnotou nápověda

pro vyplnění komunikačního parametru) a navíc na základě zadaného komunikačního parametru ještě do výstupní proměnné index (do tohoto slovníku) pro odpovídající komponentu.

Vlastnosti Komunikace

Třída `VlastnostiKomunikace` spojuje všechny možné vlastnosti, které může komunikační komponenta potřebovat nastavit od `Protokolu`. Ten je má možnost nastavit (typicky nastaví jen některé z nich) před inicializací komunikace a komponenta je (opět typicky jen ty, které potřebuje) může následně při své inicializaci využít.

Obecně využitelným parametrem komunikace je řetězec `NewLine` pro symboly konce řádku. Další vlastnosti specifické pro sériový port jsou celočíselné hodnoty `BaudRate` a `DataBits`, dále pak výčtové `StopBits`, `Parity` a `Handshake`. Naopak pro TCP klienta jsou typické vlastnosti `Encoding` pro specifikaci kódování a celočíselný `BufferSize` pro nastavení velikosti pracovního bufferu.

V současné době jsou implementovány komunikační komponenty pro sériový port (`SerialPortKomunikace`) a pro TCP klienta (`TcpClientKomunikace`).

SerialPortKomunikace

Třída `SerialPortKomunikace` představuje komunikační komponentu pro práci se sériovým portem. V konstruktoru vyžaduje název portu (typicky ve tvaru např. *COM1*) a delegáta na metodu pro příjem dat, kterou použije v EventHandleru pro událost `SerialDataReceived`.

Při inicializaci dojde k otevření nového portu a nastavení všech jeho vlastností. Při deinicializaci naopak port uzavřeme a uvolníme jeho systémové prostředky. Start a Stop vlastní činnosti komponenty spočívá v přiřazení, resp. odřazení výše zmíněného EventHandleru pro událost `SerialDataReceived`. Pro zápis dat je využita přímo metoda `WriteLine` sériového portu.

TcpClientKomunikace

Třída `TcpClientKomunikace` je komponentou pro komunikaci přes TCP spojení. V konstruktoru vyžaduje IP adresu (typicky ve tvaru např. *127.0.0.1*), číslo portu a delegáta na přijímací metodu.

Při inicializaci dojde k založení nového TCP klienta, získání jeho síťového proudu a nastavení všech vlastností (oddělovač řádku, kódování a velikost bufferu). Při deinicializaci naopak proud i klienta uzavřeme. Start a Stop vlastní činnosti komponenty spočívá v nastartování, resp. ukončení naslouchacího vlákna. To má na starosti čtení dat ze streamu (pomocí `ReadByte()`). Zápis dat probíhá opět přímo do streamu metodou `Write` (po přidání symbolu `NewLine`).

3.1.7 Protokol

Třída `Protokol` má na starosti především zpracovávání příkazů od a do zařízení, tedy z přijatého příkazu rozpoznat `Veličinu` (s hodnotou) nebo `Událost` (s hodnotou) a poslat je na globální monitoring (ten spravuje `Okno měřeni`), případně

reagovat odesláním dalších příkazů (odpovědí) do zařízení. Dále obsluhuje zařízení na základě tikání časovače (typicky periodické dotazy na hodnoty či přímo posílání hodnot některých veličin) a obsluhuje i reakce na `Události` (akce). Také může nastavovat vlastnosti komunikace a parametry zařízení. Ještě definuje příkazy pro ovládání zařízení. Je opět abstraktní a pro potomky definuje tyto metody:

- `string VypisJmeno();` – vrátí jméno
- `void NastavKomunikaci(VlastnostiKomunikace vlastnosti);` – nastaví vlastnosti komunikace
- `void NastavZarizeni(Zarizeni zarizeni);` – nastaví další požadované parametry zařízení
- `void ObsluzZarizeni(Zarizeni zarizeni);` – obslouží zařízení (při tiknutí časovače)
- `string ObsluzUdalost(Udalost udalost, string hodnota, Zarizeni komu);` – reaguje na `Událost` (akci), když nerozpozná, může vrátit chybovou zprávu
- `string ZpracujPrikaz(string prikaz, Zarizeni odkud);` – zpracovává přijatý příkaz, když nerozpozná, může vrátit chybovou zprávu
- `Dictionary<string, string> DefinujPrikazyOvladani();` – vrátí slovník příkazů pro ovládání zařízení, klíčem je název či popis příkazu, hodnotou je přesné znění příkazu

Kromě toho ještě sama přepisuje `ToString()` s pomocí metody `VypisJmeno()` od potomků.

Tvůrce Protokolu

Obdobně jako u `Komunikace` slouží i zde pro vytváření nových instancí `Protokolů` statická třída `TvurceProtokolu`. Má metodu `static Protokol VratProtokol (string protokolParametr)`, kterou volá `Zařízení` a na základě zadaného názvu protokolu dostane příslušnou instanci `Protokolu`. Pro konfiguraci ve formuláři `SprávaZařízeníForm` je zde metoda `static List<string> VratSeznam (string protokolParametr, out int index)`, která vydává tentokrát seznam (názvů) všech dostupných `Protokolů` a opět navíc na základě zadaného názvu protokolu ještě do výstupní proměnné index (do tohoto seznamu) pro odpovídající `Protokol`.

Master a Slave

Při implementaci rozlišujeme a implementujeme odlišně protokoly obsluhující zařízení typu Master a typu Slave. Zařízení typu Slave jsou fyzické přístroje (např. Kolo), které programem ovládáme. Typem Master naopak rozumíme typicky vyhodnocovací programy (např. Oxycon), které posílají ovládací příkazy do našeho programu a my pro ně emulujeme fyzické zařízení.

V současné době jsou implementovány následující protokoly: `Ergoline_Slave`, `Ergoline_Master`, `Nonin_Slave`, `Nonin_Master`, `T2000_Slave`, `T2000_Master`, `BtlEcg_Slave`, `ManualControl_Master` a `ChartPresent_Slave`.

Ergoline_Slave – Obsluhuje bicyklový ergometr (tzv. kolo). Určuje přesně vlastnosti sériového portu, definuje i slovník příkazů pro ovládání. V obsluze zařízení mu periodicky posílá nastavení výkonu a dotazuje se na hodnoty výstupních veličin, proto vyžaduje u `Zařízení` použití časovače. Zařízení ale nedokáže přijmout více příkazů souběžně, proto musí odesílat postupně a s prodlevami. To platí i pro příkazy `Událostí`, z nichž obsluhuje tyto: `start`, `start_with_BP` a `stop`. Přijímá příkazy s aktuálními hodnotami `Veličin`, které vkládá na monitoring.

Ergoline_Master – Emuluje chování bicyklového ergometru vůči vyhodnocovacím programům. Sám tedy `Zařízení` ani `Události` neobsluhuje, pouze přijímá požadavky pro nastavení hodnot či `Událostí` nebo dotazy na aktuální hodnoty, na které obratem odpovídá.

Nonin_Slave – Obsluhuje pulsní oxymetr. Toto zařízení žádné příkazy nepřijímá, pouze periodicky posílá naměřené hodnoty saturace a tepové frekvence, které ukládáme na monitoring. Pokud jsou hodnotami pouze pomlčky, tak je naopak z monitoringu mažeme (vložením hodnoty `null`). Vlastnosti sériového portu jsou opět přesně určeny.

Nonin_Master – Emuluje chování pulsního oxymetru. Používá tedy časovač u `Zařízení` a v jeho obsluze periodicky posílá hodnoty saturace a tepové frekvence z monitoringu, příp. pomlčky, pokud nejsou hodnoty známy. Sám už nic nepřijímá.

T2000_Slave – Obsluhuje pohyblivý pás (tzv. běhátko). Určuje vlastnosti sériového portu i bohatý slovník ovládacích příkazů. V obsluze zařízení mu periodicky posílá nastavení vstupních veličin a dotazuje se na aktuální naměřené hodnoty, proto vyžaduje u `Zařízení` použití časovače. Z `Událostí` obsluhuje tyto: `init`, `start` a `stop`. Přijaté aktuální hodnoty `Veličin` opět vkládá na monitoring.

T2000_Master – Emuluje chování pohyblivého pásu vůči vyhodnocovacím programům. `Zařízení` ani `Události` tedy sám neobsluhuje, jen přijímá nastavení hodnot, `Událostí`, dotaz na aktuální hodnoty a další servisní příkazy, na což hned odpovídá.

BtlEcg_Slave – Obsluhuje samostatný měřič EKG. Ten se chová podobně jako oxymetr Nonin, tedy žádné příkazy nepřijímá, pouze periodicky posílá naměřené hodnoty tepové frekvence, které ukládáme na monitoring. Pokud jsou však hodnoty neznámé, tak zařízení na rozdíl od Noninu neposílá nic, takže musíme provádět kontrolu aktuálnosti údajů. K té využíváme u `Zařízení` časovač s intervalem 10 sekund a pokud je hodnota starší, tak ji z monitoringu vymažeme (vložíme hodnotu `null`). Vlastnosti sériového portu jsou i zde jasně určeny.

ManualControl_Master – Komunikuje se samostatným programem ManualController (viz kapitola 3.2), a to pomocí TCP spojení, jehož vlastnosti určuje. Při obsluze zařízení využívá jeho časovač, protože mu periodicky posílá hodnoty vstupních veličin. Přicházející hodnoty `Veličin` i `Událostí` ukládá na monitoring.

ChartPresent_Slave – Zajišťuje data pro doplňkovou aplikaci ChartPresenter (kapitola 3.3), opět přes TCP spojení s danými vlastnostmi. Od ní nic nepřijímá, jen jí periodicky posílá hodnoty všech (vstupních) veličin dostupných na monitoringu, k čemuž využívá časovač u `Zařízení`.

3.1.8 TaskQueue

Pro realizaci fronty sloužící ke zpracování příchozích příkazů byla vytvořena třída `TaskQueue<T>`, která představuje frontu typu *producent – konzument*. Typový parametr `T` představuje typ úkolu, pro který bude fronta použita (obdobně jako klasická `Queue<T>`). V programu se využívá jednak pro zpracování přijatých příkazů ze zařízení (to je `TaskQueue<string>`), dále pak ještě v `Okně měření` pro zpracování `Veličin` přicházejících na globální monitoring (to je `TaskQueue<Tuple <Velicina, string, Zarizeni>>`) a též pro zpracování přicházejících `Událostí` (to je `TaskQueue<Tuple<Udalost, string, Zarizeni>>`).

Fronta se vytvoří použitím příslušného konstruktoru při inicializaci zařízení, resp. při zahájení měření v `Okně měření`. Následně konzumentské vlákno začne pracovat, tedy vybírat úkoly z fronty a na základě typového parametru volat jejich obslužné metody (tedy buď `ZpracujData` aktuálního `Zařízení`, nebo `UlozData` či `RozesliUdalost` aktuálního `Okna měření`).

Pro vložení nového úkolu daného typu slouží metoda `EnqueueTask(T task)`. Tu volá `Zařízení` ve své metodě `PrijmiDataKomunikace`, resp. `Okno měření` ve svých metodách `vlozDoFrontyVelicin` a `vlozDoFrontyUdalosti`. K ukončení práce fronty slouží metoda `Dispose()`, která se volá při deinicializaci zařízení, resp. při ukončení měření.

3.1.9 Okno zařízení

Formulář `ZařízeníForm` představuje monitorovací (logovací) a ovládací okno pro každé zařízení, které se účastní měření. Dá se v něm sledovat log příchozích a odchozích příkazů, řídit používání časovače a posílání nulových hodnot veličin a odesílat příkazy pro ovládání zařízení.

Okno je zkonstruováno hned po konstrukci zařízení (resp. po deserializaci) a ve svém konstruktoru přímo dostane své `Zařízení`. Z něj si vyčte název a pomocí jeho protokolu dostane slovník příkazů ovládání, kterým naplní příslušný `ListBox`. Při inicializaci zařízení mu pak jsou poslány hodnoty pro nastavení řídících `CheckBoxů` a okno je zobrazeno.

Pro zápis na log poskytuje okno metodu `DisplayData(string msg)`, kterou jeho `Zařízení` v případě potřeby volá. Při deinicializaci zařízení se okno nezavírá, toto je ponecháno na uživateli. Při novém měření se pak případně okno opět zkonstruuje a zobrazí.

3.1.10 Konfigurace

Zastřešujícím prvkem pro konfigurování programu je formulář `KonfiguraceForm`. Uživatelsky je možno spravovat všechny 4 základní objekty programu – `Veličiny`, `Události` (akce), `Zařízení` a `Scénáře`. Tím se rozumí přidávání nových a editace i odstraňování stávajících.

Odstraňování zajišťuje přímo formulář `KonfiguraceForm`, přičemž jsou kontrolovány kolize, tedy nelze smazat výše postavený objekt, pokud je použit jako součást níže postaveného (např. `Veličina` nastavená na vstupu či výstupu nějakého `Zařízení`, nebo `Zařízení` zařazené v nějakém `Scénáři`). Před odstraněním je ještě položen dotaz uživateli.

Jinak se otevře pomocí `ShowDialog()` formulář pro správu příslušného objektu, a to buď bezparametrickým konstruktorem (v případě nového objektu), nebo je daný objekt předán v konstruktoru (v případě editace stávajícího).

Formuláře pro správu mohou být tedy zkonstruovány dvěma způsoby, dle toho si případně formulář načte datové položky stávajícího objektu, nebo bude zakládat nový. Konfigurovat lze vždy základní datové položky objektů, většinou jsou to řetězcové hodnoty a seznamy. U povinných položek (vyznačeno ve formuláři, např. `Název`) je kontrolováno vyplnění, případně i unikátnost, v případě chyby je údaj doplněn. Při zavírání je zobrazen dotaz na uložení údajů.

Poznamenejme ještě, že položka `Id` u `Veličin` a `Událostí` je důležitá proto, že pomocí ní `Protokol` dané objekty rozpoznává. Pokud se tedy změní, může být potřeba i změna v `Protokolu`. Pro párování mezi příkazy a `Id` (a naopak) obsahuje `Protokol` slovníky `idPrikazy` a `prikazyId`.

3.1.11 Okno měření

Vstupním bodem do programu je formulář `MěřeníForm`. Jeho hlavní starostí je průběh konkrétního měření. Můžeme tedy především spouštět a ukončovat měření dle vybraného scénáře, dále máme přístup ke konfiguraci. Při probíhajícím měření formulář spravuje tzv. monitoring, tedy sledování aktuálních hodnot měřených veličin. Přicházející požadavky se volitelně zapisují do logu. `CheckBoxem` můžeme zapisování do logovacích `TextBoxů` nastavit (platí i pro logy zařazených zařízení). Další `CheckBox` nastavuje, jestli se mají zobrazovat ovládací okna zařízení. Vyvolávat příjem `Veličin` a `Událostí`, tedy ovládat monitoring, můžeme i manuálně. Dále máme možnost vybrané zařízení znovu nebo dodatečně připojit do běžícího scénáře.

Zpracování argumentů příkazového řádku

Pokud je program spouštěn z příkazového řádku, mohou mu být předány ještě některé další argumenty:

- Prvním z nich je parametr `-c`, za kterým následuje jméno konfiguračního souboru (očekává se formát XML). Pokud není tento parametr zadán, použije se jméno `souborSeznamy.xml`.

Z daného konfiguračního souboru se v dalším kroku deserializují 4 základní seznamy `Veličin`, `Událostí`, `Zařízení` a `Scénářů`. Pokud došlo při deserializaci k chybě, nastaví se předpřipravená konfigurace seznamů *Modelové zapojení*

(metodou `NastavModeloveZapojeni()`). Pak se můžou naplnit i `ComboBoxy` pro výběr `Scénářů` měření i `Veličin`, `Událostí` a `Zařízení` (ty jsou pro manuální ovládání).

- Dalším parametrem je `-o`, za kterým následuje `Id` scénáře, který má být předvybrán v `ComboBoxu` scénářů.
- Dále je k dispozici parametr `-s`, který vyvolá automatický start měření dle scénáře vybraného v `ComboBoxu`.
- Pokud byl parametr `-s` využit a měření se automaticky spustilo, dá se ještě automaticky provést nějaká `Událost` pomocí zadaného parametru `-e`, za kterým následuje `Id` požadované `Události`.

Monitoring měřených veličin

Pro vkládání měřených hodnot `Veličin` na monitoring poskytne formulář metodu `void vlozDoFrontyVelicin(Tuple<Velicina, string, Zarizeni> task)`, ta přijímá trojici údajů `Veličina`, její hodnota a `Zařízení`, odkud tato veličina pochází (využívají ji tedy především `Protokoly`). Pro sběr těchto požadavků je opět využita fronta `TaskQueue`.

Při jejím vybírání se pak volá metoda `UlozData`, která má na starosti samotné ukládání na monitoring. Ta tedy volitelně zapíše na log (k tomu je zde opět metoda `DisplayData(string msg)`), dále pak zkontroluje, jestli má dané `Zařízení` danou `Veličinu` v aktuálním `Scénáři` na výstupu a jen tehdy ji na monitoring opravdu zapíše.

K realizaci monitoringu je využit slovník `aktVelicinyHodnoty`, který je také poskytován ven prostřednictvím metody `vratAktVelicinyHodnoty()` (opět využíváno `Protokoly`). Uživateli se pak zobrazuje jako `DataGridView` (to řídí metoda `DisplayMonitorData()`).

Ještě je možné, aby přijatá hodnota byla `null`, což potom znamená výmaz dané veličiny z monitoringu. I tohle může udělat jen zařízení, které má danou veličinu v aktuálním scénáři na výstupu.

Monitoring Událostí

Obdobným způsobem se přijímají `Události` (akce). Opět je zde fronta přijatých `Událostí` a metoda pro příjem `void vlozDoFrontyUdalosti(Tuple<Udalost, string, Zarizeni> task)`.

Při výběru se volá metoda `RozesliUdalost`, která `Událost` rovnou rozešle do všech zúčastněných zařízení, která ji mají nastavenou v aktuálním `Scénáři` na vstupu (tedy zavolá jejich metodu `void ProvedUdalost(Udalost udalost, string hodnota)`). I toto však provádí pouze, pokud byla `Událost` vyvolána `Zařízením`, které ji má nastavenou v aktuálním `Scénáři` na výstupu.

Aktuální scénář

Podobně jako slovník monitoringu poskytuje `Okno měření` ještě aktuální `Scénář` (položka `aktScenar`) metodou `vratAktScenar()` (též využívána `Protokoly`).

Dále je poskytováno aktuální nastavení ohledně zápisu do logu (položka `bool ukazovat`) metodou `vratUkazovat()` a příznak `bool ukazovatOknaZarizeni` metodou `vratUkazovatOknaZarizeni()` pro zobrazování ovládacích oken (využíváno `Zařízeními`).

Start a stop měření

Start a stop konkrétního měření zajišťuje metoda `startStopButton_Click`. V případě startu provede tyto kroky: nastaví aktuální `Scénář` dle výběru v `ComboBoxu`, nastaví příznak zápisu do logu dle `CheckBoxu`, inicializuje fronty pro `Veličiny` a `Události`, pro všechna zařazená `Zařízení` zkusí inicializovat `Komunikaci`, pokud se povede, zařadí je do seznamu aktivních zařízení, pro všechna aktivní zařízení zavolá jejich inicializaci, inicializuje monitoring, deaktivuje konfigurační prvky a aktivuje ovládací prvky formuláře.

Při požadavku na Stop pak provede následující kroky: deinicializuje aktivní `Zařízení` a pak deinicializuje jejich `Komunikace`, deinicializuje fronty, deaktivuje ovládací prvky a aktivuje konfigurační prvky formuláře.

Dodatečné připojení zařízení

V průběhu měření se někdy může vyskytnout potřeba dodatečně připojit další zařízení, nebo znovu připojit takové, které se odpojilo, a to bez přerušení běhu scénáře. K tomu zde máme metodu `reConnectButton_Click`. Jestliže vybrané zařízení již bylo mezi aktivními, tak jej nejdříve deinicializuje (včetně komunikace). Po kontrole, zda je zařazeno v aktuálním scénáři, se pro něj následně provedou stejné inicializační kroky jako při startu scénáře, čímž se opět dostane do seznamu aktivních zařízení.

Přístup ke konfiguraci

Metoda `konfiguraceButton_Click` má za úkol otevřít pomocí `ShowDialog()` formulář `KonfiguraceForm`. Po dokončení jeho práce ještě aktualizuje `ComboBoxy` s příslušnými seznamy a provede serializaci všech seznamů do konfiguračního souboru.

Zavírání okna programu

Při zavření `Okna měření` se předně ukončí měření, pokud ještě probíhá. Následně se provede serializace seznamů. Použije se již dříve nastavené jméno pro vytvoření konfiguračního souboru.

Ukládání konfiguračních dat

Pro uložení konfiguračních dat programu využíváme serializaci 4 základních seznamů `Veličin`, `Událostí`, `Zařízení` a `Scénářů`, které pro tento účel sdružujeme do třídy `ObjektKSerializaci`. U každého objektu základní struktury se serializují vždy ty položky, které daný objekt vyžaduje v konstruktoru (tedy tytéž, které spravujeme v konfiguraci). Tyto položky jsou označeny atributem `DataMember`, celá třída pak atributem `DataContract`.

Aby byla serializace slovníků ve `Scénáři` přehlednější, vytvořili jsme speciální třídy odvozené od odpovídajících slovníků. To umožňuje určit, jak mají být pojmenovány elementy označující položky, klíče a hodnoty (nastaveno v atributu `CollectionDataContract`).

K serializaci je využit `DataContractSerializer`, serializujeme do XML souboru, zápis provádí `XmlWriter` s nastavením indentace. Nastaveno je rovněž zachovávání referencí mezi objekty.

3.2 Manuální ovládání

Program `ManualController` slouží k rozšířenému, příp. vzdálenému manuálnímu ovládání některých zařízení. Tzv. `Požadované veličiny`, které můžeme těmto zařízením nastavovat, rozlišujeme dále na typ `Auto` a typ `Manual`. Veličiny typu `Auto` jsou vypočítávány vyhodnocovacími programy a tato aplikace je přijímá a zobrazuje. Naopak typ `Manual` můžeme touto aplikací sami generovat a odesílat zpět do integrátoru.

3.2.1 Struktura programu

Vstupním bodem do programu je formulář `OvládáníForm` (viz kapitola 3.2.4), který zároveň z důvodu praktické jednoúčelovosti aplikace sdružuje i veškerou další funkčnost. Jedinou výjimkou je samostatná „servisní“ třída `TaskQueue` (v kapitole 3.2.3), která stejně jako u hlavního programu CPX DI zajišťuje frontu pro zpracování příkazů došlých do aplikace.

3.2.2 Komunikace

Program komunikuje přes TCP spojení a zaujímá postavení TCP serveru. Po startu naslouchacího vlákna se tudíž dle zadané (a předem zkontrolované) IP adresy a čísla portu založí nový `TcpListener`, který následně čeká na připojení TCP klienta. Poté si vyzvedneme síťový proud, ze kterého již čteme data pomocí `ReadByte()`. Když se klient odpojí, uzavřeme stream a opět přejdeme do stavu čekání. Zápis dat provádíme do streamu metodou `Write` po přidání koncového symbolu.

3.2.3 TaskQueue

Tato třída je totožná s třídou `TaskQueue` (kapitola 3.1.8) hlavního programu CPX DI, pouze je zúžena pro použití jen v `Okně ovládání` pro přijímání příkazů (typ `TaskQueue<string>`).

3.2.4 Okno ovládání

Start a stop činnosti

Start a stop činnosti zajišťuje metoda `startStopButton_Click`. V případě startu provede inicializaci přijímací fronty a nastartování naslouchacího vlákna. Při

požadavku na Stop pak deinicializuje frontu a uzavře síťový proud i celé spojení. Rovněž při přímém zavírání okna programu se nejdříve zastaví TCP spojení, pokud ještě běží.

Zpracování příkazů

Pokud přijatý příkaz odpovídá očekávanému tvaru, což je ověřováno pomocí regulárních výrazů, tak získanou hodnotu zobrazíme v příslušném políčku (k tomu máme metodu `DisplayValue(string value, Label label)`). Případnou chybu můžeme zobrazit v logovacím boxu.

Jestliže máme pro danou veličinu nastaven automatický režim, tak přijatou hodnotu hned odešleme zpět do integrátoru. Režim je určován příslušnými `CheckBoxy` označenými `Auto`, jejichž změnou dochází k aktivaci resp. deaktivaci ovládacích tlačítek na přenášení a odesílání hodnot veličin i políček pro zápis hodnoty a kroku.

Při přijetí události zobrazíme zprávu v logovacím boxu a vyvoláme obsluhu tlačítka pro změnu stavu na základě obdržené události. Tu zajišťuje metoda `zmenaStavuUdalostiButton`, která v případě zaškrtnutého `CheckBoxu` pro automatický režim událostí zkontroluje aktivaci tlačítek událostí `Start`, `Start with BP` a `Stop`.

Odesílání nastavených hodnot

Manuálně zapsané hodnoty daných veličin se vydávají ven sadou příslušných odesílacích tlačítek, které obsluhuje metoda `obsluzOdeslatButton`, stejně jako tlačítka pro vyvolání některých `Událostí`. Pro jednoduché přenesení přijatých hodnot do políček k odeslání slouží sada příslušných přenášecích tlačítek obsluhovaná metodou `obsluzPrenestButton`. Další možností je plynulé zvětšování či zmenšování nastavených hodnot o předem zvolený krok, k čemuž slouží příslušná tlačítka označená + a −. Jejich obsluhu provádí metoda `obsluzKrok`, která zároveň kontroluje validitu hodnot a jejich případné zarovnání na určený počet cifer dle protokolu. Pro tuto funkcionalitu je možno do budoucna uvažovat o napojení na hardwarový ovladač.

3.3 Grafická vizualizace

Program `ChartPresenter` slouží pro grafickou vizualizaci naměřených hodnot a jejich záznam v čase. Z integrátoru jsou do něj periodicky posílány hodnoty veličin obsažené na monitoringu. Zde je pro každou z nich vedena samostatná datová řada, která je průběžně naplňována. Nasbíraná data můžeme z grafu také exportovat do XML.

3.3.1 Struktura programu

Úvodní formulář `GrafForm` (viz kapitola 3.3.3) opět sdružuje téměř veškerou funkčnost, jelikož i tato aplikace je prakticky jednoúčelová. Samostatná zůstala jen „servisní“ třída `TaskQueue` fungující stejně jako v programu Manuální ovládání (kapitola 3.2.3).

3.3.2 Komunikace

Komunikace programu probíhá přes TCP spojení, a to opět totožně jako v programu Manuální ovládání (v kapitole 3.2.2).

3.3.3 Okno grafu

Start a stop činnosti

Metoda `startStopButton_Click` zajišťuje v případě startu inicializaci fronty pro přijímání příkazů a započetí práce naslouchacího vlákna. Na požadavek stop reaguje opět ukončením práce fronty, uzavřením streamu i spojení. Totéž zkontroluje při zavírání okna.

Zpracování příkazů

Nejdříve se ověří správný tvar přijatého příkazu dle regulárních výrazů. Následně přidáme získanou hodnotu spolu s aktuálním časovým údajem do odpovídající datové řady. Pak se ještě provádí kontrola automatického posunutí grafu. Je nastavena tak, aby se vždy zobrazilo posledních 30 sekund na ose X (zbytkem se dá projíždět pomocí scrollbaru). Tuto kontrolu je možno uživatelsky vypnout odškrtnutím příslušného `CheckBoxu`).

Pokud byl přijat příkaz pro událost, tak se jeho zpracování (např. vymazání grafu) provede pouze v případě, kdy je zaškrtnut `CheckBox` pro automatické řízení grafu událostmi.

Vlastnosti grafu

Pro vykreslení grafu je využita komponenta `Chart` z prostředí WinForms aplikací v jazyce C# na platformě .NET. Graf se konstruuje hned po načtení formuláře a je v něm použita jedna grafová plocha. Na ose X je zobrazen časový údaj (`DateTime`) s popisky ve tvaru „HH:mm:ss“ a osovým intervalem 1 sekunda. Viditelná oblast je v základu nastavena na posledních 30 sekund, pro pohyb v celém rozsahu dat je zobrazen scrollbar. Osa Y zobrazuje číselné hodnoty a využívá možnosti automatického smršťování pro optimální využití prostoru všemi datovými řadami. Pod grafem je umístěna legenda.

Pro každou podporovanou veličinu, což je určeno slovníkem `prikazyId` obdobně jako u `Protokolů`, je pak vytvořena samostatná datová řada liniového typu s kulatými značkami. Značkám je doplněna bublinová nápověda zobrazující hodnoty daného bodu a řada je přiřazena do legendy. Všechny řady jsou zároveň sdruženy ve slovníku pro snadný přístup při zpracování příkazů. Graf je ještě možno uživatelsky vymazat, přičemž se všechny body jeho konstrukce provedou znovu od začátku.

Export dat

Graf umožňuje přímý export všech obsažených dat do formátu XML. Nejdříve pomocí `DataManipulatoru` z grafu vytvoříme `DataSet` s exportovanými hodnotami, kterému přiřadíme jméno na základě aktuálního data a času. Pak z něj získáme

XML řetězec, který následně načteme do XML dokumentu. Tento dokument už může uživatel uložit, k čemuž mu zobrazíme `SaveFileDialog`.

3.4 Simulační programy

Pro potřeby simulace reálného fungování programu a sledované komunikace i tehdy, když nejsou připojena žádná fyzická zařízení, byly vytvořeny doplňkové programy, které simulují chování jednotlivých přístrojů nebo programů, které se mohou v laboratoři vyskytovat. Jedná se konkrétně o `Behatko_model`, `BtlEcg_model`, `EKG_model`, `Kolo_model`, `Oxycon_model` a `Pulzak_model`.

3.4.1 Komunikace

Tyto programy, stejně jako fyzická zařízení, komunikují přes sériový port. Konkrétní název portu je v každém programu přímo zadán a odpovídá reálnému názvu portu používanému v zátěžové laboratoři. Zadaný port je příslušným programem otevírán hned po spuštění. V každém programu je implementován příslušný protokol obdobně jako v hlavním integrátoru. Jinak programy obsahují různé druhy boxů pro zobrazování přijatých příkazů a zapisování příkazů k odeslání, aby mohly sloužit k testování správné funkčnosti komunikace.

4. Ukázka použití

4.1 Integrační program CPX DI

Průběh měření s využitím integračního programu CPX DI se skládá z těchto kroků:

- Prvně si nakonfigurujeme potřebné veličiny a události, dále zařízení, se kterými chceme pracovat, a poté scénáře měření, ve kterých určíme zařazeným zařízením jejich vstupní a výstupní veličiny. Jako základ můžeme využít přednastavené modelové zapojení.
- Pak už můžeme vybraný scénář spustit. Tím navážeme spojení se zařízeními, přičemž případné chyby jsou při kontrole spojení ihned oznámeny). Pro správně připojená zařízení se zobrazí jejich ovládací okna (pokud to máme zaškrtnuto).
- V průběhu měření se v hlavním okně ukazují přijaté hodnoty veličin, které se zapisují do přehledné tabulky (tzv. monitoring). Záznam kompletní komunikace s každým zařízením se zobrazuje v jeho okně, odkud mu také můžeme posílat ovládací příkazy.
- Hodnoty veličin a událostí lze na monitoring zadávat i manuálně, ale je potřeba dodržet pravidlo, že jejich zdrojem musí být zařízení, které je má nastaveno v aktuálním scénáři na výstupu.
- Dalším tlačítkem lze za běhu scénáře znovu připojit vybrané zařízení, např. pokud bylo odpojeno, nebo i dodatečně připojit nové zařízení (ale musí být v konfiguraci scénáře zařazeno).

4.2 Manuální ovládání

Jak mohou konkrétně vypadat jednotlivé kroky postupu při použití programu si můžeme ukázat na příkladu manuálního ovládání bicyklového ergometru:

- Spustíme síťové spojení s požadovanou IP adresou a číslem portu. Ovládací prvky jsou však zatím neaktivní.
- Startujeme měření dle daného scénáře v integrátoru, tím navážeme spojení s aplikací manuálního ovládání. Teprve v tomto okamžiku se její ovládací prvky zaktivují.
- Výchozí situace vypadá tak, že do aplikace z integrátoru přichází hodnoty požadovaného výkonu typu Auto, které předtím vypočetl vyhodnocovací program Oxycon a zobrazují se v horní části.
- My potřebujeme příchozí hodnotu typu Auto upravit, proto si ji tlačítkem `Přenést` zapíšeme do vedle umístěného textového pole, které je určeno pro požadovanou hodnotu typu Manual.

- Nyní chceme zapsané číslo plynule zvětšovat, proto budeme klikat na tlačítko +. Nové hodnoty se zároveň budou odesílat zpět do integrátoru, který už je automaticky předává ergometru. Analogicky funguje i snižování hodnot tlačítkem −.
- Kromě toho si můžeme zvolit režim Auto, ve kterém požadované hodnoty nenastavujeme ručně, ale aplikace ihned po přijetí aktuální hodnotu zasílá zpět do integrátoru.
- V dolní části okna máme možnost ještě vyvolávat některé události. U nich je při zaškrtnutí režimu Auto uchováván stav, takže např. stisknutím `Startu` je jeho tlačítko deaktivováno a teprve po kliknutí na `Stop` se stane znovu aktivním (a naopak).

4.3 Grafická vizualizace

Když si chceme zobrazovat naměřené hodnoty veličin do grafu, použijeme tento postup:

- Spustíme síťové spojení s požadovanou IP adresou a číslem portu.
- Odstartujeme měření dle scénáře v integrátoru, tím navážeme spojení s grafickou vizualizací.
- Pokud máme zaškrtnutý režim Auto, tak se zobrazování veličin v grafu řídí přijatými událostmi (start, stop).
- Jinak se hodnoty vykreslují ihned po obdržení. Pokud to máme povoleno, tak se graf automaticky posunuje, aby byla vždy vidět data za posledních 30 sekund.
- Dále máme možnost graf exportovat do XML souboru nebo vymazat.

4.4 Simulační programy

Tyto doplňkové programy slouží k simulaci chování fyzických zařízení v laboratoři. Postup použití je tedy následující:

- Před spuštěním zkontrolujeme, zda je aktivní sériový port, který je nastaven v programu. Následně program spustíme.
- Nastartujeme měření dle příslušného scénáře v integrátoru, tím navážeme spojení se simulačním programem.
- Nyní už simulační program přijímá a odesílá veličiny dle nastavení ve scénáři stejně jako fyzické zařízení.

Závěr

V této práci jsme se zabývali vytvořením integračního nástroje, který řídí komunikaci a ovládá měřicí zařízení používaná při vyšetřování pacientů v zátěžové laboratoři na Klinice rehabilitace a tělovýchovného lékařství UK 2.LF a Fakultní nemocnice v Motole.

Ve spolupráci se zadavatelem projektu MUDr. Kryštofem Slabým, který se věnoval průběžnému testování vyvíjených programů při nasazení v reálných podmínkách zátěžové laboratoře a na základě získaných zkušeností dále přinášel nové nápady a funkční požadavky, se podařilo vyvinout skutečně funkční produkt. Skládá se ze tří samostatných programů, které se vzájemně funkčně doplňují a jako celek přinášejí podstatné zlepšení a rozšíření možností při realizaci měření na přístrojích v laboratoři.

Podstatou zlepšení je především možnost současného zapojení více zařízení, u nichž můžeme dle aktuálních potřeb podrobně nastavovat, jaké informace od nich budeme přijímat i co se jim naopak bude odesílat. K zajištění funkcionalit manuálního ovládání některých zařízení a grafické vizualizace naměřených dat byly vytvořeny samostatné aplikace, aby bylo možno s integrátorem komunikovat i vzdáleně (přes síťové spojení).

Mezi problémy, kterým jsme při vývoji museli věnovat pozornost, patřil už samotný návrh základní struktury objektů v programu, aby byly pokryty všechny funkční požadavky a zachována snadná rozšiřitelnost. Dále bylo potřeba vyřešit způsob uchovávání konfiguračních dat, aby byly zachovány vztahy mezi uloženými objekty a zároveň byl formát konfiguračního souboru i uživatelsky čitelný a editovatelný.

Možnosti vylepšování a rozšiřování programu můžeme v zásadě rozdělit na dvě části. První z nich je možnost rozšíření programu o implementace nových protokolů k zařízením a nových způsobů komunikace. Proto byl u příslušných tříd kladen velký důraz na modulární návrh, třídy jsou tedy dle návrhového vzoru *Factory method pattern* provedeny jako abstraktní a je možno jednoduše implementovat jejich potomky.

Druhou skupinu pak tvoří náměty na nové funkce samotného programu. Ty se jistě objeví i během jeho praktického používání. Z těch, co již byly zadavatelem zmíněny, uveďme zavedení funkce časomíry (stopky) nebo přidání zátěžových profilů. Ty by umožnily automatizovanější řízení průběhu měření, což zahrnuje např. nastavení požadovaných hodnot veličin v čase či naplánování jednotlivých událostí. U programu pro manuální ovládání je zase uvažováno o napojení na hardwarový ovladač.

Seznam použité literatury

[1] Máček, Miloš a Radvanský, Jiří. *Fyziologie a klinické aspekty pohybové aktivity*. 1. vydání. Praha: Galén, 2011. ISBN 978-80-7262-695-3.

[2] *Ergoline Operation Manual*. [online]. [cit. 2015-05-03]. Dostupné z: http://www.ergoline.com/en_GB/produkte/sitzergometer/ergoselect100

[3] *Nonin Operator's Manual*. [online]. [cit. 2015-05-03]. Dostupné z: http://www.nonin.com/Model7500

[4] *Lode Valiant Plus with Control Unit*. [online]. [cit. 2015-05-03]. Dostupné z: http://www.lode.nl/en/product/valiant-plus-with-control-unit/192

[5] Bishop, Judith. *C# 3.0 design patterns*. 1. vydání. Sebastopol, CA: O'Reilly, 2008. ISBN 0-596-52773-X.

[6] Albahari, Joseph a Albahari, Ben. *C# 5.0 in a Nutshell*. 5. vydání. Sebastopol, CA: O'Reilly, 2012. ISBN 978-1-449-32010-2.

[7] *Doxygen*. [online]. [cit. 2015-05-03]. Dostupné z: http://www.stack.nl/~dimitri/doxygen/index.html

Seznam obrázků

A. Implementované protokoly

V této příloze popisujeme protokoly, které jsme již do programu implementovali, včetně přehledu jejich příkazů a požadavků na nastavení komunikace.

Nonin

Podobu tohoto protokolu stanovila společnost Nonin pro svůj pulzní oxymetr. Textový formát zahrnuje ASCII hlavičku obsahující číslo přístoje, datum a čas. Pro následné periodické posílání datových zpráv protokol [3] definuje jediný příkaz ve tvaru:

- SPO2=XXX HR=YYY

To odpovídá veličinám saturace a tepová frekvence. Každý řádek je definován pomocí CR/LF. Pro komunikaci po sériovém portu jsou pak určeny vlastnosti uvedené v tabulce A.1.

Baud rate	9600
Format	ASCII
Start bit	1
Data bits	8
Stop bits	2
Parity	None
Handshake	None

Tabulka A.1: Přehled vlastností portu pro Nonin

Upravenou podobu využívá ještě zařízení BTL ECG. Liší se tím, že v příkazu je obsažena pouze část pro tepovou frekvenci (HR).

ManualControl

Tento protokol jsme si definovali sami pro potřeby komunikace se samostatným programem Manuální ovládání. Předpokládáme u něj komunikaci přes TCP spojení s oddělovačem CR, kódováním ASCII a velikostí bufferu 32. Tvary příkazů pro zpracovávané veličiny jsme převzali z existujících protokolů, tedy pro požadovaný výkon je stejný jako v Ergoline a pro rychlost i sklon stejný jako v T2000. Kromě toho máme obsaženy ještě příkazy pro události (P – start_with_BP, s – start, F – stop a R – remark).

ChartPresent

Analogicky k ManualControlu jsme definovali i protokol pro komunikaci s aplikací Grafická vizualizace. Požadavky na TCP spojení má tedy stejné. Tvary příkazů opět vycházejí z již existujících, shodují se s těmi v Ergoline. Pro další veličiny neobsažené v Ergoline byly počáteční písmena zvoleny takto: w – požadovaný výkon Manual, L – saturace, S – požadovaná rychlost Auto, s – požadovaná rychlost Manual, V – aktuální rychlost, G – požadovaný sklon Auto, g – požadovaný sklon Manual a p – aktuální sklon. Též jsou zde příkazy událostí: eStart, eStartWithBP a eStop.

Ergoline

Tento protokol vytvořila společnost Ergoline pro své bicyklové ergometry. Kromě toho jej používá i vyhodnocovací program Oxycon. Předpokládá komunikaci přes sériový port, pro který definuje vlastnosti dle tabulky A.2. Přehled příkazů protokolu Ergoline [2] uvádíme v tabulce A.3.

Baud rate	4800
Format	ASCII
Start bit	1
Data bits	8
Stop bits	1
Parity	None
Handshake	None

Tabulka A.2: Přehled vlastností portu pro Ergoline

Command from PC / ECG	Response from ergoselect	Description
Wxxx <CR>	---	Sets a load target of xxx Watts.
S <CR>	---	Start of blood pressure measurement at rest (only with module NIBP) and subsequent start of ergometry with initial load Po.
s <CR>	---	Start of ergometer with the initial load Po without initial blood pressure measurement.
B <CR>	B xxx <CR>	Query for current load. Output of current load in W.
D <CR>	n xxx <CR>	Query for current rpm value. Output of current rpm value in 1/min.
H <CR>	H xxx <CR>	Query for current heart rate. Output of current heart rate in 1/min.
O <CR>	O xxx <CR>	Query for current systolic value. Output of current systolic value in mmHg.
U <CR>	U xxx <CR>	Query for current diastolic value. Output of current diastolic value in mmHg.
F <CR>	- - -	End of ergometry process.
I <CR>	erxxxP10Vyyy <CR>	Identification string of the ergometer. up to software version 2.9: „er900P10VV243“ from software version 3.0: adjustable as “er900” or “er800”
K <CR>	- - -	Beeper signal of duration 300ms is generated.

Tabulka A.3: Přehled příkazů Ergoline

T2000

Protokol T2000 vyvinula společnost Lode pro svůj běžecký pás [4]. Z důvodu zajištění bezpečnosti na pásu obsahuje důležitou podmínku, dle které je potřeba alespoň každé 2 sekundy poslat do zařízení nějaký ovládací příkaz, jinak se pás bezpečně zastaví (protože došlo ke ztrátě komunikace). Požadavky na vlastnosti sériového portu ukazuje tabulka A.4 a přehled ovládacích příkazů je vypsán v tabulce A.5.

Baud rate	9600
Start bit	1
Data bits	8
Stop bits	1
Parity	None
Handshake	None

Tabulka A.4: Přehled vlastností portu pro T2000

A<cr>	nastaví jednotky rychlosti na MPH
B<cr>	Start pásu
C<cr>	vrátí číslo verze software
D<cr>	inkrementální snižování sklonu o 0,1 %
E<cr>E<cr>	způsobí ESTOP (emergency – nouzové zastavení)
G0000<cr>	nastavení sklonu (číslo v setinách % a končí nulou, max. 2550)
H<cr>	High Speed komunikace
I<cr>	inkrementální zvyšování rychlosti o 0,1 MPH
J<cr>	Junk – zpráva testového módu
K<cr>	nastaví jednotky rychlosti na km/h
L<cr>	inkrementální snižování rychlosti o 0,1 MPH
M007007<cr>	nastaví míru zrychlení a zpomalení (pouze během kalibrace) – 2 čísla v rozsahu 001..010 (desetiny MPH/s)
N<cr>	normální Stop pásu, zpomaluje dle nastavené míry zpomalení
P<cr>	pro více informací
R<cr>R<cr>	soft reset
S0000<cr>	nastavení rychlosti (v setinách MPH, končí nulou, max. 2550)
U<cr>	inkrementální zvyšování sklonu o 0,1 %
W<cr>	What's up: vrátí aktuální rychlost, sklon a jednotky ve tvaru "S0000<cr>G0000<cr>A<cr>"
W<cr>	Low speed character output
Y<cr>Y<cr>	přepíná mezi normálním a debugovacím módem
u<cr>	(normálně neužíváno): vyvolá EEUPDATE
c<cr>c<cr>	start self-cal (pouze když neběží)

Tabulka A.5: Přehled příkazů protokolu T2000

B. Obsah CD

Obsah přiloženého CD je rozdělen do následujících adresářů:

- `zdroj` – zdrojové kódy všech vytvořených programů
- `dokumentace` – dokumentační stránky vygenerované ze zdrojových kódů nástrojem Doxygen [7]
- `instalace` – instalační soubory programů
- `text` – elektronická podoba textu práce ve formátu PDF