Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarové Inženýrství

Diplomová práce

Návrh a implementace software pro adaptaci parametrů modelů tavby oceli

Bc. Jan Kuchař

Vedoucí práce: Ing. Jan Knobloch

11. ledna 2016

Poděkování

Děkuji svému vedoucímu práce Ing. Janu Knoblochovi a svému kolegu Ing. Pavlu Skopcovi, že mi věnovali velké množství času a trpělivosti.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl 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

Kuchař, Jan. *Návrh a implementace software pro adaptaci parametrů modelů tavby oceli*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstract

In my thesis I describe development of software for adaptation of mathematical model parameters for steel melting shop. I deal here with choosing an ideal optimization algorithm and implementation of this algorithm as well. Also I deal here with filtering large amount of data, their choosing and the ideal way how to show them in charts. Software is made as a modul of already made system, so here is also dealed with the integreating and communication with the rest of the system. Nevertheless is put effort on making the software as independant as it can be.

Keywords Optimization, filtering, steel melting shop, adaptation, identification

Abstrakt

V mé práci popisuji vývoj software na adaptaci parametrů matematického modelu pro ocelárnu. Zabývám se zde výběrem ideálního optimalizačního algoritmu pro daný problém i samotné implementaci vybraného algoritmu. Dále se zde zabývám filtrováním velkého množství dat, jejich výběrem a zobrazením do grafů. Software je dělaný jako modul již fungujícího systému, tudíž je zde také řešena integrace a komunikace s již zaběhnutým systémem. Nicméně je dbán důraz na to, aby byl tento produkt co nejméně závislý na ostatních komponentác, již fungujícího systému.

Klíčová slova Optimalizace, filtrovaní, ocelárna, adaptace, identifikace

Obsah

Úvod **1**

1 Specifikace cílů **3**
1.1 State of Art . . . 3
1.2 Problém . . . 4
1.3 Motivace . . . 5
1.4 Cíl . . . 5
1.5 Ocelárna . . . 6
1.6 Požadavky . . . 7

2 Návrh řešení **13**
2.1 Aktuální stav . . . 13
2.2 Softwarový návrh . . . 13
2.3 Databáze . . . 34
2.4 IDE . . . 37

3 Funkce a Implementace **39**
3.1 Syntaxe . . . 39
3.2 Obecně . . . 40
3.3 Filtr . . . 40
3.4 Vyhodnocení chyb . . . 48
3.5 Optimalizace . . . 50
3.6 Databáze . . . 55
3.7 Tavby . . . 57
3.8 Práce s parametry . . . 58
3.9 Práce s prvky chemické analýzy . . . 60
3.10 Zobrazování chyby . . . 60

3.11 Testování 62

Závěr **65**

Literatura **67**

A Seznam použitých zkratek **69**

B Obsah přiloženého CD **71**

Seznam obrázků

1.1 Identifikace modelu 4
1.2 Blokové schéma modelu ocelárny - 1 6
1.3 Blokové schéma modelu ocelárny - 2 7

2.1 Tabulky C_RECORD_DESC,PE_RECORD_DET, PE_RECORD 16
2.2 Composite CFilter CFilterGroup CFilterGroupList 18
2.3 Strategy Pattern 19
2.4 Strategy Pattern v Automatickém filtru 20
2.5 Automatický filtr 21
2.6 Manuální filtr 22
2.7 Graf závislostí ve filtru 23
2.8 Optimalizace 25
2.9 Vyhodnocení chyby 30
2.10 Návrhový vzor Visitor 31
2.11 Výpočet chyby - Visitor 32
2.12 Čítač 33
2.13 Graf závislostí ve vyhodnocení chyb 34
2.14 Databázový návrh filtru 35
2.15 Databázový návrh filtru 36
2.16 Databázový návrh M_PARAMETER_ADAPT 37

3.1 Hlavní obrazovka 41
3.2 Filter 42
3.3 Save Filter Dialog 43
3.4 Automatic Filter 44
3.5 Specify Filter Dialog 45
3.6 Manuální filtr - zobrazení celková hodnota 46
3.7 Manuální filtr - zobrazení jednoduchá hodnota 47

3.8 Time Filter 48
3.9 Optimalizace - Genetický algoritmus 52
3.10 Optimalizace - Simulované ochlazování 53
3.11 Optimalizace - Brute Force 53
3.12 Brute Force - výsledek 54
3.13 Ukládání a načítání konfigurací 57
3.14 Tavby 58
3.15 Parametry 59
3.16 Chemická analýza 60
3.17 Zobrazování chyby 61

Úvod

Pro dosažení přínosu v provozu ocelárny - zpřesnění řízení ohřevu elektrickým obloukem, úspora měřících sond, snížení počtu provedených chemických analýz, zrychlení ohřevu a podpora pro plánování a koordinaci taveb - byly sestaveny matematické modely procesů elektrické obloukové pece (EAF), pánvové pece (LF) a vakuové jednotky (VD). Tyto modely predikují teplotu a materiálové složení na základě energetické a materiálové bilance. Na bázi těchto modelů je možno sestavit řídící algoritmy, které automatizují a optimalizují proces výroby oceli.

S pomocí identifikace, respektive adaptace vybraných fyzikálních parametrů, ale i počátečního stavu tavby lze zlepšit výsledky modelů. Adaptace spočívá v postupném nahrazování parametrů modelu parametry, které lépe vyhovují zvolené kriteriální funkci. Odhad parametrů vychází z optimalizace přes velké množství dostupných dat z reálného běžícího procesu. Problém malého počtu měření, nepřesnosti a nedostupnosti měření, neznalosti počáteční teploty je možné řešit zpracováním mnoha realizovaných taveb.

V této práci je popsán postup při navrhování a implementaci softwaru pro identifikaci resp. adaptaci parametrů matematického modelu. Jaké technologie byly použity a je zde taktéž popis vlastních komponent, datového a grafického rozhraní tohoto softwaru.

Kapitola 1

Specifikace cílů

1.1 State of Art

1.1.1 Identifikace

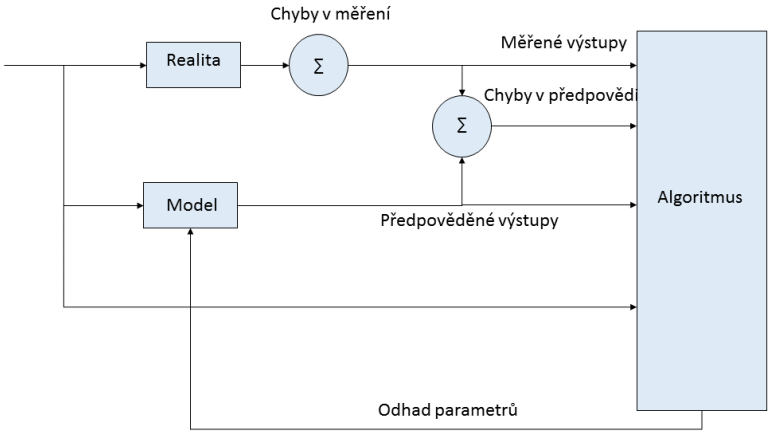
„Pod pojmem identifikace modelu se skrývají 3 pojmy.

- Odhad parametrů - proces hledání hodnot parametrů, které vedou k co nejlepší shodě výsledků modelu s měřenými daty pomocí některé ze statistických metod (např. minima čtverců).
- Citlivostní analýza - cílem citlivostní analýzy je vyšetření změn výsledků modelu v závislosti na parametru modelu
- Propagace chyb v modelu - vyhodnocení nejistot jednotlivých parametrů na celkovou nejistotu modelu

“[1]
Diagram identifikace modelu je na obrázku 1.1

1.1.2 Adaptace

„Modelová adaptace je zaměření spojené se „Machine learning“. Tento případ nastává, když se zaměříme na naučení se distribuci zdrojových dat a dobře fungujícího modelu na jiné (ale související) distribuce výsledků. Například jeden z úkolů běžného výběru „spamu“ obsahuje adaptaci modelu jednoho uživatele (zdrojová distribuce) k novému uživateli, který přijímá značně rozdílný email (cílová distribuce). V případě, kdy je k dispozici více než jedna zdrojová distribuce, hovoříme o adaptaci „multi-source domény.““ [2]



Obrázek 1.1: Identifikace modelu

1.2 Problém

Zákazník by rád optimalizoval svou výrobu. Ta by měla být optimalizována ve smyslu vyrobit co maximální počet výrobků v co největší možné kvalitě. Tento problém byl zadán firmě PTSW s požadavkem na určitý softwarový nástroj pro automatizaci a optimalizaci nějaké části výroby. Proces výroby oceli je složitý a nejde snadno fyzikálně popsat, nicméně určité hlavní veličiny tohoto procesu lze měřit, a to jak vstupy, tak jejich výstupy.
V ocelárně se během tavby oceli sbírá spoustu Během testování se ukázalo, že určité komponenty je možné ještě upravit, aby byla efektivita softwaru větší. Do budoucna se tedy plánuje s tímto softwarem ještě pracovat a upravovat jeho komponenty. Komponenta, která se ukázala jako velmi efektivní a dala by se použít i v jiných částech systému, je komponenta filtr taveb. Celý software je navržen tak, aby v okamžiku jakéhokoli rozšíření o algoritmus nebo o určitou strategii nebylo třeba velkého zásahu do systému. Celý software je patřičně okomentován a byla snaha psát kód co nejsrozumitelněji, jelikož se dá předpokládat, že v průběhu času jej bude upravovat spousta dalších programátorů. dat, měření však nejsou vždy úplně přesná, jelikož k nim nemusí docházet vždy ve stejnou dobu nebo ve stejné teplotě oceli. Dalšími vlivy může být porucha měřidel, komunikace mezi operátory, špatně vložené hodnot atd. Taktéž chemická analýza se neprovádí vždy

stejně. Tudíž data se nenachází vždy v konzistentním stavu.

1.3 Motivace

Vedoucím této externě zadané práce je Ing. Jan Knobloch z firmy PTSW. Firma PTSW za zaměřuje na automatizaci výroby a vývoj procesního systému do továren všeho druhu. Jeden z jejích dlouhodobých zákazníků je ocelárna Bushan v Indii, kam firma PTSW implementovala svůj procesní systém na řízení výroby a jiné zakázky podle požadavků zákazníka.
Návrh na tuto práci vzešel z domluvy ve vedení firmy PTSW, a to zejména z iniciativy Ing. Pavla Skopce a z jeho znalostí tavebních procesů a jejich matematických modelování v ocelárně. Tyto modely se snaží matematicky modelovat tavbu oceli, ale setkávají se s různými úskalími a jedním z nich je, že tyto modely potřebují určitou sadu parametrů, které velmi ovlivňují výsledky modelu. Tyto paramtry se však musí v současné době nastavovat ručně. V tomto procesu nastavování parametrů je hlavním problémem určit hodnotu všech parametrů tak, aby výsledná chyba modelu byla co nejmenší. Další vlna iniciativy vzešla i od zákazníka, který by rád optimalizoval svou činnost ve smyslu max. počet a kvalita výrobků za minimální náklady. Na základě popsání tohoto problému byly popsány požadavky na software, jenž by řešil tento problém co nejefektivněji. Z těchto požadavků tedy vzešlo zadání projektu, které mi bylo následně svěřeno a jehož postup řešení popisuji v této práci.

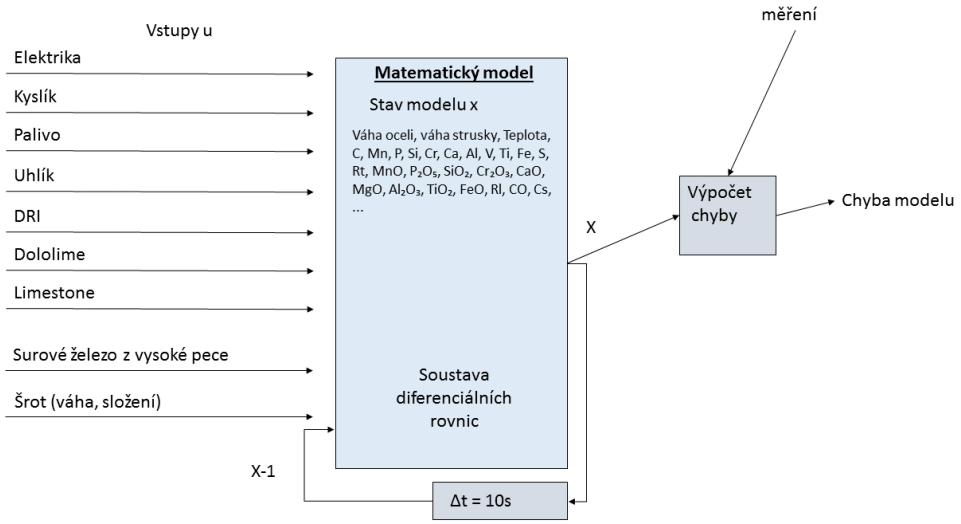
1.4 Cíl

Konečným cílem je tedy vytvořit softwarový nástroj pro automatizaci a optimalizaci výroby. Tento softwarový nástroj využívá matematický model, který je třeba nějak identifikovat. Pro zlepšení výsledků matematického modelu je třeba tento model nějak identifikovat. Bude tedy snaha najít (identifikovat) tento model jako matematický vztah mezi jeho vstupy a výstupy. Proces, kterým se toto bude provádět bude následující: zvolíme nějakou strukturu (nebo struktury) modelu, která bude mít určité parametry a budeme měnit hodnoty těchto parametrů tak, aby chyba mezi měřenými výstupy z procesu tavby oceli a výstupem z modelu procesu tavby oceli byla minimální nebo uspokojivá. Celou tuto minimalizaci (optimalizaci) je třeba automatizovat. Tato optimalizace bude prováděna přes velké množství dat, aby se z výsledků daly následně dělat různé statistiky. Proces automatické optimalizace parametrů je velice citlivý na správné vstupy (vstupy

+ výstupy procesů), v sekci 1.2 již bylo zmíněno, že data nejsou právě konzistním stavu. Když se používá například metoda nejmenších čtverců (tzv. kvadratické kritérium) k vyhodnocení chyby, tak chybné hodnoty a zejména tzv. outliers hodnoty mohou znamenat ve finále velký problém. Z tohoto důvodu je třeba věnovat velké množství času a práce do předzpracování vstupních dat. Softwarový nástroj by tedy měl být schopný odfiltrovat vstupní hodnoty modelu a provést na těchto datech následnou identifikaci parametrů a jejich optimalizaci. Toto všechno by mělo být zobrazováno a uživatel by měl mít co největší možnost ovlivnit průběh jak filtrace, tak optimalizace.

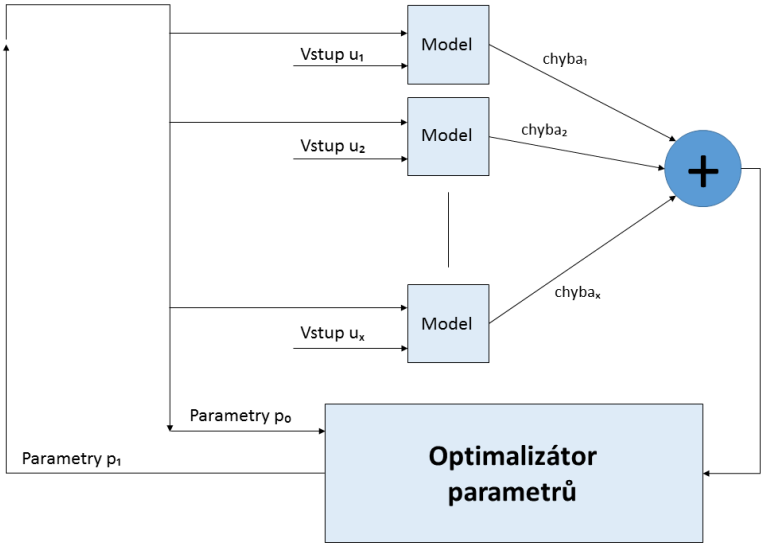
1.5 Ocelárna

V ocelárně funguje matematický model, který přijímá cyklické vstupy **u** (elektrika, kyslík, palivo atd.). Tyto cyklické vstupy se opakují každých 10 vteřin. Dále matematický model přijímá vstupy necyklické (Surové železo, šrot), které přicházejí v různých časových intervalech. Matematický model se vždy nachází v nějakém stavu, který je určen váhou oceli, váhou strusky, teplotou a chemickým složením. Pomocí soustavy diferenciálních rovnic se vyopčítává tento stav modelu. Porovnáním s reálnými hodntami měření se bude vypočítávat chyba modelu. (obrázek 1.2).



Obrázek 1.2: Blokové schéma modelu ocelárny - 1

Spuštění tohoto modelu proběhne několikrát respektive pro každou tavbu je spuštěn jeden model. Získá se tedy několik chyb modelu, které se předají „optimalizátorovi parametrů“, aby upravil parametry podle chyb. Celý proces se následně opakuje. (obrázek 1.3)



Obrázek 1.3: Blokové schéma modelu ocelárny - 2

1.6 Požadavky

Požadavky bývají kategorizovány několika způsoby. V této práci je budu rozdělovat pouze na funkční a nefunkční.

1.6.1 Funkční

„Funkční požadavky objasňují a identifikují nutné úkony, aktivity a akce, které musí být vykonány. Analýza funkčních požadavků bude použita jako základ takzvané toplevel funkce systému pro funkční analýzu“ [3] Software by se měl skládat z těchto hlavních funkčních komponent

- Filtr vstupních dat
- Optimalizace vybraných parametrů
- Vyhodnocení chyb modelu

- Výběr pece, na které chceme provádět optimalizaci
- Výběr části pece (Unit), na které chceme optimalizaci provádět
- Výběr chemického složení, na kterém se bude kromě teploty počítat chyba modelu

1.6.1.1 Filtr

Funkce filtrovaní dat (taveb) by měla mít možnost filtrovat přes všechny možné hodnoty tavby. Těchto hodnot je spousta a pro každou pec (EAF, LF, VD) se tyto hodnoty liší. Filtr by měl umět vytvořit pro každou pec všechny možné filtry přes jednotlivé hodnoty tavby. Dále by měl filtr mít tyto základní funkce.

- zobrazit všechny dostupné filtry
- vybrat konkrétní filtr
- uložit skupinu vybraných filtrů s jejich nastaveními
- načíst uložené filtry

Automatický filtr Automatický filtr by měl filtrovat tavby bez jakéhokoli zásahu uživatele. Uživatel by nicméně měl mít možnost si filtr co nejvíce přizpůsobit svým potřebám. Filtr by měl mít dvě možnosti filtrování.

- Minimální a maximální hodnota - uživatel zadá minimální a maximální hodnotu. Filtr následně otestuje všechna data tavby pro vybraný filtr a určí zda jsou data v limitu. V případě, že tomu tak není, filtruje tavbu jako závadnou.
- Odchylka od určité hodnoty - uživatel určí maximální procentuální rozdíl od vybrané hodnoty. filtr proté otestuje všechny data tavby pro vybraný filtr a určí jestli jsou data v limutu. V případě, že data nejsou v limitu, filtruje tavbu jako závadnou.
  Vybraná hodnota může být dvojího typu.
  - aritmetická hodnota
  - medián

Dále by měl mít možnost upravovat hodnoty taveb v případě, že je tak nastaveno uživatelem, a určovat kolik špatných hodnot v jedné tavbě určuje závadnou tavbu. Tudíž by filtr měl mít tu možnost nastavit procentuálně,

kolik „chyb“ je v jednotlivé tavbě dovoleno. Každá taková chyba bude poté opravena na průměrnou hodnotu tavby. V případě, že procento chyb v tavbě bude větší než hodnota zadané uživatelem, tavba bude vyřazena jako závadná.

Manuální filtr Manuální filtr by měl zobrazovat uživateli průběh tavby v čase. Uživatel by pak v přehledném grafu by měl sám vidět, která tavba se odchyluje a která naopak udržuje standartní hodnoty. Uživatel by měl mít možnost zobrazit id tavby, číslo tavby a hodnoty. Uživatel by měl mít možnost si vybrané tavby zobrazit v jednom grafu, a tak mít možnost zobrazit každou tavbu v samostatném grafu. Graf by měl mít dvě možnosti zobrazení

- Průběh hodnoty v čase - u každé tavby se ukazuje celková hodnota veličiny v čase
- Hodnota události - u každé tavby se ukazuje hodnota jednotlivé události veličiny v čase, kdy se udála

Uživatel by měl mít také možnost tavbu odfiltrovat jako závadnou.

1.6.1.2 Optimalizace

Funkce optimalizace bude nejdůležitější částí celého softwaru. Požadavky na tuto část jsou docela jednoduché. Matematický model používá řadu parametrů, které je třeba optimalizovat tak, aby model dával co nejlepší výsledek. Optimalizace by měla mít možnost probíhat více algoritmy. Uživatel by neměl být závislý pouze na jednom algoritmu, ale měl by mít na výběr z více možností. Taktéž je třeba, aby si uživatel mohl vybrat parametry, které chce optimalizovat. Někdy není třeba nebo nemusí být žádoucí optimalizovat všechny parametry, ale chceme vybrat nějakou podskupinu parametrů. Optimalizační algoritmy mívají své vlastní nastavení a své vlastní parametry, je tedy třeba aby tyto parametry byly nastavitelné uživatelem tak, aby uživatel mohl korigovat chování algoritmů.
Je třeba, aby bylo možné vybrat parametry modelu, které se budou optimalizovat. U těchto parametrů je žádáno, aby se zde dala nastavit minimální a maximální hodnota, kterých bude hodnota parametru nabývat. Je třeba, aby byla zobrazená výsledná hodnota optimalizovaných parametrů a byla zde možnost tyto hodnoty uložit. Stejně tak je třeba, aby se daly ukládat minimální a maximální hodnoty parametrů.
Výběr optimalizačních algoritmů není určen až na algoritmus „Brute Force“, který by ale neměl za úkol přímo optimalizovat, ale ukazovat závislost.

Brute Force: Při „Brute Force“ se vybere jeden z parametrů modelu a zadají se hodnoty:

- minimální mez
- maximální mez
- krok algoritmu

Algoritmus bude následně testovat chybu modelu pro měnící se parametry od minimální po maximální, kde hodnota bude upravována vždy o hodnotu krok algoritmu. „Brute Force“ tedy získá sérii chyb, ze kterých zobrazí uživateli graf, kde bude zobrazena chyba modelu v závislosti na hodnotě parametru.

1.6.1.3 Vyhodnocení chyb

Software by měl umět vyhodnotit chyby modelu jedné, ale hlavně mnoha taveb a zobrazit je. Toto vyhodnocení by mělo být uděláno pomocí dvou metod.

- Kvadratické kritérium (kriteriální funkce metody nejmenších čtverců)
- Absolutní hodnota

Taktéž by měl mít možnost spočítat chybu modelu pro více taveb současně, a to pomocí

- Aritmetický průměr
- Medián

Software by měl chybu vyhodnocovat pro několik parametrů.

- Teplota
- Chemická analýza

V chemické analýze je obsaženo několik chemických prvků a toto vyhodnocení je třeba aplikovat na každý z těchto prvků. Uživatel by měl mít možnost si vybrat prvky, které chce použít pro měření chyby.

1.6.2 Nefunkční

- Software by měl být nezavislý na konkrétní výrobě nebo procesu
- Software by měl být co možná nejméně závsilý na zbytku systému, do budoucna by bylo dobré tuto komponentu přesunout do PTSW frameworku
- Software by měl být napsán v jazyce C# za použití Visual Studia 2010
- Databáze by měla být použita Oracle Database g11

1.6.2.1 Uživatelské rozhraní

Uživatelské rozhraní by mělo být zakomponováno do uživatelského rozhraní systému PTSW, který je již v ocelárně nainstalován. Dále by se mělo co možná nejvíc komponent nacházet na hlavní obrazovce tohoto modulu a zdřovat se co možná nejvíc vyskakovacích obrazovek (pop-up). Uživatelské rozhraní by mělo splňovat Nielsonovu heuristickou analýzu.

Kapitola 2

Návrh řešení

2.1 Aktuální stav

V aktuální verzi systému, jenž je momentálně používána v ocelárně, není žádná starší verze tohoto softwaru. O tento optimalizační a filtrovací software bylo provedeno mnoho pokusů, ale žádný z nich se nedostal do finální verze produktu PTSW. V systému sice existuje simulace procesu tavby a jsou zde vidět výstupy a grafy matematických modelů, nicméně tyto modely se nedají nijak upravovat a nemůžeme u nich ani zobrazovat průměrné chyby. Tento software tedy bude jak z hledisku filtru taveb, tak z hlediska optimalizace parametrů první.
Nejedná se přirozeně o první takový software na světě. Takovýchto optimalizačních softwarů i do ocelářských prostředí bylo sestrojeno několik. Nicméně tento software bude přesně vytvořen pro potřeby firmy PTSW, tak aby ho mohla dále nabízet svým klientům.

Procesu identifikace modelu se věnuje spousta odborné literatury. Zejména System Identification od Lennarta Ljunga ([4]) a Nonlinear System Identification od Oliver Nellesa. ([5])

2.2 Softwarový návrh

2.2.1 Obecně

Software by měl být složen ze dvou hlavních komponent.

- Filtr

- Optimalizace

V rámci optimalizace bude ještě nekolik dalších komponent.

- Vyhodnocení chyb modelu
- Výběr parametrů a jejich mezních hodnot
- Výběr prvků pro chemickou analýzu

Z důvodu velkého množství taveb a jejich různých nepřesností bude třeba filtr taveb. Tyto tavby, které budou následně filtrem ohodnoceny, že jsou v pořádku, budeme matematicky simulovat pomocí modelu, jež je již vtvořený. Tento model simuluje proces tavby oceli podle událostí, jenž se během tavby oceli vyskytly a jsou uloženy v databázi. Událost v tavbě přichází cca každých 10 vteřin. Několikrát během procesu tavby přijde i událost necyklická, ve které je změřená teplota popřípadě to znamená, že byl odebrán vzorek a změřena chemická analýza tavby. Vždy v tomto okamžiku, kdy do matematického modelu přijde událost, která naní cyklická se změří odchylka (chyba) mezi reálnou změřenou hodnotou a hodnotou simulovanou matematickým modelem. Těchto odchylek (chyb) vznikne během jedné tavby několik. Ze všech těchto odchylek (chyb) je třeba pomocí určitých metod udělat jednu finální chybu, která bude reprezentovat chybu skupiny parametrů, podle kterých matematický model počítá. Tyto parametry pak budou optimalizovány se snahou o to aby výsledná chyba matematického modelu byla co nejmenší.

2.2.2 Filtr

Filtr bude jedna z nejdůležitejších částí celého projektu. Jak už bylo řečeno, pokud nebudou data správná, nebo některá data budou nepřesná a budou v nich chyby, tak celá optimalizace (a z ní pak udělané statistiky) nebudou mít pořebnou váhu, popřípadě nebudou vůbec správné. Jak už bylo uvedeno ve funkčních požadavcích (1.6.1), filtr by měl být rozdělen na dvě hlavní části.

- Automatický filtr
- Manuální filtr

Při prozkoumávání dat jsem došel ještě k názoru, že dalším filtrem, který by mohl být užitečný, je časový filtr. Proto jsem se rozhodl do návrhu zahrnout i jednoduchý časový filtr, ačkoli nebyl uveden v požadavcích na filtr.

Hlavní částí filtru bude vizuální komponenta, ve které se budou zobrazovat názvy všech filtrů. Měl by to být jakýsi seznam filtrů s možností volby filtrů. Dále podle funkčních požadavků je třeba tuto skupinu filtrů uložit, popřípadě načíst. Ukládání i načítání bude prováděno přes databázové tabulky a bude tomu více věnováno v sekci Databáze 2.3.
Abychom správně navrhli všechny třídy a celou strukturu filtru, musíme nejdříve pochopit data, která filtrujeme a kde je budeme brát. Data jsou uložena v databázi v několika tabulkách. Pro nás je velice důležitá informace, že musíme filtrovat data přes všechny tři pece (EAF, LF, VD). Z tabulek se dá zjistit, které typy dat každá pec má, a daly by se tedy všechny filtry vytvořit tzv. „nahrubo“. Nicméně to by nesplňovalo nefunkční požadavek, že by celý software měl být nezávislý na konkrétní výrobě či procesu. Tudíž celý filtr musí být tvořen dynamicky. Data pro všechny pece jsou uloženy ve stejných nebo podobných tabulkách.

- `C_RECORD_DESC` - zde je popis všech dat, které mohu přijít
- `PE_RECORD_DET` - zde se nacházejí konkrétní data pro pro pec EAF, pro pec LF je tabulka pojmenovaná (`PL_RECORD_DET` popř. `PV_RECORD_DET` pro pec VD)
- `PE_RECORD` - jednotlivé události, které postupně přichází od řídícího systému pro pec EAF (`PL_RECORD` popř. `PV_RECORD` pro pec LF popř. VD)

`PE_RECORD` Tabulky `PE_RECORD` pro EAF, `PL_RECORD` popř. `PV_RECORD` pro LF popř. VD, mají všechny stejnou strukturu.

- ID - id jednotlivé události
- `PROCUCT_ID` - id tavby, pod tímto údajem je tavba uváděna v systému
- `C_PLANT_UNIT_ID` - id pece, na které je tavba prováděna
- `TAG_NAME` - jméno události (tagu), která byla změřena

V tabulce jsou uchovávány události o jednotlivých tavbách, které se v peci zpracovávají. Tyto události se dělí na tzv. „cyklické“ a „necyklické“ události. „Cyklické“ události se ukládají zhruba každých deset vteřin. Zatímco události „necyklické“, jako například měření teploty, přichází nezávisle. To, která událost je uložena, vidíme právě ve sloupci `TAG_NAME`. Každá událost patří do určité skupiny událostí. Pro každou tavbu `PROCUCT_ID` je v této tabulce uloženo několik stovek záznamů.

Obrázek 2.1: Tabulky `C_RECORD_DESC`,`PE_RECORD_DET`, `PE_RECORD`

`PE_RECORD_DET` Tabulky `PE_RECORD_DET` pro EAF, `PL_RECORD_DET` popř. `PV_RECORD_DET` pro LF popř. VD, mají všechny stejnou strukturu.

- ID - id jednotlivého záznamu
- `MASTER_RECORD_ID` - id události ze `PE_RECORD`
- `GROUP_NAME` název skupiny událostí, do které daná událost patří.
- ...následuje několik desítek sloupců pojmenovaných `DATA_INDX0` - `DATA_INDX29`

V této tabulce jsou ukládána data, jednotlivé hodnoty pro tavby a jejich události. Hodnoty jsou ukládány do různých (DATA) sloupců.

`C_RECORD_DESC` V této tabulce je uložen popis jednotlivých událostí. Je to konfigurační tabulka a nic se do ní již nepřidává.

- ID - id záznamu
- `C_PLANT_UNIT_ID` - id pece, na které je událost prováděna

- GROUP_NAME - název skupiny událostí, do které daná událost patří.
- L2_TAG - celkový název události
- TAG_INDEX - index sloupce, ve kterém se nacházejí data pro tuto událost, v tabulce PE_RECORD_DET popř. PL_RECORD_DET popř. PV_RECORD_DET
- TABLE_NAME - Ve které tabulce se událost bude nacházet PE_RECORD popř. PL_RECORD popř. PV_RECORD

Shrnutí: V každé peci při každé tavbě se ukládá mnoho událostí s mnoho hodnotami. Typy těchto událostí jsou známé a jsou uložené v tabulce C_RECORD_DESC. Každá jednotlivá událost je uložena do tabulky PE_RECORD (popř. PL_RECORD nebo PV_RECORD). Data každé události jsou uložena v tabulce PE_RECORD_DET (popř. PL_RECORD_DET nebo PV_RECORD_DET).

2.2.2.1 Výroba filtru

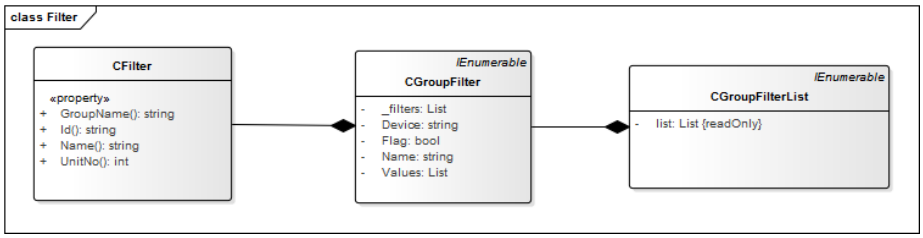
Filtr bude prováděn podle jednotlivých událostí. Tyto události jsou udržovány v určitých skupinách. Tudíž víme, že pro každou pec budeme mít seznam skupin, kde každá skupina bude obsahovat minimálně jeden filtr. Problém nastává, že u některých skupin na stejnou událost mohou přijít různé hodnoty. Toto se týká např. skupiny BURNER, která na peci EAF kontroluje stav hořáků. Nicméně tyto „hořáky“ jsou na peci 3. Tudíž filtr ještě musí rozlišovat mezi těmito rozdíly.

Filtry budou připraveny následovně:

1. V cyklu přes všechny názvy skupin, které jsou pro určitou pec v tabulce C_RECORD_DESC
2. Z tabulky C_RECORD_DESC se načtou údaje ze sloupce L2_TAG.
   - L2_TAG je ve formátu ....DEVICE.GROUP_NAME.TAG_NAME, kde DEVICE je jméno pece, ve které se událost nachází, GROUP_NAME je název skupiny a TAG_NAME je název události.
     (např. ...EAF.ELECTRIC_AC.TotalEnergy)
   - V několika případech se stane, že název skupiny nesouhlasí s názvem skupiny, která se nachází v L2_TAG. V takovém případě je většinou za názvem skupiny číslo konkretizující skupinu, např. pro skupinu BURNERN ...EAF.BURNER1.TotalFuel). Toto je tedy ten případ, kdy je třeba konkretizovat skupinu jako takovou. Čísla za názvem skupiny jsou hodnoty, které konkretizují událost.

3. při postupném procházení této `C_RECORD_DESC` se bude vytvářet objektová struktura reprezentující filtr. Každý filtr bude ukládat Id události z `C_RECORD_DESC`, název, číslo sloupce, kde jsou uložena příslušná data v tabulce `..._RECORD_DET`, číslo zařízení a název skupiny. Každý jednotlivý objekt bude uložen v kolekci filtrů, které patří do stejné skupiny. Tato skupina bude ukládat název zařízení, na kterém se nachází název skupiny, jednotlivé filtry do skupiny patřící. Z důvodu skupin typu BURNER, které jsou tzv. nejednoznačné, se bude v těchto skupinách uchovávat flag jednoznačnosti a pole konkretizujících hodnot. Třídu CGroupFilter bude třeba taky používat v kolekci, pro tento účel bude vytvořena třída CFilterGroupList. Ttřída CGrouFilterList bude tedy obsahovat všechny možné filtry pro jedno určité zařízení.

Takto bude vypadat objektová struktura filtru pro jednotlivé zařízení.



Obrázek 2.2: Composite CFilter CFilterGroup CFilterGroupList

2.2.2.2 Automatický filtr

Automatický filtr bude filtrovat hodnoty automaticky na základě určené strategie. Podle požadavků máme strategie dvě.

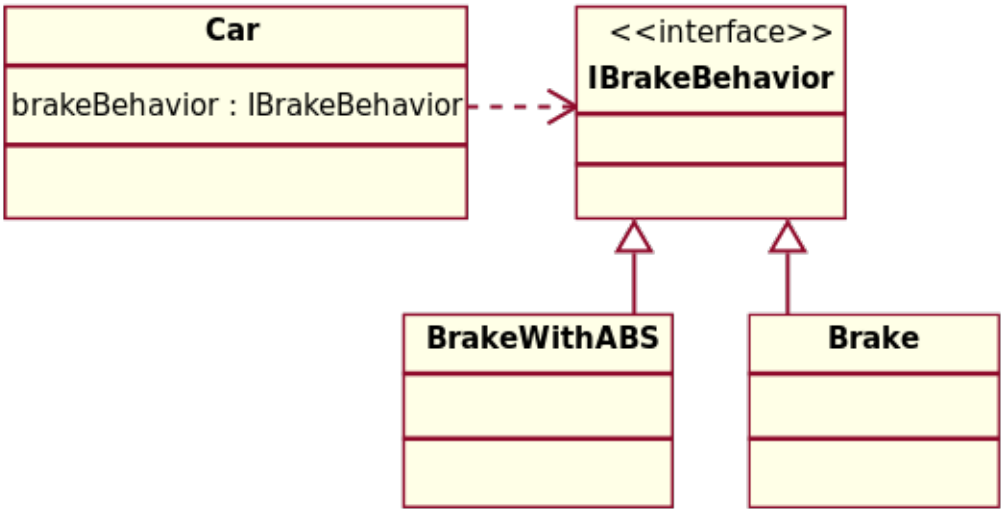
- Minimální maximální hodnota
- odchylka od hodnoty

Filtr bude tvořen jednoduchým grafickým rozhraním, kde uživatel zvolí pro každý vybraný filtr typ strategie a příslušné hodnoty. Dále zde bude pole, kde uživatel zadá procento nevyhovujících hodnot.
Filtr bude fungovat v cyklu, v němž bude postupně procházet jednotlivé hodnoty taveb a vždy rozhodovat, zda daná hodnota splňuje kritéria zadaná uživatelem. V případě, že hodnota nebude splňovat kritéria, zvýší se čítač chyb. V případě, že tento čítač překročí povolenou hranici nastavenou uživatelem, tavba se prohlásí za „závadnou“. V opačném případě se

pouze hodnota nastaví na průměrnou hodnotu tavby a tavba bude uznána za „nezávadnou“. O nastavování hodnot se píše v sekci Úprava hodnot tavby(2.2.2.5)

Strategie filtru budou řešeny Strategy Patternem.

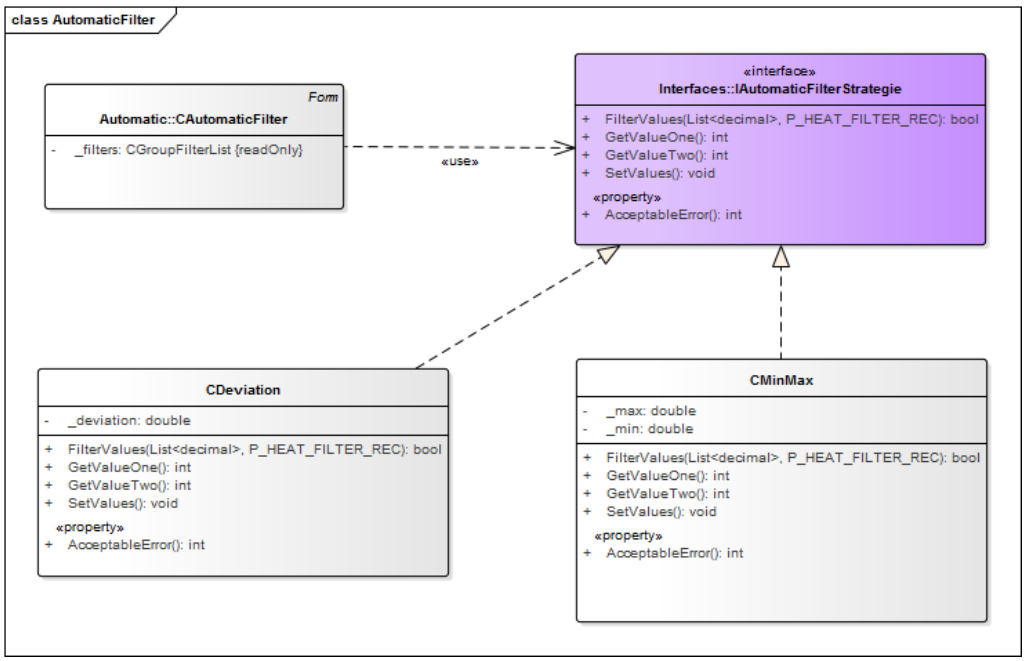
Strategy Pattern „V programování Strategy pattern je softwarový návrhový vzor, který umožňuje, aby chování algoritmu bylo vybráno za běhu programu. Strategy pattern nechává algoritmus nezávislý na konkrétním klientovi, který ho využívá.“ Objektový návrh Strategy Patternu je vidět na obrázku 2.3.



Obrázek 2.3: Strategy Pattern

Objektový návrh mého řešení Strategy patternu je vidět na obrázku 2.4.

Automatický filtr bude řešen hlavní vizuální třídou AutomaticFilter. Tato třída bude sbírat data od uživatele a předávat vše třídě CAutomaticFilter, která bude zpracovávat filtrovací logiku pomocí výše zmíněného Strategy Patternu a tříd, které se nacházejí v něm.
Na třídě CAutomatic filter bude ještě závislá jednoduchá vizuální třída, ve které bude uživatel u nekonkrétních skupin konkretizovat svůj výběr. Např. u skupiny BURNER se dá uživateli na výběr, ze kterého hořáku se mají data brát. Návrh automatického filtru je na obrázku 2.5

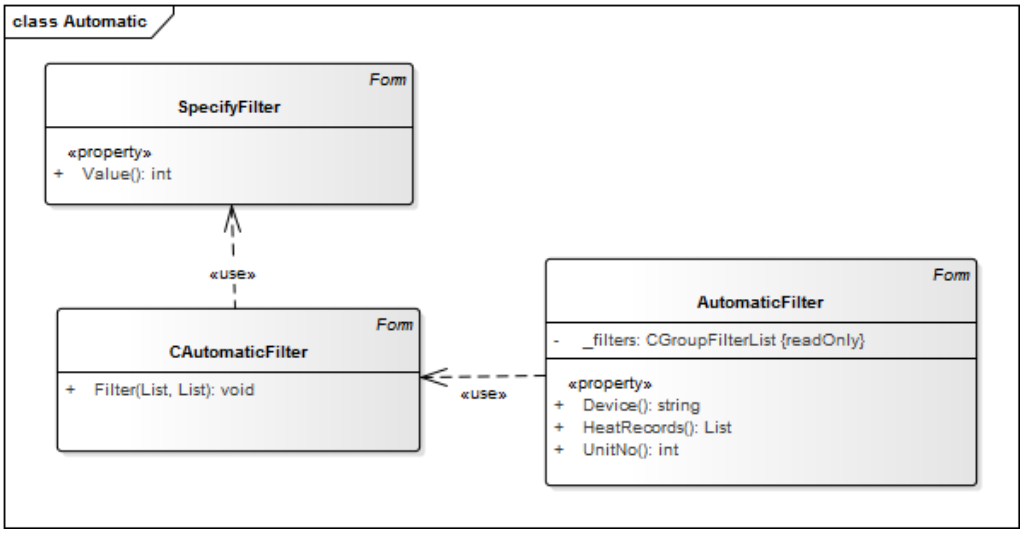


Obrázek 2.4: Strategy Pattern v Automatickém filtru

2.2.2.3 Manuální filtr

Manuální filtr bude zobrazovat všechny hodnoty tavby v grafu. Uživatel bude mít následně možnost si data v grafu prohlédnout a určit, které jsou nevyhovující a které jsou naopak přípustné. Uživatel by měl mít možnost v grafu vidět, o kterou tavbu se jedná, popřípadě danou hodnotu tavby.

V Manuálním filtru tedy bude uživatelské rozhraní, které by mělo zobrazovat vybrané filtry a možnost jejich zobrazení do grafu. Dále by zde měl být graf s možností vymazávání nevhodné tavby ze skupiny filtrovaných taveb. Manuální filtr bude načítat data z databáze podle stanoveného filtru a vybraných taveb. Tato data budou muset být předzpracována, aby byla snadno zobrazitelné v grafu. Další problém manuálního filtru bude „Filtrovací menu“. Toto menu bude vytvořeno na základě výběru filtrů uživatelem. Pro každý filtr bude třeba vytvořit spouštěcí tlačítko, případně u filtrů, jež nejsou konkrétní, i dodatečný výběr. Jednotlivé filtry bude třeba zobrazovat ve skupinách, do kterých patří.



Obrázek 2.5: Automatický filtr

Graf: Graf bude mít dvě možnosti zobrazení hodnot, které se budou do grafu načítat.

- časová posloupnost - graf bude zobrazovat celkovou hodnotu v závislosti na čase
- hodnota události - graf bude zobrazovat pouze hodnotu událostí v závisloti na čase

Graf bude mít dvě možnosti zobrazení taveb.

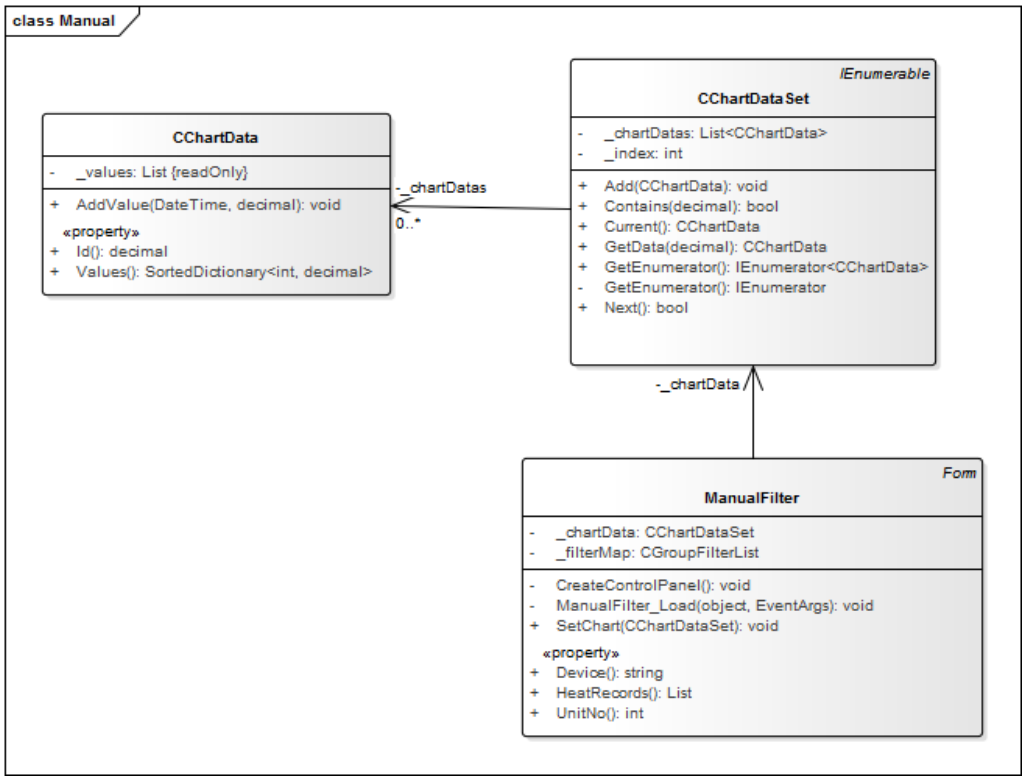
- všechny tavby v jednom grafu
- každá tavba bude mít svůj samostatný graf

Hlavní třídou u grafu bude třída, kterou nazveme CChartData. Tato třída bude obsahovat data pro jednu tavbu vybranou podle příslušného filtru. Data, která se budou z databáze brát, budou dvoje. Jednak to bude samotná hodnota události, jež se stala, a také to bude čas události, tedy časový údaj, kdy událost proběhla. Na základě tohoto časového údaje budeme moct zobrazit hodnoty v grafu závisle na času, a to v obou variantách zobrazení. Třída CChartData bude napojena na třídu CChartDataSet, což bude kontainer pro tuto třídu.

Třída CChartDataSet bude obsahovat všechna data, která bude graf zobrazovat.

Filtrovací menu: Za vytvoření filtrovacího menu bude zodpovědná třída CGroupFilter. Tato třída představuje skupinu filtrů vybraných uživatelem. Pro každý filtr bude tato třída dynamicky tvořit „tlačitko“ a všechna tato „tlačitka“ vloží do „GroupBoxu“, který bude taktéž dynamicky vytvořen. „Tlačitku“ bude přidán „handler“, který bude zodpovědný za chování programu po „stistknutí tlačítka“. Tento „handler“ načte příslušná data z databáze do struktury popsané v paragrafu 2.2.2.3 Po vytvoření bude tato struktura předána třídě ManualFilter, která data zobrazí do grafu.

Manuální filtr tedy bude řešen vizuální třídou ManualFilter, která bude zobrazovat patřičná data a načítat je ze struktury popsané v paragrafu 2.2.2.3.



Obrázek 2.6: Manuální filtr

2.2.2.4 Časový filtr

Časový filtr bude jednoduchý dialog, kde uživatel pomocí komponentů na vybraní časového údaje zadá dvě data.

- startovní datum
- koncové datum

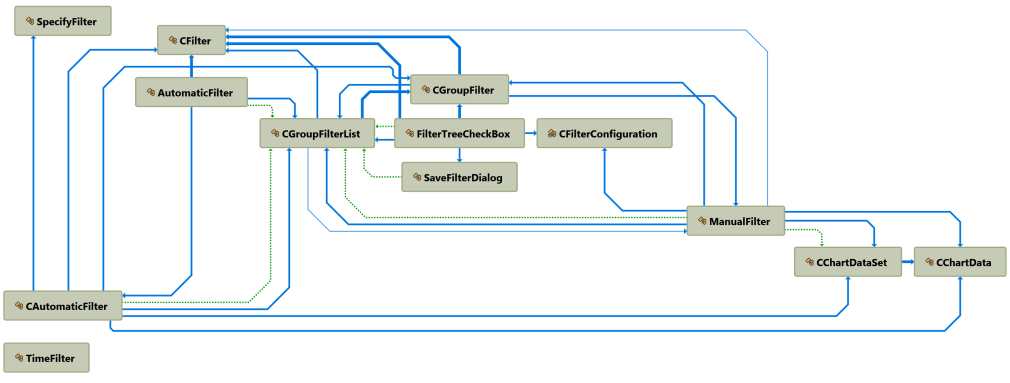
Množina taveb, jež byla do té doby využívána, bude vyfiltrována, aby v ní zůstaly pouze tavby, jejichž datum vzniku bylo mezi těmito dvěma daty.

2.2.2.5 Úprava hodnot tavby

U některých taveb bude třeba některé hodnoty přizpůsobit, aby příliš nevybočovaly svou odchylkou od ostatních hodnot. Je to pouze v případech, kdy se uživatel tak rozhodne. Tato hodnota pak bude uložena i do databáze pro příslušný filtr a dále se bude pracovat s upravenou hodnotou tavby.

2.2.2.6 Shrnutí

Celkový filtr bude velice důležitá komponenta celého softwaru. Na následujícím grafu 2.7 je vidět závislost všech softwarových komponent v části filtr.



Obrázek 2.7: Graf závislostí ve filtru

2.2.3 Optimalizace

Na základě funkčních požadavků je známo, že bude třeba použít více optimalizačních algoritmů. Použije se zde tedy opět návrhový vzor Strategy (více 2.3). Pro nynější verzi softwaru budou naimplementovány dva optimalizační algoritmy a algoritmus „Brute Force“.

- Genetický algoritmus
- Simulované žíhání
- Brute Force

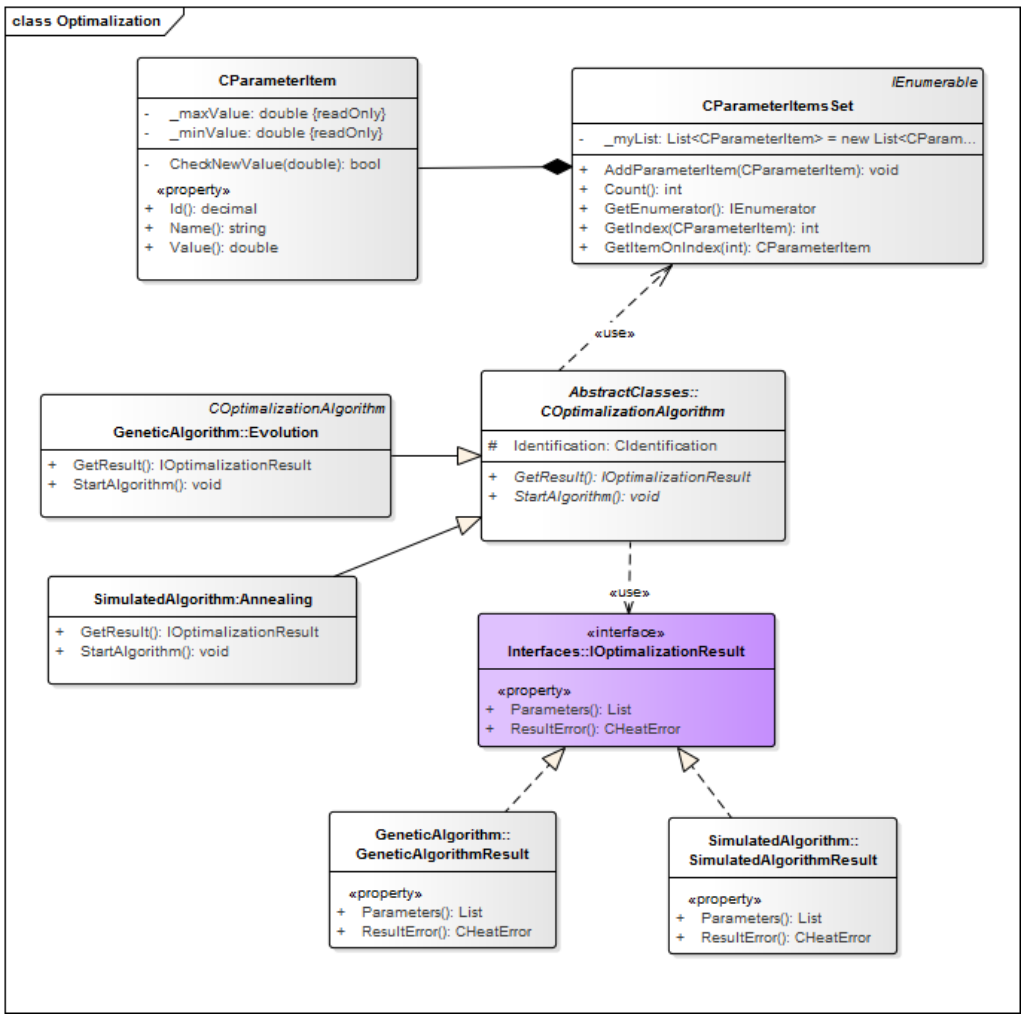
Všechny tyto algortimy budou podléhat určitému rozhraní, aby si zbytek programu nemusel pamatovat, se kterým algoritmem pracuje. Tento interface, nebo spíše abstraktní třída, aby se zde daly implementovat ostatní přípravné metody, bude mít dvě abstraktní metody.

- StartAlgorithm - metoda, která bude spouštět výpočet algoritmu
- GetResult - metoda, jež bude vracet výsledek algoritmu.

Tato abstraktní třída bude nazvána COptimalizationAlgorithm. Na metodu, StartAlgorithm bude využívat jednotlivých algoritmů a následně bude přes interface IOptimalizationResult, vracet výsledky daného algoritmu.
Optimalizační algoritmus bude uset pracovat s paramtetry, jenž budou načteny a zobrazeny na hlavní obrazovce aplikace. Parametry modelu, již jsou v systému reprezentovány třídou `M_PARAMETER`. Nicméně tato třída, neobsahuje hodnoty, kterých může algoritmus nabývat. Proto bude třeba vytvořit třídu se kterou bude schopen algoritmus pracovat. Proto bude vytvořena třída CParameterItem, jenž bude reprezentovat parameter a budou zde uloženy hlavní atributy třídy parametr, a taktéž zde budou uloženy ony rozsahy hodnot. Další problém, který by se vyskytl v případě použití v algoritmech třídy `M_PARAMETER`, by byla úprava hodnty, parametru. Třída CParameterItem bude sdružována v composite třídě CParameterItemSet.
Celý diagram návrhu Optimalizace je na obrázku 2.8.

2.2.3.1 Genetický algoritmus

Genetické algoritmy jsou typem evolučních algoritmů, algoritmů, jež jsou inspirovány evolučními procesy v přírodě. Jedná se o iterativní algoritmy, které v každé iteraci zlepšují množinu řešení (populaci) pomocí operací selekce, křížení nebo mutace. Výhodou těchto algoritmů je jejich nezávislost na typu problému. Jedinou jejich znalostí je výpočet fitness funkce, která



Obrázek 2.8: Optimalizace

reprezentuje kvalitu řešení (potomka).
Pro řešení jsem zvolil Generational metodu genetického algoritmu, tedy metodu, které v každé iteraci generuje novou populaci. Její princip naznačuje následující pseudokód a popis fází algoritmu.

Pseudokód :

- Vygeneruj výchozí náhodnou populaci P(0).
- N-krát opakuj

  - Vytvoř novou prázdnou populaci P(N).
    * Selekce - vyber rodiče z populace P(N-1)
    * Křížení - proveď s rodiči operaci křížení
    * Mutace - proveď s potomky operaci mutace
    * Přidej potomky do nové populace
  - Nahraď starou populaci novou populací P(N-1)=P(N)
- Potomek z populace P(N) s nejlepší kondicí je řešením

Fáze algoritmu

Fitness :
Fitness, neboli kondice genotypu, určuje kvalitu řešení. Např. v případě problému batohu je to cena věcí v batohu s podmínkou překročení váhy. V případě překročení váhy je kondice nejhorší možná. V případě problému tohoto matematického modelu bude fitness funkce složitější. V případě výpočtu fitness funkce se bude muset spustit simulace modelu a sbírat data do tříd CDataHeat. Výslednou fitness funkci bude tedy reprezentovat třída CHeatError. Nicméně tato třída bude mít v sobě teplotní chybu a seznam chyb chemické analýzy. Pro potřeby algoritmu bude tedy třeba tuto třídu umět vyjádřit jedním číslem, které bude reprezentovat celkovou chybu dat tavby.
Další problém se vyskytuje, že např. u problému batohu chceme, aby byla cena batohu, a tedy fitness funkce, co nejvyšší, nicméně u chyby tavby chceme, aby tato chyba byla co nejmenší. Tudíž ftiness funkce se bude rovnat převrácené hodnotě celkové chyby tavby. Tímto převrácením dosáhneme efektu, kde velká chyba bude udávat malou fitness funkci a malá chyba naopak velkou.

$$fitness = \frac{1}{Error}$$

Více o počítání fitness funkce v sekci Vyhodnocení chyb (2.2.4)

Selekce :
Pro selekci jsem implementoval dvě strategie

- strategii rulety
  - všem jedincům je podle jejich fitness funkce dána hodnota, která by se dala definovat jako rozmezí na ruletě. Čím rozmezí větší, tím větší šance úspěchu.

  - náhodně se vybrou 2 čísla od 0 do 1. Tato čísla reprezentují kuličku rulety, která spadne do nějaké rozhraní. Vybraný jedinec se pak podílí na vytváření nové generace

- universální stochastickou strategii
  - všechny jedince rozdělím stejně jako u strategie rulety.
  - náhodně vyberu jedno číslo a k tomuto číslo postupně přičítám konstantu.

Křížení :
Implementoval jsem jednobodové křížení.

- Náhodně je zvolena bariéra
- Bere se vše z jednoho genomu před bariérou a vše z druhého genomu za bariérou.

Dále jsem zde implementoval 2 způsoby kontroly populace

- En Bloc, kde je vyměněna celá generace jejich potomky
- Elitismus, kde se bere `10%` nejlepších jedinců z generace předchozí a kopírují se namísto `10%` nejhorších jedinců, ze staré generace.

Mutace :
Každý gen genomu zmutuje s pravděpodobností, jež lze parametricky určit.

2.2.3.2 Simulované žíhání

„Simulované žíhání (simulované ochlazování, anglicky Simulated annealing, SA) je pravděpodobnostní optimalizační metoda prohledávání stavového prostoru založená na simulaci žíhání oceli. Při prohledávání stavového prostoru se může snadno stát, že algoritmus uvázne v lokálním minimu. V metodě se tomu snažíme zabránit tím, že provádíme i změny k horšímu, velké hlavně zpočátku, a díky tomu se můžeme dostat z lokálního minima. Velikost změny záleží na teplotě. Čím větší je teplota, tím větší se provádí změny. Algoritmus pracuje pouze s jedním kandidátním řešením. Obyčejný gradientní algoritmus přijímá nové řešení pouze, pokud je lepší než řešení stávající. Při simulovaném žíhání jsou s určitou pravděpodobností přijímána i řešení horší. Pravděpodobnost přijetí i horšího řešení je přímo závislá na teplotě a nepřímo na velikosti zhoršení. V průběhu výpočtu algoritmu je teplota postupně snižována na základě rychlosti konvergence (přibližování

se cíli). Pokud algoritmus konverguje rychle (hodnocení stavů rychle klesá), snižuje se teplota také rychle a algoritmu je tak bráněno pokračovat do hůře hodnoceného stavu. Konverguje-li algoritmus pomalu (hodnocení stavů moc neklesá), zpomalí se snižování teploty, aby se případně podařilo vyprostit z lokálního minima.“ [7]

Pseudokód:

- Vygeneruj náhodný počáteční stav
- Opakuj dokud není teplota menší než finální teplota
  - Opakuj pro počet vnitřních stavů
    - Získej nový stav :
      - Vygeneruj náhodný stav
      - $\delta = Cena_{nový_stav} - Cena_{starý_stav}$
      - if $\delta < 0$ vrať nový stav
      - jinak vygeneruj náhodné číslo $Rand$
      - if $Rand > exp(\frac{\delta}{Teplota})$ vrať nový stav
      - jinak vrať starý stav
    - V případě, že nový stav má lepší „cenu“ než do té doby nejlepší nalezený ulož jako nejlepší nalezený
    - ochlaď teplotu

Nový stav je přijat vždy, když je lepší než předchozí. Pokud není, je přijat s pravděpodobností $exp(\frac{\delta}{Teplota})$. Tento vzorec říká, že je přijat, pokud je teplota vysoká nebo rozdíl v ceně malý. Pravděpodobnost přijetí horšího stavu klesá s teplotou.

Cena (Cost): Cena (Cost) je fitness funkce algoritmu, která ohodnocuje kvalitu stavů. Cena stavu, je vypočítána jako chyba matematického modelu (více 2.2.4).

2.2.4 Vyhodnocení chyb

Vyhodnocení chyby je jedním z hlavním stavebních kamenů optimalizačních algoritmů. Vyhodnocení chyb modelu se bude používat u všech algoritmů jako „Fitness funkce“ (více 2.2.3.1).

Chybu jedné tavby nazveme Γ_H (H - jako Heat - anglicky tavba) a zadefinujeme ji jako

$$\Gamma_H = \frac{1}{N_T} * \varepsilon^T + \frac{1}{N_A}\varepsilon^A * Q_A$$

Chyba jedné tavby je tedy součet teplotní chyby ε^T a chyby chemické analýzy ε^A, která je vynásobena koeficientem Q_A, zadaným uživatelem. N_A a N_T jsou počty provedených měření.

Teplotní chyba

$$\varepsilon^T = \sum_{i=1}^{N} \epsilon_i$$

- Absolutní hodnoda - $\epsilon_i = |T_{measured,i} - T_{model,i}|$
- Kvadratické kritérium - $\epsilon_i = (T_{measured,i} - T_{model,i})^2$

Chyba chemické analýzy

$$\varepsilon^A = \sum_{i=1}^{N} \epsilon_i$$

$$\epsilon_i = \sum_{i}^{N} \delta_i$$

- Absolutní hotnoda -

$$\delta_{Ch,i} = |a_{measured,i} - a_{model,i}| * q_{Ch}$$

- Kvadratické kritérium -

$$\delta_{Ch,i} = (a_{measured,i} - a_{model,i})^2 * q_{Ch}$$

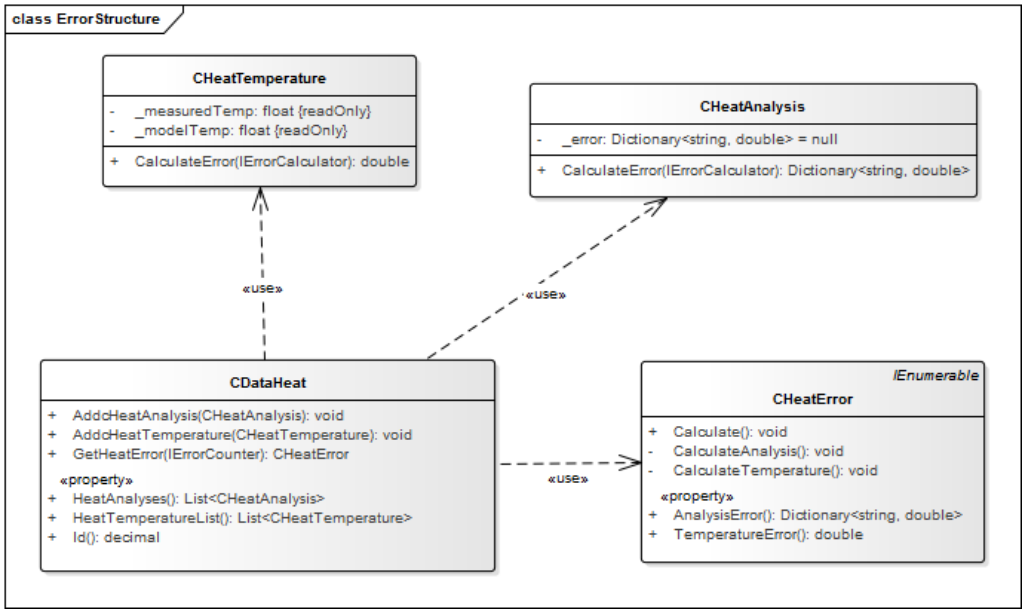
$$Ch \in \{C, Mn, Si, ...\}$$

q_{Ch} je koeficient jednotlivého chemického prvku.

Celková chyba Γ bude součet Γ_H podělená počtem taveb.

$$\Gamma = \sum_{i=1}^{H} \frac{\Gamma_H}{N_H}$$

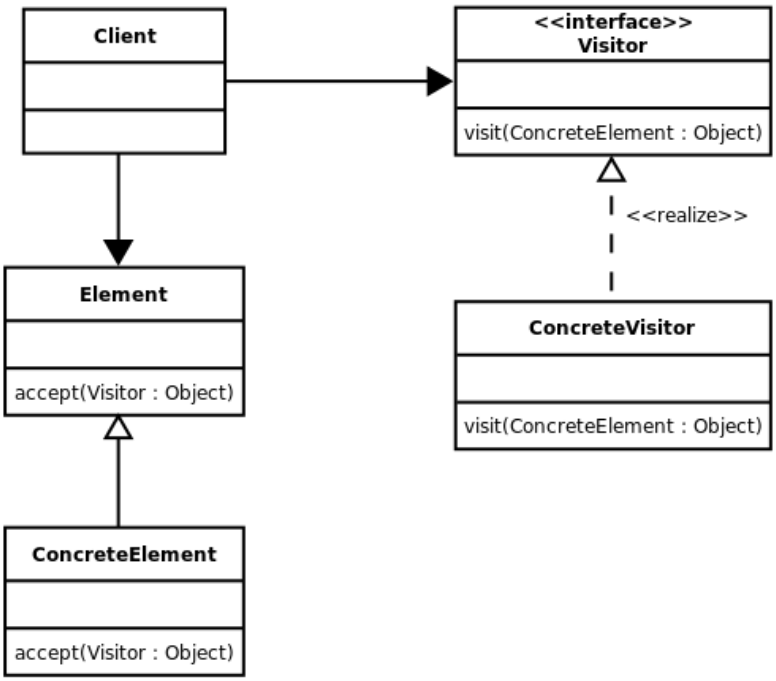
Pro vyhodnocení chyby je třeba si nejdříve vytvořit strukturu, ve které bude daná chyba uložená. Z požadavků je dáno, že zjišťujeme 2 chyby, teplotní a chemickou analýzu. Z toho tedy vyplývá, že bude třeba dvou tříd, ve kterých budou uložena data pro jednotlivé tavby. Tyto třídy nazveme CHeatTemperature a CHeatAnalysis. Obě tyto třídy budou mít také metodu na vypočítání chyb z uložených dat. Z důvodu více možných způsobů výpočtů chyby a vzhledem k tomu, že v budoucnu by se výpočet chyb mohl rozrůst na více metod, nicméně není pravděpodobné, že by se počítaly chyby z více údajů, bude na výpočet chyby použit návrhový vzor Visitor. (více 2.2.4). Chyba se bude počítat pro určitou skupinu parametrů, které se budou optimalizovat. Třída pro tuto skupinu byla již zmíněna v sekci Optimalizace (2.2.3). Jednotlivá tavba bude uložena ve třídě CDataHeat, do které se budou postupně přidávat instance tříd CHeatTemperature a CHeatAnalysis. Třída CDataHeat bude ještě mít ukazatel na třídu CHeatError, kde se bude uchovávat a přes kterou se bude vypočítávat celková chyba tavby.



Obrázek 2.9: Vyhodnocení chyby

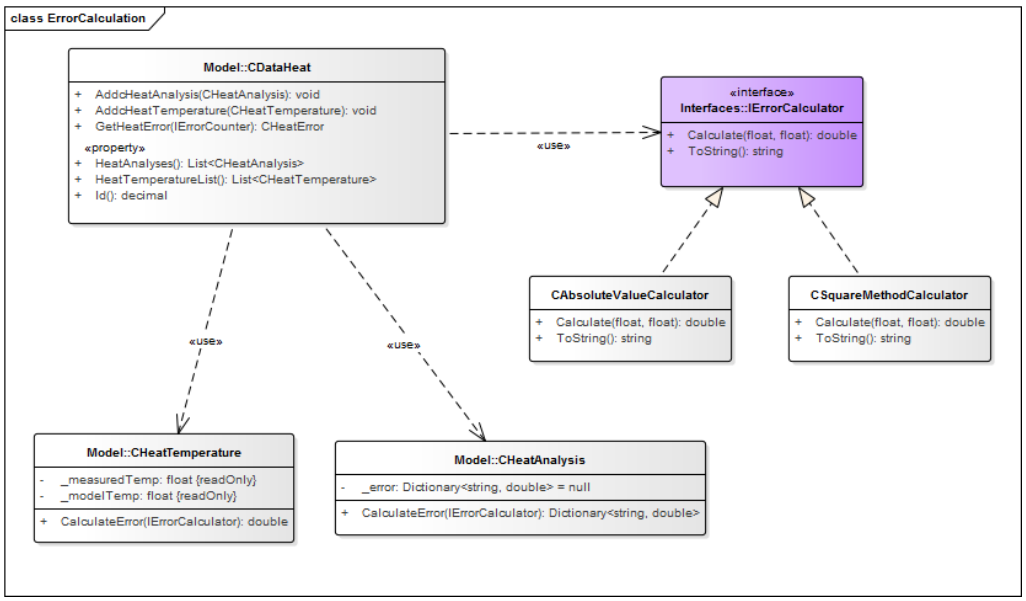
Visitor „Návrhový vzor Visitor umožňuje rozšiřovat možnosti objektu bez nutnosti modifikace jeho třídy. Snaží se o podobný cíl jako aspektově orientované programování. Visitor patří mezi návrhové vzory, které ovlivňují chování tříd a jejich instancí, tzv. behavioral patterns. Návrhový vzor Visi-

tor lze využít v situaci, kdy navrhujeme množinu tříd, do které již nebudeme žádnou třídu přidávat, ale je pravděpodobné, že budeme potřebovat přidat nějakou funkcionalitu. Připravíme se tedy na situaci, kdy jsme nuceni do všech tříd naší konečné množiny přidat další metodu. Principem je, že pro každou novou akci, kterou chceme dodat původní množině tříd, vytvoříme novou třídu. Tato nová třída představuje „návštěvníka“. Instanci tohoto návštěvníka pak předáme původní třídě a ta v podstatě sama na sebe zavolá odpovídající metodu návštěvníka. Původní třída tedy představuje „navštíveného“. Jinými slovy: návštěvník umí vykonat novou akci, navštívený ho přijme a nechá ho se sebou vykonat onu novou akci. Takže platí, že kolik bude dodatečných akcí, tolik bude návštěvnických tříd.“ [6]



Obrázek 2.10: Návrhový vzor Visitor

Na obrázku 2.10 je vidět návrh pro návrhový vzor Visitor. Na obrázku 2.11, je vidět návrh aplikace návrhového vzoru Visitor pro výpočet chyby.



Obrázek 2.11: Výpočet chyby - Visitor

2.2.4.1 Kalkulace

Mezi kalkulací a čítačem je jeden velký rozdíl. Kalkulace vypočítává jednu hodnotu. Jednu odchylku (chybu) z naměřených a odsimulovaných hodnot. Jak již je popsáno výše, kalkulace je řešena návrhovým vzorem visitor. Dá se taky napsat, že kalkulace využívá návrhový vzor Strategy (více 2.3). Celkový návrh kalkulace je na obrázku 2.11. Podle požadavků budou zatím navrženy pouze dvě možnosti kalkulace.

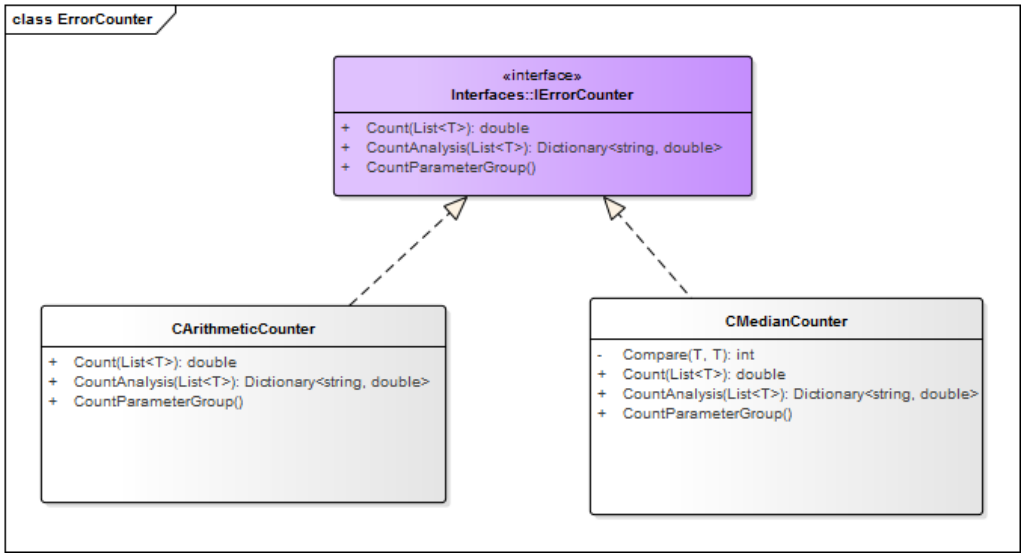
- Absolutní hodnota
- Metoda nejmenších čtverců (více 2.2.4.1)

Metoda nejmenších čtverců „Metoda nejmenších čtverců „Metoda nejmenších čtverců je matematicko-statistická metoda pro aproximaci řešení přeurčených soustav rovnic (tj. soustav, kde je více rovnic než neznámých). "Nejmenší čtverce"znamenají, že výsledné řešení má minimalizovat součet čtverců odchylek vůči každé rovnici. Metoda je v základní podobě určená pro řešení nekompatibilních soustav lineárních rovnic (v obecnější podobě hovoříme o nelineární metodě nejmenších čtverců), díky čemuž je fakticky ekvivalentní tzv. lineární regresi.“ [8]

2.2.4.2 Čítač

Čítač pracuje s výsledky kalkulace. Kalkulace počítá s jednou hodnotou. Takovýchto hodnot má jedna tavba několik a je třeba z nich udělat výslednou hodnotu pro tavbu, popřípadě u více taveb pro celou skupinu. Čítač bude taktéž jako kalkulace tvořen návrhovým vzorem Visitor, jelikož se dá předpokládat, že budou přicházet další možnosti jak vypočítávat výslednou hodnotu. Čítač bude taktéž tvořen návrhovým vzorem Strategy. Celkový návrh čítače je na obrázku 2.12. Podle požadavků budou zatím navrženy pouze dvě možnosti čítače.

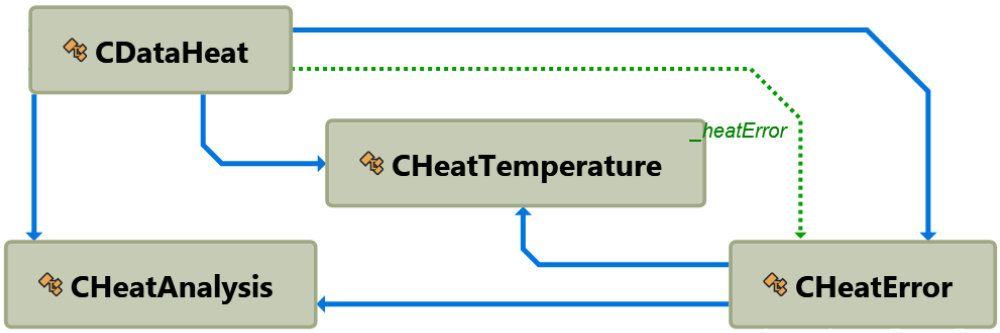
- Aritmetický průměr
- Medián



Obrázek 2.12: Čítač

2.2.4.3 Shrnutí

Na následujícím grafu 2.13, je vidět závislost všech softwarových komponent v této části.



Obrázek 2.13: Graf závislostí ve vyhodnocení chyb

2.3 Databáze

Celý systém je již na databázi napojen a má i své rozsáhlé databázové schéma. Já se budu zabývat pouze návrhem nových částí databáze, která bude vytvořena specificky pro tento software. Pro práci s databází bude použit framework od společnosti Microsoft s názvem EntityFramework (více A). Tento framework slouží k namapování databázových tabulek na objekty a naopak. Pomůže taktéž vytvořit databázovou strukturu na základě tříd (Entit) nebo vytvoří strukturu tříd (Entit) na základě databázového schématu. V našem případě použijeme druhý postup. Bude se využívat ještě jeden přístup k databázi, a to klasický přístup pro framework .Net. tzv. ADO.Net. Tímto přístupem se budou získávat data z tabulek, které nejsou vytvořeny pomocí EntityFramework, přřípadně když bude SQL dotaz vyžadovat složité klauzule JOIN, popřípadě složitější dotazovací podmínky. Pro účely tohoto softwaru bude databázový návrh rozdělen na tři části.

- FilterModel - databázový model pro část filter
- HeatModel - databázový model pro část ukládání taveb
- ParameterModel - databázový model pro část načítání parametrů

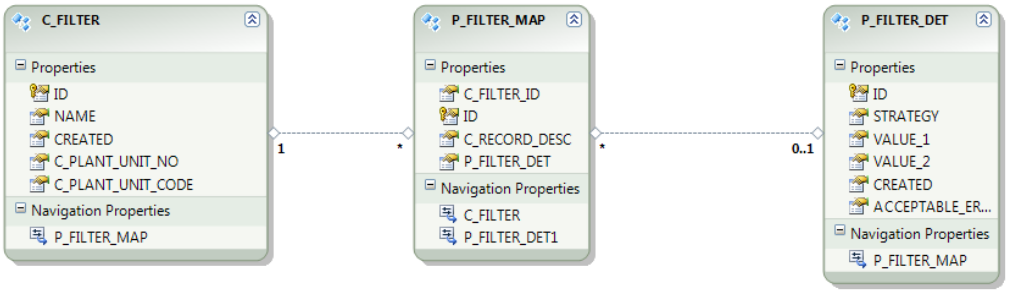
2.3.1 FilterModel

Filtry bude třeba ukládat a načítat z databáze. Jeden filtr se vždy bude skládat z několika jednotlivých filtrů. O těchto jednotlivých filtrech by se dalo říct, že se jedná o sloupce za tabulky `P_RECORD_DET`. Ze sekce Výroba

filtru (2.2.2.1) je zřejmé, že sloupce v tabulce `P_RECORD_DET` jsou popsány v tabulce `C_RECORD_DESC`, tudíž v rámci jednotlivého filtru je třeba si pro jeho zpětné vytvoření pamatovat ID záznamu v tabulce `C_RECORD_DESC`. U filtrů, které se budou používat automaticky, bude třeba ukládat rozsah hodnot a strategii, která je na automatické filtrování určená, taktéž bude třeba si ukládat, kolik procent hodnot může být „nevalidních“. Nakonec je třeba si celou skupinu filtrů uložit s tím na kterém zařízení a které jednotce filtr pracuje. Celkově to tedy dělá návrh tří tabulek.

- `C_FILTER` - zde se bude ukládat skupina filtrů. Každá skupina filtrů bude mít id, jméno, zařízení, na kterém je použita i jednotka tohoto zařízení. Každý záznam v tabulce `C_FILTER` bude mít několik záznamů v tabulce `P_FILTER_MAP`.
- `P_FILTER_MAP` - zde se bude ukládat jednotlivý filtr. Bude zde id skupiny filtrů, kam jednotlivý filtr patří. Bude zde odkaz na záznam v tabulce `C_RECORD_DESC`, v případě, že se bude jednat o automatický filtr, bude zde odkaz na tabulku `P_FILTER_DET`.
- `P_FILTER_DET` - tato tabulka bude sloužit na ukládání automatických filtrů, jak již bylo popsáno výše.

Celý databázový návrh FilterModelu je vidět na obrázku 2.14.



Obrázek 2.14: Databázový návrh filtru

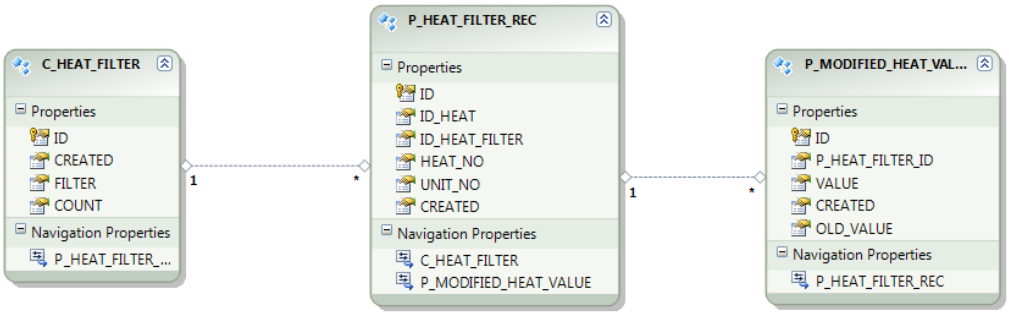
2.3.2 HeatModel

HeatModel se bude zabývat ukládáním a načítáním taveb. Může se jednat o odfiltrované tavby, které chceme použít pro optimalizaci modelu. Všechny tavby se v systému ukládají v tabulce `S_HEAT`. Z tohoto důvodu nebude

třeba ukládat celou tavbu, ale pouze ukazatel na tuto tabulku. Bude třeba taktéž ukládat upravené hodnoty taveb. Stejně jako v případě filtrů se i tavby sdružují do skupin, které se ukládají. Máme tedy tři tabulky, do kterých se bude ukládat.

- `C_HEAT_FILTER` - zde se bude ukládat skupina TAVEB. Každá skupina TAVEB bude mít id, jméno filtru, který byl na ni použit, a počet taveb, jež skupina obsahuje. Každý záznam v tabulce `C_HEAT_FILTER` bude mít několik záznamů v tabulce `P_HEAT_FILTER_REC`.
- `P_HEAT_FILTER_REC` - zde se bude ukládat jednotlivá tavba. Bude zde id skupiny taveb, kam jednotlivá tavba patří. Bude zde odkaz na záznam v tabulce `S_HEAT`, v případě, že se bude jednat o tavbu, ve které byla upravena hodnota nebo hodnoty. Bude zde odkaz na tabulku `P_MODIFIED_HEAT_VALUE`.
- `P_MODIFIED_HEAT_VALUE` - tato tabulka bude sloužit na úkládání upravených hodnot taveb, jak již bylo popsáno v sekci Úprava hodnot tavby (2.2.2.5)

Celý databázový návrh HeatModelu je vidět na obrázku 2.14.

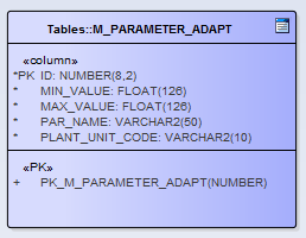


Obrázek 2.15: Databázový návrh filtru

2.3.3 ParameterModel

Parametry se nebudou ukládat do žádného nastavení, jelikož už budou předem vybrány operátory. Všechny parametry matematického modelu jsou nyní uloženy v tabulce `M_PARAMETER`, nicméně ne všechny parametry se budou optimalizovat. Optimalizovat se budou pouze parametry uložené v tabulce `M_PARAMETER_ADAPT`. V případě, že hodnoty parametrů budou upraveny tak, aby chyba modelu byla minimální a uživatel si bude přát toto

nastavení uložit, parametry se uloží do tabulky `M_PARAMETER`. V tabulce `M_PARAMETER_ADAPT` bude uložen vždy název parametru a hodnoty, kterých může nabývat.



Obrázek 2.16: Databázový návrh `M_PARAMETER_ADAPT`

2.4 IDE

Jako vývojové prostředí nebo IDE je zvoleno Visual Studio 2010. Pro práci s databázemi se bude používat Oracle SQL Developer. Ve Visul Studiu se bude používat ještě nástroj ReSharper od společnosti JetBrains, jenž umožňuje několik vylepšení.

Kapitola 3

Funkce a Implementace

Na následujících stránkách je popsána funkcionalita a implementace vyvýjeného software. Postupně budou procházeny jednotlivé komponenty softwaru a bude popisována jejich funkcionalita a jak byla tato funkcionalita implementována.

3.1 Syntaxe

V kódu software je použita jednotná sytnaxe, která je standartem pro psaní kódu v jazyku C#. Při udržování této syntaxe byl velmi nápomocný nástroj ReSharper, který mimo jiné kontroluje i tuto syntaxi.

- Vizuální komponenty / dialogy - u vytvořených dialogů nebo vizuálních komponent je využita synatexe „CamelCase“
- Třídy - třídy vždy začínají velkým písmenem C a následuje název třídy - CClass
- Metody - metody v jazyce C# vždy začínají velkým písmenem - Metoda()
- Property - jazyk C# umožňuje něco, co se nazývá Property třídy. Jsou to atributy třídy, které mají veřejný „getter“ nebo „setter“. Property vždy začínájí s velkým písmenem - Property
- atributy - soukromé („privátní“) atributy třídy se označují malým písmenem a s „podtržítkem“ na začátku slova. - `_atribut`.
- proměnná lokální proměnná se v bloku kódu označuje velkým písmenem s malým „l“ na začátku - lVariable

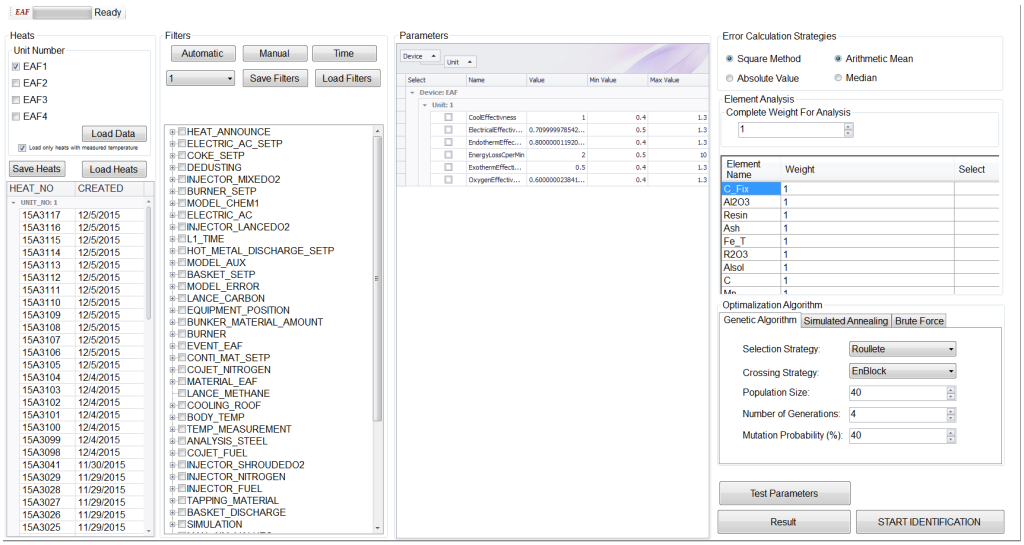
3.2 Obecně

Při implantování softwaru se přišlo na několik nedostatků návrhu a byl někdy částečně upraven. Taktéž se při implantaci objevilo mnoho dalších problémů, kde některé byly vyřešeny a jiné ne.
Software obecně vychází z vizuální komponenty Identifikace. Tato komponenta je vizuální třída, jež sdružuje celý program. V této vizuální komponentě jsou vloženy další vizuální komponenty, jež zaštiťují jednotlivé funkcionality softwaru. Tato komponenta je naimplementována jako abstraktní třída. To je z důvodu, že je třeba dělat tuto adaptaci parametrů pro více typů pecí. (EAF, LF a VD). Tyto vizuální komponenty budou mít vždy uživatelské rozhraní stejné, nicméně jejich funkčnost bude trošku jiná. Největší rozdíl bude v přístupu k databázi v tabulkách jednotlivých pecí.
Samotná Identifikace modelu bude probíhat v abstraktní třídě CIdentification, tato třída bude mít konkrétní implementace opět pro každou jednotlivou pec. Toto řešení je nutné vzhledem k načítání dat z databáze, kde každá pec má jiné identifikační prvky. Ty zde budou konkrétně řešeny v jednotlivých třídách. Simulace modelu probíhá ve firemním frameworku, který se spouští z této abstraktní třídy, a data, která se zachytávají z modelu, se taktéž ukládají ve třídě CIdentification. Třídu CIdentification budou tedy využívat optimalizační algoritmy v okamžiku, kdy chtějí získat „fitness funkci“ (více 2.2.3.1). Třída CIdentification taktéž ukládá celou strukturu na vyhodnocení chyb.
Je tedy vytvořen „namespace“ „AbstractClasses“, který obsahuje abstraktní třídy celého softwaru. Pro každou pec je vytvořen speciální namespace, kde se nacházejí konkrétní implementace těchto tříd.

3.3 Filtr

Část Filtr byla naimplementována do namespace „Filter“. U filtru se podařilo dodržet jeho návrh, a tudíž struktura tříd odpovídá diagramům uvedených v návrhu. Hlavní třídou je zde vizuální komponenta „FilterTreeViewCheckBox“, která je vložena do hlavní vizuální komponenty programu. Komponenta je tvořena čtyřmi hlavními vizuálními prvky.
Komponenta je tvořena čtyřmi hlavními vizuálními prvky.

- Ukládací tlačítko - po stisknutí tohoto tlačítka se vytváří instance třídy SaveFilterDialog, který uloží selekci vybraných filtrů do databáze. O implementaci ukládní více v sekci Databáze(3.6)



Obrázek 3.1: Hlavní obrazovka

- Načítací tlačítko - po stisknutí tohoto tlačítka se objeví dialog LoadConfiguration, který je více popsán v sekci Ukládání a načítání konfigurací (3.6.2). Díky tomuto dialogu se načte do stromové struktury filtrů konfigurace filtru, jenž byla načtena z databáze.

- Rozevírací seznam - rozevírací seznam určuje, ze které části(UnitNumber), se budou načítat filtry. Filtry jsou různé pro každou pec i pro každou část pece. Po určení části pece se od stromového seznamu načte seznam filtrů pro danou konfiguraci.

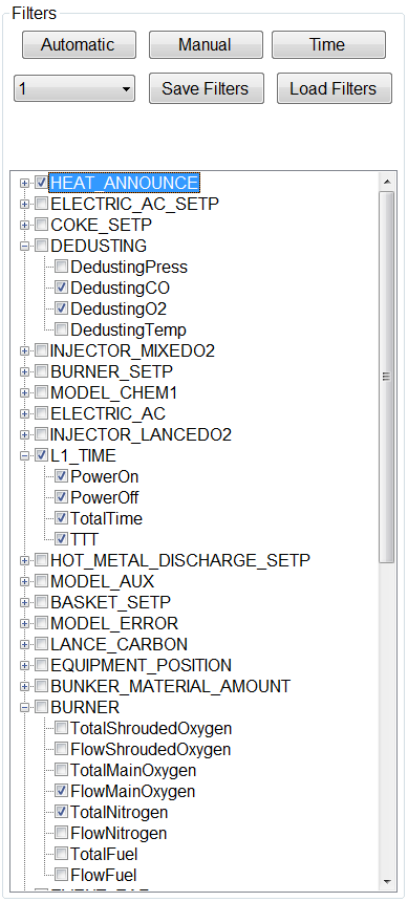
- Stromový seznam - zobrazuje seznam filtrů. Každý filtr patří do skupiny nějakých filtrů. Filtry lze vybrat jednotlivě i celou skupinu.

„FilterTreeViewCheckBox“ vrací ostatním třídám, zejména jednotlivým filtrům seznamy filtrů, jenž byly vybrány pro filtrování.

V namespacu „Filter“ se nacházejí ještě tyto třídy.

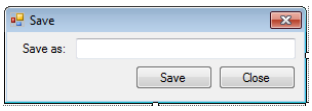
- **CFilter** - Třída, jež reprezentuje samostatný jeden filtr. Obsahuje všechny atributy jednotlivého filtru (id, name, groupName, unitNo...).

- **CFilterConfiguration** - Vizuální třída, která je odvozona od abstraktní třídy LoadConfigurations. CFilterConfigurations zařizuje ukládání a načítání filtrů z databáze. (více o LoadConfigurations zde 3.6.2)



Obrázek 3.2: Filter

- **CGroupFilter** - Třída, jež seskupuje filtry reprezentované třídou CFilter, do skupin podle property GroupName. Důležité metody:
  - CreateFilters - metoda, jež vytvoří skupinu filtrů se stejným GroupName.
  - CreateGroupBox - metoda, jež vytvoří pro každou skupinu vizuální komponentu GroupBox a do ní vloží tlačítka pro uživatelem vybrané filtry. Těmto tlačítkům rovněž přidá EventHandler s danou funkcionalitou.
- **CGroupFilterList** - Iterátor, přes třídu CGroupFilter
- **SaveFilterDialog** - Dialogové okno při ukládání filtru do databáze. Dotazuje se, na jméno jak má být filtr uložen.



Obrázek 3.3: Save Filter Dialog

3.3.1 Automatický filtr

3.3.1.1 Funkce

Automatický filtr filtruje tavby pomocí předem nastavených parametrů. V dialogovém okně automatického filtru se nastavují jednotlivým filtrům jejich hodnoty a následně je filtr spuštěn. V následujících komponentách se nastavují následující parametry.

- Výběr filtru (rozevírací seznam) - zde se nastavuje hodnota pro určitý filtr. Pro správný běh automatického filtru je třeba, aby pro každý filtr byly nastaveny hodnoty, podle kterých by se mělo filtrovat.

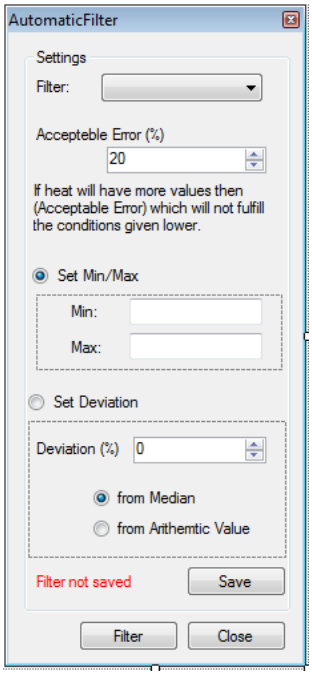
- Akceptovaná chyba (číselný výběr) - zde se procentuálně nastavuje, kolik procent chyb, bude u tavby akceptovatelný. V případě nastavení hodnoty 0, nebude akceptovatelná žádná chyba u tavby a tavba bude, v případě chyby, odfiltrovnána. V případě nastavení nějaké hodnoty vyší než 0 např. n. (n $>$ 0). Bude tavba, která má méně než n% hodnot, a tedy nevyhovuje nastaveným podmínkám, stále brána jako „validní“ tavba. Tyto hodnoty budou následně upraveny na průměrnou hodnotu tavby.

- Výběr strategie - pro určení mezních hodnot, podle kterých se bude filtrovat.

  - Minimum maximum - do políček se nastaví minimální a maximální hodnota tavby
  - Směrodatná odchylka - do políčka pro odchylku se nastaví procentuální velikost odchylky od určitého čísla. Následně se vybere, od kterého čísla se bude odchylka počítat.
    - Medián
    - Aritmetický průměr

- Ukládací tlačítko - Po nastavení každého filtru je třeba filtr uložit. Kliknutí na toto tlačítko se filtr uloží do paměti.

- Tlačítko filter - stisknutím tohoto tlačítka se spustí samotný filtr. Filtr začne procházet jednotlivé tavby a určovat, jestli jejich hodnoty jsou v nastavených limitech. Výsledkem je seznam taveb, které vyhovují stanoveným limitám.
- Tlačítko Cancel - ukončí celý automatický filtr a zavře dialog.



Obrázek 3.4: Automatic Filter

3.3.1.2 Implementace

Implementce automatického filtru je ve dvou „namespacech“

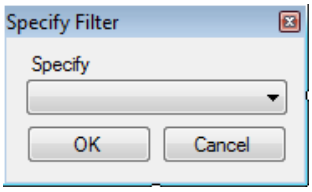
- Filter.AutomaticFilter - zde se nachází hlavní třídy automatického filtru, vizuální komponenta a pomocný dialog SpecifyFilter
  - **AutomaticFilter** - Vizuální komponenta pro nastavení automatického filtru. Do třídy se při vytváření předá seznam taveb, filtry, kteráma má probíhat filtrování a mezní hodnoty nastavené v dialogu.
  - **CAutomaticFilter** - Třída, kde probíhá samotná filtrace, pomocí dat ze třídy AutomaticFilter a vybraných strategií z Strategies.AutomaticFilter. AutomaticFilter má interface IFilter.

  - **SpecifyFilter**

- Strategies.AutomaticFilter - zde se nacházejí třídy, kde probíhá samotné vyhodnocování hodnot jestli, splňují nebo nesplňují dané kritéria. V metodě FilterValue, procházejí data určité tavby a podle nastavené strategie určují tavbu za „validní“ či nikoli.
  - **CDeviation**
  - **CMinMax**

CAutomaticFilter Třída CAutomaticFilter se stará o vlastní filtraci. Prochází všechny filtry a pro každý filtr prochází každou tavbu. V případě, že tavba v předchozím filtru byla prohlášena za „nevalidní“, do dalších filtrů se již nepřidává. V případě nejednoznačnosti filtru (více 2.2.2.1) se vytvoří jednoduchý dialog (více 3.3.1.2). Každý filtr má nastaveny mezní hodnoty, třída CAutomaticFilter využívá třídy CMinMax a CDeviation pro kontrolování hodnot tavby oproti těmto mezním hodnotám.

SpecifyFilterDialog Po vytvoření pouze zobrazí uživateli hodnoty, které si uživatel vybere z rozevíracího seznamu a potvrdí tlačítkem OK. Tato hodnota je dále zpracovávaná v automatickém filtru.



Obrázek 3.5: Specify Filter Dialog

3.3.2 Manuální filtr

3.3.2.1 Funkce

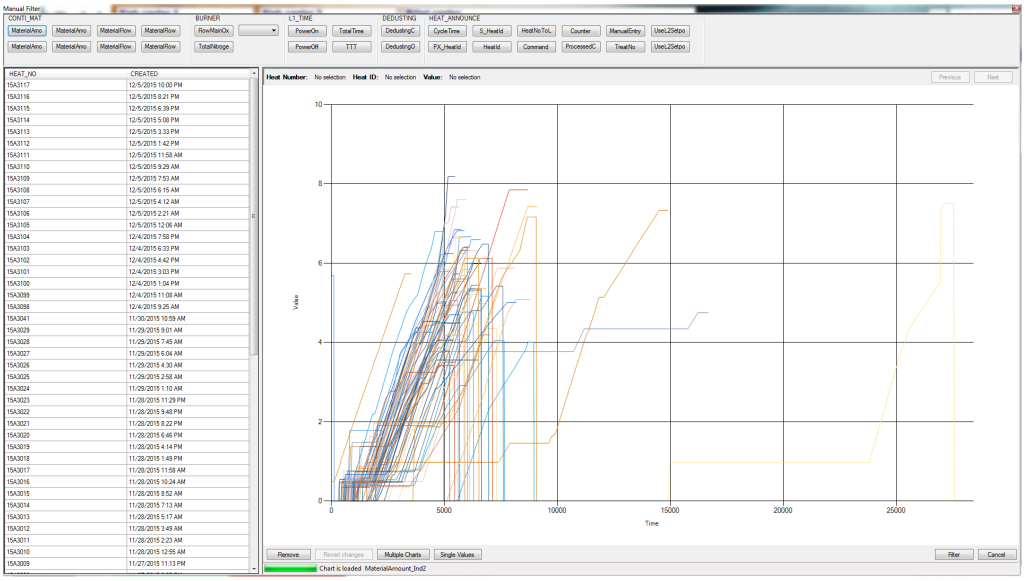
Při vytvoření manuálního filtru se automaticky vytvoří ovládací prvky pro filtry, již byly vybrané v hlavní vizuální komponentě. V horní části okna dialogu pro manuální filtr jsou zobrazeny tlačítka pro jednotlivé filtry. V levé části dialogu je seznam taveb, které se budou manuálně filtrovat. V hlavní části dialogu je místo pro grafické zobrazení průběhu taveb. Po kliknutí na určité tlačítko, které reprezentuje určitý filtr, se zobrazí graf hodnot, kde různé křivky značí různé tavby. Hodnoty grafu jsou uváděny v čase a jsou

to hodnoty, podle kterých funguje vybraný filtr. Zobrazení grafu jsou dvě možné varianty.

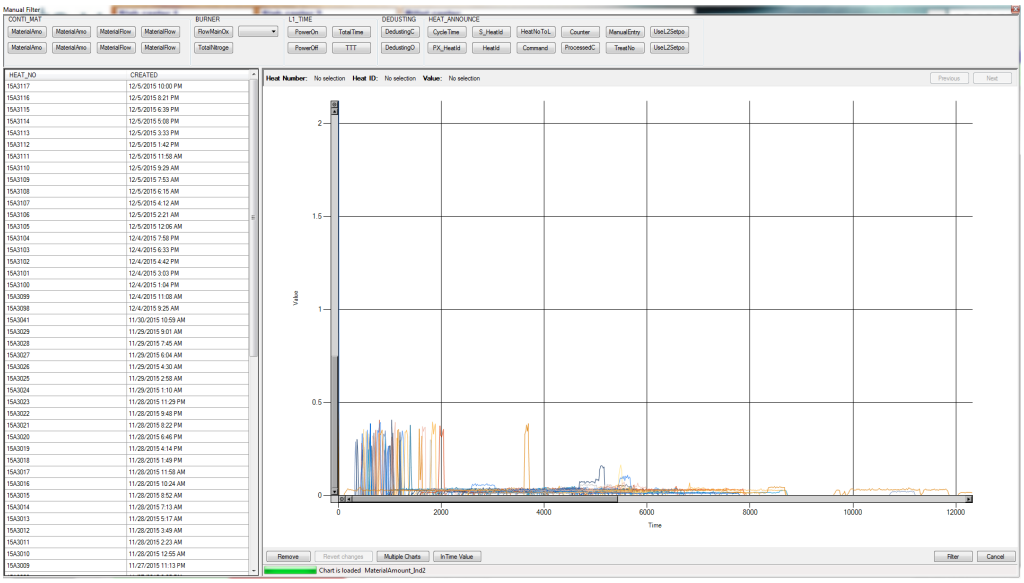
- Celková hodnota v čase - ukazuje celkovou hodnotu tavby v čase
- Jednoduchá hodnota - ukazuje pouze hodnotu, jež přibyla v v daném čase.

U grafu je také možno oddělit jednotlivé tavby, takže vidíme pouze jednu tavbu v grafu a lze mezi tavbami přepínat. Základní nastavení je takové, že v jednom grafu jsou hodnoty všech taveb.

Graf je možno přibližovat a oddalovat. V případě kliknutí na nějakou křivku grafu se v levé tabulce označí tavba, pro kterou je daná křivka platná. To samé platí i opačným způsobem. Pokud bude označená tavba v tabulce vlevo, bude zvýrazněna i křivka v grafu. Jakoukoli křivku lze z grafu odebrat, tímto odebereme i tavbu, v případě špatného odebrání je zde tlačítko „Reverse“, které tavbu vrátí zpět.
Tlačítkem „Filter“ v pravém dolním rohu ukončíme manuální filtrování a ve výběru taveb zůstanou pouze tavby, které nebyly v manuálním filtru odebrány. Tlačítkem „Cancel“ v pravém dolním rohu celou operaci zrušíme.



Obrázek 3.6: Manuální filtr - zobrazení celková hodnota



Obrázek 3.7: Manuální filtr - zobrazení jednoduchá hodnota

3.3.2.2 Implementace

Manuální filtr je tvořen třemi třídami.

- **ManualFilter** - Hlavní vizuální komponenta manuálního filtru. Využívá interface IFilter.
- **CChartData** - Obsahuje data pro zobrazovaný graf.
- **CChartDataSet** - Kontainer pro třídu CChartData.

ManualFilter Manualní filtr zobrazuje data, která jsou uložena ve třídě CChartDataSet. Tato data zpracuje a zobrazuje do grafů. Pomocí tříd CGroupFilter a CGroupFilterGroup se v horní části komponenty ManualFilter vytvoří ovládací panel, díky kterému je možno filtrovat tavby, jež jsou načteny v tabulce v levé čísti komponenty.

3.3.3 Časový filtr

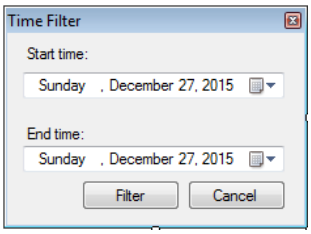
3.3.3.1 Funkce

V časovém filtru si uživatel nastaví časové rozpěti, ze kterých chce zobrazit tavby. Filtr zobrazí datum „nejmladší“ tavby a datum „nejstarší“ tavby.

Obě tato data se po té zobrazí a uživatel má možnost obě tato data změnit. Filtr poté v sekci pro tavby zobrazí pouze tavby, které splňují výběr.

3.3.3.2 Implementace

Time filtr je vizuální komponenta, tvořená dvěma komponentami TimePicker. V těch si uživatel vybere počáteční a koncové datum. Tato data jsou pak vrácena do hlavní třídy Identification, která zobrazí pouze tavby, které jsou mezi těmito dvěma daty.



Obrázek 3.8: Time Filter

3.4 Vyhodnocení chyb

3.4.1 Funkce

Vyhodnocení chyby se používá v optimalizačních algoritmech jako fitnes funkce (více 2.2.3.1). Vyhodnocení chyb spouští celou simulaci procesu tavby a sbírá jednotlivé chyby. Vyhodnocení chyb ale nepracuje pouze s jednou tavbou, ale s kolekcí mnoha taveb. V případě, že je v optimalizačním algoritmu vyžádána fitness hodnota, je spuštěno vyhodnocení chyby. Tím se spustí matematický model a celá simulace vybraných taveb. Simulátor taveb zaznamenává události, které v tavbě probíhají, a v případě zaznamenání události měření teploty nebo chemické analýzy se zjistí daná hodnota z modelu a obě hodnoty se uloží. Po ukončení všech simulací taveb se spočítá hodnota chyby modelu na základě parametrů, již jsou zadány uživatelem. Funkcionalita vyhodnocení chyby může být vyzkoušena přes funkci „Test parametrů“, který spustí matematický model s hodnotami parametrů zadaných v tabulce s parametry. A po skončení zobrazí chybu modelu na těchto parametrech.

3.4.2 Implementace

Vyhodnocení chyb je naimplementováno v mnoha namespacech.

- Heats
  - **CDataHeat** - Shromažďuje data jedné tavby. Obsahuje dva seznamy hodnot. Seznam tříd CHeatTemperature pro teplotu a seznam tříd CHeatAnalysis pro chemickou analýzu.
  - **CHeatAnalysis** - Třída, jež vypočítává a ukládá chybu chemické analýzy. Výpočet chyby se provádí v metodě CalculateError, do které jako parametr přichází příslušný kalkulátor.
  - **CHeatTemperature** - Třída, jež vypočítává a ukládá chybu teploty tavby. Výpočet chyby se provádí v metodě CalculateError, do které jako parametr přichází příslušný kalkulátor.
  - **CHeatError** - Třída, jež obsahuje chybové hodnoty analýzy a teploty pro jednu tavbu. Třída obsahuje metodu GetErrorValue, která vypočítává celkovou chybu tavby. Jako parametr se do této metody předává koeficient analýzy.
- AbstractClasses
  - **CIdentification** - Abstraktní třída, která spouští simulaci modelu a zapisuje data teploty a chemické analýzy. V hlavní metodě třídy RunIdentification se spouští pro každou tavbu samostatný model. V případě více taveb se pro každou tavbu spouští model na samostatném vláknu. Vlákno má vždy za úkol spustit model jedné tavby, v případě, že po skončení jsou ještě nějaké tavby nezpracované, se na vlákně spouští další model s jinou tavbou. Teprve, když jsou všechny tvaby skončené, program dále pokračuje.
- Model
  - **CParameterGroup** - Třída, jnež reprezentuje skupinu taveb. Property ErrorValue vrací celkovou chybu pro všechny tavby.
  - **CParameterError** - Je naimplementována jako wrapper okolo třídy CHeatError, jenž ho rozšiřuje o funkcionalitu pracovat pro více taveb.
- Strategies.ErrorCalculation
  - **CAbsoluteValueCalculator** - Třída, jež podléhá interfacu IErrorCalculator. V metodě Calculate počítá rozdíl mezi změřenou hodnotou a modelovou hodnotou.

  - **CSquereMethodCalculator** - Třída, jež podléhá interfacu IErrorCalculator. V metodě Calculate počítá rozdíl mezi změřenou hodnotou a modelovou hodnotou.
  - **CArithmeticCounter** - Třída, jež podléhá interfacu IErrorCounter. Třída obsahuje metody pro součet chyb teploty a analýzy.
  - **CMedianCounter** - Třída, jež podléhá interfacu IErrorCounter. Třída obsahuje metody pro součet chyb teploty a analýzy.

- Util
  - **HeatPlayer** - Jednoduchá třída, jež simuluje pomocí firemního frameworku tavbu a zachycuje události.

3.5 Optimalizace

3.5.1 Funkce

Optimalizace by měla optimalizovat parametry matematického modelu. K tomu používá dva algoritmy

- Genetický algoritmus
- Simulované ochlazování

V rámci optimalizace také lze vidět vliv hodnoty jednotlivých parametrů na chybu modelu. Tato funkcionalita je více popsána v podsekci „Brute Force“ (zde 3.5.1.3).

Oba optimalizační algoritmy po zadání vstupních parametrů těchto algoritmů a spuštění celé identifikace se začnou podle určitých pravidel testovat jednotlivé hodnoty vybraných parametrů ve snaze najít co nejmenší chybu matematického modelu. Po ukončení obou algoritmů se zobrazí ta konfigurace, která má nejmenší chybu parametrů v modelu.

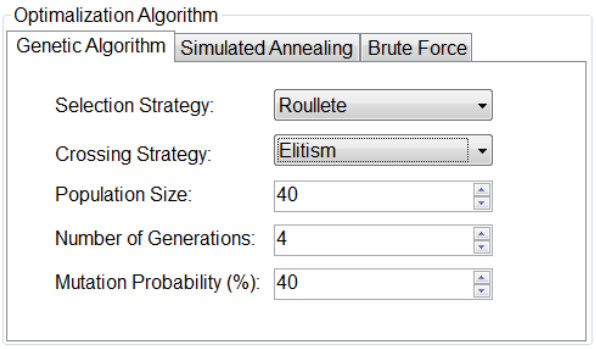
3.5.1.1 Genetický Algoritmus

Genetický algoritmus pracuje jako na prinicpu simulované evoluce. Algoritmus vytvoří „populaci“, to jest několik „jedinců“, kteří jsou reprezentování různou konfigurací parametrů. Díky tomu má každý „jedinec“ svou vlastní „fitness funkci“. Z celé generace se vyberou jedinci s nejlepší možnou hodnotou „fitness funkce“ a z nich se pomocí „křížení a mutace“ vytvoří další

generace. Jedinci se vybírají podle selekčních strategií a následně se nová generace ještě upravuje podle strategie na kontrolu populace. Podle evoluční teorie, by každá generace měla být silnější než ta předešlá, jelikož se vybírá z těch nejlepších jedinců. Algoritmus končí, pokud většina populace má stejnou konfiguraci parametrů, nebo že byl dosažen maximálního počtu populací.

Genetický algoritmus má několik parametrů, které ovlivňují jeho chování.

- Velikost populace - počet jedinců, již mají být v jedné populaci
- Pravděpodobnost mutace - s jakou pravděpodobností bude konfigurace jedinců náhodně upravována.
- Počet generací - kolik běhů algoritmu proběhne
- Selekční strategie
  - universální strategie - „fitness funkce“ každého jedince se převede na procentuální hodnotu celku. Náhodně zvolím hodnotu v rozmezí 0-1 a k ní přičítám určitou konstantu. Na kruhu, kde kruh je součet všech „fitness funkcí“, má jedinec s tou největší, větší šanci, že bude vybrán jako rodič příští generace.
  - strategie rulety - „fitness funkce“ každého jedince, se převede na procentuální hodnotu celku. Takže na „pomyslné ruletě“ nebo kruhu, kde kruh je součet všech „fitness funkcí“, má jedince s největší „fitness funkcí“ největší díl. Náhodně se poté vyberou dvě čísla v rozmezí 0-1. Čím větší rozpětí na kruhu jedinec má, tím větší šance, že bude vybrán. Vyberou se dva jedincí, kteří pak budou tvořit „rodiče“ jedince z další generace.
- Kontrola populace
  - Elitismus - z minulé generace se vybere 10% nejlepších jedinců, kteří se nakopírují do nové generace. Z té se poté odebere určitý počet nejslabších jedinců, aby byla zachována velikost populace.
  - Bloková - používají se jenom jedinci nově vytvoření.



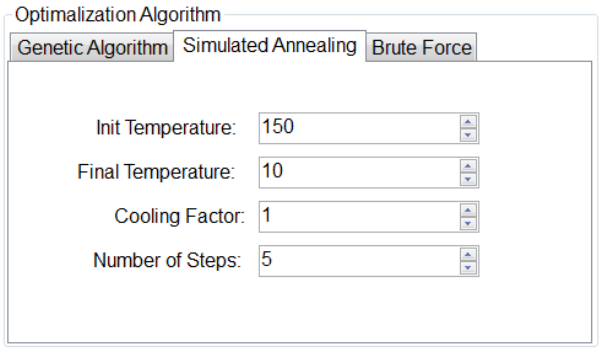
Obrázek 3.9: Optimalizace - Genetický algoritmus

3.5.1.2 Simulované ochlazování

Algoritmus Simulované ochlazování se inspiroval stejnojmennou technikou z metalurgie. Kov se při této metodě ochlazuje pozvolna, čímž se zabraňuje vzniku krystalů a zvyšuje se pevnost a odolnost materiálu. Atomy se při vyšší teplotě mohou pohybovat materiálem a hledat vhodné místo s nižší energii. Jak teplota klesá, počet pohybujících se atomů také klesá a usazují se ve vhodnějších pozicích. Na konci dává metoda větší šanci na nalezení stavu s celkově menší energií.
Algoritmus vždy v jednom kroku algoritmu vygeneruje další stav, v případě, že je tento stav lepší (má lepší „fitness funci“) než stav předchozí, tak ho přijme, v opačném případě se stále může náhodně přijmout. Tato náhodnost závisí na „teplotě algoritmu“, která je na začátku nastavená a v každém kroku algoritmu se snižuje. Čím větší teplota, tím větší šance na změnu stavu.

Algoritmus Simulované ochlazování má několik parametrů, které ovlivňují jeho chování.

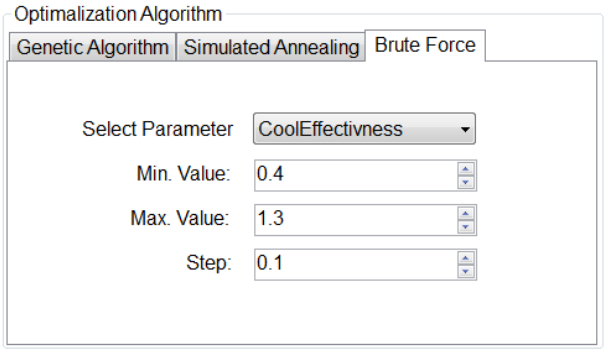
- Počáteční teplota - teplota, jakou nastavíme algoritmus na začátku běhu
- Koncová teplota - teplota, u níž algoritmus skončí výpočet
- Ochlazovací faktor - hodnota, o kterou se bude teplota oclazovat
- Počet kroků - počet kroků algoritmu. Čím větší počet kroků, tím déle se hledá stav bez ochlazování



Obrázek 3.10: Optimalizace - Simulované ochlazování

3.5.1.3 Brute Force

Funkce „Brute Force“ slouží k otestování vlivu jenotlivého parametru na chybu matematického modelu. Užiatel si vybere parameter, hodnoty jakých má nabávat a krok o který se má hodnota navyšovat. „Brute Force“ následně spouští model vždy s jinou hodnotou parametru. Po vyzkoušení všech hodnot se zobrazí graf, kde je vidět závislost chyby modelu na hodnotě parametru (obrázek 3.12).

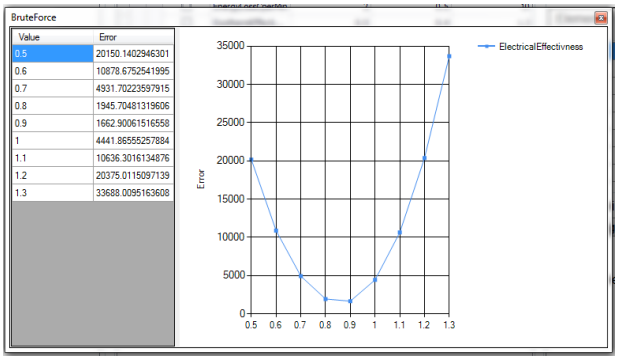


Obrázek 3.11: Optimalizace - Brute Force

3.5.2 Implementace

Implementace optimalizace je ve čtyřech namespacech.

- Optimization



Obrázek 3.12: Brute Force - výsledek

  - **Optimization** - Hlavní třída optimalizace. Vizuální komponenta zobrazena na hlavní obrazovce. Načítá data a předává zabalené v interfacu IOptimalizationParameters.
  - **COptimalizationResult** - Třída, jež je návratovou hodnotou všech optimalizačních algoritmů. Obsahuje celkovou chybu e formě třídy CHeatError a seznam výsledných hodnot parametrů ve třídě CParameterItemSet.

- Optimalization.BruteForce
  - **BruteForce** - Vizuální komponenta, kde se zobrazuje výsledná závislost chyby matematického modelu a hodnoty jednoho parametru.
  - **CBruteForce** - Hlavní třída algoritmu Brute Force. Zde probíhá volání matematického modelu a shromažďování výsledků. Třída podléhá abstraktní třídě COptimalizationAlgorithm.
  - **CBruteForceParam** - Třída parametrů, jež se předává třídě CBruteForce. Třída podléhá interfacu IOptimalizationParameters.

- Optimalization.GeneticAlgorithm
  - **Crossing.CrossingClass** - Abstraktní třída, která sdružuje strategie řízení populace.
  - **Crossing.CElitsmCrossing** - Elitismus.
  - **Crossing.CEnBlockCrossing** - Blokové řízení.
  - **Selection.SelectionClass** - Abstraktní třída, která sdružuje strategie vybírání rodičů nové generace.

  - **Selection.CRoulleteSelection** - Strategie rulety.
  - **Selection.CUniversalStochastic** - Univerzální strategie.
  - **CChromozon** - Třída reprezentující jednoho jedince.
  - **CGeneration** - Třída reprezentující jednu generaci jedinců.
  - **CGenericParam** - Třída parametrů, jež se předává třídě CEvolution. Třída podléhá interfacu IOptimalizationParameters.
  - **CParent** - Třída reprezentující dvojici chromozonů ze kterých vzniká nový potomek.
  - **CEvolution** - Hlavní třída genetického algoritmu. Je odvozená od abstraktní třídy COptimalizationAlgorithm. Obdrží vstupní parametry algoritmu a spouští hlavní kostru algoritmu.

- Optimalization.SimulatedAnnealing
  - **CAnnealing** - Hlavní třída algoritmu simulovaného ochlazování. Je odvozená od abstraktní třídy COptimalizationAlgorithm. Obdrží vstupní parametry algoritmu a spouští hlavní kostru algoritmu.
  - **CState** - Třída, jež reprezentuje stav algoritmu. Vytváří nové stavy a počítá jejich váhu.
  - **CAnnealingParams** - Třída parametrů, jež se předává třídě CAnnealing. Třída podléhá interfacu IOptimalizationParameters.

3.6 Databáze

3.6.0.1 Funkce

Funkce databáze je ukládat a načítat data do a z databáze.

3.6.1 Implementace

Implementace přístupu do databáze je tvořena třemi přístupy.

- Microsoft EntityFramework - umí namapovat databázové tabulky jako třídy (Entity) do programu. To při práci s databází velmi usnadňuje práci. EntityFramework umožňuje několik přístupů implementace.
  - Generování databáze - programátor nejprve napíše programové třídy (Entity) a poté pomocí EntityFramework vygeneruje databázovou strukturu.

  - Generování Entit - programátor nejprve vytvoří databázové tabulky a z nich poté pomocí EntityFrameworku vygeneruje programové třídy(Entity). V tomto případě byl použit druhý přístup.

- OleDbConnection - v ojedinělých případech, kdy se nehodilo použít přístup EntityFramework, šlo třeba o spojení několika tabulek a složitějších SQL dotazů. Byl použit klasický přístup do databáze přes OleDbConnection. Veškeré složitější SQL scripty jsou uloženy v souboru SqlScripts.xml.

- PTSW Framework - U několika málo tříd, jenž již byly v systému naimplementovány, je použit přístup na databázi přes firemní PTSW Framework.

Seznam tříd.

- **CEntityFrameworkUtil** - Statická třída, jenž přistupuje k databázi a vrací data.

- **ConnectionPoolManager** - Konkrétní implementace ObjectPool.

- **DatabaseManager** - Statická třída, jenž vykonává jednotlivé SQL příkazy.

- **ObjectPool** - Abstraktní třída, která udržuje připojení k databázi.

3.6.2 Ukládání a načítání konfigurací

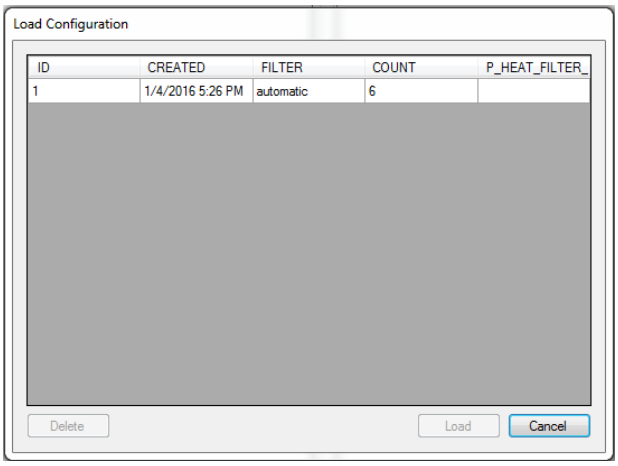
3.6.2.1 Funkce

Ukládání a načítání konfigurací je funkce, jež umožňuje načítat a ukládat různé uživatelem nadefinované konfigurace. Tato funkcionalita je u Taveb a u Filtrů. V obou případech se v okamžiku vyfiltrování taveb, popřípadě výběru seznamu filtrů, po stisku tlačítka „Save“ daný výběr uloží. Při dalším spuštění už proto není třeba vytvářet ten stejný výběr, ale pouze při stisknutí tlačítka „Load“ se zobrazí dialogové okno, ve kterém se zobrazí všechny uložené konfigurace. Jakoukoli z těchto konfigurací lze načíst.

3.6.2.2 Implementace

Implementace této funkcionality je tvořena v abstraktní vizuální třídě **LoadConfiguration**. Konkrétní implementace této třídy

- **CHeatFilterConfiguration**



Obrázek 3.13: Ukládání a načítání konfigurací

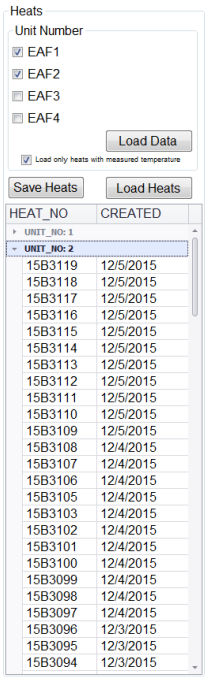
- **CFilterConfiguration**

naplní vizuální komponentu daty. Vizuální komponenta poté vrací příslušnou vybranou hodnotu.

3.7 Tavby

3.7.1 Funkce

V levé části hlavní vizuální komponenty je část, která je zodpovědná za načítání taveb. Tavby mohou být již odfiltrované a uložené v databázi. Ukládání a načítání taveb je blíže řešeno v sekci Databáze (3.6). Tavby, které jsou zobrazeny ve vizuální komponentě, jsou tavby, pro které se bude adaptovat matematický model. Tavby jsou seřazeny podle části pece (Unit number), z níž jsou. V případě, že uživatel nechce načítat již vyfiltrovaná data, ale chce načíst všechny tavby, je třeba v horní části vybrat, ze kterých částí pece se budou tavby načítat. Na obrázku 3.14 je zobrazeno vybírání částí pece.



Obrázek 3.14: Tavby

3.7.2 Implementace

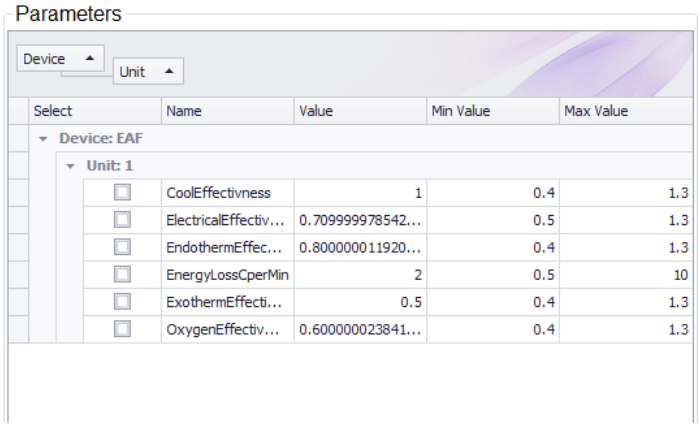
Sekce taveb je z části implementována přímo do hlavní vizuální třídy programu Identification. Jde o načítání taveb z databáze. Tavby se načítají v metodě LoadHeats. Načítají se pouze ty tavby, které odpovídají peci a části pece, které byly vybrány uživatelem. Další třídou, jež je součástí implementace taveb, je CHeatFilterConfiguration. Vizuální třída, která je odvozena od abstraktní třídy LoadConfigurations, CHeatFilterConfiguration zařizuje ukládání a načítání filtrů z databáze. (více o LoadConfigurations zde 3.6.2)

3.8 Práce s parametry

3.8.1 Funkce

Na hlavní obrazovce, jsou při zadání sekce načteny parametry modelu. Načteny jsou pouze parametry, které souvisejí pouze s částí pece, která byla vybrána. Z těchto načtených parametru lze vybírat jednotlivé parametry, které poté budou dále zpracovány v optimalizaci. U parametrů lze měnit mezní hodnoty, kterých může parametr dosahovat. Hodnota, která je

v tabulce u parametrů, je aktuální hodnota, která se v modelu používá. V případě, že by se testovala pouze chyba parametrů přes funkci „Test parametrů“, pak lze hodnotu parametru měnit.



Obrázek 3.15: Parametry

3.8.2 Implementace

Implementace je rozdělena do těchto tříd.

- **ParameterSelection** - Vizuální komponenta, která je zobrazena na hlavním panelu. Při inicializaci se volá metoda LoadData, která načítá do tabulky všechny parametry z databázové tabulky `M_PARAMETER_ADAPT`.

- **CParameterItem** - Třída, jež reprezentuje jednotlivý parametr. Obsahuje metodu Mutate, která je využívána optimalizačními algoritmy při náhodné změně hodnoty parametru.

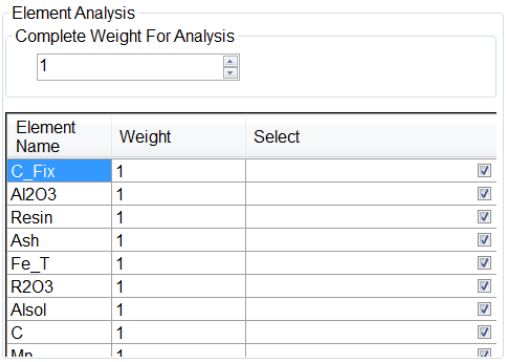
- **CParameterItemSet** - Kontainer pro třídu CParameterItem

- **CNullAlgorithm** - Třída, jež je děděná ze třídy COptimalizationAlgorithm. Tato třída zajišťuje funkcionalitu pro test parametrů. Nejde tedy o žádný optimalizační algoritmus, ale simulace modelu se pouští pouze přez optimalizační agloritmy, takže zde musel být do návrhu vložen tzv. „Null Pattern“. CNullAlgorithm, se „na venek“ chová jako optimalizační algoritmus, nicméně pouze spouští matematický model s hodnotami parametrů, které jsou nastaveny v ParameterSelection.

3.9 Práce s prvky chemické analýzy

3.9.1 Funkce

Při načítání hlavní obrazovky, jsou načteny prvky chemické analýzy, na kterých je při simulaci tavby měřena chyba. Z počátku jsou vybrány všechny prvky, ale tento výběr lze změnit a lze vybrat pouze některé prvky. Prvků i celé chemické analýze se dá přiřadit koeficient nebo-li váha, která je následně použita při výpočtu chyby modelu (více 2.2.4)



Obrázek 3.16: Chemická analýza

3.9.2 Implementace

Implementace je rozdělena do tří tříd.

- **AnalysisConfiguration** - Vizuální komponenta, která je zobrazena na hlavním panelu. Při inicializaci načítá z databáze chemické prvky. Při inicializaci adaptace vrací třídu CAnalysisConfiguration
- **CAnalysisConfiguration** - Třída udržuje koeficient chemické analýzy a vybrané chemické prvky s jejich koeficienty.
- **CAnalysisElement** - Reprezentuje jeden chemický prvek a jeho koeficient.

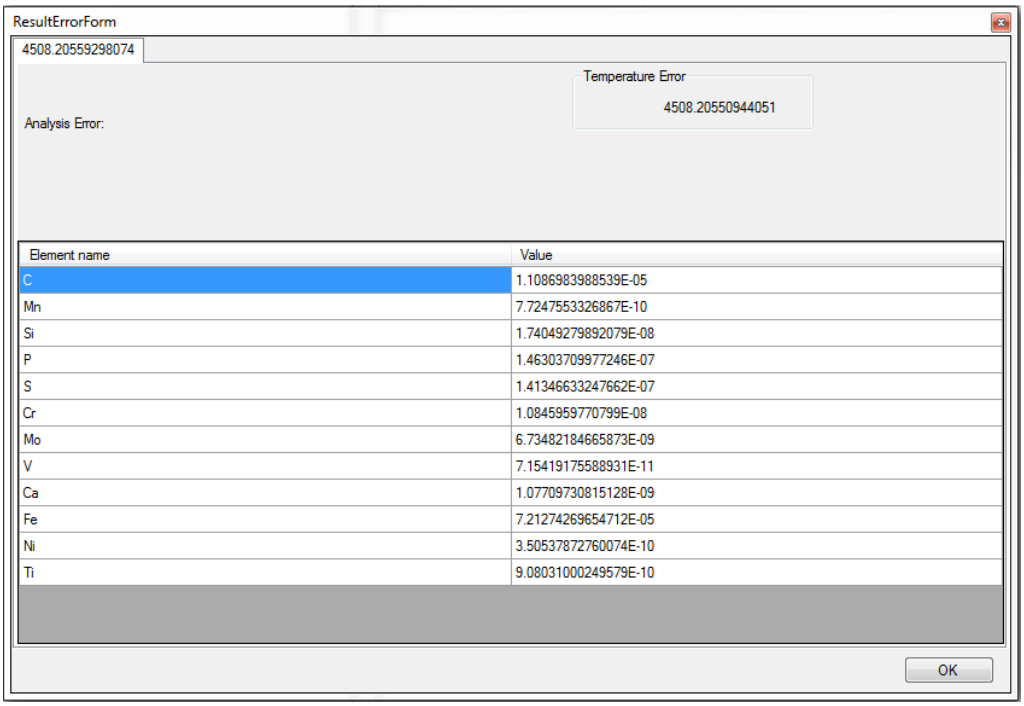
3.10 Zobrazování chyby

3.10.1 Funkce

Zobrazení chyby se spustí ve dvou případech.

- Při ukončení identifikace
- Při ukončení testování parametrů

V obou případech se otevře jednoduché okno, kde je zobrazena hodnota celkové chyby modelu a hodnoty parametrů při této chybě. V případě ukončení identifikace jsou tyto hodnoty optimalizovány. Z tohoto okna může uživatel stiskem tlačítka „uložit“, uložit hodnoty parametrů do databáze.



Obrázek 3.17: Zobrazování chyby

3.10.2 Implementace

Implementace zobrazování chyby je tvořena dvěma vizuálními komponentami.

- ResultErrorForm - Vizuální komponenta, která dostane seznam COptimalizationResult a pro každý tento výsledek vytvoří novou instanci vizuální komponenty ResultError, která zobrazuje jednotlivou chybu skupiny taveb. V případě, že by byla optimalizace spuštěna pro více

částí jedné pece, bude mít ResultErrorForm několik záložek s výsledky pro každou část pece.

- ResultError - Vizuální komponenta, jež je použita jako záložka v ResultErrorForm

3.11 Testování

Testování softwaru probíhalo v několika fázích a bylo s ním spojeno mnoho problémů. Jeden z velkých problémů byla závislost vyvíjeného softwaru na systému, do kterého je integrován. Software samotný nebylo možno spouštět bez spouštění celého systému. V případech, kdy to bylo možno, se funkcionalita otestovala mimo systém, jestli funguje správně, a až následně byla testována v rámci systému, zda v integraci s ostatním systémem nevykazuje nečekané chování. Příkladem takového testování bylo například testování algoritmů, kde algoritmus byl napsán a otestován na jiný problém. Následně byl vložen do systému a byla upravena jeho „Fitness“ funkce tak, aby vyhovovala problému tohoto softwaru. Až následně se kontrolovalo, jestli algoritmus má předpokládané odezvy, nicméně správnost implementace algoritmu již byla předpokládána.

3.11.1 Developer's Unit Testing

Základní testování bylo prováděno na úrovni metod a tříd. Výhodou tohoto testování v prostředí Visual Studia je automatické vytváření testovacích tříd, kde se nastaví, které metody se mají volat se kterými parametry, a kontroluje se výsledek. Tento způsob testování byl použit na všechny třídy/metody, u nichž se dala předpokládat algoritmická chyba. Většina chyb se projeví již během tohoto testování. Nevýhodou někdy bývá, že napsat správný test je někdy těžší než napsání samotné metody.

3.11.2 Feature testing

Pojmem „Feature testing“ se myslí testování funkcionalit. V této části se testují jednotlivé funkce systému. Zde se přichází na komplexnější chyby v komunikaci jednotlivých objektů. Taktéž se zde testuje, zda daná funkcionalita se chová tak, jak se od ní očekává. V případě chyby se musí upravit kód a tento kód většinou znovu otestovat na úrovni unit testů.

3.11.3 Integration testing

Při „Integration testing“ se testuje celková funkcionalita aplikace. V tomto případě tedy celkový software. Byla zde testována komunikace mezi jednotlivými funkcemi a komunikace mezi jednotlivými vizuálními komponenty.

Závěr

Úkolem této diplomové práce bylo vytvoření daného softwaru. Tato zpráva popisuje proces tvorby takového softwaru od návrhu až po jeho implementaci. Výsledná aplikace má 86 tříd na 5001 řádcích kódu s průměrným indexem udržitelnosti 75,2 podle Visual Studia 2010.

Během vytváření tohoto softwaru, se vyskytlo mnoho problémů, které musely být vyřešeny nebo řešeny v průběhu implementace. Největší z těchto problémů byl problém matematického modelu, který je svou současnou implementací velmi pomalý. Software nicméně není fixován na tento matematický model. V případě změny matematického modelu ve firemním frameworku PTSW bude tento software fungovat stejně. Software, jenž je výstupní částí tohoto projektu, je funkční a splňuje veškeré požadavky, již na něj byly kladeny v návrhu a splňuje i požadavky ze zadání diplomové práce. Optimalizace parametrů funguje a software vrací výsledky parametrů, které vracejí menší chybu matematického modelu než současné hodnoty parametrů.

Během testování se ukázalo, že určité komponenty je možné ještě upravit, aby byla efektivita softwaru větší. Do budoucna se tedy plánuje s tímto softwarem ještě pracovat a upravovat jeho komponenty. Komponenta, která se ukázala jako velmi efektivní a dala by se použít i v jiných částech systému, je komponenta filtr taveb. Celý software je navržen tak, aby v okamžiku jakéhokoli rozšíření o algoritmus nebo o určitou strategii nebylo třeba velkého zásahu do systému. Celý software je patřičně okomentován a byla snaha psát kód co nejsrozumitelněji, jelikož se dá předpokládat, že v průběhu času jej bude upravovat spousta dalších programátorů.

Literatura

[1] Ing. David Stránský, P.: Identifikace modelu. 2013, doi:http://kzei.fsv.cvut.cz/pdf/YMMO_pr_11.pdf.

[2] Ben-David, S.; Blitzer, J.; Crammer, K.; aj.: A theory of learning from different domains. *Machine Learning Journal*, 2010: s. 1–2.

[3] Nuseibeh, B.; Easterbrook, S.: Requirements Engineering: A Roadmap. 2015, doi:http://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf.

[4] Ljung, L.: *System Identification, Theory for the User, Second Edition*. Prentice Hall Ptr, 1999.

[5] Nelles, O.: *Nonlinear System Identification*. Springer-Verlag Berlin Heidelberg GmbH, 2001.

[6] Pecinovský, R.: *Návrhové Vzory*. Computer Press, 2015.

[7] Mundhenk, N.: Simulated Annealing Project. 2007. Dostupné z: `https://en.wikiversity.org/wiki/Simulated_Annealing_Project`

[8] Likeš, J.; Machek, J.: *Matematická statistika*. SNTL Praha, 1988.

[9] Microsoft: Overview of the .NET Framework. 2013. Dostupné z: `http://msdn.microsoft.com/en-us/library/zw4w595w.aspx`

[10] JetBrains: Productivity Tool for Visual Studio. 2015. Dostupné z: `https://www.jetbrains.com/resharper/`

[11] Microsoft: namespace (C# Reference). 2015. Dostupné z: https://msdn.microsoft.com/en-us/library/z2kcy19k.aspx

Příloha **A**

Seznam použitých zkratek

.Net Framework „.Net framework je technologie společnosti Microsoft, která podporuje vytváření a běh aplikací. .Net je součástí mnoha aplikací bežících na systému Windows, které poskytují stejnou funkcionalitu.“[9]

PTSW Framework (popř. pouze Framework) Framework, který byl vyvinut firmou PTSW, pro zajišťování správného fungování jejich aplikací.

C# Je programovací jazyk, který byl navržen a vytvořen speciálně pro prostředí .NET. V tomto jazyce mužete vytváret dynamické webové stránky, komponenty distribuovaných aplikací, komponenty pro přístup k datům, nebo klasické aplikace pro systém Windows.

DevExpress Framework poskytující vizuální komponenty do frameworku .Net.

LINQ „Language Inegrated Query - množina funkcí pro práci s řetezci v jazyce C# nebo Visual Basic.“ [9]

Entity Framwork „object-relation mapper, který umožňuje vývojářům na frameworku .Net pracovat s relačními daty používáním domain-specific objektů. To vylučuje většinu kódu data-access, který vývojáři museli psát.“ [9]

ReSharper „ReSharper rozšiřuje prostředí Visual Studia s více než 1700 kontroly kódu pro C# , VB.NET, ASP.NET, JavaScript, TypeScript a další technologie. Pro mnoho těchto kontrol ReSharper poskytuje quick-fix na zlepšení kódu.“ [10]

EAF Elektrická oblouková pec

LF Pánvová pec

VD Vakuová jednotka

namespace - „Klíčové slovo, které je používáno v rozshahu, který obsahuje množinu společných objektů. Namespace se dá použít pro organizování částí kódu a globálně unifikovat typy.“ [11]

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
  - `readme.txt` ..... stručný popis spuštění aplikace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `doku` ..... programová dokumentace