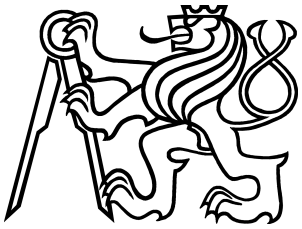


ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ

Fakulta elektrotechnická

Diplomová práce

Archiv experimentů

Jméno : Jan Lehký

Vedoucí : Ing. Libor Waszniowský

červen 2008

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky *Školní rok:* 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Jan Lehký

Obor: Technická kybernetika

Název tématu: Archiv experimentů

Zásady pro vypracování:

1. Seznamte se s možnostmi vytváření grafického uživatelského rozhraní v programu Matlab, s prací s databázemi a vytvářením aplikací pro platformu .Net.
2. Navrhněte a implementujte archiv experimentů provedených v Matlabu, případně v Hardware in the Loop simulaci. Archiv bude umožňovat ukládat uživatelem vybraná vstupní a výstupní data, poznámky a modely.
3. Navrhněte a implementujte samostatné grafické rozhraní, které i bez Matlabu bude umožňovat parametrizovat a ovládat simulátor pro Hardware in the Loop simulaci a pracovat s archivem.
4. Navrhněte a implementujte způsob prohledávání experimentů uložených v archivu podle uživatelem zadaných parametrů.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Libor Waszniowski, Ph.D.

Termín zadání diplomové práce: zimní semestr 2006/2007

Termín odevzdání diplomové práce: leden 2008

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze dne 21.02.2007

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 22. 5. 2008 .....

.....
podpis

Poděkování

Tímto bych chtěl poděkovat ing. Liboru Waszniowskému za odbornou pomoc a cenné rady při vypracování této diplomové práci a jeho trpělivost. Dále bych chtěl poděkovat svým rodičům za podporu kterou mi poskytli za celou dobu studia.

Abstrakt

Tato diplomová práce se zabývá vytvořením uživatelského rozhraní, které bude použito jako součást projektu pro firmu Aero Vodochody.

Uživatelské rozhraní umožňuje archivaci a zpracování dat získaných z programu Matlab za pomocí COM rozhraní a jeho Automation serveru. Data jsou ukládána do databáze PostgreSQL, do databáze je také možné uložit jakýkoli soubor např. nastavení experimetnu, dokumentaci, grafy.

Pro práci z daty je zde připravena možnost vykreslení libovolných grafů z naměřených dat, jejich porovnání a ukládání na disk za pomocí jednoduchých nástrojů. Dále je možné pomocí tohoto nástroje generovat částečné dokumenty, které po doplnění mohou sloužit jako reporty ke změřeným experimentům. Větší část tohoto programu je vytvořena v jazyce C#, v programu Matlab jsou pak implementovány funkce pro import a export dat.

Abstract

This diploma thesis implements a user interface, which will be used as a part of the project for company Aero Vodochody.

Most of this program is writen in C# programing language, but there are import and export functions implemented in Matlab program. This user interface allows archivation and manipulation with data from Matlab by using its COM interface and its Automation server. Data are stored in PostgreSQL database, it is posible to store files into the database as setting of experiment, documentation, graphs.

There are an easy to use and intuitive tools to work with data in the user interface. It is possible to plot and compare this data and save figures to the disk if it is needed. Word document can be generated by this tool. This documents can be used as reports or presentation of experiments.

Obsah

Seznam obrázků

Kapitola 1

Úvod

1.1 Představení problému a motivace

Při návrhu nového řídícího systému nebo i vylepšení stávající funkčnosti zařízení se setkáváme s tím, že v různých fázích vývoje jsou testovány vlastnosti navržené pomocí softwarových prostředků jako je například Matlab, Mathematica, Maple na reálném hardwaru. Z každého měření mohou být výstupem velké množství dat, která je někdy nutné, jindy spíše vhodné dobře uchovat pro pozdější kontrolu. Velice praktické se může jevit porovnání nových výsledků s výsledky dosaženými před delší dobou, abychom viděli jestli se vývoj ubírá správným směrem a dochází k požadovanému zlepšení.

Porovnání výsledků nemusí být zrovna snadné pokud není napsán kvalitní report z předchozího měření. Po delší době nemusí být zřejmé s jakým nastavením, vstupními hodnotami a za jakých podmínek bylo toto měření provedeno. Přitom zanedbání některého faktoru můžu vést k naprosto mylnému úsudku a výsledku předchozího testu. Bez odpovídajícího softwarového vybavení je tak kladen velký důraz na kvalitu a zodpovědnost pracovníků, aby tyto skutčnosti nezanedbali.

Proto může být návrh vhodného software pro uchování staršího nastavení, vstupních hodnot, výsledků přínosem pro vývoj. Tento software může být navržen obecně pro uchování jakýchkoli dat nebo navržen specificky pro zadané prostředí, pak ovšem může dosahovat lepších výsleků, lepšího komfortu pro obsluhu a větších rychlostí nežli obecné řešení.

V průběhu vývoje je také většinou nutné podávat zprávy o postupu a zlepšení dosažených vývojem. Software, který uchovává výsledky a je schopen je snadno a rychle exportovat do námi psaného reportu, dokáže zrychlit naši práci a tak i částečně odstranit repetitivní práci vývojáře a ponechat většinu jeho času na opravdu důležité úkoly.

1.2 Řešení problému

Tato práce je součástí projektu pro firmu Aero Vodochody, čímž je dána základní věc, a to zadroj dat a pracovní prostředí, v našem případě je to program Matlab. Proto už od začátku musíme uvažovat s tím, že vytvořený program musí být schopen komunikovat s programem Matlab, získávat z něho data a v případě potřeby je do Matlabu zpátky exportovat.

Pro uchovávní všech dat získaných měřením je dobré použít databázi, ve které jsou data uchována v požadovaném formátu a kvalitě, je také možné k ní přistupovat z více míst, čímž se vyhneme možnému zmatku se soubory a jejich aktuálním umístěním.

V databázi tak budou uchována data, která budou obsahovat nastavení systému, hodnoty vstupních a výstupních proměnných, jejich názvy a další pomocné proměnné. Tyto data jsou importována z programu Matlab do hlavního uživatelského rozhraní k dalšímu zpracování. Stejná data je pak možno exportovat zpět do programu Matlab a uskutečnit tak experiment se stejnými parametry.

V uživatelském rozhraní bude pak možné procházet tyto změřené data, jejich vykreslení a porovnání za pomocí využití funkcí obsažených v COM Automation serveru programu Matlab. Dále bude možné generovat dokumenty, vkládat do nich grafy generované tímto nástrojem ze změřených dat.

Jestliže budeme schopni generovat reporty s definovaným formátem, které budou moci sloužit buďto jako dokumentace k měření nebo jako prezentace výsledků, vyřešíme tím další z problémů uvedených výše.

Z těchto poznatků a dalšího zamyšlení vznikla architektura popsaná v části 3.

Kapitola 2

Použité technologie

Tato část se věnuje popisu použitých technologií, především je představena práce s komponentami COM jak obecně, tak její specifikace pro Matlab a Microsoft Word, které jsou spolu s kódem napsaném v jazyce C# hlavními stavebnímy kameny celé aplikace. Dále je jedna část věnována krátkému představení databáze PostgreSQL.

2.1 Obecný přehled komponent COM

COM je zkratka pro Component Object Model (model komponentových objektů). Je to rozhraní, které umožňuje komunikovat a vytvářet dynamické modely programům napsaných v jakémkoli programovacím jazyku, který podporuje tuto technologii. COM bylo vytvořeno firmou Microsoft v roce 1993.

Hlavním přínosem tohoto rozhraní je, že je jazykově neutrální, umožňuje použití kódu napsaného v jednom jazyce, použít v jazyce jiném aniž by došlo k jakémukoli problému při vzájemné interakci. Také při využití nepožaduje jakoukoli znalost vnitřního kódu k tomu, aby byly vlastnosti námi použitého COM objektu použity. Aby toto bylo zaručeno je nutné při psaní COM objektů dodržovat daná pravidla, implementace rozhraní *IUnknown* což znamená, že každý COM objekt je odvozen od *IUnknown*. Některé COM objekty implementují rozhraní *IDispatch* (Automation), které je odvozeno od *IUnknown* a doplňuje ho o další metody *GetTypeInfoCount()*, *GetTypeInfo()*, *GetIDsOfNames()* a *Invoke()*.

Každá z COM komponent musí implementovat tři metody dané rozhraním *IUnknow*. *AddRef()* a *Release()* slouží k počítání odkazů na daný objekt, čímž je ovládána životnost daného objektu v paměti, pokud počet odkazů klesne na nulu, může být objekt odstraněn. Každý objekt je zodpovědný za vyčištění jím využité paměti, pokud reference na tento objekt klesnou na nulu. Další metodou je *QueryInterface()*, která slouží k vyhledávání rozhraní, která jsou implementována v komponentě.

Metody obsažené v IDispatch dovolují klientské aplikaci zjistit jaké vlastnosti a metody jsou podporovány objektem za běhu programu. Dále obsahují informace nutné

k volání těchto vlastností a metod.

Třídy v komponentech jsou identifikovány pomocí GUID (global unique identfier). Při zavedení komponenty je textová podoba GUID zapsána do systémového registru. Záznam v registru obsahuje informace o umístění komponenty a její další atributy.

Vazby na komponenty COM

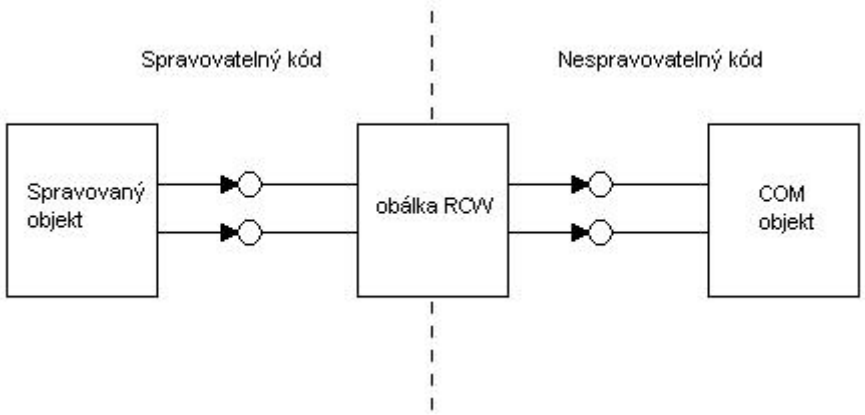
Abychom mohli danou komponentu používat je potřeba o ní vědět alespoň základní údaje. Tyto údaje je možné získat dvěma způsoby.

Časná vazba - umožňuje získat informace o komponentě již v době překladu. Tento způsob umožňuje přísnější kontrolu typů při překladu aplikace, čímž je zaručena lepší bezpečnost aplikace. Další výhodou je rychlost použití než při využití druhého způsobu.

Pozdní vazba - při tomto způsobu použití může i malý překlep při psaní kódu způsobit chybu za běhu programu. Spoléháme na to, že námi použité metody jsou v rozhraní opravdu použity a nemáme předchozí kontrolu. Pozdní vazby mohou být také velmi pomalé.

Obálky komponent

V případě aplikace založené na platformě .NET Framework, není komunikace přímá, ale je přes spravovanou obálku komponenty. Obálka je označována jako RCW(Runtime-Callable Wrapper) a chová se jako jakési proxy pro nespravovanou komponentu COM. Obálka ošetřuje volání a předávání dat mezi spravovanou aplikací a nespravovaným kódem COM komponenty, princip je naznačen na obr.2.1.

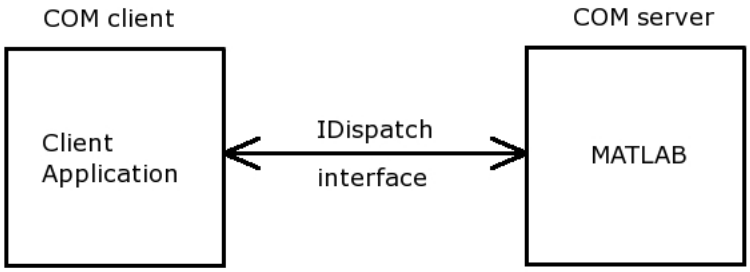


Obrázek 2.1: COM obálka

Obálka je při využití prostředí Visual Studio .NET generována automaticky a v kódu jsou pak volány metody COM již ve spravovaných obálkách. Cílem volání tedy není přímo COM komponenta, ale právě obálka této komponenty. Ve vygenerované obálce jsou obsaženy všechny třídy, které jsou k dispozici pro jazyk C#. Obálka rovněž zajišťuje převod správných typů na typy komponenty COM a následný zpětný převod.

2.2 COM rozhraní Matlab

Jak je popsáno výše, COM rozhraní slouží ke komunikaci mezi aplikacemi. Matlab může být použit dvěma způsoby, jako COM client, pak Matlab aplikace ovládá jinou aplikaci, která v tu chvíli slouží jako COM server nebo aplikace ovládá Matlab a pak se chová jako COM server. Matlab COM rozhraní implementuje rozhraní *IDispatch,* tím se řadí mezi komponenty zvané také automation. Komunikace dvou programů využívajících COM rozhraní matlabu je na obr.2.2



Obrázek 2.2: Matlab COM server

COM rozhraní Matlabu umožňuje mnoho možností jako je např. využití ActiveX pro použití programu Internet Explorer v okně Matlabu, spolupráce s programem MS Excel atd.

Pro spuštění prvku jakéhokoli COM prvku je použit příkaz : *server = actxserver(prametr).*

Jelikož je téma COM rozhraní Matlabu rozsáhlé a větší část se netýká přímo mé diplomové práce budu se dále věnovat pouze COM Automation serveru.

Po provedení příkazu uvedeného výše s parametrem v tabulce 2.1 je otevřeno okno serveru, vlastnosti Automation serveru jsou nastaveny podle parametru v příkazu. Vlastnosti se týkají pouze samotného Automation serveru, ale přístup k němu a použitelné příkazy jsou u každého totožné.

Parametr	**Vlastnosti COM serveru**
matlab.applocation	spustí nové okno s příkazovou řádkou Matlabu ve verzi v jaké byl naposledy spuštěn COM server
matlab.autoserver	spustí nové okno s příkazovou řádkou v nejvyšší verzi nainstalovaného Matlabu
matlab.desktop.application	spustí nové kompletní okno Matlabu v jeho nejvyšší verzi

Tabulka 2.1: Parametry pro spuštění Automation serveru

Matlab Automation server poskytuje možnosti k vykonávání standardních příkazů ve workspace serveru, přenos proměnných a celých matic mezi pracovními prostory

hlavního programu do pracovního prostoru Automation serveru a zpět, vzdálené spouštění funkcí. Přehled hlavních funkcí je uveden dále.

server.Execute('command') v prostoru serveru je vykonán standardní Matlab příkaz uvedený v parametru command

result=server.Feval('function',numout,x1,...,xn) v prostoru serveru je vykonána funkce specifikovaná parametrem function, numout udává počet výstupů, které funkce vrací, x1 až x2 jsou parametry vstupující do funkce

server.PutWorkspaceData('varname','workspace',data) slouží k přenosu dat z aktuálního ('current') nebo ze základního ('base') workspace do workspace Automation serveru. Při spuštění toho příkazu v hlavním okně Matlabu jsou tyto prostory stejné, ale při použití v .m souboru jsou tyto dva prostory odlišeny. Parametr varname specifikuje jaké jméno bude mít proměnná ve workspace Automation serveru, data pak název přenášené proměnné ze specifikovaného workspace.

server.PutFullMatrix('varname','workspace',xreal,ximag) do prostoru serveru je přenesena matice se jménem definovaným v parametru varname z pracovního prostoru definovaného parametrem workspace. Parametry xreal a ximag jsou hodnoty reálné a imaginární části přenášené matice.

server.GetWorkspaceData('varname','workspace') tento příkaz slouží k přenosu dat opačným směrem nežli výše uvedený příkaz PutWorkspaceData. Parametr varname specifikuje jméno přenášené proměnné z workspace Automation serveru, wokspace pak kam bude proměnná uložena.

[xreal,ximag]=server.GettFullMatrix('varname','workspace',zreal,zimag) matice se jménem uvedeným ve varname je přenesena z Automation serveru do prostoru definovaném parametrem workspace. Parametry zreal a zimag jsou stejného rozměru jako rozměr přenášené matice a jsou většinou nulové.

Pro použití těchto příkazů v programu napsaného v prostředí Visual Studio musí být vložena příslušná knihovna Matlabu pro COM server. V jazyce C# je vytvořen Automation server pomocí příkazu *matlab = new MLApp.MLAppClass(),* pokud byl již vytvořen Automation server jiným programem, je pomocí tohoto příkazu navázáno spojení s tímto serverem a můžeme začít s tímto spojením pracovat. Pomocí proměnné *matlab* se odkazujeme na referenci daného Automation serveru a můžeme vykonávat příkazy uvedené výše.

2.3 COM rozhraní MS Word

Firma Microsoft vytvořila COM rozhraní pro celou rodinu produktů Office, aby mohly být ovládány jejich prvky z jiných aplikací a mohlo docházet ke komunikaci mezi těmito programy a námi psanými aplikacemi.

Opět se zaměříme na část využitou v této diplomové práci a to COM rozhraní pro MS Word.

Většina potřebných funkcí a vlastností je uložena v knihovně *Micrsoft.Office.Core* a dále knihovně *Word*. Dále je uvedeno jak je možné programově otevřít, vytvořit nový dokument, vkládání textů, obrázků a tabulek. Programově je také možno nastavit vlastnosti dokumentu, jeho vzhled a využít eventu generovaných v dokumentu k dalšímu chodu aplikace.

Navázání komunikace s aplikací MS Word

Pro navázání spojení je nutné vytvořit nový objekt pomocí příkazu *wrdClass = Microsoft.Office.Interop.Word.ApplicationClass()*, který vrací objekt typu Microsoft.Office.Interop.Word. V tuto chvíli je vytvořen objekt *wrdClass,* který je schopen ovládat MS Word dokumenty, ale zatím nemáme aktivní odkaz na konkrétní dokument.

Základní práce s dokumentem - založení, otevření, uložení

Pro práci s konkrétním dokumentem je nutné alokovat objekt jak je popsáno výše.

wrdDoc = wrdClass.Documents.Open(ref fileName, ref ConfirmConversions, ..., ref XMLTransform) - otevírá již existující soubor uložený v podporovaném formátu se jménem uvedeným v parametru fileName, další parametry modifikují metody přístupu k dokumentu, heslo, formát atd., celá deklarace je uvedena v dokumentaci.

wrdDoc = wrdClass.Documents.Add(ref Template, ref NewTemplate, ref DocumentType, ref Visible) - vytváří zcela nový dokument, parametry udávají jaký vzorový dokument je použit nebo zda ho otevřít jako vzor, jakého typu je daný dokument, např. prázdný, e-mail. Visible udává jestli je dokument otevřen ve viditelném okně.

wrdDoc.Save(ref NoPrompt, ref OriginalFormat) - uloží již existující soubor. NoPromt udává jestli Word ukládá všechny dokumenty automaticky nebo jestli je musí uložit uživatel. OriginalFormat udává v jakém formátu bude dokument uložen.

wrdDoc.SaveAs(ref fileName, ref FileFormat, ..., ref AddBiDiMarks) - uloží dokument do souboru uvedeného v parametru fileName. Zbývající parametry udávají způsob uložení, heslo, jestli má být uložen jako read-only atd.

Ve všech výše uvedených metodách je možné větší část jejích parametrů vynechat, pak je zvolena přednastavená hodnota.

Vkládání textu do dokumentu

Pro přidávání textu do dokumentu je možné zvolit dvě cesty.

První možností je využití objektu *Range* - před psaním textu musíme určit v jaké části a délce bude Range definován, jeho první a poslední index. Poté je možné do tohoto rozmezí vložit text, pokud v tomto rozmezí již text je, bude nahrazen novým. Jelikož jsou v objektu wrdDoc, který ukazuje na otevřený dokument, obsaženy informace o členění na stránky, odstavce, věty a jejich příslušný počet je možné přesně definovat začátek, kde bude text vepsán.

Range rng = wrdDoc.Range(ref start, ref end) - vytvoří nový objekt rng v otevřeném dokumentu se začátkem/koncem udaných v parametrech start/end.

Range rng = wrdDoc.Section[1].Range - vytvoří objekt rng, který bude prvním znakem a délkou shodný s rozsahem první strany v dokumentu.

rng.Text = "Hello, world" - přidá do dokumentu na místo definované objektem rng text Hello, world

Druhou možností je použít objekt *Selection* - pro každý dokument může být definován pouze jeden objekt selection, jedná se v podstatě o pozici kurzoru v dokumentu. Proto je dobré při vkládání textu do již otevřeného dokumentu zkontrolovat opravdovou pozici a jestli není v tuto chvíli nějaká část dokumentu vybrána, jelikož by byla nahrazena.

Selection sel = wrdClass.Selection - získá odkaz na aktuální pozici v dokumentu

sel.TypeText("Hello, world") - na pozici definovanou objektem sel bude přidán text Hello, world. Pokud není nastaveno přepisování textu je námi psaný text vložen před již existující text.

Vložení obrázku do dokumentu

Pro přidaní obrázku buďto definujeme opět objekt *Range* nebo využijeme objektu *Selection.*

InlineShapes.AddPicture(FileName, ref LinkToFile, ref SaveWithDocument, ref Range) - přidá obrázek uvedený v parametru FileName do aktuálního dokumentu. Zbylé parametry jsou volitelné, LinkToFile umožňuje obrázek svázat s souborem, kde se tento obrázek nalézá, SaveWithDocument umožňuje zvolit jestli bude soubor, který je svázán s obrázkem uložen spolu s dokumentem, Range udává místo, do kterého bude obrázek vložen.

Vložení tabulky do dokumentu

Tabulky je možno vkládat jak pomocí *Range*, *Selection,* tak je možné je vložit přímo za použití objetku *Word.Document*. Každý dokument má všechny tabulky uloženy v

kolekci Tables a je tedy možné k nim přistupovat pomocí indexu.

wrdDoc.Tables.Add(Range Range, int NumRows, int NumColumns, ref DefaultTableBehavior, ref AutoFitBehavior) - přidá do dokumentu na pozici Range tabulku o počtu řádků, sloupců udaných v parametrech NumRows, NumColumns. Další parametry jsou volitelné. DefaultTableBehavior udává jestli se budou buňky automaticky přizpůsobovat obsahu, AutoFitBehavior určuje jak se tabulka bude přizpůsobovat jejímu obsahu.

Formátování textu

Programově je možné nastavit zarovnání, velikost písma, druh písma pro objekt Range. Jelikož objekt Range může být definován mnoha způsoby, celá stránka, odstavec, část textu, je možno formátování textu velice přesně programově měnit podle požadvaků.

2.4 PostgreSQL

PostgreSQL je open-source databáze vyvinutá na University of California v Berkeley Computer Science Department. Jako taková obsahuje několik prvků, které byly do komerčních databázových aplikací přidány až déle, i nadále jsou přidávány funkce, které zlepšují vlastnosti celé databáze a tím ji řadí mezi nejlepší.

Hlavními klady databáze jsou:

- Plná podpora ACID (Atomicity, Consistency, Isolation, Durability) - vlastnosti, které umožňují, aby velké operace proběhly spolehlivě.
- Podpora standardu ANSI SQL
- Rozhraní pro ODBC, JDBC, C++, Pearl atd.
- Unicode
- Podpora SSL
- Nezávislost na platformě
- Uživatelem definovatelné typy, funkce, operátory

Jelikož jsou používány standartní příkazy jazyka SQL, není přechod na tuto databázi vůbec náročný. Ke spojení s databází je používáno protokolu ODBC, použití v programovacím jazyce C# není příliš odlišné od databáze Microsoft SQL. K samotné databázi je také naistalováno grafické rozhraní pgAdmin pro jednoduchou manipulaci s databází a její správu, které dále zjednodušuje manipulaci s ní.

Jedním z kladů této databáze je také možnost ukládat pole prvků přímo do jedné buňky a to i vícerozměrná.

Kapitola 3

Architektura programu

V této kapitole je popsáno jakým způsobem jsem řešil daný problém a důvody, které mě k tomuto řešení vedly. Je zde také uvedeno blokové schéma charakterizující chod celého systému.

Zadáním této práce je implementovat rozhraní pro uchování dat, které jsou generována při provádění experimentů. Jejich uchování ve vhodném formátu. Možnost vyhledávání a procházení těchto dat a jejich pozdější zpracování.

Ze zadání práce také vyplývají dva nutné prvky systému, jimiž jsou přístup k datům v programu Matlab a zařazení vhodné databáze do celého systému pro ukládání dat.

3.1 Možná řešení

V této části budou popsány možné varianty řešení, jejich klady a zápory, odůvodnění výběru databáze.

3.1.1 Matlab + databáze

Program matlab umožňuje vytvoření základního grafického rozhraní pomocí GUI Layout Editor, které by umožňovalo uživateli manipulovat s daty v prostředí matlabu bez nutnosti použít další programovací jazyk. Avšak možnosti, které máme při tvorbě tohoto rozhraní, nejsou zdaleka tak rozsáhlé jako v moderních programovacích jazycích jako je C# nebo Java. Také nastávají problémy s uchováním dat sloužící pouze pro interní potřeby programu.

Pro spojení matlabu s databází není v základní instalaci vytvořena podpora a spojení je možné pouze je-li naistalován Database Toolbox, který je nutné zakoupit zvlášť, čímž vznikají další výdaje. Další možností pro spojení s databází je napsání příslušné .mex funkce, která by zajišťovala přístup k databázi. I když je navrženo, aby .mex funkce byly psány v programovacích jazycích C, C++ je možné v nich po správném nastavení překladače použít knihovny z .NET Framework. Takto by bylo možné přis-

tupovat k databázi za pomocí těchto nástrojů a využití jiných technologií obsažených v prostředí .NET. Při pokusu o implementaci takovéhoto rozhraní však nastávaly problémy se spustitelností daného kódu a celkové jeho přenositelnosti na další počítače.

Dalším nedostatkem, který by se objevil při implementaci celého programu v jazyce matlab by zajisté byly nepříliš snadné možnosti při psaní reportu například ve spojení s programem MS Word.

3.1.2 Matlab + C# + databáze

Přidání programovacího jazyku C# ve spojení se silným návrhovým systémem Visual Studio odstraníme několik neduhů, které mělo předchozí řešení. Volba mezi využitím programovacích jazyků C# nebo Java byla dána spíše mými většími zkušenostmi s jazykem C#, zajisté by bylo možné dosáhnout podobného výsledku v jazyce Java.

Prvním nedostatkem, který je odstraněn, je původně slabší grafické rozhraní, které nyní za použití prostředí Visual Studia a jazyka C# poskytne mnohem větší možnosti. Toto rozhraní bude příjemnější pro běžného uživatele, který je již zvyklý na podobný druh aplikací.

Druhým vylepšním je bezesporu lepší přístup k databázím, než-li je v programu Matlab. Je možné využít podporovaný MS SQL Server nebo i další při použití příslušného rozhraní. Není nutné dokupovat další software pouze pro tuto část.

Třetí je možnost snadnějšího přístupu k aplikaci MS Word, a tak dosáhnout možnosti generování dokumentů s vloženým textem a vybranými grafy. U takto generovaného dokumentu se může uživatel soustředit více na dopsání teoretické části a popsání výsledků, data z měření jsou již automaticky vložena. Pro přístup k této aplikaci je použito COM rozhraní, které bylo popsáno v části 2.3.

Použitím daného návrhu nám však vzniká nezanedbatelný problém nutnosti implementace vrstvy, která nám umožní komunikovat s prostředím programu Matlab a přenášet data oběma směry. Tento problém můžeme řešit pomocí jednoduchého spojení za použití protokolu TCP/IP nebo použitím Matlab Automation Serveru. U obou těchto řešení je nutné implementovat menší grafické rozhraní v programu Matlab pro výběr dat, pro přenos a jejich zpětné získávání. Z těchto dvou možností se jeví jako lepší využití Automation Serveru, hlavní výhodou je možnost vykreslení grafů a provádění příkazů v prostředí Automation serveru aniž by muselo být spuštěno hlavní okno programu Matlab a tím částečná nezávislost na této části. Oproti TCP/IP se nemusíme starat o správnou interpretaci paketů a přijatých dat, vše je definováno v rozhraní Automation serveru.

3.1.3 Volba databáze

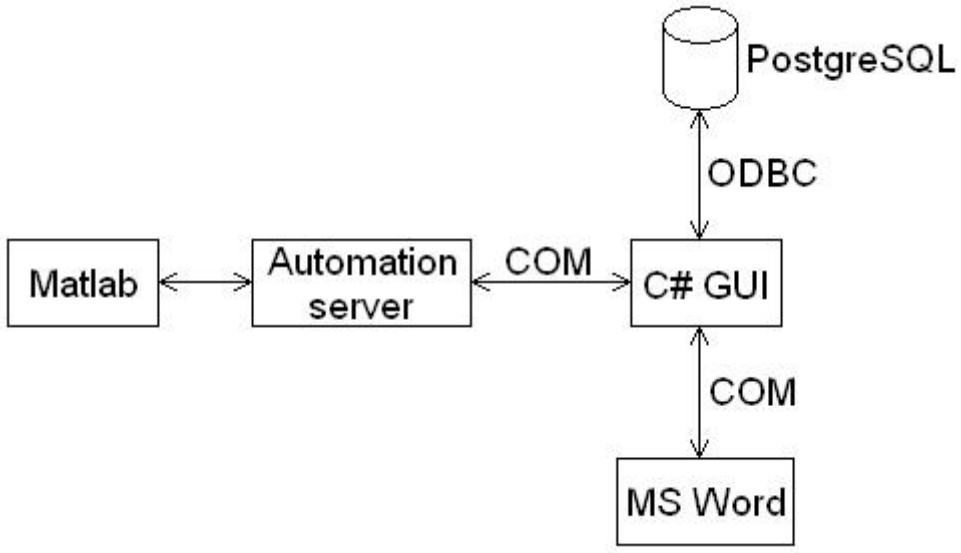
Zde přicházejí v úvahu opět minimálně dvě varianty.

Databáze Microsoft SQL Server - tento SQL server, jelikož je produktem firmy Microsoft, je provázán s prostředím Visual Studio a jsou v něm připraveny možnosti pro lepší spolupráci s programy napsanými právě v tomto prostředí. Nedostatkem této databáze je nemožnost práce přímo s maticemi. Také se nejedná o volně šířitelnou databázi, čímž se opět zvyšují náklady na zavedení aplikace.

Databáze PostgreSQL - tato databáze je blíže popsána v části 2.4. Hlavním přínosem pro naší aplikaci je možnost snadné práce s vícerozměrnými poli v databázi. Přitom většina dat, které budeme používat jsou matice, a to jak ty, které popisují chování systému a jeho nastavení, tak výsledky měření. Použití této databáze přináší tedy velké usnadnění práce s větší částí dat. Pro komunikaci s databází je použito rozhraní ODBC.

3.2 Struktura řešení

Na následujícím obrázku je znázorněno vybrané řešení a komunikace mezi jednotlivými bloky celého řešení.



Obrázek 3.1: Struktura řešení

Z obrázku je patrné, že jsem se rozhodl pro řešení uvedení v bodě 3.1.2 a to z toho důvodu, že klady, které nám tento způsob řešení přináší jsou mnohem větší, nežli popsaný zápor nutnosti implementace mezivrstvy pro přenos dat z Matlabu.

Pro tuto mezivrstvu jsem zvolil možnost Automation serveru, který je pro svoji jednoduchost a částečnou nezávislost lepším řešením pro daný problém.

Při návrhu systému jsme narazili na problém, jestli a jakým způsobem umožnit uživateli ukládat soubory k příslušnému experimentu. Možnost ukládání reportů je velice praktická věc, k námi napsanému dokumentu má pak kdokoli s příslušným oprávněním

přístup a nemusíme řešit jejich správné archivování. Větší problém je s ukládáním souborů Simulinku a jiné soubory sloužící ke správné funkci zařízení. Chceme-li uložit soubor s nastavením k příslušnému experimentu musíme se držet jasných pravidel, jak udržovat správnou verzi a každý s přístupem k těmto souborům musí podle nich postupovat. V jiném případě by mohlo dojít k modifikaci souboru s nastavením a uložená data pak nemusejí odpovídat skutečné odezvě zařízení při tomto nastavení. Jinou možností je ukládat pouze odkazy na aktuální soubor, pak by při jeho změně odkaz ukazoval na stále stejný soubor. Problém by však nastal při neopatrné manipulaci, když by byl například soubor přesunut nebo smazán, pak bychom ztratili jakékoli informace o tom, jaké další nastavení bylo využito. Rozhodl jsem se využít možnosti ukládání souborů do databáze, při zodpovědnosti obsluhy, tak bude vždy zřejmé za jakých okolností daný experient vznikl a je možná pozdější reprodukovatelnost.

Kapitola 4

Interface pro práci s daty v programu Matlab

V této kapitole je popsána část řešení, které zajišťuje přenost dat z programu Matlab na Automation server pro další zpracovíní v hlavním pracovním prostředí a jejich opětovné načtení. Také je zde popsána struktura změřených dat, která je vstupem do celého zpracování a archivace dat.

4.1 Struktura experimentu

Po provedení experimentu na reálném zařízení je vrácena struktura obsahující následující položky:

- date - datum a čas měření
- cfg - tato položka je opět strukturou obsahující data popisující blíže nastavení celého systému, najdeme zde následující nastavení
  - Switches - přepínače, které volí požadované zapojení z několika možných
  - Signums - známénka, která udávají jak bude brána proměnná za daným přepínačem
  - Model Names - jména bloků, která jsou zapojena do obvodu regulátoru
  - Model Vars - hodnoty pro nastavení chování jednotlivých bloků regulátorů
  - Aero Force Model Names
  - Aero Force Model Vars
- data - struktura obsahující změřená data na zařízení
  - Time - časy, ve kterých jsou data změřena

– Data - hodnoty na sledovaných vstupech, výstupech

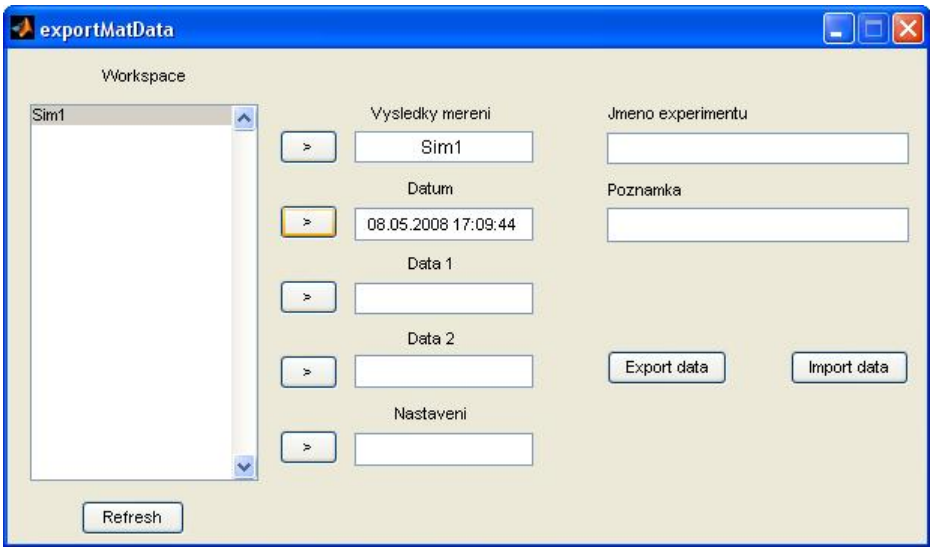
– Names - jména sledovaných vstupních a výstupních proměnných

Pomocí této struktury můžeme velice dobře specifikovat nastavení celého měřeného systému a i po delší době jsme schopni reprodukovat měření na základě načtené struktury.

Položka data je zde uvedena pouze jednou, ale je možné specifikovat několik takovýchto struktur, které budeme chtít uložit do databáze, jedinou podmínkou je, že musí obsahovat položky uvedené výše, aby bylo možné ji bez problému identifikovat.

4.2 Export dat

Pro vytvoření funkce exportující data z Matlabu jsem využil možnosti použít grafické rozhraní, které ulehčí uživateli volbu správných dat, umoží připsání jména experimentu a další poznámky. Její náhled je uveden na obr 4.1. Před spuštěním musí být vytvořen Automation server příkazem *prom = actxserver('matlab.aplication')*. K tomuto serveru se pak připojuje samotný program a hlavní grafické rozhraní. Tento server nesmí být ukončen v době přenosu dat, jinak se musí celá operace opakovat od začátku.



Obrázek 4.1: exportMatData

Jak je vidět z obrázku, grafické rozhraní umožní vybrat proměnnou z hlavního pracovního prostoru Matlabu a její vložení do příslušné položky ve formuláři.

Vysvětlení jednotlivých položek :

- Vysledky mereni - do této položky se vkládá celá struktura jak je uvedena v části 4.1, jestliže je zadána není brán zřetel na další položky kromě jména experimentu a poznámky ve formuláři a exportována je právě tato struktura.

- Datum - datum uložení experimentu, datum a čas je automaticky doplněn po stisknutí tlačítka příslušícího k této položce
- Data 1, Data 2 - místa pro vložení dat splňující definici příslušné struktury (musí obsahovat Time, Data, Names)
- Nastavení - struktura nastavení pro měřené zařízení, ve formátu uvedeném výše
- Jméno experimentu - uživatelem volený název experimentu
- Poznámka - položka pro jakoukoli poznámku k dokumentu

Položky Datum, Data 1, Data 2, Nastavení jsou využita, jestliže nemáme celou strukturu měření a chceme ji složit z jednotlivých částí. Pokud bude přítomen správný název celé struktury tato data nejsou využita.

Tlačítka :

- Export data - tímto jsou vybraná data exportována do Automation serveru a připravena ve správné struktuře pro převzetí hlavním programem.
- Import data - je možné ho využít k přečtení dat z Automation serveru, další popis této funkce je uveden v další části.
- Refresh - jestliže jsou mezi spuštěním tohoto prostředí vložena do hlavního pracovního prostoru další data (proměnné) je možné tímto tlačítkem aktualizovat jejich seznam a využívat i nové proměnné.

Pro funkci refresh je využito dvou jednoduchých příkazů.

```
vars = evalin('base','who');
set(handles.listbox1,'String',vars);
```

Prvním příkazem zjistíme všechny příkazy v hlavním okně, které následně vložíme do listboxu.

Implementace exportu dat

Při začátku exportu dat z programu matlab musíme zajistit, že jsou vybrány nějaké proměnné a jestli uživatel opravdu vložil strukturu, kterou očekáváme. Kontrola vložené struktury není prováděna do hloubky, jelikož nemusí vždy obsahovat stejný počet datových položek se stejnými jmény.

Struktury není možné přenášet mezi Matlabem a Automation serverem pomocí již existující funkce jako je to u matic a jiných datových typů, proto bylo nutné se zamyslet nad řešením tohoto problému. Řešením bylo napsání rekurzivní funkce *exportStruct(input, jmeno, serverCom),* která prochází položky přijaté struktury.

```
function exportStruct(input, jmeno, serverCom)
  names = fieldnames(input);      %ziskani jmen polozek ve strukture
  noNames = size(names,1);
  for i=1:noNames      %projduti vsech polozek
    jmenoD = sprintf('%s.%s', jmeno, names{i});      %ulozene jmeno
    nextField = getfield(input, names{i});      %ziskani polozky ze struktury
    if isstruct(nextField)      %overeni jestli je to struktura
      exportStruct(nextField, jmenoD, serverCom);
    else if(size(nextField,1)~=0)      %overeni platne polozky
      sendField = getfield(input, names{i});
      serverCom.PutWorkspaceData('temp','caller',sendField);
      exStr = sprintf('%s.%s=temp', jmeno, names{i});
      serverCom.Execute(exStr);
    else serverCom.PutWorkspaceData('temp','caller',0);
      exStr = sprintf('%s.%s=temp', jmeno, names{i});
      serverCom.Execute(exStr);
    end
  end
end
```

Každá položka je testována, jestli se jedná a strukturu nebo jednoduchý datový typ. Je-li položka jednoduchým datovým typem, je odeslána na Automation server, pokude se jedná o strukturu je opět zavolána funkce *exportStruct*. Tímto procházíme celou strukturu nejdříve do hloubky a pak následuje její postupná kompletace na straně Automation serveru.

Pokud je zadána celá struktura je ve výsledku uložena s názvem *Simulation,* což je očekávaný formát pro čtení, jak v aplikaci napsané v jazyce C#, tak pro zpětné načtení do programu Matlab.

Jestliže je zadána struktura po částech, musí být provedena kontrola, jestli se nacházejí všechny nutné položky a to datum, konfigurace a data. Množství dat uložených tímto způsobem je omezeno na dvě struktury oproti neomezenému počtu je-li již hotova kompletní struktura. Názvy proměnných jsou upraveny a doplněny tak, aby ve výsledku byla na straně Automation serveru struktura se standardním formátem a obsahem dat.

Po ukončení této funkce jsou na straně Automation serveru tři proměnné:

- Simulation - struktura obsahující data, která popisují daný experiment
- jmenoEx - jméno experimentu
- Poznamka - poznámky k experimentu

Tyto proměnné by neměly být modifikovány, jinak může dojít k problémům s jejich následným čtením. Pokud exportujeme více experimentů je nutné toto provést postupně. Exportovat experiment na Automation server v hlavním rozhraní ho importovat a případně uložit do databáze. Takto bychom postupovali u každého experimentu.

4.3 Import dat

Pro importování dat z Automation serveru je využita funkce, která je částečně podobná funkci pro export dat. Opět není možné přenést celou strukturu experimentu v jednom kroku, proto je nutné ji přenést za použití rekurzivní funkce.

První však musí nastat kontrola jestli jsou na straně Automation serveru obsažena nějaká data, která by bylo možné přenést. Předpokládám, že do dat obsažených na Automation serveru nezasahuje obsluha a jeho obsah je tak plně pod kontrolou hlavní aplikace nebo funkce pro export dat. V tomto případě pak vím, že jestliže jsou na serveru přítomna data ve struktuře Simulation, je možné je přenést do pracovního prostoru Matlabu.

sim = importMatData(comServer)

Tímto příkazem je zavolána funkce pro import dat, parametr comServer je odkazem na existující instanci Automation serveru. Funkcí vrácená struktura experimentu je vrácena do proměnné sim.

```
function vysl = importStruct(jmeno, serverCom)
cmd = sprintf('temp=fieldnames(%s)', jmeno);
serverCom.Execute(cmd);
names = serverCom.GetWorkspaceData('temp', 'caller');
for i=1:size(names,1)
   cmd = sprintf('temp=isstruct(%s.%s)',jmeno, names{i});
   serverCom.Execute(cmd);
   result = serverCom.GetWorkspaceData('temp','caller');
   if(result==1)
      cmd=sprintf('%s.%s',jmeno, names{i});
      vysl.(names{i})=importStruct(cmd, serverCom);
   else
      cmd=sprintf('temp=%s.%s',jmeno, names{i});
      serverCom.Execute(cmd);
      vysl.(names{i})=serverCom.GetWorkspaceData('temp','caller');
   end
end
```

Funkce importStruct je opět volána rekurzivně jestliže víme, že proměnná se jménem v parametru jmeno je strukturou. Nejdříve jsou zjištěna jména položek ve struktuře,

pak pro každou položku zjistíme jestli je opět strukturou. Pokud ano, je opět zavolána funkce importStruct se jménem dané položky. Pokud se již jedná o základní datový typ, je položka přenesena z Automation serveru. Aby byla tato funkce více variabilní a použitelná na různé typy struktur je pro uložení do lokálního workspace využita možnost dynamického pojmenování proměnné, jak je vidět v příkazu *vysl.(names{i}) = serverCom.GetWorkspaceData('temp','caller')*. Prohledávání struktury je opět do hloubky jako u funkce pro export dat.

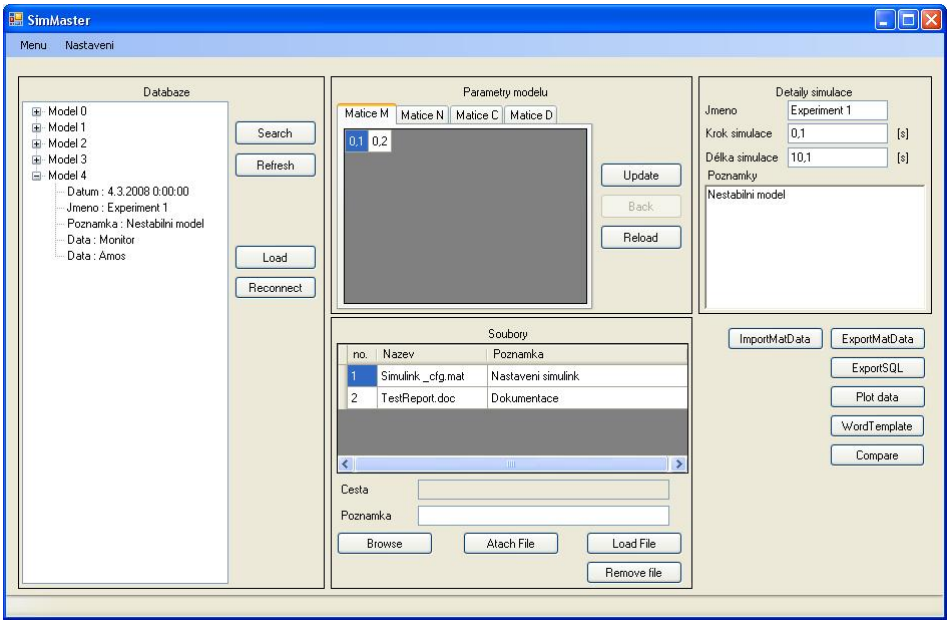
Kapitola 5

Interface pro práci s naměřenými daty

V této kapitole je popsáno grafické uživatelské rozhraní pro zpracování naměřených dat. Dále je popsána implementace jednotlivých částí tohoto rozhraní a možnosti těchto částí.

5.1 Obecný popis

Hlavní část grafického prostředí je na obrázku 5.1, v tomto okně je možné načíst data z databáze nebo z Automation serveru případně je exportovat. Jsou zobrazeny hodnoty matic, které definují daný model a další informace zadané při exportu dat. Také je zde možno přiložit jakýkoli soubor až do velikosti 2GB, který bude uložen do databáze a svázán s tímto experimentem.



Obrázek 5.1: Hlavní okno aplikace

Nyní se blíže podíváme na implementaci jednotlivých části tohoto prostředí.

5.2 Třída pro práci s COM rozhraním Matlabu

Na obrázku 5.1je vidět dvě tlačítka pro import/export dat. O komunikaci s prostředím Automation serveru se stará třída *Simulation,* která obsahuje obě fukce pro tato tlačítka a také možnost načtení dat z databáze. Spojení s Automation severem je navázáno hned po startu aplikace. Pokud není v tuto chvíli server aktivní je vytvořen nový. K tomuto serveru pak přistupují všechny funkce.

5.2.1 Import dat

Na začátku běhu celého programu je vytvořen jeden prázdný objekt. Po stisknutí tlačítka ImportMatData nastává volání funkce třídy Simulation - *fillSimulation(MLApp.-MLAppClass matlab)* a předem vytvořený objekt je naplněn.

Pro získání dat z Automation serveru používáme funkci Execute, které v pracovním prostředí vykoná námi zadaný příkaz a vrací výsledek v datovém typu string, který je standardně vidět v okně programu Matlab. Jelikož importovaná data nejsou stejného typu, je pro každý typ vytvořena příslušná funkce, která zajistí správnou interpretaci získaných dat.

Při čtení jména experimentu a poznámky se nejedná o těžký problém, jelikož data jsou ve správném formátu a pouze stačí odstranit znaky, které program Matlab automaticky přidává na začátek a konec dat.

Budeme-li však chtít získat část struktury, která definuje nastavení experimentu, nejsou námi čtená data pouhé stringy, ale jedná se například o matice čísel nebo stringů. Je zde také předpokládáno, že struktura na straně Automation serveru splňuje specifikaci popsanou v části 4.1.

Pole stringů je po vykonání příkazu Execute vráceno ve formě dlouhého řetězce, kde námi požadované stringy jsou v uvozovkách oddělených mezerou, proto projdeme celý string a znaky mezi uvozovkami bereme jako námi požadované stringy.

Ještě větší problém však nastává při získávání hodnot, například v části struktury ModelVars. Tyto hodnoty mohou být jak jedno číslo, tak dvourozměrná matice neznámé velikosti. Jestliže je přítomno pouze jedno číslo není jeho přečtení žádný problém. Pokud se však jedná o matici, nemá string, který dostaneme po vykonání příkazu Execute definován přesně. U malých matic jsou hodnoty v řádku odděleny mezerou a další řádek je oddělen znakem odřádkování. Ve větších maticích toto však není dodrženo a nemusí být celý řádek vložen za sebe, ale může být rozdělen dalším znakem což znemožňuje snadné přečtení hodnot. Tento problém jsem odstranil čtením matice po sloupcích, kde je vždy zajištěno, že je další prvek oddělen odřádkováním a získal jsem hodnoty následujícím cyklem.

```
for (i = 0; i < strTable.Length-1; i++)
{
   str = strTable[i];
   str = str.Remove(0, 2);
   str = str.Replace('.', ',');
   cislo = float.Parse(str);
   temp[i] = cislo;
   pocetVal++;
}
```

Program Matlab používá pro oddělení desetinných míst tečku kdežto v jazyce C# je k tomuto účelu použita čárka. Proto musí být u každého prvku nahrazena desetinná tečka čárkou.

Po implementaci výše zmíněných částí již lze získat strukturu z Automation serveru.

5.2.2 Export dat

Export dat nastává po stisknutí tlačítka ExportMatData. Opět je volána funkce třídy Simulation, tentokrát *exportSimulation(MLApp.MLAppClass matlab)*. Je-li objekt, který má být exportován prázdný je výsledná struktura na straně Automation serveru také prázdná.

První je exportováno jméno experimentu, poznámka, datum. U těchto položek stačí vytvořit krátký string s příkazem platným v programu Matlab a pomocí funkce Execute tento příkaz provést v prostoru Automation serveru.

Pro exportování matic je ve třídě Simulation vytvořena funkce *setTable(MLApp.MLAppClass serverCon, float[,] Table, string param)*. Parametry této funkce jsou serverCon - spojení s aktivním Automation serverem, na který budou data uložena, param - jméno proměnné, pod kterou budou data uložena, Table - proměnná, která je exportována.

Ve funkci je nejdříve zjištěna velikost zadaného pole, pokud se jedná pouze o jednu hodnotu je přímo uložena na Automation server. Jedná-li se o větší pole hodnot je vytvořen řetězec, který reprezentuje daná data. Musíme převést reprezentaci desetinných čísel na tu, která je akceptovatelná v programu Matlab, tedy nahradit desetinnou čárku tečkou, toho docílíme použitím funkce Replace, která je definovaná ve třídě String.

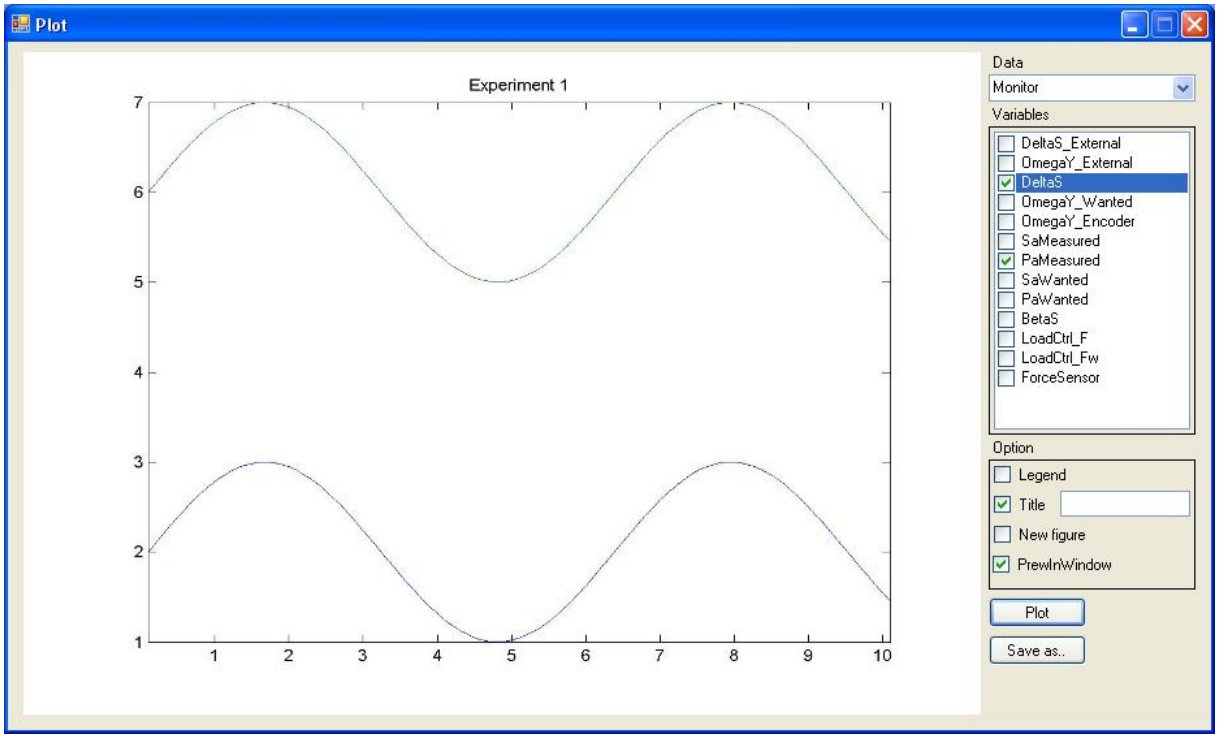
Pro exportování polí řetězců je použita funkce *setStringTable(MLApp.MLAppClass serverCon, string[] Table, string param)*. Parametry jsou podobné jako u předchozí funkce setTable, jedinou odlišností je typ parametru Table, který je v tomto případě string[].

```
for (int index = 0; index < pocetStr; index++)
{
   if (index == 0)
      tmp = "{\'" + Table[index] + "\'";
   else
      tmp = tmp + ",\'" + Table[index] + "\'";
}
tmp = tmp + "}";
cmd = string.Format("{0}={1}", param, tmp);
serverCon.Execute(cmd);
```

Jak je vidět z uvedeného kódu této funkce, jedná se o postupné přidávání požadovaných řetězců do uvozovek a jejich správné oddělení. Tímto cyklem vzniká delší řetězec, který je na straně Matlabu identifikován podle našich požadavků.

5.2.3 Vykreslení uložených dat

Program umožňuje vykreslení dat, která jsou aktuálně načtena, rozhraní pro toto vykreslení je uvedeno na obrázku 5.2.



Obrázek 5.2: Vykreslení uložených dat

V combo boxu Data jsou na výběr struktury změřených dat aktuálního experimentu s jejich odpovídajícími názvy. Pokud jsou vybrána data v tomto combo boxu, jsou do seznamu Variables vloženy všechny názvy proměnných z těchto dat. Ze seznamu je

pak možné vybrat proměnné, které budou vykresleny v grafu. Pomocí výběru v části Option můžeme přidat k obrázku libovolný nadpis, přidat popisky k zobrazeným datům a otevřít náhled výsledného grafu v tomto okně.

Po navolení všech dat a spuštění vykreslení pomocí tlačítka Plot, jsou vygenerovány řetězce obsahující příslušná vybraná data, data pro časovou osu. Nastaveny limity pro osu x a pokud byla navolena nějaká další možnost pak přidány další možnosti k vykreslenému grafu.

Získání zmíněných řetězců je implementováno ve funkcích getTimeForPlot a getDataForPlot, které jsou obsaženy ve třídě Simulation.

getTimeForPlot:

```
string getTimeForPlot(int indexData)
{
   int pocetRadku;
   string tmp = "[";
   pocetRadku = this.data[indexData].time.GetLength(0);
   for (int i = 0; i < pocetRadku; i++)
   {
      if (i == 0)
         tmp += this.data[indexData].time[i,0].ToString().Replace(',', '.');
      else
         tmp = tmp + ","
                + this.data[indexData].time[i,0].ToString().Replace(',', '.');
   }
   tmp += "]";
   return tmp;
}
```

Parametr indexData je určen tím, která data byla vybrána. Z časových údajů je utvořena řádková matice, která je pak přidána jako jeden z parametrů do příkazu k vykreslení.

Funkce getDataForPlot je velice podobná této, rozdílem je, že data jsou vícerozměrná a je nutné dbát na správné řazení hodnot za sebou tak, aby ve výsledku nedošlo k záměně dat při vykreslení.

Po vygenerování těchto řetězců je celý příkaz odeslán na Automation server. Zde je otevřen nový "figure", ve kterém jsou zobrazena zadaná data, pokud uživatel nezvolí, že chce vytvořit náhled obrázku v programu, pak je tímto funkce ukončena.

Je-li zvolen náhled, je vykreslený "figure" uložen do souboru s pracovním názvem pic.jpg, místo uložení je součástí konfiguračního souboru a může být změněno v nastavení programu v položce Work directory. K ukládání vykreslených grafů je využíváno funkce programu Matlab print s parametrem -djpeg. Dále je možné daný obrázek uložit

do námi požadovaného souboru k pozdějšímu využití, soubor pic.jpg bude při dalším použití nahrazen novým obrázkem.

5.3 Komunikace s databází PostgreSQL

Komunikace s databází PostgreSQL je založena na protokolu ODBC (open database connectivity). Komunikace je navázána po startu aplikace, kdy je potřeba naplnit treeview, které zobrazuje základní informace o experimentech uložených v databázi. K dalšímu využití spojení databáze dochází při ukládání/načítání experimentů. Pro připojení k databázi je použit následující řetězec:

```
"Dsn=PostgreSQL30W;database=AeroMdl_0_1;server=localhost;port=5500;
uid=loginname;password=pasword"
```

Dsn musí být definováno na tomto počítači jako platný datový zdroj. Pokud není možné toto spojení navázat je zobrazeno chybové hlášení. Parametry připojení je také možno nastavit v konfiguraci programu.

5.3.1 Přehled experimentů v treview

Informace v treeview jsou uloženy jako kolekce tříd, každá tato třída musí být odvozena od třídy TreeNode. V tomto případě je tato třída velmi jednoduchá a definuje pouze jméno uzlu a zobrazovaný text. Do jména uzlu je uložena informace o tom, který experiment v databázi tento uzel představuje. V zobrazovaném textu jsou pak informace pro uživatele jako je datum, poznámka atd. Při vytváření celého stromu pak je jeden hlavní uzel, ke kterému jsou postupně přidávány uzly tvořící požadovanou strukturu.

Pro naplnění treeview je použita funkce *FillTreeview()*. Nejdříve je vytvořen SQL příkaz pro získání všech experimentů:

```
select model_id, date, poznamka, jmenoex, data from model
```

Pokud je aplikován filtr, jsou k němu dále přidávány další podmínky, které specifikují, které experimenty jsou vyžádány uživatelem. K tomuto účelu je vytvořena třída *Filtr*. Pokud uživatel zvolí nějaké specifikace vyhledávání je nastaven parametr *Filtr.IsUsed* na true a dochází k přiřazení podmínek. Lze vyhledávat i části textů, pak je použit SQL příkaz *like*, který za použití znaku '%' vyhledá jakýkoli string, který v jakékoli části obsahuje zadaný řetězec. Naprogramovaný filtr tak umožňuje vyhledávání experimentů pomocí data, části jména experimentu nebo poznámky. Jestliže jsou nalezeny nějaké experimenty, jsou zobrazeny v treeview.

5.3.2 Načtení experimentu z databáze

Po vyvolání eventu stisknutím tlačítka Load získáme číslo požadovaného experimentu, které je uloženo ve jméně příslušného uzlu treeview. Číslo uložené v tomto jméně

odpovídá privátnímu klíči.

Pro snadnější prohledávání výsledku SQL příkazu je využit ODBC data adapter, který po zadání příkazu může naplnit proměnnou typu DataTable. Výsledkem je pak proměnná, která obsahuje řádky vrácené databází. V našem případě je vrácen jeden řádek obsahující parametry a data experimentu. Ve třídě Simulation je pak vytvořen konstruktor, který má jako vstupní parametr právě proměnnou typu DataRow.

V získaném řádku jsou z větší části pouze čísla privátních klíčů, které jsou dále uloženy v jednotlivých tabulkách databáze. Proto je ve zmíněném konstruktoru vytvořeno několik dalších příkazů, které z databáze za použití zmíněných privátních klíčů získají přesné nastavení experimentu a uloží ho do objektu typu Simulation. U většiny parametrů nemusí docházet k dalším úpravám.

U parametrů, které jsou uloženy v databázi jako pole je nutné ještě string, který je vrácen databází obsahující konkrétní data příslušnou funkcí rozdělit a získat data v dále použitelném formátu. Pro tuto činnost jsou implementovány dvě funkce.

První pro rozdělení polí obsahující string : *string[] parseSqlStrTable(string tabulka).*

```
private string[] parseSqlStrTable(string tabulka)
{
  string[] strTable;
  tabulka = tabulka.Remove(0,1);
  tabulka = tabulka.Remove(tabulka.Length - 1, 1);
  strTable = tabulka.Split(',');
  return strTable;
}
```

Druhá pro rozdělení číselných hodnot a to jak jednotlivých čísel nebo celých matic : *float[,] parseSqlTable(string tabulka).* Tato funkce je principielně velmi podobná předcházející funkci. Nutnou částí však je zjištění jestli je ve vstupním řetězci pouze jedno číslo, řádkový vektor nebo větší matice a podle tohoto rozdělení dále daný řetězec rozdělit.

Posledním bodem při načtení experimentu je získání informací o přiložených souborech, privátní klíče přiložených souborů jsou součástí výsledku prvního SQL příkazu. Proto je pouze přečteme z databáze, uložíme ve správném formátu do dataGridView, tento prvek umožňuje snadné vkládání dat do tabulky. Vytvoříme tabulku stringů, které reprezentují jednotlivé buňky v řádku, a pak je přidáme do kolekce řádků prvku dataGridView.

5.3.3 Uložení dat do databáze

K uložení dat dochází poté, je-li v paměti uložen objekt obsahující data o experimentu a uživatel stiskne tlačítku ExportSQL. Funkce vykonávající obsluhu tohoto eventu je

ve třídě Simulation.

```
void exportSimulationToSQL(OdbcConnection sqlConnection)
```

V této funkci dochází postupně k uložení všech dat příslušného experimentu, soubory jsou přikládány zvlášť, a proto bude tato funkce popsána až dále. Implementaci této funkce budeme demonstrovat na jednoduchém příkladu uložení nastavení přepínačů při tomto experimentu.

```
cmd = string.Format("SELECT MAX(switches_id) FROM switches");
sqlCmd = new OdbcCommand(cmd, sqlConnection);
result = sqlCmd.ExecuteScalar().ToString();
if (result == "")
  switchesID = 0;
else
  switchesID = Convert.ToInt16(result) + 1;
cmd = string.Format("INSERT INTO switches (switches_id, deltas_ex
  ternal_switchtoggleb, deltas_switchtoggleb, model_switchtoggleb,
  omegay_external_switchtoggleb, aeroforcemodel_switchtoggleb,
  aeroforce_switchtoggleb) VALUES ('{0}','{1}','{2}','{3}','{4}','{5}','{6}')",
  switchesID, this.cfg.switches.DeltaS_External_SwitchToggleB, this.cfg.
  switches.DeltaS_SwitchToggleB, this.cfg.switches.Model_SwitchTogg
  leB, this.cfg.switches.OmegaY_External_SwitchToggleB, this.cfg.swi
  tches.AeroForceModel_SwitchToggleB, this.cfg.switches.AeroForce_
  SwitchToggleB);
sqlCmd = new OdbcCommand(cmd, sqlConnection);
sqlCmd.ExecuteNonQuery();
```

U zbylých položek je postupováno obdobně, vždy je zjištěn poslední index v dané tabulce a na příští pozici je vložen momentálně aktivní experiment. Pro uložení matic jsou zde implementovány dvě funkce a to : *string strArray(float[,] table)*, *string strStrArray(string[] table)*. Tyto funkce obstarávají převod matic a polí uložených v paměti na formát akceptovaný v databázi PostgreSQL tak, aby byly uloženy jako matice. Pro ukládání matic je v programu PostgreSQL zaveden příkaz *ARRAY*. Po celou dobu ukládání dat jsou uchovávány indexy privátních klíčů, naposledy jsou uloženy data do tabulky model, kam jsou vloženy tyto indexy, aby byly aktivní odkazy na další tabulky s konkrétními daty, přesná struktura databáze je popsána v části 6.

5.3.4 Práce se soubory

Aby bylo možné přidávat soubory s nastavením nebo hotové reporty k již uloženému experimentu, je ukládání implementováno odlišně nežli ukládání zbylých dat. Také načítání souborů se děje v oddělené části kódu.

Pro ukládání souborů a velkých binárních dat jsou v databázi PostgreSQL dva způsoby, buďto uložíme data přímo do sloupce tabulky, který má typ *Bitea* nebo je soubor uložen do datového prostoru nazvaného *Large object* a v tabulce je pak uložen pouze odkaz na tento objekt. V jiných databázích je k tomuto účelu použit datový typ *Blob*, který je zde podobný spíše datovému typu *Bitea*. Zvolil jsem druhý přístup a to uložení souboru mezi Large objects a uchování odkazu na tento soubor. Zvolený přístup je jednodušší v tom, že existují funkce na straně PostgreSQL serveru, které se o transfér dat postarají a uloží je v příslušném datovém prostoru, velikost jednoho souboru je omezena na 2GB.

nazev	popis funkce
lo_import('file')	uloží do databáze soubor defnovaný parametrem file, a vráti ukazatel na něj
lo_export(oid, 'file')	načte z databáze soubor se zadaným ukazatelem oid a uloží ho do souboru file
lo_unlink(oid)	vymaže soubor s ukazatelem oid z databáze

Tabulka 5.1: Server funkce PostgreSQl pro praci se soubory

Při ukládání souboru je nejdříve uživatelem vybrán příslušný soubor pomocí standardního dialogu pro vybrání souboru. Poté je možno přidat k souboru jakoukoli poznámku a uložit ho do databáze stiskem tlačítka Attach file.

```
cmd = string.Format("INSERT INTO files (files_id, file_oid, filename, poznamka)
VALUES ({0}, lo_import('{1}'), '{2}', '{3}')", fileid, openFileDialog1.FileName,
filename, textBox5.Text);
```

Pomocí toho SQL příkazu je vytvořen nový řádek v tabulce, do tohoto řádku je uložen vybraný soubor a poznámka k souboru. Dále je nutné aktualizovat seznam přiložených souborů v tabulce model.

```
cmd = string.Format("UPDATE model SET files = '{0}' WHERE model_id =
{1}", result, sim1.sim_id);
```

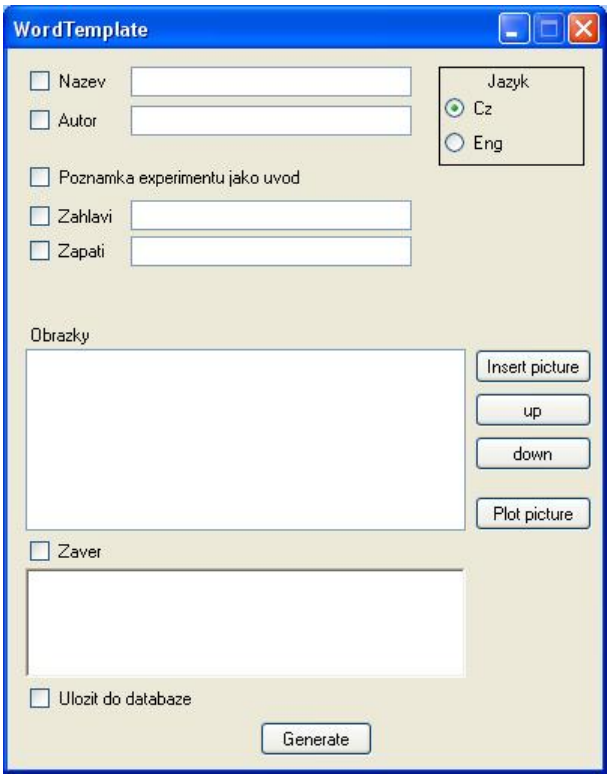
Tímto je soubor přiložen ke správnému experimentu k pozdějšímu využití.

K načtení souboru je nutné vybrat v seznamu přiložených souborů uživatelem požadovaný soubor, příslušný řádek označit a stisknout tlačítko Load File. Nejdříve je v dialogovém okně nutné specifikovat do kterého souboru budou data uložena, pak je pomocí funkce PostgreSQL lo_export soubor uložen na vybrané místo.

5.4 Generování dokumentů

Po stisknutí tlačítka WordTeplate v hlavním okně, je otevřeno okno, které nám umožní specifikovat detaily vygenerovaného dokumentu, jeho detail je na obr. 5.3. V tomto okně můžeme specifikovat obsah úvodní strany položkami Nazev, Autor. Položky Zahlaví a

Zapati dovolují specifikovat záhlaví a zápati v dokumentu, kromě úvodní strany která tyto parametry neobsahuje. V časti Obrazky je možné vybrat soubory které budou vloženy do dokumentu, pořadí v tomto okně udává i pořadí vložení do dokumentu, ke zvolení správného pořadí slouží tlačítka up/down. Podporovány jsou sobory .jpg a .tif. V části závěr je možné napsat závěr celého dokumentu. Po stisknutí tlačítka Generate je vytvořen dokument podle těchto voleb, a uložen do souboru specifikovaného uživatelem.



Obrázek 5.3: WordTemplate

Implementace

Větší část dokumentu je vytvořena pomocí funkcí popsaných v části 2.3. Pomocí objektu Range a Selection jsou vloženy texty a obrázky do dokumentů, definované při natavení okna. Přepínač Cz/Eng slouží k možnosti vytvořit dokument obsahující české nebo anglické názvy kapitol, jelikož je možné vygenerovat dokument s anglickými názvy kapitol nemusí uživatel v nutnosti napsat anglický report přepsat všechen text, pak by docházelo k velkému znehodnocení nástroje.

Tlačítko Plot picture slouží k vygenerování nového obrázku z naměřených dat, který bychom chtěli vložit do dokumentu. Pokud chceme využít tuto funkci je nutné mít načtený některý experiment z databáze. Princip jakým jsou generovány obrázky je popsán v části 5.2.3.

Nastavení záhlaví všech stránek kromě úvodní je ukázáno v následujícím kódu.

```
object start = Word.WdSectionStart.wdSectionNewPage;
formPoint.wrdDoc.Sections.Add(ref nothing, ref start);
Word.Section sec = formPoint.wrdDoc.Sections[2];

//zahlavi

if (checkBox1.Checked == true)
{
  sec.Headers[Word.WdHeaderFooterIndex.wdHeaderFooterPrimary].
    LinkToPrevious = false;
  sec.Headers[Word.WdHeaderFooterIndex.wdHeaderFooterPrimary].
    Range.Text = textBox1.Text;
  sec.Headers[Word.WdHeaderFooterIndex.wdHeaderFooterPrimary].
    Range.ParagraphFormat.Alignment=Word.WdParagraphAlignment.
    wdAlignParagraphCenter;
}
//zapati
if (checkBox2.Checked == true)
{
  sec.Footers[Word.WdHeaderFooterIndex.wdHeaderFooterPrimary].
    LinkToPrevious = false;
  sec.Footers[Word.WdHeaderFooterIndex.wdHeaderFooterPrimary].
    Range.Text = textBox2.Text;
  sec.Footers[Word.WdHeaderFooterIndex.wdHeaderFooterPrimary].
    Range.ParagraphFormat.Alignment=Word.WdParagraphAlignment.
    wdAlignParagraphCenter;
}
```

Nejdříve je nutné vytvořit novou sekci v dokumentu aby bylo možné nastavit parametry záhlaví a zápatí jinak pro úvodní stranu a jinak pro ostatní strany v dokumentu. Pro každou sekci je pak dále můžné nastavit správné záhlaví a zápatí. Jelikož celé záhlaví a zápatí je bráno jako jeden objekt range je pro něj možné nastavit parametry zmíněné výše.

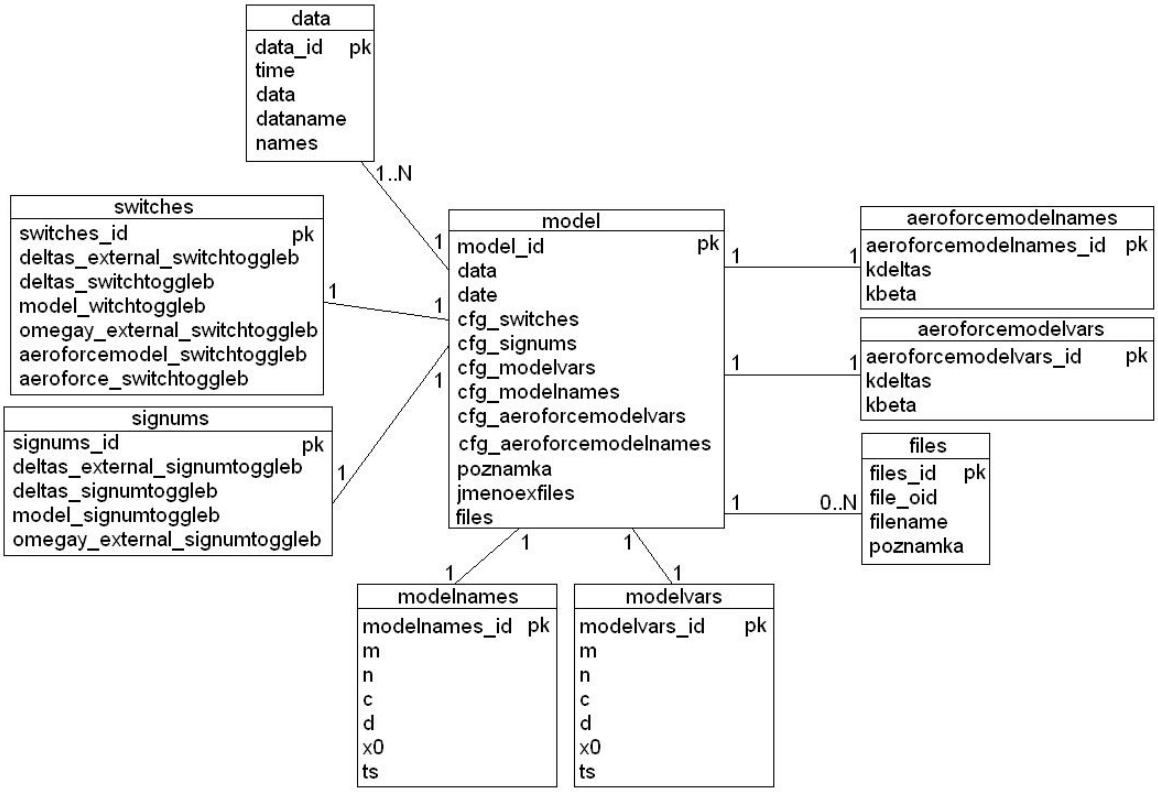
Pokud máme vybrán experiment, jé také možné soubor přímo přiložit k tomuto experimentu.

Kapitola 6

Návrh struktury databáze

V této kapitole je popsána navržená struktura databáze a její uspořádání.

Celkové schéma databáze je vidět na obr. 6.1. Celkový návrh struktury vychází ze snahy ponechat tuto část v co nejjednodušším formátu. Proto je zde hlavní tabulka model, která obsahuje odkazy na další tabulky, ve kterých jsou obsažena data. U většiny tabulek se jedná o relaci 1..1 jelikož není možné uložit jeden experiment s více jak jedním nastavení. U tabulky data je relace 1..N, jelikož je možné připojit jednu až N struktur dat. U Tabulky files je relace 0..N, není nutné přikládat nějaké soubory k experimentu, ale je možné jich uložit N.



Obrázek 6.1: Struktura databáze

V tabulkách modelvars, data, aeroforcemodelvars jsou použity pro uchování matic vícerozměrné pole typu double precision. Na obrázku není znázorněn prostor pro uchování přiložených souborů, jelikož ten je plně spravován serverem, nám jsou přístupné pouze identifikátory jednotlivých souborů.

Kapitola 7

Závěr

Cílem této práce bylo navrhnout a implementovat nástroj, který by v celkovém projektu pro firmu Aero Vodochody tvořilo výstupní bod a uživatelské rozhraní, který by umožňoval vykreslení dat, jejich zpracování a archivaci. V původním návrhu dále vystupovala možnost konfigurace zařízení pro simulaci HIL (hardware in the loop), tato část byla po jednu dobu vývoje součástí i tohoto programu, ale s postupem vývoje byl původní záměr změněn právě na ono uživatelské rozhraní. Vytvořený program je schopen komunikovat s programem Matlab, získat z jeho prostředí změřená data a posléze je zpracovat a archivovat.

Zpracování experimentů bylo doplněno možností generování vzorů reportů v programu MS Word, které umožní uživateli jednodušeji vytvářet celkové reporty a prezentaci dosažených výsledků.

Vzhledem k návrhu struktury aplikace a typu využitých technologii, je možno ji nasadit bez dalších nákladů a nutnosti zakoupit doplňkový software k pracovištím vývoje.

Literatura

[1] Simon Robinson, K. Scott Allen, Ollie Cornes, Jay Glynn, Zach GreenVoss, Burton Harvey, Christian Nagel, Morgan Skinner, Karli Watson : C# Programujeme profesionálně, Wrox, Computer Press, 2003, 1. vydání, ISBN 80-251-0085-5

[2] Karel Zaplatílek, Bohuslav Doňar : Matlab tvorba uživatelských aplikací, BEN, 2004, 1.vydání, ISBN 80-7300-133-0

[3] The MathWoks, Inc. [online], dostupné na : `www.mathworks.com`

[4] PostgreSQL 8.2 Documentation [online], dostupné na : `http://www.postgresql.org/docs/8.2/static/`

[5] MSDN Library [online], dostupné na : `http://msdn.microsoft.com/en-us/library/default.aspx`

[6] MS Word Tasks [online], dostupné na : `http://msdn.microsoft.com/en-us/library/78whx7s6(VS.80).aspx`

[7] LYX [online], dostupné na : `www.lyx.org`

Použité programy

1. Microsoft Visual Studio 2005
2. Matlab 2006a
3. PostgreSQL 8.2
4. Microsoft Word 2003

A Popis instalace

1. Instalace .Net Framework 2.0 : jelikož je tato aplikace napsána za pomocí Visal Studia v jayzce C# je nutné nainstallovat toto rozhraní. .Net Framework 2.0 je možno volně získat na stránkách : `http://www.microsoft.com/downloads/` nebo na přiloženém CD

2. Instalace databáze PostgreSQL 8.2 : databázi je možno volně získat na stránkách `http://www.postgresql.org/download/`nebo na přiloženém CD. Při instalaci nejsou nutná zvláštní nastavení, stačí postupovat podle udělovaných pokynů. Pouze je nutné zapamatovat si port na kterém nastavíme datbázový server a další přístupové údaje jako logovací jméno a heslo.

3. Založení databáze : zpustíme program pgAdmin v menu kam je nainstalováná databáze PostgreSQL. Po spuštění navážeme spojení se serverem. Při prvním spuštění, je vyžadováno heslo zvolené při instalaci. Pravým tlačítkem klikneme na položku Databases a dáme vytvořit novou databázi příkazem New Database... Tabulky a jejich nastavení získáme spuštěním souboru setup.sql obsaženého na přiloženém CD.

4. Datový zdroj ODBC : v Ovládacích panelech programu MS Windows zvolíme možnost Výkon a údržba - Nástroje pro správu - Datové zdroje (ODBC). V záložce Systémové DNS zvolíme přidat. Z nabízených možností zvolíme PostgreSQL Unicode a pokračujeme Dokončit. Po otevření okna s nastavením tohoto zdroje nastavíme jeho jméno, adresu a přístupové údaje jako logovací jméno a heslo. Spojení je možné v tomto okně vyzkoušet, pokud se připojení nezdaří je nutné modifikovat údaje.

5. Microsoft Office : pro správnou funkci je potřeba naistalovat Microsoft Office 2003 Profesional edition. Při instalaci zvolit možnost vlastního nastavení a v Nástrojích sady Office přidat k instalaci možnosti : Podpora programovatelnosti aplikace Microsoft Forms 2.0 v rozhraní .NET a Podpora programovatelnosti inteligentních značek v rozhraní .NET.

6. Matlab : pro správnou funkci je potřeba program Matlab R2006a, při instalaci je možné ponechat standardní nastavení. Pro snadné spouštění příkazů tohoto

programu je nutné přidat do pracovníc cest adresář obsahující soubory exportu a importu dat.

7. Spuštění programu : při prvním spuštění asi nebude hned možné využít databázi jelikož nastavení nebude správné. Proto v záložce Nastavení zvolíme položku Conf. database. Zobrazené menu je velmi podobné jako při nastavení datového zdroje. Data nastavíme podle námi zvolených parametrů při instalaci datového zdroje. Pokud je vše správně nastaveno je možné tlačítkem Test conn. vyzkoušet funkčnost spojení nebo přímo tlačítkem Ok potvrdit toto nastavení a pokračovat v dalším využívání programu.

B Obsah přiloženého CD

1. .Net Framework 2.0
2. PostgreSQL 8.2
3. Funkce pro program Matlab
4. Spustitelná verze hlavního programu
5. Zdrojové kódy hlavního programu
6. Diplomová práce ve formátu .PDF