Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Informační systém pro sběr a analýzu dat v oblasti radiačních měření

Bc. Tomáš Rybáček

Vedoucí práce: Ing. Tomáš Vaňát

11. ledna 2016

Poděkování

Chtěl bych poděkovat všem pracovníkům ÚJF Řež, kteří mi byli nápomocni při tvorbě této diplomové práce. Především bych chtěl poděkovat RNDr. Jozefu Ferenceiovi CSc., RNDr. Filipu Křížkovi Ph.D. a mému vedoucímu diplomové práce Ing. Tomášovi Vaňátovi.

Práce byla součástí širšího projektu zaměřeného na radiační odolnost silikonových pixelových čipů a komponent pro inovaci vnitřního dráhového systému detektoru ALICE v CERNu v Ženevě. Měření se realizuje na cyklotronu ÚJF AV ČR v Řeži v rámci projektu CANAM, podpořeného grantem MŠMT číslo LM2011019.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. ledna 2016 .....

Odkaz na tuto práci

Rybáček, Tomáš. *Informační systém pro sběr a analýzu dat v oblasti radiačních měření.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato práce se zabývá analýzou procesů, které probíhají v Ústavu jaderné fyziky Akademie věd České republiky. Důraz je kladen především na oblast radiačních měření. Po této analýze se práce věnuje tvorbě informačního systému, který tyto procesy optimalizuje a zjednodušuje. V závěru je shrnuto testování softwaru a manažersko-ekonomický přínos aplikace.

Klíčová slova Knihovna MFC, framework ROOT, DEMO modelování, radiační měření

Abstract

This thesis deals with analysis of processes, which are taking place in Nuclear Physics Institute of the Czech Academy of Sciences. Main focus lies on radiation measurements. After these analyses, thesis deals with creation of Information system, which optimizes identified processes. Software testing and economical benefits are summarized in conclusion.

Keywords MFC Library, framework ROOT, DEMO modelling, radiation measurements

Obsah

Seznam obrázků

Seznam tabulek

Úvod

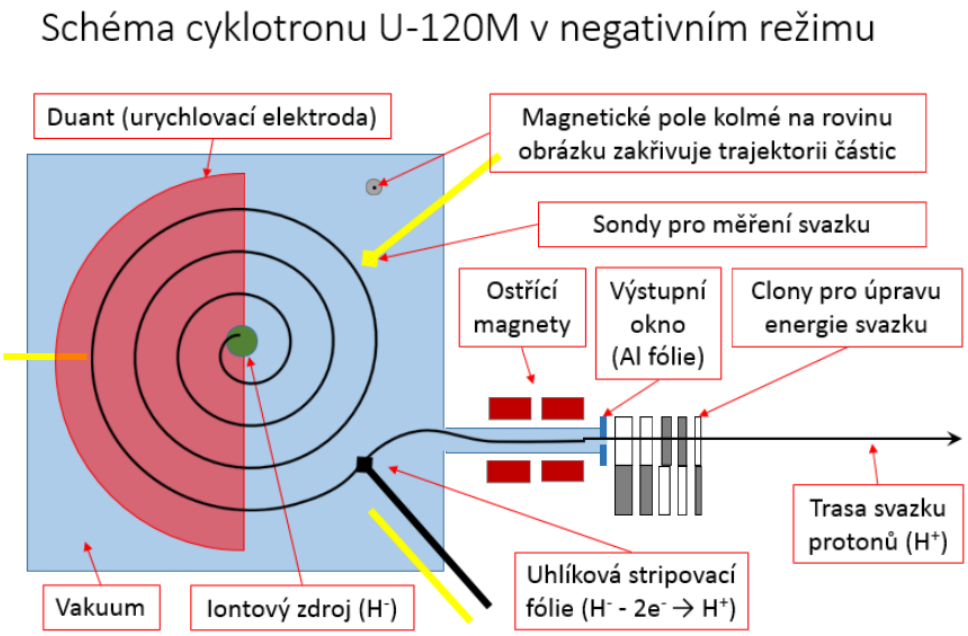
Tato diplomová práce si klade za cíl podpořit radiační měření probíhající v Ústavu jaderné fyziky Akademie věd ČR. V současné době je měření spravováno velmi jednoduchým softwarem, který v podstatě pouze obsluhuje sériovou linku, zapisuje na ni příkazy periferiím a čte z ní vrácené hodnoty. Neumožňuje jakékoliv logování dat ani správu uživatelů. Program dokonce často zhavaruje nebo má jiné kritické nedostatky.

V první části této práce se zabývám analýzou současného řešení, která je doprovázena modelováním procesů, jež v ÚJF probíhají, metodou DEMO. Dále je pak detailně popsána implementace informačního systému, společně s jednoduchými návody, jak aplikace dále spravovat a rozšiřovat. Tato část tedy bude zastávat i roli dokumentace aplikace. V závěru hodlám provést a zhodnotit akceptační testy a systémové nároky softwaru. V poslední kapitole je shrnut manažersko-ekonomický přínos diplomové práce.

O Ústavu jaderné fyziky AV ČR

Ústav jaderné fyziky AV ČR sídlí společně s akciovou společností UJV ve výzkumném areálu v Řeži. Provádí zde jak experimentální, tak teoretické výzkumy, v nichž se zaměřuje především na vlastnosti jaderné hmoty ve srážkách těžkých iontů, teorii jader, hyperjádra nebo interakce elementárních částic s jádry. [1]

Velice důležitou částí výzkumu jsou i experimentální pokusy. Ty umožňuje provádět mnoho zařízení, které se v areálu nacházejí. Pro tuto diplomovou práci je nejdůležitější cyklotron. Cyklotron slouží k urychlování těžkých nabitých částic pomocí vysokofrekvenčního elektrického pole. Tato práce si v žádném případě neklade za cíl ovládat celý cyklotron, k tomu je potřeba tým vyškolených pracovníků a velín o velikosti větší místnosti. Na cyklotronu se ovšem měří pomocí spousty periferií, která aplikace vyvinutá v rámci této práce bude spravovat. Samotný cyklotron se nemůže nacházet ve stejné míst-



Obrázek 0.1: Schéma cyklotronu používaného v ÚJF Řež

nosti jako velín, protože míra radiace v jeho okolí je životu nebezpečná. V celé budově cyklotronu se tak musí dodržovat poměrně přísné hygienické předpisy a samotný cyklotron je pak izolován zhruba tři metry širokou zdí.

Typickým pokusem, který v ÚJF může probíhat, je měření maximální životnosti programovatelného hradlového pole (FPGA), jež je cyklotronem ozařováno postupně zvětšující se dávkou.

Periferie ovládané během radiačních měření

Pohybový mechanismus

Protože s ozařovaným vzorkem se musí často manipulovat, je potřeba vzdáleně ovládat pohybový mechanismus, který se vzorkem hýbe. Aktuálně se v Řeži využívá „Precision position Control System MCL“ (dále jen „MCL“). Krokový motor MCL dokáže s ozařovaným vzorkem pohybovat s přesností na 0,0001 mm. Motor je schopný se otáčet rychlostí až 9 otáček za vteřinu, přičemž nevydává žádný velký hluk. Mechanismus využívá techniku lineární interpolace, takže po zadání nové souřadnice dorazí na cílovou pozici všechny (přesněji řečeno oba, MCL v řeži se hýbe pouze v rovině) směry zároveň. Pozicovací systém podporuje sériovou linku RS-232, takže se dá ovládat programově, případně je dostupný i joystick.

Unidos E PTW – Universal dosemeter

Pokud je ozařovaný vzorek správně umístěn ve svazku cyklotronu, je potřeba měřit přijatou dávku. K tomuto účelu v Řeži slouží zařízení „Unidos E PTW – Universal Dosemeter“ (dále jen „Unidos“). K Unidosu je možné připojit ionizační komůrku nebo jiný druh sondy. Unidos je obecně velmi komplexní zařízení, čili kromě teoretické fyziky můžeme jeho uplatnění najít v mnoha dalších oborech, především v biologii a lékařství.

Pro diplomovou práci je podstatné, že Unidos je schopný měřit elektrický proud, vyvolaný svazkem ionizujících částic, prolétávajícím ionizační komůrkou. Na základě parametrů konkrétní ionizační komůrky lze tento proud přepočítat zpět na intenzitu svazku částic. Má standardní ovládací panel, z nějž je možné naměřené hodnoty číst a nastavit mnoho režimů měření. Důležité je, že stejně jako MCL, Unidos umožňuje komunikaci po RS-232.

Po sériové lince se tak dá nastavit mnoho parametrů, například jeden ze tří rozsahů, typ měření (dávka/náboj) nebo mód (zadržení aktuální hodnoty, měření za předchozí časový úsek atp.). Především je ale možné číst naměřené hodnoty, společně s časovou značkou a jinými atributy.

Kapitola 1

Cíl práce

Cílem diplomové práce je analýza procesu radiačních měření, na jejímž základě vznikne informační systém, který nejenže bude ovládat poskytnuté periferie, ale umožní celkovou správu radiačního měření. Základní předpoklad úspěchu je, že nově implementovaný systém překoná existující z hlediska funkčnosti a rozšíří ho o nové podstatné funkce. Po jeho implementaci je nutné otestovat, jestli splňuje očekávané systémové nároky. V závěru práce je zároveň důležité shrnout manažersko-ekonomický přínos implementované aplikace.

Podstatné je, aby systém prošel akceptačními testy, které realizují samotní pracovníci ÚJF Řež.

Kapitola 2

Analýza a návrh

2.1 Analýza stávajícího řešení

Aktuální řešení používané v ÚJF je založené pouze na frameworku ROOT. Program není nijak kompilovaný (v podstatě se jedná o posloupnost ROOTovských funkcí) a k jeho spuštění je potřeba nějaký C/C++ interpret, například CINT. Celý program je navíc napsán v jediném souboru, a ačkoliv je členěn do funkcí, chybí mu jakýkoliv objektový návrh. Z těchto důvodů je program velice obtížný na správu a rozšiřitelnost.

Největším problémem je ale časová náročnost programu. Jelikož v samotném ROOTu nelze ovládat sériovou linku, bylo nutné z programu volat externí pythonovské skripty, do nichž se vstupní parametry vkládaly pomocí vytvářených souborů. Další velké zdržení spočívalo v aktivním čekání, které program v hojné míře využíval.

Na druhou stranu z hlediska uživatele se jedná o snadno uchopitelné řešení s jasně daným work–flow. Celý program je v podstatě jedno okno, které je logicky členěno na ovládání posuvného mechanismu, měření proudu a hledání svazku. Z počátku je nutné vybrat, na jakých portech běží patřičné periferie – bohužel program nenabízí seznam připojených zařízení, takže je v podstatě nutné zkoušet jeden port po druhém. Po tomto kroku je nutné ještě periferie spustit, čímž se automaticky spustí jejich kalibrace. Ovládání je poměrně strohé, ovšem splňuje minimální požadavky – pohyb MCL, spuštění měření proudu a celé rozhraní pro nastavení hodnot pro automatické hledání svazku. V pravém dolním rohu pak program uživatelsky přívětivě vypisuje svůj stav a stav periferií.

Bohužel celkové hodnocení z pohledu uživatele je značně degradováno tím, že program velice často „padá“, protože samotný ROOT není vhodný pro komplexní softwarové řešení středního a velkého rozsahu, navíc často nastávají problémy v komunikaci s externími skripty. Dalším problémem jsou aktivní čekání, kdy program přestane třeba i na desítky sekund odpovídat a není s ním tedy možné jakkoliv manipulovat.

K tomu, aby diplomová práce byla úspěšná, je mimo jiné nutné, aby výsledný informační systém tyto chyby odstraňoval.

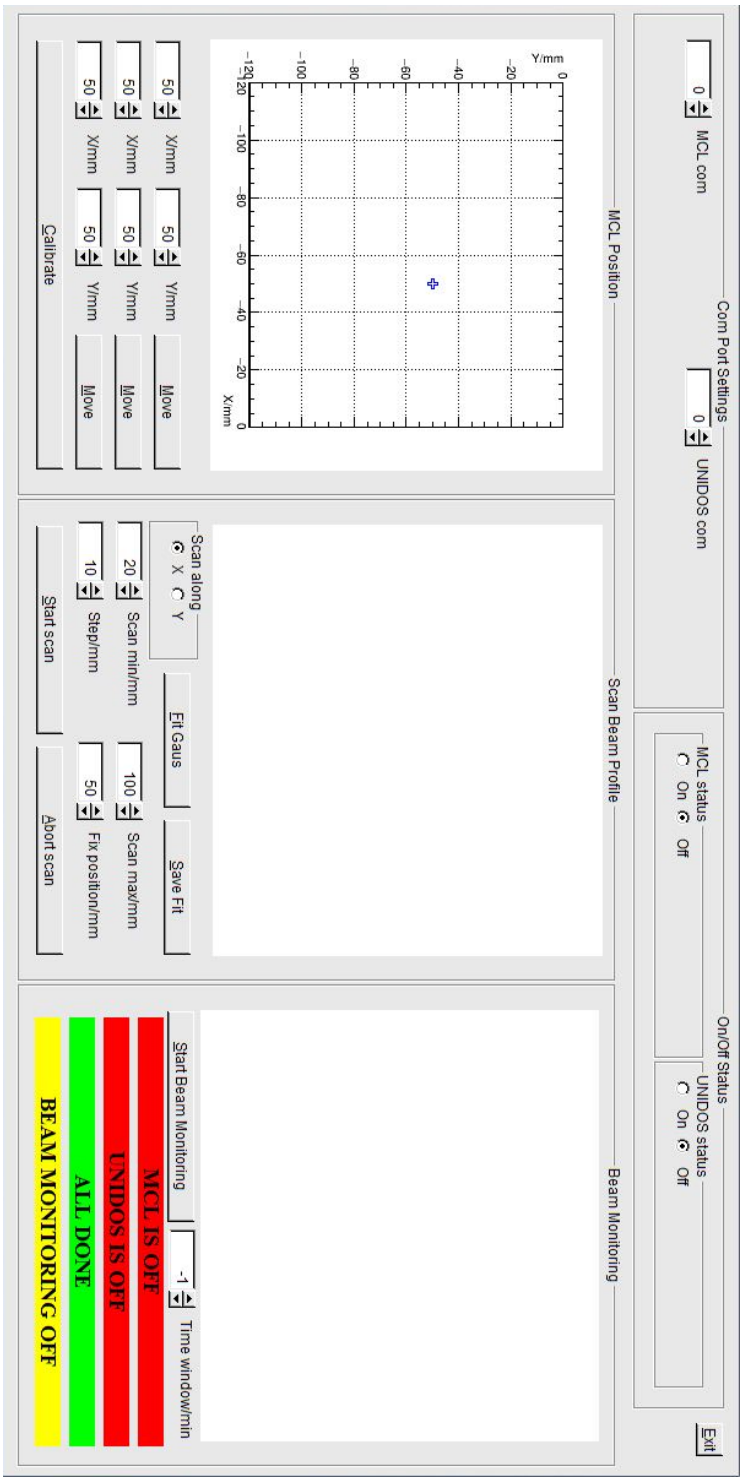
2.2 Požadavky na aplikaci

2.2.1 Funkční požadavky

- F1 Správa uživatelů – aplikace bude umožňovat přihlášení a přidání uživatelů. Aplikace musí rozlišovat minimálně tři typy uživatelů – správce, přihlášený uživatel a nepřihlášený uživatel.
- F2 Výběr ovladače (driveru) pro ovládání periferie.
- F3 Správa pohybového mechanismu
  - F3.1 Kalibrace
  - F3.2 Pohyb pohybového mechanismu s rozlišením na mikrometry
  - F3.3 Načtení aktuální polohy mechanismu
  - F3.4 Možnost vložení polohy ozařovaného vzorku a středu svazku
  - F3.5 Vizualizace
  - F3.6 Logování dat do souboru
- F4 Správa měření proudu
  - F4.1 Možnost spustit a vypnout průběžné měření
  - F4.2 Zakreslování naměřených hodnot do grafu
  - F4.3 Logování dat do databáze nebo souboru
  - F4.4 Možnost automatického nalezení středu svazku

2.2.2 Nefunkční požadavky

- NF1 Programovací jazyk Python nebo C++
- NF2 Software bude koncipován modulárně, předpokládá se další rozšiřování
- NF3 Aplikace bude funkční minimálně pod OS Windows
- NF4 Aplikace bude nezávislá na databázové platformě
- NF5 Vizualizace a matematické výpočty budou realizovány pomocí frameworku ROOT
- NF6 Bude se jednat o standalone aplikaci



Obrázek 2.1: Snímek aktuálního řešení, které je v ÚJF používáno.

2.3 DEMO modelování

Zkratka DEMO má význam „Design and Engineering Method for Organisations“. Tato metoda se v poslední době velice dynamicky rozvíjí, protože má oporu jak ve vědeckých teoriích, tak byl dokázán i její praktický přínos v činnostech organizace. [2] Bohužel stále v ní neexistuje jednotná česká terminologie. Bylo rovněž dokázáno, že DEMO výrazně snižuje složitost modelů (až o 90 % běžné dokumentace) a také umožňuje odhalit tacitně prováděné koordinační akty. [3]

Základními stavebními bloky DEMO modelování jsou diagramy OCD (Organization Construction Diagram)a PSD (Process Structure Diagram). Pomocí těchto diagramů jsem se tedy rozhodl modelovat procesy probíhající v ÚJF Řež.

2.3.1 OCD

V diagramu OCD jde o zachycení rolí a ontologických transakcí mezi nimi. Tento diagram je zpravidla doprovázen TPT – Transaction Product Table, který k jednotlivým transakcím udává jejich produkty. OCD se zpravidla skládá z:

- Rolí interních uživatelů
- Rolí externích uživatelů
- Interních transakcí a tzv. „hraničních“ transakcí
- Iniciačních komunikačních linek
- Linek vedoucích od aktéra k jeho vykonavateli

V modelu se také setkáme s pojmy „sebeaktivační“ a „kompozitní“ role. Sebeaktivační role je taková, kdy iniciátor a vykonavatel transakce je tentýž aktér. Kompozitní role se pak skládá z několika dílčích atomických aktérů. V modelu níže jsem sebeaktivační role označil smyčkou v levém horním rohu, kompozitní aktéři jsou poté vyznačeni šedě. [4]

2.3.2 PSD

V diagramu PSD najdeme rovněž role a transakce mezi nimi. Výhodou tohoto modelu je fakt, že v něm můžeme sledovat konkrétní tok informací, které jsou v DEMO standardizované. Konkrétně se jedná o tyto možnosti komunikace:

- Request (Rq) – žádost o spuštění transakce
- Promise (Pm) – příslib dodání produktu transakce
- Decline (Dc) – zamítnutí dodání produktu

- State (St) – předání produktu transakce
- Accept (Ac) – přijetí hotového produktu
- Reject (Rj) – nepřijetí hotového produktu

Jé též možné některé již učiněné kroky odvolat, v takovém případě provádíme tzv. „Revoke“ – například Revoke Request.

Diagram PSD se používá především k definici všech možných kroků, které mohou v organizaci nastat.

2.4 Návrh technologií pro implementaci

Jelikož je nutné, aby aplikace fungovala pod OS Windows, rozhodl jsem se k implementaci využít technologie Microsoftu a seznámit se s nimi. Velice lákavé bylo využití C#, ale nakonec jsem se rozhodl využít technologii MFC, která umožňuje velice jednoduše vytvářet a spravovat standardní Windows ovládací prvky (okna, tlačítka, input boxy, select boxy aj.).

V programu bylo též nutné data vizualizovat, to znamená vytvářet různé grafické elementy a různá grafická primitiva, z nichž následně vznikají grafy. Dále pak bylo nutné využívat netriviálních matematických operací, jako prokládání bodů funkcí. Oba tyto problémy pomáhá řešit framework ROOT, a protože s ním má většina pracovníků v ÚJF Řež zkušenosti, bylo téměř nutné ho použít v kombinaci s výše zmíněnou knihovnou MFC. Propojení těchto technologií nebylo úplně triviální, proto se mu podrobně věnuji v kapitole 5.

Nečekaně komplikovaným problémem se ukázalo nalezení softwaru pro efektivní kreslení a správu DEMO diagramů, které jsou součástí zadání diplomové práce. Jelikož tato metoda není příliš rozšířená, najít vhodné nástroje je poměrně obtížné. Nakonec byl vybrán software Demoworld od společnosti Formetis a zásuvný modul pro vývojové prostředí IDE. Obě tyto řešení jsou zdarma, ačkoliv u Demoworldu je nutné zaplatit za plnohodnotnou licenci kredity, dají se tyto kredity získat zdarma, jelikož software stále běží ve zkušebním režimu.

2.4.1 Knihovna MFC

Microsoft Foundation Class Library (MFC) je knihovna funkcí, která obaluje část Windows API do tříd C++. Ty jsou definovány pro většinu tzv. „handlů“ – abstraktních odkazů na zdroje (obrázky, okna atp.) – a pro předdefinovaná okna a většinu standardních ovládacích prvků. [5]

První verze knihovny MFC byla představena v roce 1992 společně s Microsoftím kompilátorem C/C++, nicméně jednalo se o pouze velmi úzkou množinu tříd obalující Windows API. Je ovšem nutné si uvědomit, že v této době objektový návrh a C++ byly teprve na počátku svého vzestupu a začínaly pomalu vytlačovat standardní jazyk C. Je zajímavé, že již z této doby se

do současných verzí MFC dostal zajímavý rozmar, a sice prefixování většiny funkcí předponou „Afx“, ačkoliv postrádá význam – v době raného vývoje MFC se totiž knihovna označovala „Application Framework Extensions“, název MFC byl přijat až příliš pozdě. [6] Diplomová práce byla implementována ve verzi 14.0.23026.0 z července 2015, ovšem existuje již i verze novější.

Jako příklad můžeme uvést třídy CWinApp a CWnd. Instancí třídy CWinApp získáváme běžící aplikaci, v níž pak můžeme vytvářet třídy CWnd. Ty mají mnoho členských proměnných a metod, pomocí nichž můžeme spravovat okna. Vše se tedy odehrává pomocí C++ tříd a přímé volání Windows API je nutné velice výjimečně.

Společně s Microsoft Visual Studio 2008 byl distribuován tzv. MFC Feature Pack, který rozšiřuje existující třídy, ty bývají nově označovány s postfixem „Ex“ – například tedy CWinAppEx. Toto rozšíření umožňuje implementaci moderního uživatelského rozhraní, jako je například Office Fluent UI. Toto rozšíření ovšem pro diplomovou práci není podstatné. [7]

2.4.1.1 Výhody knihovny MFC

- Velice jednoduchá správa GUI
- Dokáže zpracovávat Windows zprávy a tím komunikovat i s ostatními programy
- Jednoduchá přenosnost na jiné operační systémy (existuje i Unixová verze MFC) a architektury.
- Zjednodušuje práce s databází pomocí Data Access Objects (DAO) a Open Database Connectivity (ODBC). Pomocí těchto technologií je možné získat přístup do databáze nezávisle na programovacím jazyku nebo operačním systému.

2.4.1.2 Nevýhody knihovny MFC

- Programy vytvořené pomocí MFC potřebují ke správné činnosti .dll knihovnu, kterou je nutné buďto staticky nalinkovat nebo distribuovat společně s programem. Každopádně je pak výsledná aplikace poměrně veliká.
- MFC obsahuje veliké množství datových typů, mezi kterými je často potřeba netriviálně konvertovat.
- Aplikace naprogramované v MFC jsou pomalejší než obdobné aplikace naprogramované v jiném frameworku.

2.4.2 Framework ROOT

Framework ROOT se zabývá především pokročilým zpracováním dat. Byl vyvinut v CERNu, s nímž ÚJF úzce spolupracuje. Aktuálně ho celosvětově používají tisíce fyziků, protože umožňuje mimo jiné: [8]

- Uložení dat – je možné uložit data a jakýkoliv C++ objekt v komprimované binární formě. Tato forma je tzv. self–descriptive (samo–popisná), takže soubor v sám sobě uchovává informace o struktuře dat. Čili i pokud zdrojová data popisující soubor nejsou dostupná, uložená data budou stále bez problémů čitelná. K interní reprezentaci je použit n-ární strom, takže prohledávání souborů je velice rychlé i pro velké objemy dat.

- Snadný přístup k datům – data uložená v jednom nebo několika ROOT souborech jsou dostupná z PC, z webu a z mnoha jiných systémů. Stromy použité k reprezentaci dat se rozkládají po několika souborech a mohou být zřetězeny a spravovány jako unikátní objekt, což umožňuje iterovat přes velké množství dat.

- Získávání dat – ROOT obsahuje velice sofistikované nástroje pro matematickou analýzu a statistiku. Dále je též možné data generovat, například funkce nebo náhodné veličiny s libovolným rozdělením. Toto byl hlavní důvod, proč jsem ROOT zvolil do své diplomové práce. Všechny tyto nástroje se dají používat v paralelním režimu, ovšem to většinou nebylo nutné a většinu problémů s rychlostí jsem vyřešil způsoby, které nabízelo MFC.

- Zveřejnění a vizualizaci výsledků – výsledky výpočtů a měření mohou být zobrazeny v histogramech a grafech. Většina těchto elementů obsahuje vlastní menu, v němž si uživatel v run–time může nastavit mnoho parametrů (rozsah od – do atp.). Zároveň ROOT umožňuje výsledky snadno převádět do PDF.

- Interpretovaný režim nebo kompilování vlastního projektu – může být použit C++ interpret pro interaktivní sezení s vlastními makry, ale je též možné celý program zkompilovat. V obou režimech je možné vytvářet GUI, ovšem i pro tento účel bylo vhodnější použít MFC, které je k tomu ostatně navrženo.

- Použít ROOT s jinými jazyky – framework obsahuje nástroje, jak ho spojit s jinými jazyky, například Python, R nebo Mathematica. Jak jsem ovšem v diplomové práci zjistil, spojení s knihovnou MFC nebylo příliš triviální.

Framework ROOT je distribuován pod licencí LGPL 2.1, takže je možné ho použít v široké škále uzavřených i otevřených prostředí.

2.4.2.1 Výhody frameworku ROOT

- Velice jednoduchá tvorba grafů a histogramů.
- Použitelnost s jinými programovacími jazyky a frameworky.
- Možnost efektivního zpracování velikých souborů.

2.4.2.2 Nevýhody framework ROOT

- Špatná správa paměti – framework produkuje veliké množství memory leaků.
- Nutnost použití globálních proměnných.
- Malá komunita oproti jiným matematickým frameworkům, tudíž obtížné řešení problémů.

2.5 Popis ODBC

Práci s databází v diplomové práce probíhá pomocí rozhraní ODBC – Open Database Connectivity. ODBC je standardní aplikační rozhraní, tzv. API, pro přístup k datům. Jeho velkou výhodou je fakt, že umožňuje aplikaci přistupovat k datovým zdrojům nezávisle na tom, jaký systém pro řízení báze dat (anglicky Database Management Systém – DBMS) používá datový zdroj. Z pohledu aplikace tak přistupujeme k Middlewaru, který nám zabezpečí možnost použití datového zdroje, ať už používá jakýkoliv DBMS nebo formát dat pro uložení.

Nezávislost ODBC na DBMS je realizována pomocí ODBC ovladačů. Ten je potřeba v aplikaci definovat a mít ho v podobě například .dll knihovny.

Architektura ODBC se dělí do čtyř základních vrstev:

1. Aplikace – aplikace získává data pomocí volání SQL dotazů, které jsou v ODBC implementovány.
2. Správce ovladačů – jak bylo zmíněno výše, ODBC ke korektní činnosti potřebuje konkrétní ovladače jednotlivých DBMS. Je tedy potřeba vrstva, která zajistí propojení mezi aplikací a příslušným ODBC ovladačem. O toto se stará právě Správce ovladačů. Jeho dalším úkolem je zjištění, jaké funkce poskytuje jaký ovladač. Tyto funkce si pak ukládá do tabulky pomocí níž je implementována i možnost přístupu aplikace k několika datovým zdrojům najednou.
3. Ovladač – již zmíněnou třetí vrstvou jsou konkrétní ovladače. Jejich úloha je překlad dotazů z aplikace na SQL volání, kterému bude rozumět DBMS.

4. Datový zdroj – konkrétní DBMS, které zpracovává požadavky aplikace.

Přestože za vývojem ODBC stojí především společnost Microsoft, není jeho použití nijak platformově omezeno. Je tedy možné ho použít i na jiných operačních systémech včetně UNIXu nebo MacOS.

2.5.1 Výhody ODBC

- Efektivní třídění a filtrace informací
- Přístup ke konfiguraci pomocí externích programovacích nástrojů
- Lze vytvářet vlastní mini-aplikace pomocí vestavěných funkcí, jako jsou dotazy nebo makra

2.5.2 Nevýhody ODBC

- Zmenšená rychlost a celkové snížení výkonu i u triviálních dotazů.
- Přidání další databázové vrstvy a zvýšení komplexnosti aplikace
- Nutnost zadání DNS

2.6 OLE DB

Zkratka OLE DB znamená Object Link and Embedding Database. Podobně jako ODBC se jedná o aplikační rozhraní vyvinuté Microsoftem pro přístup k datům nezávisle na programovacím jazyku nebo platformě.

Aby API vycházelo z technologie OLE, je nutné, aby poskytovalo rozhraní implementované pomocí Component Object Model (COM). OLE DB byla od počátku vyvíjená jakožto nástupce ODBC, protože má podporovat všechny funkce jako ODBC, a navíc ho rozšířit o podporu široké škály nerelačních databází, jako jsou objektové databáze a tabulkové procesory, které nutně nemusí implementovat jazyk SQL.

Princip Ole DB spočívá v oddělení datového zdroje a aplikace pomocí několika úrovní abstrakce, které se nazývají relace, příkazy a množiny řádků. Důležité je též konceptuální rozdělení na poskytovatele a konzumenty . Roli konzumentů v tomto případě zastávají aplikace potřebující data, zatímco poskytovatelé jsou softwarové komponenty implementující určitě rozhraní a tudíž jsou schopné poskytnout data konzumentovi.

Moderní verze Visual Studia umožňují práci s ODBC a OLE DB, proto je důležité alespoň teoreticky znát obě varianty. Jelikož žádné funkce, o které OLE DB rozšiřuje starší ODBC nebylo nutné využít, rozhodl jsem se tedy využít model ODBC.

2.7 Sériová linka – teorie

Velice důležitou součástí práce je komunikace po sériové lince. Rozhodl jsem se proto v této kapitole popsat obecné parametry sériové linky, přesněji standardu RS-232, zatímco její implementace v jazyku C++ bude popsána v kapitole 5.

2.7.1 Obecně o RS-232

Sériová linka se používá především pro komunikaci osobních počítačů s jinou elektronikou. Tato elektronika může být z různých odvětví lidské činnosti, a protože je tento standard i relativně jednoduchý, je stále v průmyslu velice často využíván, ačkoliv poslední varianta RS-232C byla představena již v roce 1969.

V některých oblastech je RS-232 je v poslední době často vytlačován výkonnějším Univerzálním sériovým rozhraním (USB), ovšem pro jednoduché komunikace je RS-232 přece jen ještě výhodnější právě díky své jednoduchosti – standard pouze definuje, jak přenést určitou sekvenci bitů po médiu, nezabývá se tedy vyššími vrstvami komunikace. V referenčním ISO/OSI modelu tedy komunikuje pouze na nejnižší fyzické vrstvě. Jeho další výhodou je skutečnost, že standard je ze své podstaty bezkolizní, na rozdíl od výše zmíněného USB nebo třeba Ethernetu.

Nejčastější vývod, kterým je RS-232 z počítače vyvedena, je realizován pomocí konektoru typu D-Sub typu DE-9 M, připojen tedy bývá variantou DE-9 F. Bohužel jak již bylo zmíněno, tyto konektory se v PC již často nenachází, protože bývají nahrazeny USB. Často tedy bývá potřeba použít převodník USB-RS232, jako tomu bylo během implementace diplomové práce. Z pohledu operačního systému se poté s takto připojenou periferií pracuje stejně, jako by byla připojena přímo pomocí RS-232. [9]

Pro komunikace na velké vzdálenosti (více než 15 metrů nebo délku vodiče o kapacitě 2500pF, jak udává norma) je vhodné použít některé z novějších standardů sériové komunikace, konkrétně RS-422 nebo RS-485, ačkoliv i na RS-232 lze použít opakovače signálu pro zesílení dosahu. Každopádně řešení této problematiky nebylo v diplomové práci zapotřebí. [10]

2.7.2 Technická specifikace RS-232

Standard RS-232 pro svoji komunikaci používá dvě napěťové úrovně – log. 0 a log. 1. Logická 1 je často označována jako tzv. marking state, neboli česky klidový stav. Naopak logické 0 se přezdívá space state. Log. 1 je implementovaná jako napěťově záporná hodnota, zatímco log. 0 je signalizována kladným napětím. Běžně se jedná o hodnoty +10 V a -10 V.

Komunikace je realizována pomocí níže popsaných signálů. Zdrojové zařízení (DCE) v případě diplomové práce zastupuje periferie (posuvný mechanis-

mus, unidos, ...) a koncové zařízení (DTE) je počítač, na kterém je spuštěn informační systém.

- DCD – Data Carrier Detect – Detekce nosného kmitočtu
- RXD – Receive Data – Tok dat z DCE do DTE
- TXD – Transmit Data – Tok dat z DTE do DCE
- DTR – Data Terminal Ready – Koncové zařízení oznamuje, že je připravené komunikovat
- SGND – Signal Ground – Signálová zem
- DSR – Data Set Ready – DCE signalizuje DTE, že je schopné komunikovat
- RTS – Request to Send – DTE signalizuje DCE, že je schopné komunikovat
- CTS – Clear to Send – DCE signalizuje DTE, že je komunikační cesta je volná
- RI – Ring Indicator – Indikátor zvonění

Z popsaných signálů není příliš patrné, zda-li RS-232 komunikuje asynchronně nebo synchronně. To z toho důvodu, že komunikace probíhá ve dvou fázích – zahájení komunikace proběhne asynchronně, kdežto samotná komunikace pak funguje synchronně.

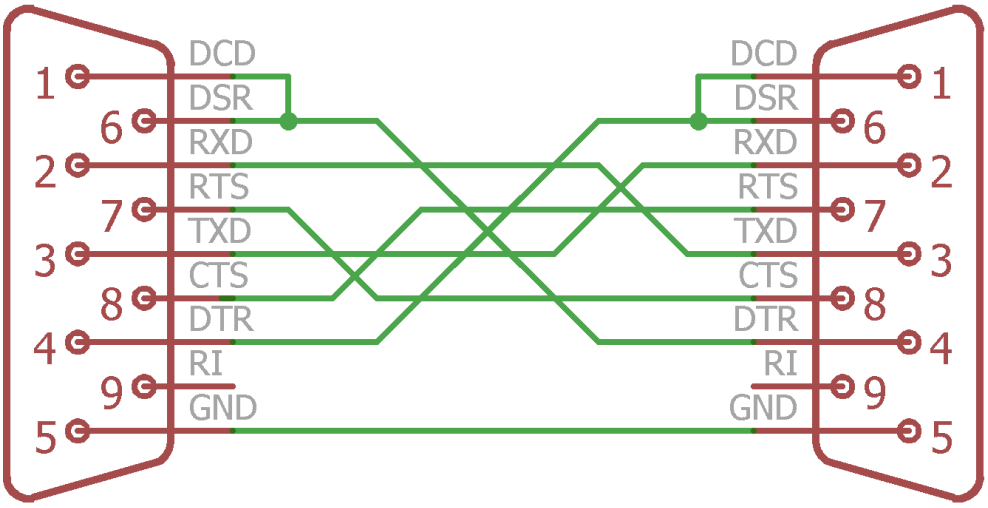
V diplomové práci byla implementace sériové linky testována za použití loopback zapojení – odchozí a příchozí data byla posílána v rámci jednoho PC – to znamená, že DCE i DTE je totéž zařízení. Z tohoto důvodu je znalost výše uvedených pinů zásadní – musíme si totiž vytvořit (nebo zakoupit) tzv. Null modem, který některé z uvedených pinů překříží (viz obrázek 2.2).

Název null modem je historický – někdy bylo nutné propojit dva terminály nebo počítače napřímo mezi sebou a komunikovat. V takovém případě bylo potřeba mezi ně připojit ještě jedno zařízení, které postupně nahradil null modem, neboli „prázdný“ modem.

2.7.3 Základní parametry RS-232

Pro úspěšnou implementaci diplomové práce je nutné porozumět především parametrům, které jsou potřeba během komunikace nastavit, aby periferie komunikovala správně. Mezi ty základní (a často nutné) patří:

- Baud Rate – Baud je jednotka, pomocí níž se v informatice měří přenosová rychlost média. Jeden Baud odpovídá jedné změně signálu na



Obrázek 2.2: Schéma zapojení null modemu.

přenosovém médiu. Baud rate (Bd) poté udává počet změn signálu během jedné vteřiny. Vzhledem k tomu, že do jedné signálové změny lze zakódovat i více než jeden bit, nelze tuto metriku slučovat s častěji používanou bps – bits per second. Nejčastější hodnoty baudrate jsou 2400, 4800, 9600 a 19200 Bd.

Jednotka Baud je pojmenována po francouzském vynálezci Jean-Maurice-Émile Baudotovi.

- Velikost rámce – počet bitů v jednom rámci dat. V současných systémech se používá takřka výhradně osmibitový přenos, ale v některých starších zařízeních se používaly i sedmibitové rámce.

- Parita – Parita bez vysokých nároků na výpočetní výkon zabezpečuje základní kontrolu dat. Princip spočívá v součtu jedničkových bitů a doplní se paritním bitem tak, aby byl výsledek buď sudý, nebo lichý. Existují i varianty, kdy se paritní bit nastavuje buď vždy na log. 0 (tzv. „space parity“) nebo naopak na log. 1 („mark parity“). V tomto případě se jedná o tzv. single error detection code, čili kód detekující jen jednu chybu.

- Stop bit – Stop bitem se indikuje ukončení přenášeného rámce. Zároveň tento bit indikuje, že příjemce má určitý čas na zpracování přijatých dat. Stop bit může být buďto jediný, nebo zdvojený, který se používal spíše u starších zařízení, které potřebovaly větší množství času na zpracování přijatého rámce.

Kapitola 3

Realizace

3.1 Použité softwarové nástroje

3.1.1 Microsoft Visual Studio

Visual Studio je vývojové prostředí od společnosti Microsoft, ve kterém je možné vyvíjet širokou škálu aplikací, od standardních stand–alone programů až po webové aplikace a služby. Tyto produkty lze psát v několika jazycích, které MSVS podporuje, konkrétně se jedná o C, C++, C# VB.NET a C++/CLI, k němuž je ovšem potřeba Microsoftí implementace C++ kompilátoru nazývaná Microsoft Visual C++. Tento software je sice komerční, ale existují verze, které jsou zdarma pro malé výzkumné týmy a školní účely, což přesně diplomová práce splňuje.

IDE používá IntelliSense, čili je schopné automaticky doplňovat kusy kódu. Dále pak vývojové prostředí umožňuje jednoduše provádět refaktoring. Visual Studio pochopitelně obsahuje i textový editor, debugger a mnoho designerů pro tvorbu GUI. Visual Studio existuje v mnoha verzích, pro implementaci diplomové práce bylo použito Community edition 2015.

3.1.2 Oracle SQL Developer Data Modeler

Pomocí Oracle SQL Developer Data Modeler lze jednoduše modelovat databázi, kterou bude aplikace používat. Ačkoliv datový model není příliš složitý, je vhodné ho mít zdokumentovaný a přehledně namodelovaný. Tento nástroj umožňuje modelovat jak na úrovni konceptuálního, tak relačního modelu. Jeho velkou předností je možnost vygenerovat DLL skript pro vytvoření SQL databáze.

3.1.3 Demoworld.nl

Demoworld je online modelovací nástroj pro modelování procesů a jejich simulaci. Tento software byl vyvinut především pro studijní účely a bohužel působí

místy nedodělaně. Naštěstí obsahuje krátké video–tutorialy, pomocí nichž lze jeho použití zvládnout. Program umožňuje i týmovou spolupráci na projektu a několik dalších možností, které ovšem nebyly podstatné pro diplomovou práci.

Jelikož je software spíše ve fázi testování, je možné si pořídit verzi Student i Professor zdarma a používat tak aplikaci v plném rozsahu. [11]

3.1.4 CINT

CINT je jednoduchý interpret kódu napsaného v C a C++. Umožňuje interpretovat většinu kódu ANSI C a ISO C++ 2003. Ve skriptu je možné volat kompilované třídy a funkce, zatímco i kompilovaný kód může zpětně volat interpretované funkce pomocí CINT. Tento nástroj je vhodný použít tam, kde je rychlý vývoj softwaru přednější než jeho efektivní zpracování ovlivňující rychlost programu. Pomocí něho byl interpretován původní software v ÚJF Řež.

V diplomové práci byl použit především pro testování kódu napsaného v ROOT. [12]

3.2 Popis případů užití

Tato podkapitola se zabývá všemi případy užití, které mohou v aplikaci nastat. U každého případu je nejprve uveden aktér (účastník), který může daný případ použití vykonat, a poté uveden návod krok po kroku, jak dosáhnout cíle.

Důležitá je rovněž hierarchie účastníků znázorněná na obrázku 3.1 a pro přehlednost jsou důležité případy užití znázorněny na obrázku 3.2.

3.2.1 Přihlášení uživatele

Aktér: Nepřihlášený uživatel / Přihlášený uživatel / Správce aplikace

1. Případ užití začíná v momentě, kdy se nepřihlášený uživatel rozhodne přihlásit do aplikace.

2. Uživatel zobrazí tabulku pro přihlášení v hlavním okně.

3. Aktér vyplní své uživatelské jméno s heslem a stiskne tlačítko Login.

4. Pokud byly údaje zadány správně, systém uživatele přihlásí a oznámí tuto skutečnost vyskakovacím oknem. V opačném případě aktér pokračuje bodem 3 tohoto scénáře.

3.2.2 Přidání nového uživatele

Aktér: Správce aplikace

1. Tato situace nastává, pokud chce správce aplikace přidat do systému nového uživatele.

2. Správce aplikace zobrazí tabulku pro přihlášení v hlavním okně.

3. Aktér vyplní nové uživatelské jméno s heslem a stiskne tlačítko Add user.

4. Pokud je zadané jméno unikátní (žádné takové již v aplikaci neexistuje), systém nového uživatele vytvoří a uloží hash hesla. Pokud je jméno již obsazené, aplikace tuto skutečnost oznámí a je potřeba vybrat jiné jméno.

3.2.3 Vybrání ovladače pro posuvný mechanismus nebo systém měření proudu

Aktér: Přihlášený uživatel / Správce aplikace

1. Scénář začíná v okamžiku, kdy chce aktér připojit nový ovladač pro posuvný stůl nebo systém měření proudu.

2. Aktér zobrazí tabulku ovladačů v hlavním okně.

3. Aktér vybere ovladač pro požadované zařízení společně s číslem portu a potvrdí stisknutím Vybrat.

4. Systém vytvoří ovladač a pokusí se ho připojit na požadovaný port.

3.2.4 Otevření rozhraní pro ovládání pohybového mechanismu

Aktér: Přihlášený uživatel / Správce aplikace

1. Případ užití začíná, když se aktér rozhodne pracovat s pohybovým mechanismem.

2. Aktér pomocí scénáře 3.2.3 vybere ovladač pohybového mechanismu.

3. Aktér v hlavním menu vybere kategorii Measurement a dále pak Positioning System Controls.

4. Je-li vybraný ovladač připojen k zařízení, systém otevře ovládací rozhraní pro pohybový mechanismus. Pokud není, aplikace aktéra upozorní. Dále je nutné vybrat jiný ovladač, případně změnit port, na kterém je pohybový mechanismus připojen.

3.2.5 Kalibrace pohybového mechanismu

Aktér: Přihlášený uživatel / Správce aplikace

1. Scénář začíná, když se aktér rozhodne zkalibrovat pohybový mechanismus.
2. Aktér postupně vykoná všechny kroky popsané ve scénáři 3.2.4.
3. Aktér stiskne tlačítko Calibrate table.
4. Aktér počká maximálně dvacet vteřin, než se pohybový mechanismus zkalibruje. Systém úspěšnou kalibraci oznámí vyplněním maximálního rozsahu v boxu Actual Position.

3.2.6 Nastavení polohy pohybového mechanismu

Aktér: Přihlášený uživatel / Správce aplikace

1. Případ užití začíná v okamžiku, kdy se aktér rozhodne nastavit polohu pohybového mechanismu.
2. Aktér postupně vykoná všechny kroky popsané ve scénáři 3.2.4.
3. Aktér vyplní souřadnice v boxu Table Movement nebo vyberu jednu z uložených pozic, dále pak vybere požadované jednotky. Chce-li nastavit stůl do vybraných souřadnic bez ohledu na aktuální polohu, zaškrtne možnost Move Absulutely. Pokud chce stůl posunout o vybranou délku, nechá políčko Move Absolutely prázdné.
4. Aktér stiskne tlačítko MOVE a vyčká na nastavení pohybového mechanismu.
5. Systém novou polohu oznámí v boxu Actual position a zobrazí ji v grafickém rozhraní. Zároveň zaloguje všechny relevantní hodnoty.

3.2.7 Zadání polohy vzorku

Aktér: Přihlášený uživatel / Správce aplikace

1. Scénář začne, pokud chce aktér zadat nebo změnit polohu zkoumaného vzorku relativně k poloze stolu.
2. Aktér postupně vykoná všechny kroky popsané ve scénáři 3.2.4.
3. Aktér vyplní vzdálenosti vzorku od pohybového mechanismu.
4. Aktér stiskne tlačítko SET, systém zobrazí polohu vzorku v grafu.

3.2.8 Zadání polohy středu svazku

Aktér: Přihlášený uživatel / Správce aplikace

1. Tato situace nastane, pokud chce aktér zadat nebo změnit polohu středu svazku.
2. Aktér postupně vykoná všechny kroky popsané v 3.2.4.
3. Aktér vyplní obě souřadnice středu svazku.
4. Aktér stiskne tlačítko SET, systém zobrazí polohu vzorku v grafu.

3.2.9 Odpojení pohybového mechanismu od aplikace

Aktér: Přihlášený uživatel / Správce aplikace

1. Případ užití začíná v okamžiku, kdy se aktér rozhodne odpojit aktuálně používaný pohybový mechanismus.
2. Aktér postupně vykoná všechny kroky popsané ve scénáři 3.2.4.
3. Aktér stiskne tlačítko Disconnect. Systém pohybový mechanismus odpojí a pro další práci s pohybovým mechanismem je nutné připojit jiný ovladač – viz 3.2.3.

Alternativní scénář:

1. Alternativní scénář začíná po druhém kroku hlavního scénáře.
2. Aktér stiskne tlačítko Exit – systém zařízení odpojí a zároveň zavře rozhraní pro ovládání pohybového mechanismu. Při stisknutí tlačítka OK se rozhraní pouze zavře a ovladač zůstane připojen.

3.2.10 Otevření rozhraní pro monitorování intenzity proudu

Aktér: Přihlášený uživatel / Správce aplikace

1. Scénář začíná v okamžiku, kdy se aktér rozhodne otevřít rozhraní pro monitorování intenzity proudu.
2. Aktér pomocí scénáře 3.2.3 vybere ovladač zařízení pro měření proudu.
3. Aktér vybere z hlavní nabídky možnosti Measurement – Flux measurement.
4. Pokud je aktuálně vybrán připojený ovladač pro měření proudu, systém otevře rozhraní pro monitorování intenzity. V opačném případě je nutný vybrat jiný ovladač pomocí 3.2.3.

3.2.11 Spuštění monitorování intenzity

Aktér: Přihlášený uživatel / Správce aplikace

1. Tato situace nastává, když se aktér rozhodne spustit monitorování intenzity svazku.
2. Aktér postupně vykoná všechny kroky popsané v části 3.2.10.
3. Aktér stiskne tlačítko START MEASUREMENT. Systém zobrazí graf a začne do něho zakreslovat hodnoty s třívteřinovým rozestupem. Dále pak všechny naměřené hodnoty loguje do databáze.

3.2.12 Pozastavení monitorování intenzity

Aktér: Přihlášený uživatel / Správce aplikace

1. Scénář užití začíná v okamžiku, kdy se aktér rozhodne přerušit monitorování intenzity svazku.
2. Aktér vykoná všechny kroky popsané v části 3.2.10
3. Aktér stiskne tlačítko SUSPEND MEASUREMENT, které je přístupné pouze v okamžiku, kdy monitorování běží. Systém veškeré měření přeruší až do okamžiku, kdy se ho uživatel rozhodne opět zapnout.

3.2.13 Nastavení časového okna

Aktér: Přihlášený uživatel / Správce aplikace

1. Tento případ užití nastává ve chvíli, kdy se aktér rozhodne změnit uplynulou dobu, za kterou jsou do grafu zpětně vypisovány hodnoty.
2. Aktér postupně vykoná všechny kroky popsané v podkapitole 3.2.10.
3. Aktér v kolonce Time window nastaví počet minut, za které bude program vypisovat hodnoty. Pokud uživatel nenastaví nic nebo nulu, budou se vypisovat všechny naměřené hodnoty.

3.2.14 Odpojení zařízení pro monitorování intenzity proudu od aplikace

Aktér: Přihlášený uživatel / Správce aplikace

1. Případ užití začíná v okamžiku, kdy se aktér rozhodne odpojit aktuálně používané zařízení pro monitorování intenzity proudu od aplikace.
2. Aktér postupně vykoná všechny kroky popsané ve scénáři 3.2.10.

3. Aktér stiskne tlačítko Disconnect. Systém zařízení pro monitorování intenzity proudu odpojí a pro další práci se zařízením je nutné připojit jiný ovladač – viz 3.2.3.

Alternativní scénář:

1. Alternativní scénář začíná po druhém kroku hlavního scénáře.

2. Aktér stiskne tlačítko Exit – systém zařízení odpojí a zároveň zavře rozhraní zařízení pro monitorování intenzity proudu.

3.2.15 Otevření rozhraní pro automatické hledání středu svazku

Aktér: Přihlášený uživatel / Správce aplikace

1. Tato situace nastává, když se aktér rozhodne otevřít rozhraní pro automatické hledání středu svazku.

2. Aktér pomocí scénáře 3.2.3 vybere ovladač zařízení pro měření proudu a zároveň ovladač pro posuvný mechanismus.

3. Aktér v hlavním menu vybere možnosti Measurement – Scan beam profile.

4. Systém zkontroluje připojení obou periferií a je-li v pořádku, otevře rozhraní pro automatické hledání středu svazku. V opačném případě je nutné nastavit chybějící ovladač.

3.2.16 Zadání parametrů pro automatické hledání středu svazku

Aktér: Přihlášený uživatel / Správce aplikace

1. Případ užití začíná v okamžiku, kdy uživatel chce nastavit parametry pro automatické hledání středu svazku.

2. Aktér postupně vykoná všechny body ze scénáře 3.2.15.

3. Aktér zadá následující parametry:

   - Scan along – výběr osy, na které bude hledán střed svazku.
   - Units – jednotky (mikrometry nebo milimetry), v nichž uživatel zadává všechny souřadnice.
   - Scan min – souřadnice, na níž hledání svazku začne.

- Scan max – souřadnice, jejíž překročení signalizuje ukončení skenování. Hodnota musí být menší, než rozsah posuvného mechanismus, který je získán kalibrací. Pro správnou funkci je tedy nutné vykonat scénář 3.2.5.
- Scan step – krok, s jakým se bude skenovat vybraná osa.
- Fixed axis – souřadnice zafixované osy.

4. Aktér stiskne tlačítko SET PARAMETERS.

5. Systém nastaví pohybový mechanismus do vybrané polohy a uloží vybrané parametry skenování.

3.2.17 Spuštění hledání středu svazku

Aktér: Přihlášený uživatel / Správce aplikace

1. Scénář začne, když se aktér rozhodne spustit automatické hledání středu svazku.

2. Aktér provede všechny kroky ze scénáře 3.2.16.

3. Aktér stiskne tlačítko START SCAN.

4. Systém spustí skenování svazku a zakresluje jednotlivé hodnoty proudu do grafu.

3.2.18 Proložení naměřených bodů Gaussovou křivkou

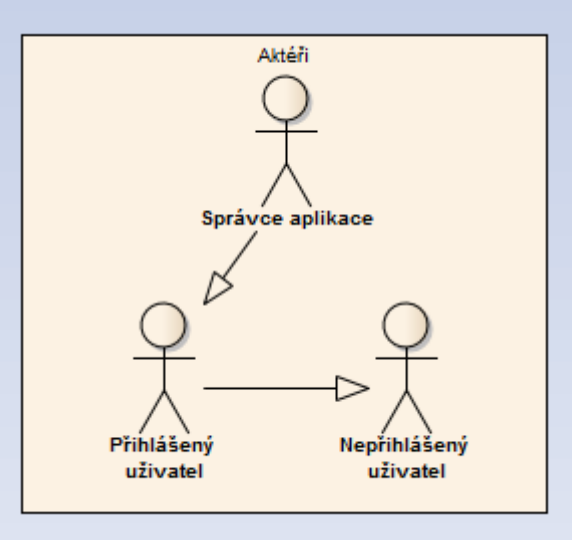
Aktér: Přihlášený uživatel / Správce aplikace

1. Případ užití začíná v momentě, kdy se aktér rozhodne proložit naměřené body z hledání středu svazku Gaussovou křivkou.

2. Aktér provede všechny kroky ze scénáře 3.2.15.

3. Aktér stiskne tlačítko FIT GAUSS.

4. Je-li naměřeno alespoň pět bodů, systém je proloží Gaussovo křivkou. V opačném případě je nutné naměřit další body pomocí scénáře 3.2.17. Prokládání je iterativní, při opakovaném stisknutí tlačítka FIT GAUSS může být dosaženo jiných – a lepších – výsledků.

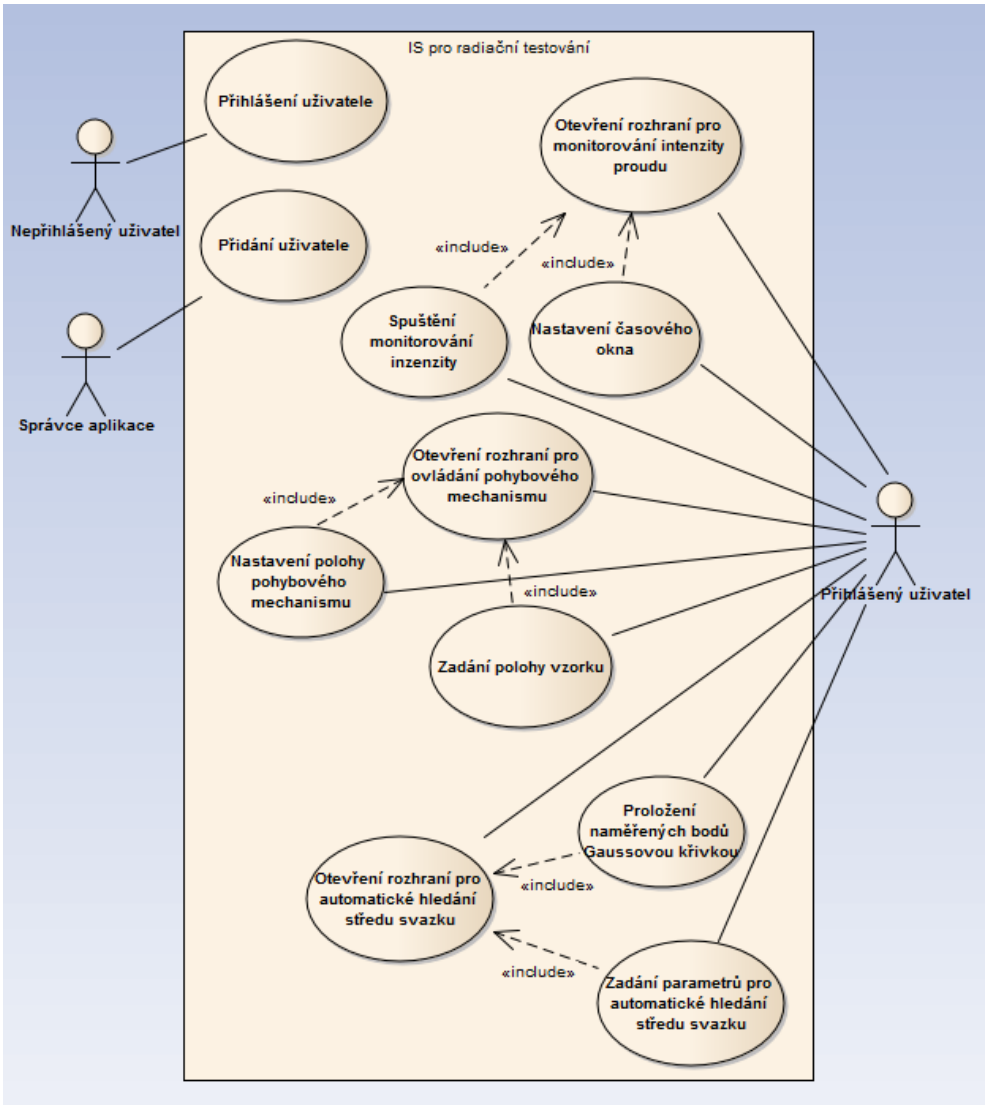


Obrázek 3.1: Hierarchické uspořádání všech účastníků, kteří se v informačním systému vyskytují. Výše postavený účastník v hierarchii může vykonávat všechny případy užití níže postaveného.

3.2.19 Ukončení programu

Aktér: Nepřihlášený uživatel, Přihlášený uživatel a Správce aplikace

1. Scénář začíná v okamžiku, kdy se aktér rozhodne ukončit program.
2. Aktér v hlavním menu zvolí možnosti File – Quit.
3. Systém program ukončí.



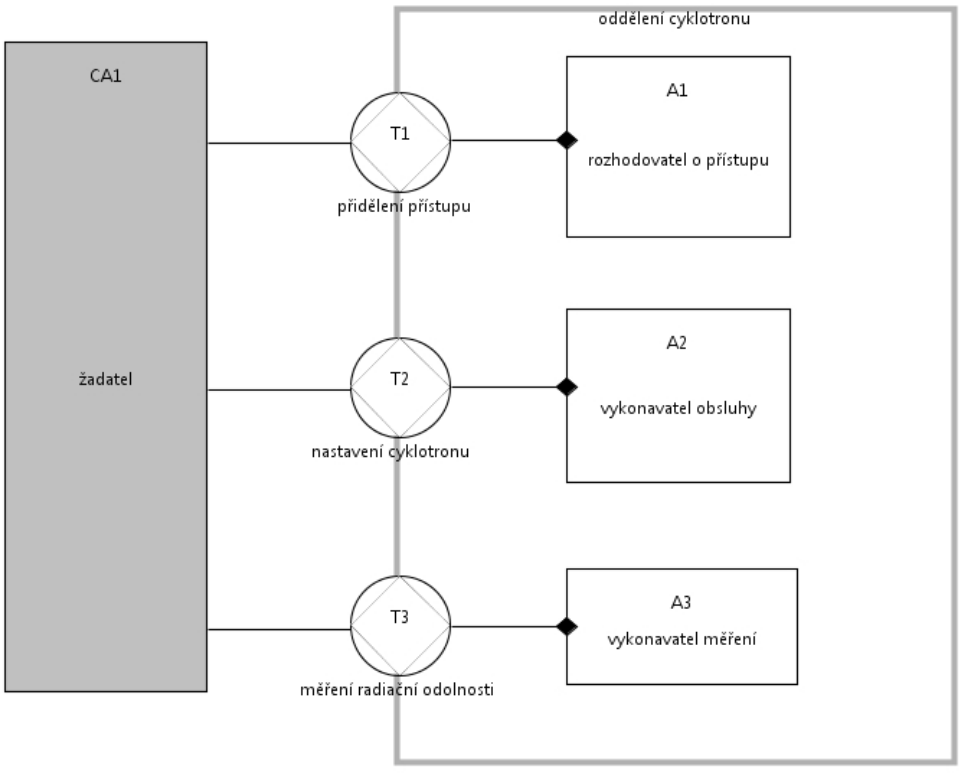
Obrázek 3.2: Schéma případů užití. Z úsporných důvodů jsou zaznamenány pouze nejpodstatnější případy užití.

Kapitola 4

DEMO modelování

4.1 Model rezervace cyklotronu

4.1.1 OCD



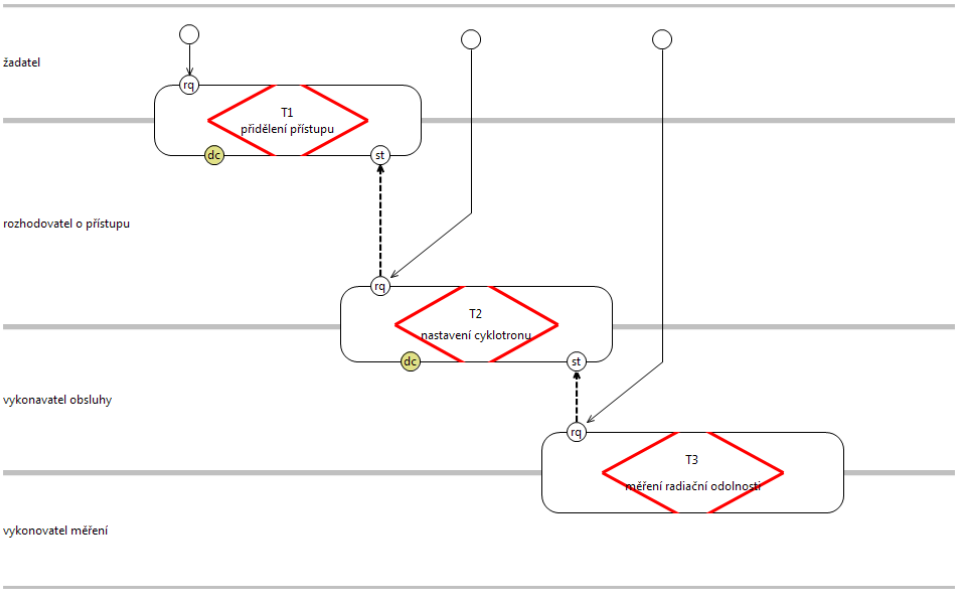
Obrázek 4.1: OCD diagram rezervace cyklotronu

4.1.2 TPT

Tabulka 4.1: TPT tabulka rezervace cyklotronu

ID transakce	Jméno transakce	Produkt transakce
T1	přidělení přístupu	Přístup byl přidělen.
T2	nastavení cyklotronu	Cyklotron byl nastaven.
T3	měření radiační odolnosti	Radiační odolnost byla změřena.

4.1.3 PSD



Obrázek 4.2: PSD diagram rezervace cyklotronu

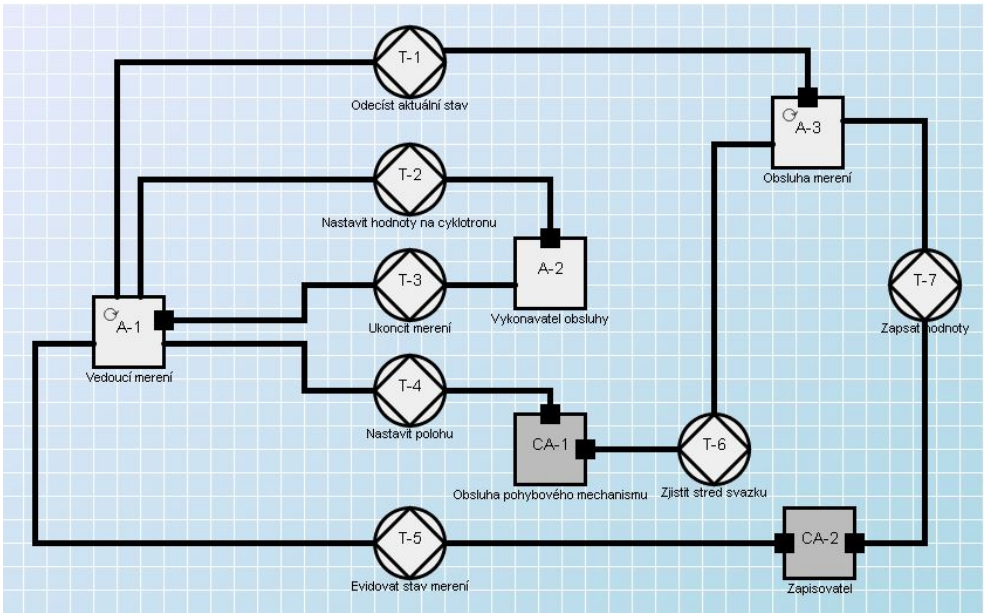
4.1.4 Tabulka mapování rolí na aktéry

Tabulka 4.2: Mapování rolí na aktéry modelu rezervace

ID role	Jméno role	Jméno aktéra
CA1	žadatel	Vedoucí oddělení ÚJF
A1	rozhodovatel o přístupu	Vedoucí cyklotronu
A2	vykonavatel obsluhy	Obsluha cyklotronu
A3	vykonavatel měření	Pracovníci ÚJF

4.2 Model radiačního měření

4.2.1 OCD



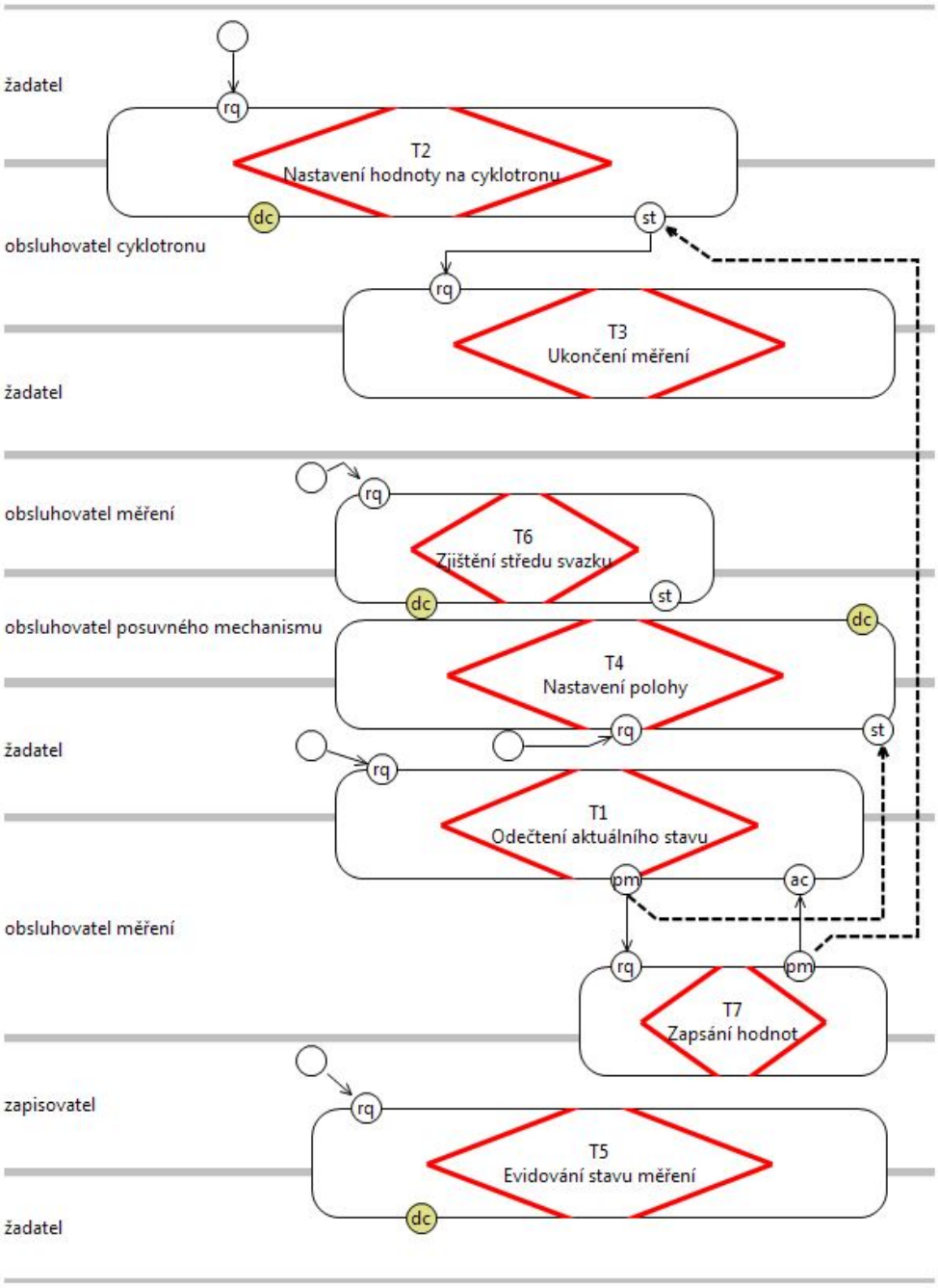
Obrázek 4.3: OCD diagram radiačního měření

4.2.2 TPT

Tabulka 4.3: TPT tabulka radiačního měření

ID transakce	Jméno transakce	Produkt transakce
T1	Odečtení aktuálního stavu	Aktuální stav byl odečten.
T2	Nastavení hodnoty na cyklotronu	Cyklotron byl nastaven na požadovanou hodnotu.
T3	Ukončení měření	Měření bylo ukončeno.
T4	Nastavení polohy	Poloha pohybového mechanismu byla nastavena.
T5	Evidování stavu měření	Aktuální stav měření byl zapsán.
T6	Zjištění středu svazku	Poloha středu svazku byla zjištěna.
T7	Zapsání hodnot	Aktuální hodnoty byly zapsány.

4.2.3 PSD



Obrázek 4.4: PSD diagram radiačního měření

4.2.4 Tabulka mapování rolí na aktéry

Tabulka 4.4: Mapování rolí na aktéry modelu měření

ID role	Jméno role	Jméno aktéra
CA-1	Obsluha pohybového mechanismu	Pracovník ÚJF
CA-2	Zapisovatel	Odborný asistent
A-1	Žadatel	Vedoucí měření
A-2	Vykonavatel obsluhy	Obsluha cyklotronu
A-3	Obsluha měření	Pracovník ÚJF

4.3 Shrnutí

K úspěšnému vytvoření DEMO modelů bylo potřeba překonat jednu zásadní výzvu, a sice správně identifikovat ontologické transakce. Na první pohled se může zdát, že mnoho transakcí v modelu 4.3 je pouze datalogických, či infologických, ale po pečlivém přezkoumání zjistíme, že tomu tak není. Tyto transakce, ačkoliv se mohou zdát triviální, tvoří klíčové produkty ÚJF. Nemůžeme tedy ani pouhé zapsání hodnot považovat za infologické.

Na modelu 4.1 a 4.2 není příliš mnoho co optimalizovat. Model 4.3 už je ale mnohem zajímavější. Především by bylo možné některé aktéry sloučit do jednoho, čímž se ušetří lidské prostředky a odpadne mnoho komunikace v modelu 4.4, což samozřejmě omezuje možnost zanešení lidské chyby.

Ideálním řešením je tak implementovat software, který bude podporovat právě transakce z těchto modelů.

Kapitola 5

Implementace

5.1 Diagram tříd

Kompletní diagram tříd je znázorněn na obrázku 5.1. Je z něho patrné, že ústřední bod celé aplikace je třída MainForm, která dále instancuje třídy ovládající jednotlivá rozhraní – TableController, BeamController a FluxController.

Je nutné si uvědomit, že tyto třídy mají na starost především uživatelské rozhraní a nestarají se o samotný sběr dat a komunikaci s hardwarem. K tomuto účelu používají abstraktní třídy ovladačů, prozatím TableDriver a FluxDriver, ovšem je možné do aplikace jednoduše implementovat další kategorii pro další zařízení. Tyto abstraktní třídy jsou následně implementovány konkrétními ovladači, například MCLDriver. TableController tak v podstatě vůbec nepotřebuje vědět, jaký konkrétní hardware ovládá, dotazuje se vždy rozhraní TableDriver a konkrétní zařízení je vybráno pomocí ovladače.

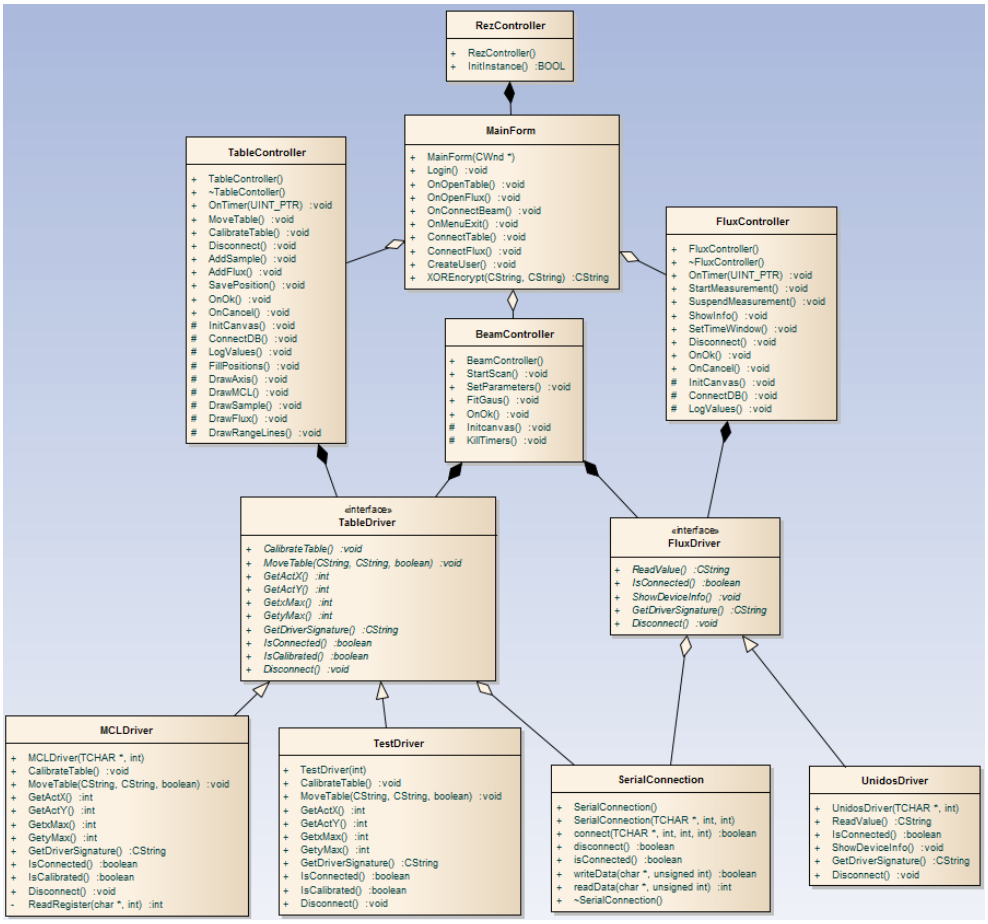
Pokud by teď tedy bylo nutné vyměnit pohybový mechanismus za jiný, stačí do aplikace připsat ovladač implementující TableDriver. Toto rozdělení je velice důležité, protože modularita byla jedna ze základních požadavků na aplikaci.

5.2 Propojení knihovny MFC s funkcemi ROOT

5.2.1 Založení projektu

Založení projektu využívající MFC lze provést dvěma způsoby – použitím standardního Wizard rozhraní Microsoft Visual Studia, nebo založit úplně prázdný projekt a vše nastavit ve vlastnostech projektu.

První možnost je velice jednoduchá, stačí vybrat File → New → Project → Visual C++ → MFC Application. Dále je nutné vybrat vhodné nastavení projektu, jako typ aplikace, styl projektu a způsob využití MFC – pokud uživatel vybere použití MFC pomocí DLL, nebude tato knihovna součástí distribuce a je nutné DLL knihovny dodávat zvlášť. Výhodou je menší velikost celého



Obrázek 5.1: Diagram tříd informačního systému.

projektu. V případě zvolení statické verze knihovny budou MFC funkce kompilovány do projektu, čímž vzroste jeho velikost, ale není je potřeba dodávat společně s programem.

Druhou možností je založit prázdný projekt a vše nastavit ručně ve vlastnostech projektu. Tato možnost je poměrně komplikovaná, má ovšem jednu zcela zásadní výhodu – vývojáři odpadne mnoho zbytečného kódu, který prvně popsaný způsob vygeneruje.

Jako optimální řešení se jeví použít Wizard MSVS, seznámit se strukturou MFC projektu a odmazat nepotřebné části. [13]

5.2.2 Instalace frameworku ROOT a linkování k MFC projektu

K instalaci je nutné stáhnout instalační balík z webu https://root.cern.ch/. Po instalaci je vhodné vyzkoušet ROOT funkce například z příkazové řádky. Pokud vše funguje správně, je nutné zadat umístění hlavičkových souborů s implementací Rootovských funkcí a nalinkovat ROOT knihovny do projektu MFC.

Přidání hlavičkových souborů je možné provést následujícím postupem: Project → Properties → C/C++ → Additional Include Directories a do tohoto pole vepsat umístění hlavičkových souborů. Ty jsou k nalezení v adresáři, do kterého byl ROOT nainstalován, například tedy C:\root_v5.34.25\include.

Linkování knihoven lze provést v nastavení Linkeru. Konkrétně tedy Project → Properties → Linker → Input → Additional Dependencies. Nyní je nutné přidat všechny knihovny, které bude aplikace používat. V krajním případě lze tedy nalinkovat všechny soubory s příponou .lib z adresáře C:\root_v5.34.25\lib\.

5.2.3 Použití funkcí ROOT v samotném kódu

V tento okamžik pravděpodobně půjde C++ kód obsahující ROOT funkce zkompilovat a spustit. Bohužel během vykonání samotné funkce program zhavaruje. Na začátku programu je proto nutné deklarovat a definovat globální proměnnou typu TApplication:

```
// Global variable declaration
TApplication *gMFCRootApp;
```

```
// Somewhere in code,
// before execution of any ROOT function
gMFCRootApp = new TApplication("Rez Controller ROOT", &
   argc, argv);
gMFCRootApp->SetReturnFromRun(true);
```

Nakonec je nutné definovat funkci, které bude v programu periodicky volána:

```
VOID CALLBACK ROOTTimer(HWND hwnd, UINT message, UINT
   idTimer, DWORD dwTime) {
        if(gApplication != NULL) {
                gApplication->StartIdleing();
                gSystem->InnerLoop();
                gApplication->StopIdleing();
        }
}
```

V tuto chvíli je spojení ROOT a MFC hotové. Mnoho funkcí frameworku ROOT ke své činnosti nicméně potřebuje vytvořený Canvas, který v MFC projektu není triviální vytvořit – jednoduše proto, že je potřeba mateřské okno, do kterého je možné ho vložit. Pro tento účel je vhodné vytvořit virtuální okno v již existujícím. Toto lze provést například následujícím kódem:

```
// Necessary header files
#include <TApplication.h>
#include <Windows4Root.h>
#include <TCanvas.h>

// Code for new window creation - must be placed
// in a class derived from CWnd
CWnd * staticWindow = new CWnd;
staticWindow->Create(_T("STATIC"), _T(""), WS_CHILD |
   WS_VISIBLE,
CRect(B_WINDOW_LEFT, B_WINDOW_TOP, B_WINDOW_WIDTH,
   B_WINDOW_HEIGHT), this, 1236);

int wid = gVirtualX->AddWindow((ULong_t)staticWindow->
   m_hWnd, B_WINDOW_WIDTH, B_WINDOW_HEIGHT);
beamCanvas = new TCanvas("Beam", B_WINDOW_WIDTH,
   B_WINDOW_HEIGHT, wid);
```

5.3 Sériová linka – implementace

Jak již bylo zmíněno v kapitole 2.7, implementace komunikace pomocí sériové linky není triviální záležitost. Bohužel knihovna MFC v tomto případě přímo příliš nepomůže, ovšem nabízí několik tříd a funkcí, které mohou alespoň trochu implementaci sériové linky usnadnit.

Mezi hlavní problémy sériové komunikace patří: [14]

- Nastavení parametrů popsaných v podkapitole 2.7.3 – pokud nebudou obě zařízení používat správný formát dat, komunikace nebude fungovat. V tomto případě naštěstí pomůže několik funkcí, které MFC nabízí. Velice problémová je pak implementace handshakingu.

- Asynchronní čtení a zápis – čtení a zápis do souboru je relativně rychlý oproti zápisu na sériový port. Může se jednat řádově až o desítky milisekund, po které je aktivní vlákno zablokováno.

- Problém při „event driven“ programování – Většina GUI aplikací používá event driven paradigma. Toto paradigma je založené na odchytávání zpráv, bohužel čekání na příchozí zprávu velice často způsobuje aktivní čekání a zablokování celé aplikace.

Několik implementací sériové linky v C++ se dá stáhnout z internetu, bohužel spousta z nich má problémy s kompatibilitou s moderními systémy, někdy nepracují dostatečně korektně nebo nesplňují všechny nároky, které jsou na sériovou linku v rámci diplomové práce kladeny. Existuje i několik velice robustních implementací sériové linky, ovšem jejich používání je naopak až příliš těžkopádné.

I z těchto důvodů jsem se rozhodl implementovat vlastní třídu podporující komunikaci. Sériová linka je v aplikaci realizována pomocí třídy SerialConnection. Tato třída ke své činnosti používá mnoho tříd z Win32 API, například HANDLE nebo COMSTAT.

5.3.1 Otevření portu

Otevření portu zajišťuje metoda connect. Ta pomocí funkce CreateFile vytvoří pro čtení i zápis soubor, který ovládá sériový port. Jelikož se jedná o zvláštní typ souboru, musíme ho pro tyto účely vytvořit s několika specifiky – parametr fdwShareMode musí být nula, protože komunikační porty nemohou být sdíleny stejným způsobem jako standardní soubory, a dále pak fdwCreate musí být nastaven na příznak OPEN_EXISTING.

V další části metoda provádí několik základních a obecných testů, zda-li se připojení zdařilo. Další důležitou částí je nastavení parametrů komunikace, které jsou do metody connect předány jakožto parametry. Naštěstí MFC obsahuje strukturu DCB, která obsahuje členské proměnné s názvy přesně pro sériovou komunikaci. Stačí tedy tyto proměnné nastavit na správné hodnoty a zapsat pomocí funkce SetCommState.

5.3.2 Zápis

Zápis v této implementaci je realizován pomocí standardní funkce pro zápis do souboru – tedy WriteFile. Pokud tato funkce vrátí false, metoda writeData přečte vrácenou chybu ze sériového portu.

5.3.3 Čtení

Čtení je na rozdíl od zápisu v této implementaci poměrně problematické, ačkoliv je implementováno pomocí standardní funkce ReadFile. Za prvé nelze odhadnout, kdy jsou na portu příchozí data a je tedy možné je přečíst. Abychom předešli aktivnímu čekání, je možné situace řešit dvěma způsoby – zavést multithreading, kdy jedno vlákno bude dokola číst sériovou linku. Toto řešení ovšem víceméně potřebuje globální proměnné, proto je v diplomové práci použito řešení pomocí časovačů, které nezpůsobují aktivní čekání, přesto jsou schopné periodicky číst ze sériové linky, viz kapitola 5.5.

Dalším problémem je situace, kdy se čtení provede přesně v okamžiku, kdy je na sériový port zapisováno. Tato možnost je velice nepravděpodobná, ovšem

nastává v momentě, kdy se čtení odpovědi provádí ihned po zápisu požadavku. V takovém případě je možné, že odpověď nestihne dorazit celá a je přečten pouze její kus, proto je vhodné vložit čekání mezi zápis a čtení, například 50 milisekund.

5.4 Způsoby předávání dat mezi okny aplikace

5.4.1 Globální proměnné

Velice triviální metodou výměny dat mezi okny je vytvoření velkého množství globálních proměnných. Jediným úskalím této metody při použití knihovny MFC a Multi-dialogové aplikace je otázka, kde a jak tyto globální proměnné nadefinovat. Bohužel je nelze nadefinovat v části kódu, kde se aplikace spouští, protože zde je většinou nutné vkládat ostatní hlavičkové soubory, které by pak musely vkládat tento výchozí bod, a vzniknul by problém křížových referencí. Tento problém je řešitelný tak, že se proměnná definuje s klíčovým slovem extern v některé z předkompilovaných hlaviček, a jelikož používáme knihovnu MFC, můžeme k tomuto účelu použít hlavičkový soubor stdafx.h. Bohužel samozřejmě přetrvávají všechny nevýhody globálních proměnných, jako je například špatná čitelnost zdrojového kódu, problémy s alokací paměti, kolize jmen atp., a z tohoto důvodu není toto řešení příliš vhodné.

5.4.2 Zpracování fronty zpráv

5.4.2.1 Fronta zpráv obecně

Druhou možností, jak komunikovat mezi jednotlivými okny aplikace, je posílání zpráv. Jedná se o jeden z nejdůležitějších prostředků výměny dat mezi běžícími programy.

Tradiční C(++) programy začínají ve funkci main(), sekvenčně provádí předepsaný kód, na jehož konci eventuálně zaniknou. Koncept Windows a jeho zprávy toto paradigma narušuje, protože běžící program reaguje na Události (Events), které budou dále v práci označovány jako Zprávy (Messages). Tyto zprávy jsou zpracovávány v tzv. smyčce zpráv, která se skládá z fronty příchozích zpráv a částí programu, která se stará o jejich vyzvedávání a předávání ke zpracování.

Tyto zprávy mohou signalizovat mnoho věcí. Mohou být vyvolány buďto uživatelem, operačním systémem, nebo jiným programem. Pro vývoj aplikace je ovšem podstatné, že vyjma standardních zpráv (WM_QUIT – ukončit program, WM_LBUTTONDOWN – stisknutí levého tlačítka myši atp.) lze používat i zprávy uživatelsky definované. Můžeme si tak definovat například vlastní zprávu WM_SENDDATA, pomocí níž budeme z jedné části aplikace (okna) posílat naměřená data do jiné části aplikace. [15]

5.4.2.2 Příklad použití fronty zpráv

V následujícím příkladě je uvedena konkrétní práce s frontou zpráv. Jedná se o modelový případ, kdy chceme z okna ovládající posuvný stůl poslat souřadnice do okna ovládající skenování středu svazku

Krok 1: Vytvoření metody pro zpracování příchozí zprávy

```
// BeamController.cpp
LRESULT BeamController::onAcceptData(WPARAM wParam, LPARAM
    lParam) {
        CString data;
        data.Format(_T("%d"), wParam);
        MessageBox(data);
        return 0;
}
```

Krok 2: Přidání konstanty, která bude představovat ID zprávy

```
//Resource.h
#define WM_SENDDATA WM_APP + 0x100
```

Jak vidíme, je nutné přičíst konstantu WM_APP, aby výsledné ID nekolidovalo s již jinými definovanými systémem.
Krok 3: Přidání makra do mapy zpráv

```
// BeamController.cpp
BEGIN_MESSAGE_MAP(BeamController, CDialogEx)
        ON_MESSAGE(WM_SENDDATA, onAcceptData)
END_MESSAGE_MAP()
```

ON_MESSAGE je generické makro, které zvládne zpracovat libovolný typ příchozí zprávy. WM_SENDDATA je konstanta definovaná v kroku 2 a onAcceptData metoda definovaná v kroku 1.
Krok 4: Poslání dat z jiného místa aplikace

```
// Napr. TableController.cpp
void TableController::sendData()
{
        CWnd * beamWindow;
        beamWindow = CWnd::FindWindowW(NULL, L"Beam");
        if (beamWindow == NULL) {
                MessageBox(L"Couldn't find Beam window");
        }
        else {
                beamWindow->SendMessage(WM_SENDDATA, 5,3);
                MessageBox(L"Sent");
        }
}
```

Metoda FindWindowW najde existující okno pomocí jeho názvu. Pokud se to podaří, data se pošlou generickou zprávou s ID WM_SENDDATA a daty 5 a 3.

5.5 Časovače v aplikaci

Pomocí časovačů je v aplikaci implementováno mnoho funkcionalit, především pomocí nich můžeme snadno realizovat neblokující čekání, aniž bychom potřebovali relativně složité vícevlaknové programování.

Pod pojmem „časovač“ v tomto kontextu rozumíme nástroj pro naplánování události, která se vyvolá po uběhnutí daného času. Jinými slovy, pokud vytvoříme časovač (timer) nastavený na n milisekund, po uběhnutí n milisekund aplikace vždy vykoná předem daný kód. Velké využití tak časovač má například při ovládání sériové linky.

Nejčastější metody použití časovače jsou následující: [16]

1. Jednorázové spuštění kódu po uplynulé době – funkce obsluhující časovač obsahuje zároveň metodu pro ukončení časovače.

2. Periodické opakování kódu – zapnutí i vypnutí časovače probíhá mimo funkci obsluhující časovač.

3. Kombinace časovačů – funkce obsluhující časovač musí rozlišovat, kterým časovačem byla spuštěna, například pomocí podmínky if nebo case.

Z programátorského hlediska se časovač nastavuje pomocí Windows API, v MFC konkrétně metodou CWnd::SetTimer(nIDEvent, nElapse, lpfnTimer), kde první parametr znamená ID časovače (mělo by být v celé aplikaci unikátní, proto je vhodné si všechny časovače evidovat, například tabulkou jako v podkapitole 5.5.1), dalším parametrem lze nastavit dobu v milisekundách, po které se časovač spustí, a pomocí posledního parametru lze nastavit funkci, které bude časovač obsluhovat. Pokud bude poslední parametr nastaven na NULL, časovač vygeneruje zprávu a umístí ji do fronty zpráv aktuálního objektu.

Spuštění časovače na úrovni kódu probíhá tak, že je vygenerována zpráva WM_TIMER. Z tohoto důvodu je nutné, aby ve třídě, kde má být časovač používán, byla v mapě zpráv tato zpráva definována, například tedy:

```
BEGIN_MESSAGE_MAP(MCLDriver, CDialog)
        ON_WM_TIMER()
END_MESSAGE_MAP()
```

Dále je nutné, aby třída měla deklarovanou a definovanou metodu

```
void OnTimer(UINT_PTR nIDEvent)
```

Tabulka 5.1: Tabulka popisující všechny časovače v aplikaci

ID časovače	Spuštěný z třídy	Popis
1	MCLDriver	Doba mezi jednotlivými aktualizacemi polohy pohybového mechanismu.
2	TableController	Doba mezi aktualizacemi GUI pohybového mechanismu.
3	FluxController	Doba mezi jednotlivými měřeními při monitorování hodnoty proudu.
4	BeamController	Interval pro pohyb stolu při skenování svazku.
5	BeamController	Doba potřebná pro ustálení hodnoty na měřícím zařízení.
10	MCLDriver	Doba potřebná pro odpověď pohybového mechanismu při žádosti o souřadnici X
11	MCLDriver	Doba potřebná pro odpověď pohybového mechanismu při žádosti o souřadnici Y
12	MCLDriver	Přečtení souřadnice Y a zapnutí časovače ID 1

Pokud je v jedné třídě více časovačů, můžeme je od sebe odlišovat pomocí proměnné nIDEvent, která odpovídá ID nastavenému v CWnd::SetTimer(...);

Časovač se po vytvoření samozřejmě dá i deaktivovat, k tomu slouží metoda CWnd::KillTimer(nIDEvent).

5.5.1 Tabulka časovačů v aplikaci

V tabulce 5.1 jsou popsány všechny časovače, které jsou v aplikaci používány. V případě dalšího rozšiřování aplikace je velice žádoucí, aby byla tabulka spravována. Za prvé nedojde ke kolizi časovačů co do unikátnosti jejich ID, za druhé poskytuje přehled, které časovače běží na pozadí při spuštění některé části aplikace.

5.6 Tvorba a správa oken v aplikaci

Aplikace je koncipována tak, že každé dialogové okno obsluhuje své zařízení. Protože těchto zařízení může být (a pravděpodobně také bude) aktivních více najednou, je nutné zajistit, aby jednotlivá okna byla tzv. nemodální.

Standardní modální (nebo též těžké) dialogové okno je grafický ovládací prvek podřízený hlavnímu oknu aplikace. Modální okno vzniká jakožto poto-

mek hlavního okna a běží v režimu, kdy není možné interagovat s rodičovským oknem, dokud nebude modální okno korektně ukončeno. Výhoda tohoto konceptu je zřejmá – programátorovi usnadňuje práci, protože na uživatele vzniká nárok na konkrétní work-flow.

Nicméně přesně toto je záležitost, která není v diplomové práci vyžadována, dokonce je přímo nežádoucí – uživatel potřebuje s více okny pracovat paralelně, protože pomocí nich ovládá jednotlivá měřící zařízení. Z tohoto důvodu je nutné prakticky všechny dialogy v aplikaci vytvářet v nemodálním režimu.

Vytvoření nemodálního dialogového okna může vypadat například takto:

```
BeamController * BeamController = new BeamController(this
    ->tableDriver, this->fluxDriver);
beamController->Create(BEAM_INTERFACE);
beamController->ShowWindow(SW_SHOW);
```

BeamController je v tomto případě třída, která ovládá patřičné rozhraní a konstanta BEAM_INTERFACE je identifikátor rozhraní, které se má zobrazit. To lze vytvořit jednoduše pomocí menu Resources v MSVS.

5.7 Postup přidání dalšího ovladače do aplikace

5.7.1 Vytvoření nového ovladače

V projektu je nutné vytvořit nový .h a .cpp soubor, ve kterém bude napsána implementace vlastního ovladače. Tento soubor bude obsahovat třídu, která bude dědit z generického ovladače pro dané zařízení – bude-li se tedy implementovat nový ovladač pro pohybový mechanismus, třída bude dědit z třídy TableDriver. Jelikož generické ovladače obsahují mnoho abstraktních metod, je nutné všechny tyto metody implementovat.

5.7.2 Zobrazení ovladače v hlavním menu

V dalším kroku je nutné přidat vytvořený ovladač do seznamu ovladačů, aby si ho uživatel mohl vybrat k používání. Seznam ovladačů se nachází na hlavní obrazovce, kterou obsluhuje třída MainForm. Během její inicializace (metoda OnInitDialog) je tedy nutné přidat do seznamu nový ovladač, výsledný kód tak může vypadat například takto:

```
        // Combo box for table driver selection
        this->tableDrivers = (CComboBox *)GetDlgItem(
            IDCO_TABLE_DRIVER);
        this->tableDrivers->AddString(L"MCL Driver");
        this->tableDrivers->AddString(L"Simple Driver");
        this->tableDrivers->AddString(L"New Driver");
        this->tableDrivers->SetCurSel(0);
```

5.7.3 Instancování nového ovladače

V posledním kroku je nutné implementovaný a vybraný ovladač instancovat. Třída MainForm obsahuje metody pro připojení jednotlivých periferií, například pro pohybový mechanismus se jedná o metodu ConnectTable. V této metodě se načte vybraná hodnota z rozbalovacího menu s ovladači, stačí pouze přidat podmínku ověřující název vybraného ovladače a instancovat ho, například tedy:

```
        if (driverSelected == L"New Driver") {
                this->tableDriver = new NewDriver(this->
                   driversCnt++);
                this->tableDriver->Create(NO_INTERFACE);
        }
```

Kapitola 6

Testování

6.1 Testování funkčnosti

Testování funkčnosti bylo prováděno na základě případů užití definovaných v části 3.2. Průchod byl třífázový, přičemž ve všech fázích se prováděly všechny případy užití krok po kroku.

Během první fáze byla aplikace mírně upravena tak, aby mohla být odlaďována pomocí modelových dat. Během této fáze bylo objeveno velké množství chyb a vznikl tedy první jakýsi bug-log. Po zapracování všech oprav se během této fáze zrealizovaly i testy použitelnosti, na jejichž základě bylo uživatelské rozhraní aplikace upraveno tak, aby nepůsobilo zmatečně.

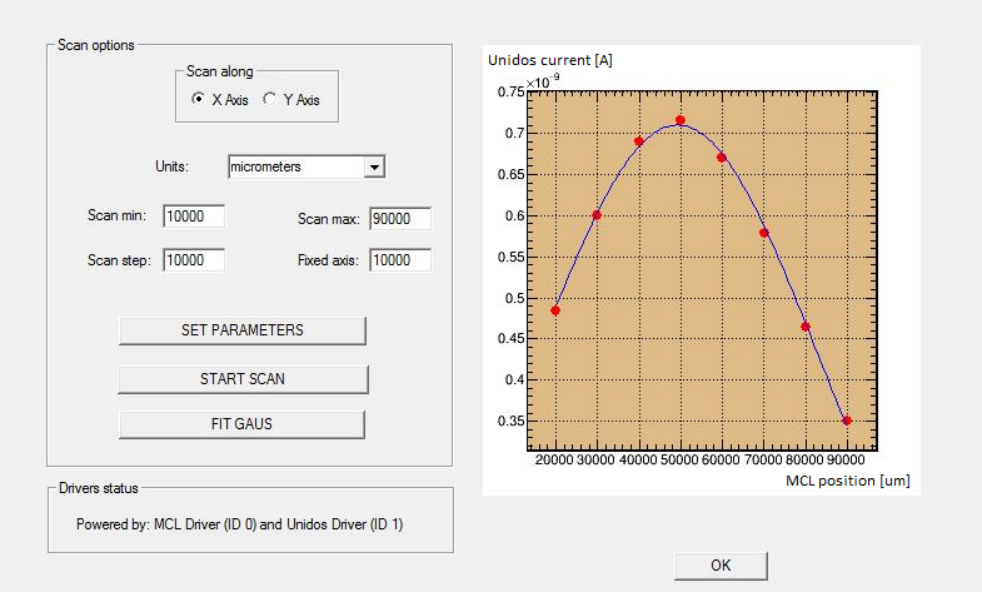
V druhé části byla aplikace poprvé nasazena v reálném prostředí cyklotronu. Bohužel i zde se vyskytlo několik kritických chyb, které se na simulovaných bězích a modelových datech neprojevily. Konkrétně se jednalo o špatnou komunikaci s pohybovým mechanismem – timeout pro čtení ze sériové linky byl příliš malý, problém byl zapříčiněn pomalou komunikací mezi mechanismem a USB serverem, který se na cyklotronu používá. Druhou kritickou chybou bylo špatné prokládání bodů Gaussovou křivkou.

V poslední fázi byla aplikace opět nasazena do reálného provozu a všechny případy užití byly provedeny úspěšně. Aplikace je tak v této fázi bez známých chyb nebo problémů. Tato fáze byla rovněž považována za akceptační testy, které prováděly samy pracovníci ÚJF Řež. Bez větších problémů se byli schopni sami orientovat v aplikaci a akceptační testy tak byly úspěšné, aplikace se plánuje nasadit zhruba od března roku 2016.

Ukázky z provádění akceptačních testů jsou na obrázcích 6.1 a 6.2.

6.2 Kontrola splnění funkčních požadavků

V tabulce 6.1 je provedena kontrola splnění funkčních požadavků. Jak je vidět, každý funkční požadavek je pokryt některým případem užití, a jelikož všechny



Obrázek 6.1: Snímek z testování skenování svazku.

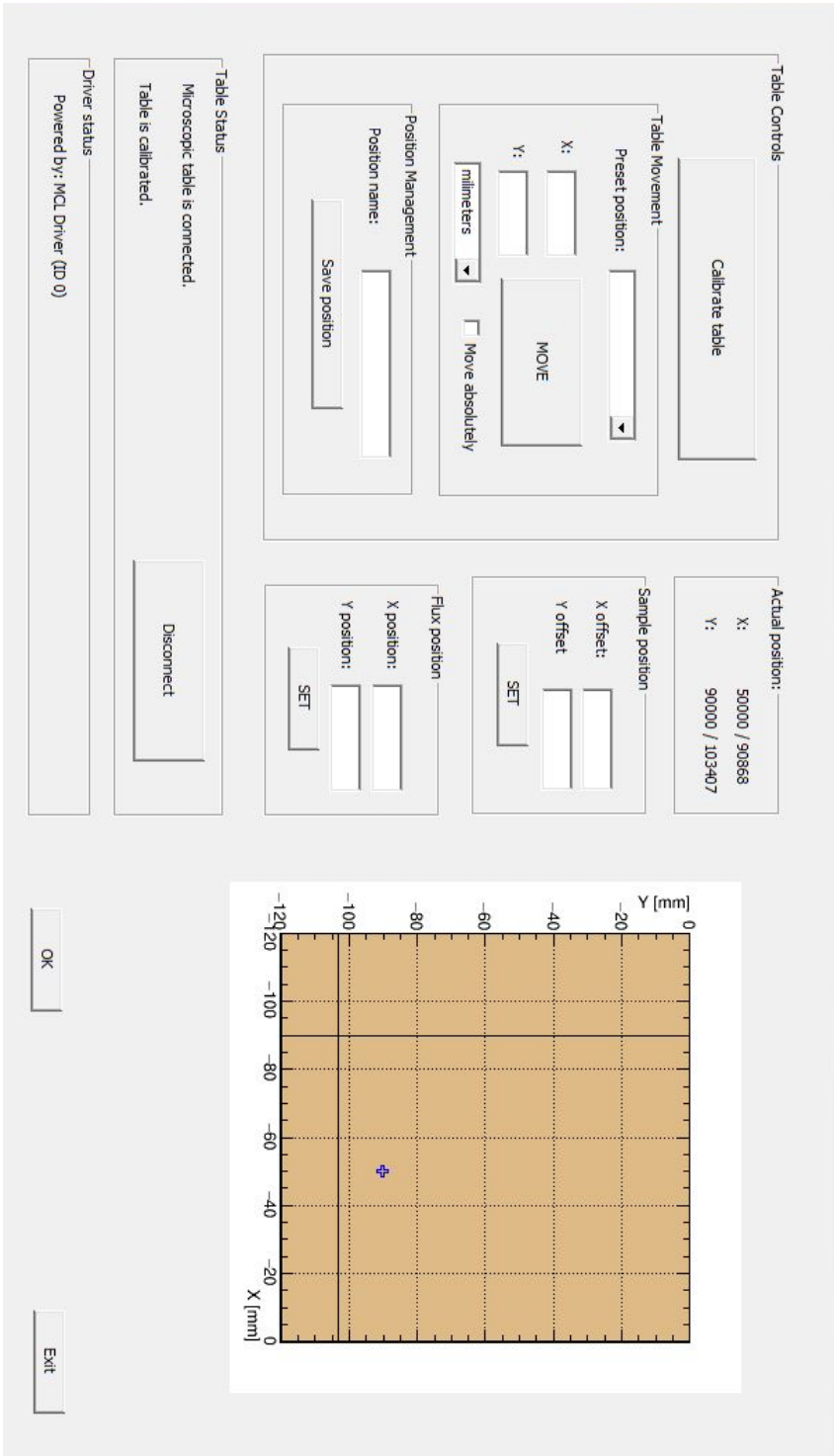
Tabulka 6.1: Pokrytí funkčních požadavků případy užití

Funkční požadavek	Pokryt případy užití	Funkční požadavek	Pokryt případy užití
F1	3.2.1, 3.2.2	F3.5	3.2.4
F2	3.2.3	F3.6	3.2.6
F3.1	3.2.5	F4.1	3.2.11, 3.2.12
F3.2	3.2.6	F4.2	3.2.11
F3.3	3.2.4	F4.3	3.2.11
F3.4	3.2.7	F4.4	3.2.17, 3.2.18

případy užití prošly akceptačními testy, tak je patrné, že se všechny požadavky podařilo implementovat úspěšně.

6.3 Měření hardwarových nároků

Veškeré testování bylo prováděno na konfiguraci 4 × 2,5GHz, 8 GB RAM paměti s operačním systémem Windows 7 Home Premium 64 bit. Testování bylo realizováno pomocí vestavěných nástrojů Microsoft Visual Studia.



Obrázek 6.2: Snímek z testování pohybového mechanismu.

Tabulka 6.2: Tabulka paměťových nároků – MCL je rozhraní pro pohybový mechanismus, MP rozhraní pro měření proudu a SS rozhraní pro skenování svazku

Spuštěná rozhraní	Spotřeba paměti [MB]
Pouze hlavní menu	25
Menu + MCL	27
Menu + MCL + MP	31
Menu + MCL + MP + SS	32
Menu + 3 × MCL + 2×MP + SS	43

6.3.1 Paměťová náročnost

První hardwarový test byl zaměřen na testování paměťových nároků. Na první pohled se můžou naměřené hodnoty zdát poněkud veliké, ovšem jsou způsobeny především používáním knihovny MFC a grafického rozhraní. Z tabulky 6.2 je patrné že nejvíce náročné je rozhraní pro průběžné měření hodnot proudu – jeho první instance potřebuje zhruba 4 MB operační paměti. Na druhou stranu nejméně paměti pak potřebuje rozhraní pro skenování svazku (1 MB), což odpovídá předpokladům, protože obsahuje nejméně ovládacích prvků.

V závěru měření jsem se rozhodl provést test, kdy bylo zároveň otevřeno několik rozhraní pro každou periferii. Jedná se o extrémní případ, který pravděpodobně nebude v praxi nikdy překročen.

6.3.2 Vytížení CPU

Druhý test z hlediska hardwarové náročnosti se týkal vytížení procesoru. Zkoumány byly především tři hlavní atributy – vytížení procesoru během spuštěného programu na pozadí, zatížení při provádění běžných operací a maximální vytížení, které program způsobil.

Příjemným překvapením bylo, že nebylo-li s programem manipulováno, nezatěžoval procesor téměř vůbec, potřeboval pouze 0–1 % celkové kapacity všech jader, a to dokonce i v okamžicích, kdy prováděl průběžnou akci, jako například skenování.

Standardní byla i hodnota vytížení CPU během provádění běžných operací (přihlášení, otevření nového rozhraní atp.). Zde bylo potřeba zhruba 4 % kapacity procesoru.

Největší naměřená hodnota pak byla 20 %. Ta nastala v momentě, kdy byl vytvořen požadavek na otevření více rozhraní najednou v jednom okamžiku, navíc na pozadí běželo již několik časovačů. Tato hodnota byla ovšem velice krátkodobá a po zpracování požadavku opět klesla pod 5 %. Navíc takovéto chování běžného uživatele není v provozu prakticky možné, takže se jedná spíše o teoretickou hodnotu.

Kapitola 7

Manažersko-ekonomické zhodnocení

7.1 Ekonomické zhodnocení

Tato diplomová práce nemá za cíl vývoj systému, který by měl být použit komerčně, nelze tedy mluvit o přímých ziscích z jeho prodeje a podobně. Ústav jaderné fyziky spadá pod Akademii věd ČR, což je organizační složka státu a jejím vrcholným cílem je provádění vědeckého výzkumu.

Aplikace generuje úspory především v časové náročnosti měření. Zbylý čas mohou pracovníci věnovat jiným činnostem a ÚJF tak šetří na platech zaměstnanců. Informační systém zastává roli zhruba jednoho pracovníka na třetinový úvazek, což pro ÚJF znamená úsporu zhruba **7 000,– Kč měsíčně**.

7.2 Náklady na licence a hardwarové zdroje

K úspěšnému nasazení systému stačí běžný notebook s operačním systémem Windows 7. Nebylo tedy nutné dokupovat žádný další speciální hardware, s výjimkou USB2.0 – RS 232 převodníku, jehož cena se pohybuje kolem 250 Kč. Dále pak je potřeba NULL modem, který stojí zhruba 90 Kč. Velice nákladnou položkou jsou samotné ovládané periferie, ovšem ty nelze do ceny vývoje započítat, protože slouží k mnoha dalším účelům.

Stejně tak nelze do celkových nákladů započítat softwarové licence, protože ÚJF má volné notebooky s Windows 7 k dispozici, nebylo tedy nutné dokupovat žádné další. K vývoji byla použita Community Edition Visual Studia, která je pro výzkumné účely zdarma. Další softwarové prostředky potřebné pro vývoj byly freeware.

7.3 Náklady na vývoj aplikace

Aplikace byla vyvíjena v rámci třetinového úvazku po dobu deseti měsíců, každý měsíc vývoje stál ÚJF 7 000 Kč. Tato doba odpovídá odhadem 400 hodinám, čili 50 MD (man-dayů). Je zřejmé, že vývoj aplikace v rámci diplomové práce je oproti tržním cenám vývojářů značně levnější, protože tyto ceny se pohybují zhruba 3 500 Kč / MD. Úspora na vývoji v rámci diplomové práce oproti běžným vývojářům tedy činí:

$$TP = (3\ 500 \times 50) - (7\ 000 \times 10) = 105\ 000\ K\check{c}$$

7.4 Ekonomické ukazatele

Celkový příjem z aplikace je tedy v rámci úspory časů pracovníků odhadnut zhruba na 7 000 Kč měsíčně, tj. 84 000 Kč ročně. Celkové náklady na vývoj systému pak činí:

$$TC = 7\ 000 \times 10 + 250 + 90 = 70\ 340\ K\check{c} \tag{7.1}$$

Protože rizikovost projektu je poměrně malá, stejně jako inflace a náklady na pořízení potřebného kapitálu, diskontní míru projektu lze nastavit na i = 5 %. Odhadovaná životnost projektu jsou tři roky (N = 3), během nultého roku byla aplikace vyvíjena. Z těchto informací můžeme určit ukazatel NPV (net present value, čistá současná hodnota) a rozhodnout tak o ekonomické úspěšnosti projektu:

$$NPV = \sum_{t=0}^{N} \frac{CF_t}{(1+i)^t} = -70\ 340 + 80\ 000 + 76\ 190 + 72\ 562 = 158\ 412\ K\check{c} \tag{7.2}$$

Jak vidíme, čistá současná hodnota projektu je kladná, čili investice je pro ÚJF výhodná. Pro detailní zhodnocení projektu bychom ovšem potřebovali znát i jiné investice probíhající v ÚJF a zvážit i ekonomický zisk projektu, ne jen účetní.

7.5 Shrnutí

Jak bylo zmíněno v úvodu této kapitoly, ekonomickou analýzu v rámci této diplomové práce můžeme považovat pouze za doprovodný ukazatel úspěšnosti projektu. Jeho hlavním smyslem je úspora času zaměstnanců, zvýšení pohodlnosti měření a obecné zjednodušení práce. Ačkoliv lze většinu z těchto faktorů víceméně ekonomicky zhodnotit, stále se bude jednat o odhady.

Závěr

Před samotnou implementací bylo nutné jednoznačně stanovit všechny funkcionality, které aplikace musí bezpodmínečně splňovat v termínu odevzdání. Všechny tyto požadavky byly zaznamenány a uvedeny v kapitole 2. V tomto ohledu lze konstatovat, že všechny požadavky byly implementovány úspěšně a v rozsahu zadání. Úspěšně byly realizovány i akceptační testy a vyvinutý software bude nasazen začátkem března 2016.

Během implementace bylo vznášeno několik dalších požadavků nad rámec původního zadání. Tyto rozšíření aplikace byly zaznamenány ve sdíleném dokumentu a jejich postupná implementace, která bude opět probíhat v rámci částečného úvazku v ÚJF Řež, je plánována na období březen–květen 2016, takže aplikaci odevzdáním neopouštím.

Během práce na diplomové práci se vyskytlo několik problémů, především během implementace. Poměrně složité bylo například zvolení vhodných nástrojů pro implementaci, protože jak se ukázalo, spojit framework ROOT s jiným projektem nebyl příliš triviální úkol. Naštěstí se vše podařilo vyřešit díky odborným radám samotných vývojářů z výzkumného pracoviště v CERNu. Další zajímavou výzvou byla neblokující implementace sériové linky nebo prokládání naměřených bodů Gaussovou křivkou. Všechny tyto záležitosti nejsou triviální a nepodařilo se mi je najít v žádných dostupných zdrojích, proto by vybrané kapitoly této práce mohly posloužit i během realizace jiných projektů. Vývoj aplikace zpočátku brzdila i nezkušenost s knihovnou MFC a jinými technologiemi Microsoftu.

Naopak některé části byly hotové podstatně dříve, než bych očekával. Například samotné ovládání periferií – jejich technická dokumentace byla kvalitní a přehledná. Navíc některé podproblémy byly částečně vyřešeny v aplikaci, kterou ÚJF aktuálně využívá. Velice jednoduchá byla i implementace GUI, byť jsem byl s Microsoft Visual Studiem v tomto ohledu mírně zklamán, naštěstí uživatelské rozhraní není příliš komplikované.

Úplným závěrem bych rád poděkoval všech pracovníkům ÚJF Řež, se kterými jsem měl díky této diplomové práci možnost se setkat. Jejich odborné

znalosti byly během vývoje informačního systému nezbytné, navíc jsem se setkal pouze se vstřícným a ochotným jednáním.

Literatura

[1] Ústav jaderné fyziky AV ČR, v.v.i.: *Stručně o ÚJF [online]*. September 2010, [cit. 2016-01-09]. Dostupné z: `http://www.ujf.cas.cz/`

[2] Náplava, P.; Pergl, R.: Empirical Study of Applying the DEMO Method for Improving BPMN Process Models in Academic Environment. 2015, [cit. 2016-01-09].

[3] Land, M. O.; Dietz, J. L. G.: Benefits of Enterprise Ontology in Governing Complex Enterprise Transformations. ročník 110, 2012: s. 77–92, [cit. 2016-01-09].

[4] Dietz, J. L. G.: Demo basis – Glossary of terms. 2014, [cit. 2016-01-09].

[5] Microsoft Corporation: *Visual Studio Dev Essentials [online]*. 2015, [cit. 2016-01-09]. Dostupné z: `https://www.visualstudio.com/products/visual-studio-dev-essentials-vs`

[6] Procise, J.: *Programming Windows with MFC*. Microsoft Press, 1999, ISBN 9781572316959, [cit. 2016-01-09].

[7] Bancila, M.: MFC Feature Pack: An Introduction [online]. March 2008, [cit. 2016-01-09]. Dostupné z: `http://www.codeguru.com/cpp/cpp/cpp_mfc/tutorials/article.php/c14929/MFC-Feature-Pack-An-Introduction.htm`

[8] ROOT Framework: *About ROOT [online]*. 2015, [cit. 2016-01-09]. Dostupné z: `https://root.cern.ch/about-root`

[9] Peacock, C.: Interfacing the Serial / RS232 Port [online]. October 2010, [cit. 2016-01-09]. Dostupné z: `http://retired.beyondlogic.org/serial/serial.htm`

[10] Olmr, V.: HW server představuje – Sériová linka RS-232 [online]. December 2005, [cit. 2016-01-09]. Dostupné z: http://vyvoj.hw.cz/rozhrani/hw-server-predstavuje-seriova-linka-rs-232.html

[11] Demo World: *About DEMOworld [online]*. 2013, [cit. 2016-01-09]. Dostupné z: https://www.demoworld.nl/Portal/Home/About

[12] ROOT Framework: *CINT [online]*. 2015, [cit. 2016-01-09]. Dostupné z: https://root.cern.ch/introduction-cint

[13] Germany, C.: Hills of Darkness 3.0 [online]. 2010, [cit. 2016-01-09]. Dostupné z: http://www.networkingprogramming.com/1024x768/MFC_HOD2010.html

[14] de Klein, R.: Serial library for C++ [online]. November 2003, [cit. 2016-01-09]. Dostupné z: http://www.codeproject.com/Articles/992/Serial-library-for-C

[15] Ford, S.: *266 tipů a triků pro Microsoft Visual Studio*. Computer press, 2009, ISBN 9788025125540, [cit. 2016-01-09].

[16] Programmer Share: *The MFC timer - millisecond [online]*. July 2012, [cit. 2016-01-09]. Dostupné z: http://www.programmershare.com/2396903/

Příloha **A**

Instalační a uživatelská příručka

A.1 Instalace aplikace

Instalace programu se provede zkopírováním obsahu adresáře „exe“ na libovolné místo v počítači. V tomto adresáři jsou rovněž všechny potřebné dynamické knihovny, které jsou ke spuštění potřeba. Ty je možné ponechat v adresáři s aplikací, nebo je překopírovat do systému, čímž budou dostupné pro všechny aplikace nainstalované v počítači.

V tento okamžik by aplikace měla jít spustit, ovšem bez funkčních ROOT funkcí. K tomu je nutné nainstalovat ROOT, jehož instalační soubor je přiložen v adresáři ROOT. Po tomto kroku by aplikace měla být 100% funkční.

A.2 Instalace databáze

V aktuální verzi je ke korektnímu logování dat potřeba mít nainstalovaný MS Access. Je vhodné otevřít soubor RezControllerDB.mdb a zkusit do něho vepsat nějaké testovací hodnoty. Pokud je soubor z nějakého důvodu v programu nepřístupný, bude většina pokusů o zápis dat do databáze vracet chybu „Nesoulad datových typů“. Přístupnost může být blokována například z důvodu stažení souboru z internetu – v Office je nutné zapisování do stažených souborů explicitně povolit.

A.3 Provoz aplikace bez připojených periferií

Do aplikace je ze všeho nejdřív nutné se přihlásit, protože pouze přihlášený uživatel může vytvářet ovladače a poté otevírat ovládací rozhraní. V distribuci na CD obsahuje aplikace jeden správcovský účet, pomocí něhož lze do aplikace přidat další. Uživatelské jméno je „TestAdmin“ a heslo „admin123“.

Bohužel testování aplikace je bez všech periferií prakticky bezvýznamné. V současné verzi aplikace obsahuje MCL Driver ovládající pohybový mechanis-

mus MCL a Unidos Driver pro ovládání Unidosu. Oba ovladače umí v omezené míře detekovat, zdali je k nim připojen správný hardware, v opačném případě při pokusu otevření rozhraní pro ovládání periferie aplikace oznámí uživatele, že ovladač není připojen. Z tohoto důvodu je v aplikaci implementován i Simple Driver, pomocí něhož lze otevřít alespoň rozhraní pro pohybový mechanismus, ačkoliv je bez periferie samozřejmě nefunkční.

Během otevírání několika rozhraní pro stejnou periferii může program občas potřebovat znovu zadat umístění databázové souboru. V tomto případě je nutné v dialogu pouze znovu vybrat soubor RezContollerDB.mdb.

Příloha B

Seznam použitých zkratek

API Application programming interface

DCE Data communications equipment

DEMO Design and Engineering Method for Organisations

DLL Dynamic-link library

DTE Data terminal equipment

GUI Graphical user interface

MFC Microsoft Foundation Class

MSVS Microsoft Visual Studio

OCD Organization Construction Diagram

PC Personal computer

PSD Process Structure Diagram

ÚJF Ústav jaderné fyziky

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `root` ..... instalační program frameworku ROOT
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `images` ..... obrázky použité v diplomové práci
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF