Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Ondřej Heřmánek

Aplikace pro zpracování stavebních rozpočtů

Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek

Studijní program: Informatika

Studijní obor: Programování

Praha rok 2011

V první řadě bych chtěl poděkovat vedoucímu práce panu Mgr. Pavlu Ježkovi za jeho čas, který mi věnoval při konzultacích, a především za jeho cenné rady a připomínky. Dále bych chtěl poděkovat svým rodičům za jejich podporu při studiu. V neposlední řadě bych rád poděkoval Martinu Mrázovi a Jiřímu Helmichovi za jejich pomoc při finální korektuře.

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

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

V Praze dne 4. srpna 2011 Ondřej Heřmánek

Název práce: Aplikace pro zpracování stavebních rozpočtů

Autor: Ondřej Heřmánek

Katedra: Katedra distribuovaných a spolehlivých systémů

Vedoucí bakalářské práce: Mgr. Pavel Ježek

Abstrakt:

Aplikace je přednostně určena pro rozpočtování staveb tak, aby vyhovovala potřebám menších stavebních firmem, které zaměstnávají pracovníky všech řemesel, mají vlastní sklad materiálu a vlastní vozový park. Největším přínosem aplikace je zautomatizování určitých výpočtů při rozpočtování stavby.

Klíčová slova: Rozpočtování staveb, Kalkulace

Title: Application for Construction Costs Calculation

Author: Ondřej Heřmánek

Department: Department of Distributed and Dependable Systems

Supervisor: Mgr. Pavel Ježek

Abstract:

Primal goal of this application is costs calculation of constructions. Application is designed to meet the requirements of small construction companies that employ workers of all crafts and have their own warehouse for materials and heavy machinery. The biggest benefit of this application is processing certain calculations automatically.

Keywords: Costs calculation of constructions

Obsah

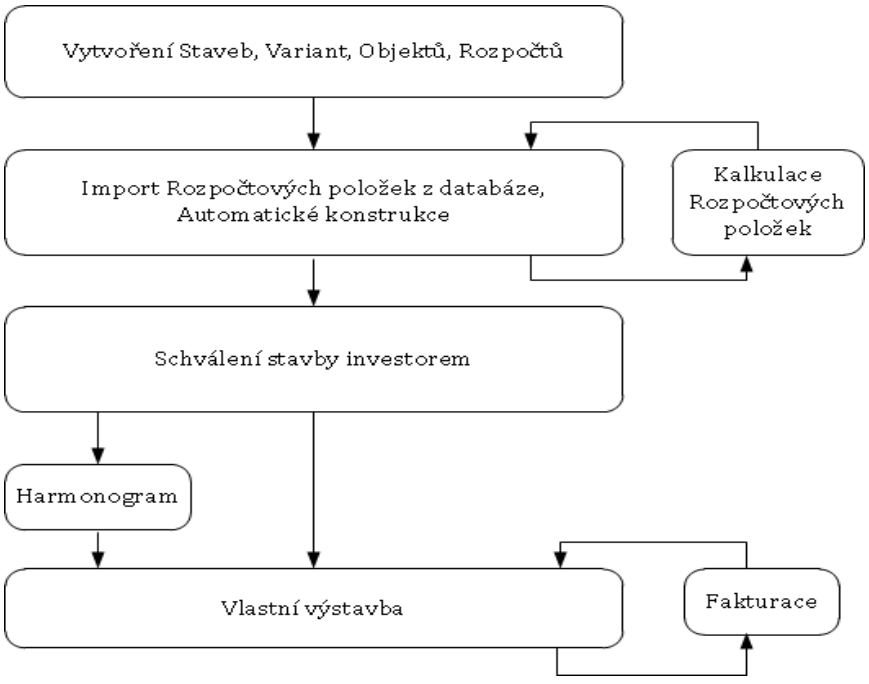
1. Úvod

Cílem této práce je vytvořit aplikaci, která stavebním firmám zjednoduší zpracovávání *Rozpočtů Staveb*, jejich správu a kontrolu průběhu. Aplikace se jmenuje KaRaFa – název spojuje slova Kalkulace, Rozpočty, Faktury - a vznikla na základě potřeb stavební firmy H&Co. Aplikace KaRaFa umožňuje zaměstnancům stavební firmy vypočítat cenu *Stavby*, vytvořit a spravovat *Harmonogram* pro kontrolu průběhu stavby, či vygenerovat faktury za provedené práce. Primárním zaměřením aplikace je výpočet ceny *Stavby*.

Pro jednodušší orientaci v textu je zvolena následující konvence: *kurzívou* jsou značeny klíčové pojmy, `jiným typem písma` jsou značeny názvy jmenných prostorů, tříd a metod, použitých ve zdrojovém kódu aplikace, a databázových objektů.

1.1 Proces rozpočtování stavby

Realizace stavby je komplexní proces. Zaměstnanec stavební firmy vytvářející *Rozpočty* - *Rozpočtář* - má na začátku k dispozici jen projekty *Stavby* vytvořené jejím *Projektantem*. Z projektu je potřeba určit *Celkovou cenu Stavby*. Cenu vypočítává uživatel *Rozpočtář* tak, že prochází postupně všechny projekty a počítá množství potřebného materiálu a práce na *Stavbu*. Když *Rozpočtář* dokončí výpočet a *Investor Stavby* vše schválí, může stavební firma začít provádět stavební práce. Kontrolu průběhu *Stavby* má starosti uživatel *Stavbyvedoucí*. Uživatelé *Stavbyvedoucí* a *Rozpočtář* můžou být – a v malých stavebních firmách většinou bývají – jeden zaměstnanec. Průběh procesu je znázorněn na obrázku číslo 1.



Obr.1 – Proces realizace stavby

Každá *Stavba* je hierarchicky rozčleněna na menší a ucelenější části, které usnadňují její realizaci. Na nejvyšším stupni myšleného stromu je Stavba, která představuje zakázku *Investora* vůči stavební firmě. *Stavbu* si lze představit jako jeden šanon, kam se ukládají všechny potřebné informace a údaje – projekty, návrhy uspořádání, údaje pro výpočet ceny, faktury ke *Stavbě*, údaje o *Investorovi*, o *Projektantovi*, datum zahájení a datum konce *Stavby*, potřebná povolení. *Stavba* se dělí na několik různých *Variant* podle přání *Investora* – dům bude s garáží nebo bez garáže, zahrada bude s grilem nebo bez grilu, a podobně. *Variant* může být více, ale vždy je alespoň jedna představující zakázku. Na základě projektu *Rozpočtář* rozdělí *Variantu* na její *Objekty*. *Objektem* se rozumí logicky oddělená část *Varianty*. V případě, že *Stavbou* je panelový dům, *Objekty* můžou být jednotlivé byty. Pokud je *Stavbou* rodinný dům, zpravidla je celý zahrnut do jednoho *Objektu*. Každá *Varianta* by měla obsahovat alespoň jeden *Objekt*. Každý *Objekt* se ještě dále dělí na menší celky – *Rozpočty*. Pomocí *Rozpočtů* může být oddělen výpočet pro přípojky inženýrských sítí, příjezdová cesta, garáž, rodinný dům.

Rozpočty obsahují položky *Materiálů*, *Dodávek*, *Strojů* a *Práce*, které určují *Celkovou cenu*. V rámci *Rozpočtu* probíhá výpočet množství - *Výměry* - těchto položek. *Rozpočtů* v *Objektu* může být více, musí vždy být alespoň jeden. Rozdělení *Objektu* na více než jeden *Rozpočet* zvyšuje celkovou přehlednost členění výpočtu.

Každý *Rozpočet* obsahuje seznam *Oddílů*. *Oddíly* člení *Rozpočet* na menší celky, které v sobě zahrnují položky pro realizaci *Stavby*. Pod pojmem *Oddíl* si lze představit „Vodorovné konstrukce“, „Svislé konstrukce“, „Truhlářské práce“, apod. *Oddíly* již zahrnují jednotlivé položky, které udávají *Celkovou cenu*.

Položky - *Materiály*, *Dodávky*, *Stroje* a *Práce* - , které se nahrávají do *Oddílů*, částí *Rozpočtů*, má stavební firma H&Co uložena v databázi ve formátu MDB [1] souboru. Tato databáze již v sobě obsahuje část výpočetní logiky – kromě tzv. *Rozborových položek* s *Materiály*, *Dodávkami*, *Stroji* nebo *Pracemi* databáze obsahuje tzv. *Rozpočtové položky*. *Rozpočtové položky* sdružují libovolný počet *Rozborových položek*, které tvoří její *Rozbor*. *Rozpočtové položky* reprezentují nejmenší ucelenou část realizace stavby.

V praxi to znamená, že uživatel může použít *Rozpočtovou položku* „Položení dlažby“, která zahrnuje *Materiál* (dlažbu) a *Práci* (poklad dlažby). Uživateli stačí vypočítat plochu, kterou je potřeba pokrýt a pomocí příslušných koeficientů *Spotřeby*, které jsou již v databázi nastaveny, se automaticky vypočítá množství potřebných dlaždic a doby, kterou na *Práci* bude pracovník potřebovat. Tento Materiál a Práce tvoří *Rozbor Rozpočtové položky*. Nahrání *Rozpočtové položky* a výpočet jejího množství je jednodušší, než nahrávat každou její *Rozborovou položku* extra a pro každou přepočítávat její množství.

Množství *Rozborových položek* se počítá z množství *Rozpočtové položky* podle jejich *Spotřeby*, vzhledem k *Rozpočtové položce*. Každá položka – *Rozpočtová* i *Rozborová* – má danou *Jednotkovou cenu*, tedy cenu za jednu svou měrnou

jednotku. *Rozborové položky* mají *Jednotkovou cenu* danou z databáze, *Jednotková cena Rozpočtové* položky se vypočítá pomocí všech jejich *Rozborových položek*. Vypočítanou hodnotu by mělo být možné případně ručně změnit podle nároků realizace *Stavby*.

Každá *Rozpočtová položka* nahraná do libovolného *Oddílu* musí mít uživatelem zadané své množství. Z tohoto množství položky - *Výměry* - se určí *Celková cena* za každou jednotlivou položku. Procesu výpočtu *Celkové ceny* se říká *Kalkulace*. *Celková cena Varianty* se počítá jako součet *Celkových cen* všech *Materiálů*, *Dodávek*, *Strojů* a *Práce* obsažených ve všech *Oddílů* všech *Rozpočtů* všech *Objektů* dané *Varianty*. Finálně zpracované *Varianty* se předloží *Investorovi*, ten z nich vybere tu, která se mu nejvíce zamlouvá.

Když dokončí *Rozpočtář* realizaci *Stavby* a je *Investorem* vybrána *Varianta*, která se mu nejvíce zamlouvá, může stavební firma zahájit stavební práce. Pro kontrolu průběhu *Stavby* slouží *Stavbyvedoucímu Harmonogram*. *Harmonogram* se skládá z jednotlivých *Etap*. Pro každou *Etapu* uživatel určí datum jejího začátku a datum jejího konce. Některé *Etapy* si může *Stavbyvedoucí* označit jako *Uzlový bod*, u nichž je potřeba dodržet datum konce *Etapy*. *Etapy* v *Harmonogramu* odpovídají *Oddílům* z *Rozpočtu* tak, že každý *Oddíl* má nějakou přednastavenou výchozí *Etapu*, do které patří. Výchozí mapování *Oddílů* do *Etap* se určuje z databáze. Harmonogram nemusí být využit pro každou *Stavbu*.

Fakturování *Investorovi* probíhá průběžně. Záleží na domluvě s *Investorem*, v jakých intervalech si přeje dostávat faktury. Všechny faktury je potřeba archivovat ke *Stavbě*, pro kterou byly vystaveny. V aplikaci bude nastavena jedna výchozí hodnota DPH, která bude nastavena všem *Rozpočtovým položkám*, u nich musí být možnost tuto výchozí hodnotu libovolně pozměnit.

Firma H&Co používá existující aplikace na zpracování stavebních rozpočtů (Aspe nebo Stavex) i přesto, že tyto aplikace mají spoustu nedostatků, které práci s nimi ztěžují. Jejich používání je časově náročné, uživatelské rozhraní je nepřehledné a obtížné na ovládání, vyhodnocování výrazu pro výpočet množství neupozorňuje na syntaktické chyby a bez upozornění ukončí výpočet, nebo umožňují nevhodné sdílení databázových záznamů mezi různými stavbami. Další nevýhodou těchto aplikací je, že umožňují práci jen na jednom počítači, neumožňují přenášet práci na více počítačů jednoho uživatele, aby mohl pracovat z kanceláře i z domova, nebo distribuovat výpočet mezi více uživatelů. Všechny tyto nedostatky by měla aplikace KaRaFa vyřešit.

1.2 Pojmy

- *Hierarchie stavby*
  - *Hierarchie stavby* je členění *Stavby* na menší části

- *Stavba*
  - *Stavba* je na vrcholu celé *Hierarchie*. Obsahuje všechny údaje potřebné pro realizaci Stavby.

- *Varianta*

- *Varianta* reprezentuje jedno možné uspořádání *Stavby*, *Stavba* může obsahovat více *Variant* podle návrhů *Investora*.

- *Objekt*
  - Každá *Varianta Stavby* se rozděluje na menší *Objekty*. *Objektů* může mít *Varianta* více, nebo může mít jen jeden *Objekt* zahrnující celou *Variantu*.

- *Rozpočet*
  - *Rozpočet* je část *Objektů Stavby* a obsahuje seznam *Oddílů*.

- *Oddíl*
  - *Oddíl* je nejmenší část *Hierarchie stavby*. *Oddíly* obsahují nahrané *Rozpočtové položky* z databáze. Pomocí *Oddílů* se vytváří *Harmonogram Stavby*.

- *Harmonogram*
  - Slouží uživateli ke kontrole a spravování průběhu *Stavby*.

- *Etapa*
  - *Etapa* reprezentuje určitou fázi *Stavby*.

- *Položka*
  - Položky se nahrávají z databáze a slouží k výpočtu *Celkové ceny*. Položkou může být *Rozpočtová položka* nesoucí výpočetní logiku, nebo *Rozborová položky* – tedy konkrétní *Materiál*, *Dodávka*, *Práce* nebo *Stroj*. Položky se do aplikace importují z databáze.

- *Rozbor Rozpočtové položky*
  - *Rozpočtová položka* se skládá z libovolného počtu *Materiálů*, *Dodávek*, *Prací* a *Strojů*. Tyto položky tvoří rozbor *Rozpočtové položky*.

- *Výměra*
  - *Výměra* je vypočítané množství položky potřebné pro realizaci *Stavby*.

- *Jednotková cena položky*
  - Cena jedné měrné jednotky dané položky.

- *Kalkulace*
  - *Kalkulace* je proces výpočtu *Celkové ceny* jednotlivých *Rozpočtových položek*.

- *Spotřeba*
  - *Spotřeba* udává množství *Rozborové položky* potřebné pro realizaci jedné měrné jednotky nadřazené *Rozpočtové položky.*

- *Uživatelská práva*
  - Aplikace KaRaFa rozlišuje uživatele s právy „Administrátor“, „Stavbyvedoucí“ a „Rozpočtář“. Jeden uživatel může mít libovolnou kombinaci práv.

1.3 Požadavky

1. Práce s existující databází

    - Hlavním požadavkem je, aby aplikace KaRaFa uměla pracovat s existující databází položek, kterou má firma k dispozici. Záznamy databáze nesou část výpočetní logiky v sobě, tu je důležité zachovat a používat.

2. Ukládání a načítání dat

    - Data načtená do aplikace je potřeba ukládat a načítat způsobem, který jednoduše umožní přenášet data mezi různými instancemi aplikace na různých pracovních stanicích. Preferované členění firmy je práce jednoho uživatele na více pracovních stanicích.

3. Jednoduché a přehledné grafické rozhraní

    - Kvalitní uživatelské rozhraní je základem efektivně fungující aplikace. Při vývoji musí být kladen velký důraz na jeho snadnou použitelnost a intuitivní ovládání.

2. Vylepšený výpočet výměr

    - Každá položka má nějakou *Výměru*, která udává její množství potřebné na stavbu. Pomocí *Výměry* položek se počítá *Celková cena* částí *Hierarchie*.

    - Vylepšení oproti jiným aplikacím bude spočívat ve vytvoření mechanismu pro vyhodnocení výrazu, který dokáže detekovat chyby (špatné uzávorkování, špatná syntaxe výrazu) a upozorní uživatele na ně.

3. Automatické výpočty konstrukcí

    - Unikátní modul, který žádná současná aplikace neimplementuje. Modul ulehčí výpočet *Výměr* položek pro komínové a sádrokartonové konstrukce. Uživatel jen vybere vlastnosti konstrukce a zadá její rozměry. Aplikace na základě těchto zadaných údajů dopočítá *Spotřebu* veškerého *Materiálu* automatiky. Tím je eliminován problém, že by uživatel mohl zapomenout započítat nějaký *Materiál* nebo špatně spočítat *Spotřebu* či *Výměru*, kdyby konstrukce počítal ručně.

4. Možnost aktualizace databáze

    - V databázi jsou uloženy všechny *Rozpočtové položky*, seznamy výchozích *Etap* a *Oddílů*, data pro automatické výpočty, proto je nezbytné ji udržovat aktuální. Aktualizace bude probíhat z aktualizačního MDB [1] souboru s pevně danou strukturou.

5. Vytváření nových částí *Staveb* zkopírováním již existující části *Stavby*

    - Aplikace umožní vytvoření nové čísti *Stavby* zkopírováním dat existující čísti *Stavby*. Zároveň bude zaručeno, že se změna nové čísti *Stavby* neprojeví v té existující a obráceně.

6. Harmonogram

    - Harmonogram umožní rozložit *Stavbu* do několika *Etap*, díky tomu může uživatel upravovat průběh stavby. Harmonogram bude upozorňovat na *Etapy*, které stále nejsou dokončené, přestože by již měly být.

7. Uživatelský přístup

    - Různé funkce aplikace budou přístupné pouze uživatelům s příslušnými právy. Jeden uživatel může mít více práv zároveň.

8. Přehledy, Tiskové sestavy a Fakturace

    - Generování přehledných výstupních informací pro interní kontrolu procesu *Kalkulace*, pro realizaci *Stavby*, pro *Investora*.

2. Analýza

2.1 Použité technologie

Jediný požadavek ze strany zadavatele na technologii je kompatibilita aplikace s operačním systémem Windows, aplikace KaRaFa by proto měla pracovat nad tímto operačním systémem. Autor si vzhledem k osobním preferencím pro vývoj aplikace zvolil programovací jazyk C# nad frameworkem .NET s využitím WPF pro grafické rozhraní. Aplikace KaRaFa je desktopová aplikace, není počítáno s webovou verzí.

2.2 Výběr databáze

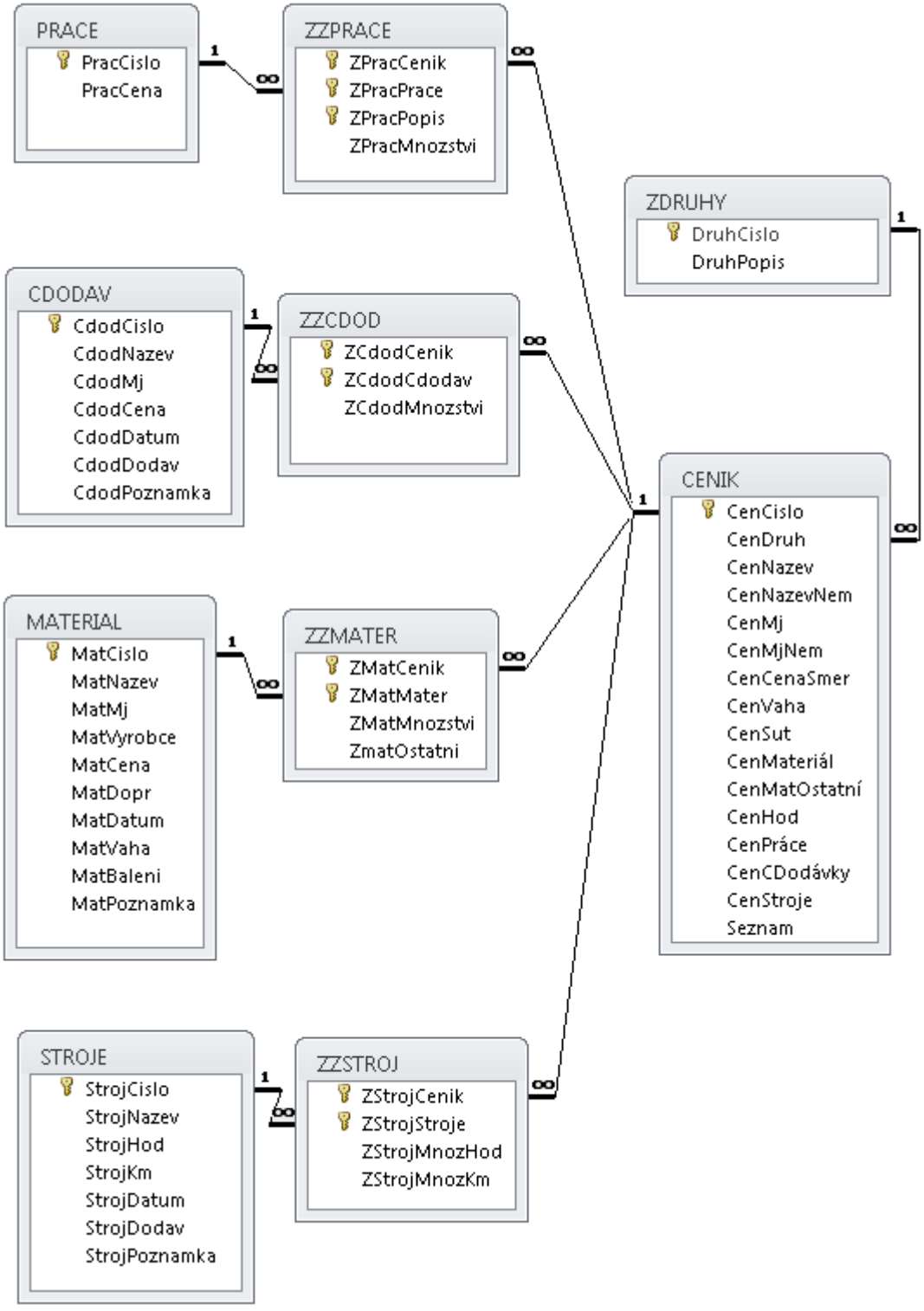
Aplikace pro práci se stavebními materiály vyžaduje databázi, kde jsou tato data uložena. Databáze může být reprezentována prostými textovými soubory. Výhodou tohoto řešení je, že tato reprezentace databáze má minimální velikost a není tak potřeba žádného speciálního nástroje na její správu a čtení dat. Nevýhodou tohoto řešení je fakt, že nad daty uložených v prostých souborech neexistuje žádný optimalizační nástroj urychlující jejich čtení. Rovněž chybí řešení průběžného ukládání dat do paměti pro rychlejší opakovaný přístup, které by v tomto případě bylo velmi užitečné. Formát takového souboru by byl pevně daný, který se při ukládání dat musí dodržet. Dotazy pro získání dat z databáze by vyžadovaly přečtení všech souborů.

Databáze, kterou má stavební firma k dispozici, je uložena v MDB souboru. Je to relační databáze pracující pod nástrojem MS Access. Výhodou nástroje MS Access je jeho jednoduchá dostupnost v rámci instalačního balíčku MS Office. Nevýhodou využití nástroje z balíčku MS Office je placená firemní licence, která by mohla představovat případnou distribuci aplikace KaRaFa i dalším firmám mohla být překážka – aplikace by neměla být závislá na komerčních aplikacích.

Dalším databázovým nástrojem pro uložení dat a práci s nimi v relační databázi je MS SQL Server. Microsoft nabízí edici MSSQL Server Express zdarma ke stažení. Tím by bylo zajištěno, že instalace aplikace KaRaFa nebude vyžadovat žádný komerční software. Další vlastností možnosti MSSQL Serveru je, že databáze může sloužit nejen jednomu uživateli, ale i jako centralizované úložiště dat pro více uživatelů jedné firmy. Tuto vlastnost využijí firmy s více uživateli, mezi kterými je práce rozdělena. Na rozdíl od sdílení MDBd [1] souboru, MSSQL Server nativně podporuje přístup více uživatelům zároveň. Další výhodnou vlastností MSSQL Serveru pro aplikaci KaRaFa je integrovaný nástroj pro import dat z databází různých formátů včetně importu dat z MDB souboru.

Aplikace KaRaFa bude využívat ke správě databáze MSSQL Server a to i přesto, že původní databáze existuje pod aplikací MS Access. První import dat z původní databáze v MDB souboru do nové databáze na SQL serveru proběhne přes výše zmíněné integrované nástroje. Poté se data ručně upraví podle potřeb aplikace KaRaFa. Z upravené databáze na MSSQL Serveru se vytvoří záloha, která bude dodávána k instalaci aplikace KaRaFa jako výchozí databáze. Databáze bude sloužit pouze jako zdroj dat při nahrávání položek do databáze. Ukládání dat bude realizováno mimo databázi, jak popisuje kapitola 2.4.

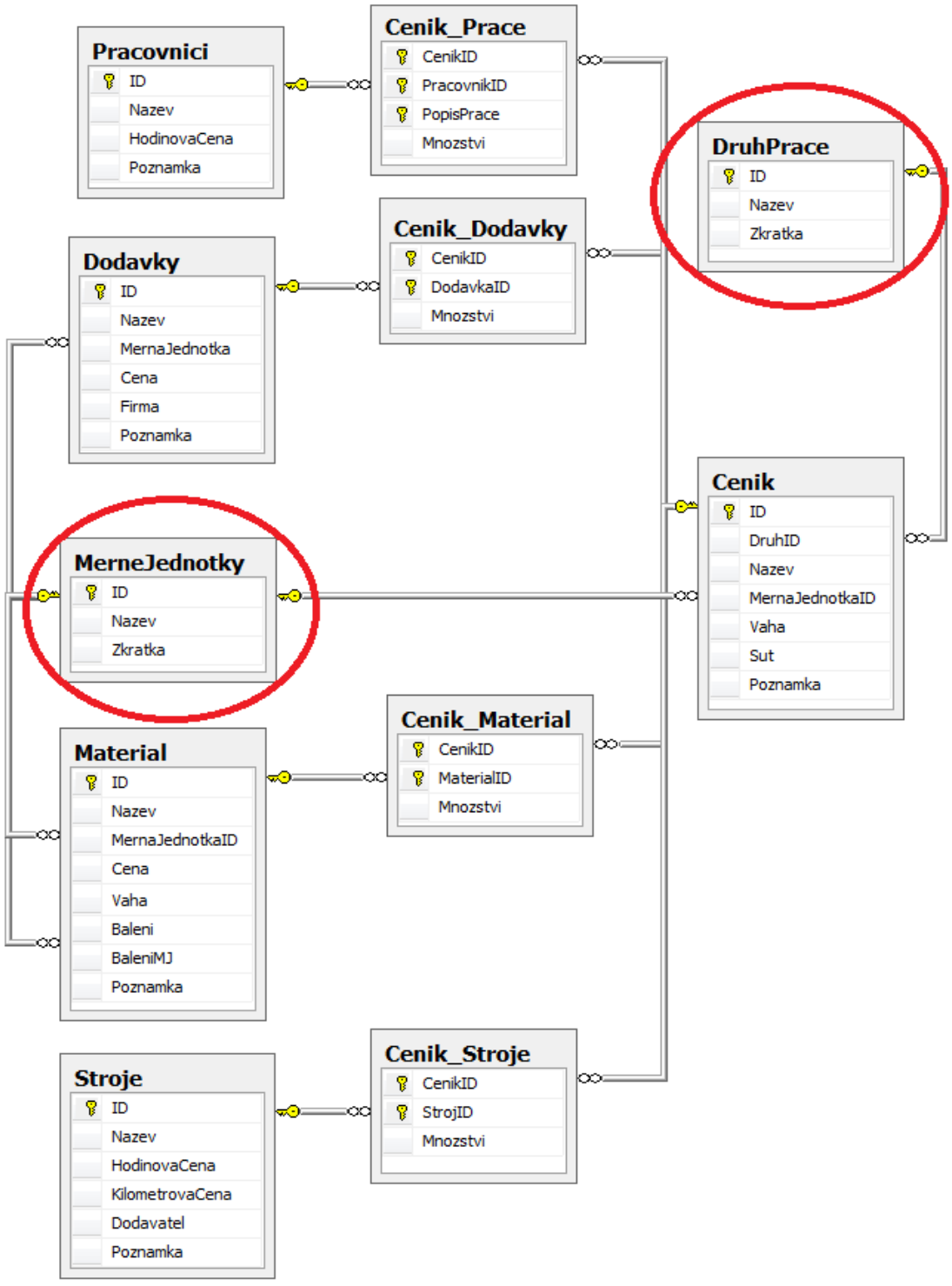
2.3 Databázové schéma



Obr. 2 – Schéma původní databáze

Obrázek číslo 2 znázorňuje původní schéma databáze, kterou má stavební firma k dispozici a se kterou má aplikace KaRaFa pracovat. Databáze obsahuje konkrétní položky, které se nahrávají do aplikace, a rovněž určuje vztahy mezi těmito položkami. Tuto funkcionalitu je potřeba využívat v aplikaci KaRaFa.

Schéma původní databáze a importované databáze zůstává téměř stejné, došlo k vynechání nepotřebných sloupců, které v původní databázi vyžadovaly jiné programy a nejsou potřeba pro aplikaci KaRaFa, a k přejmenování zbylých sloupce. Každá tabulka obsahující položky, které bude možné nahrát z databáze do aplikace, bude obsahovat sloupce `ID`, `Nazev` a `Poznamka` pro obecný přístup k základním údajům o položce. Výsledné schéma importované a upravené databáze obsahující výpočetní logiku je na obrázku číslo 3.



Obr. 3 – Schéma výsledné databáze v BCNF

Schéma původní databáze splňuje podmínky tzv. první normální formy [1]. Žádný sloupec žádné tabulky neobsahuje strukturované hodnoty, ale pouze jednoduché hodnoty. První normální forma nevylučuje redundanci některých dat. Redundantní data se nacházela v původní databázi pouze ve sloupcích pro měrné jednotky a v tabulce `Cenik` ve sloupci `CenDruh`. Proto byly přidány dvě nové tabulky – zakroužkované tabulky na obrázku číslo 2. První přidaná tabulka `MerneJednotky` obsahuje všechny měrné jednotky, jejich názvy a jejich zkratky. Druhá nová tabulka `DruhPrace` obsahuje druhy práce pro Rozpočtové položky z tabulky `Cenik`. Redundantní hodnoty ve sloupcích byly nahrazeny cizími klíči do nové tabulky. Tyto klíče jsou rovněž znázorněny na obrázku číslo 3. Ostatní tabulky a jejich provázání byly ponechány beze změny, kardinality vazeb zůstaly rovněž stejné. Schéma původní databáze v mdb souboru i nová databáze převedená do MSSQL Serveru splňují požadavky tzv. druhé normální formy [2], zároveň i tzv. třetí normální formy [3], i tzv. Boyce-Coddovy normální formy [4].

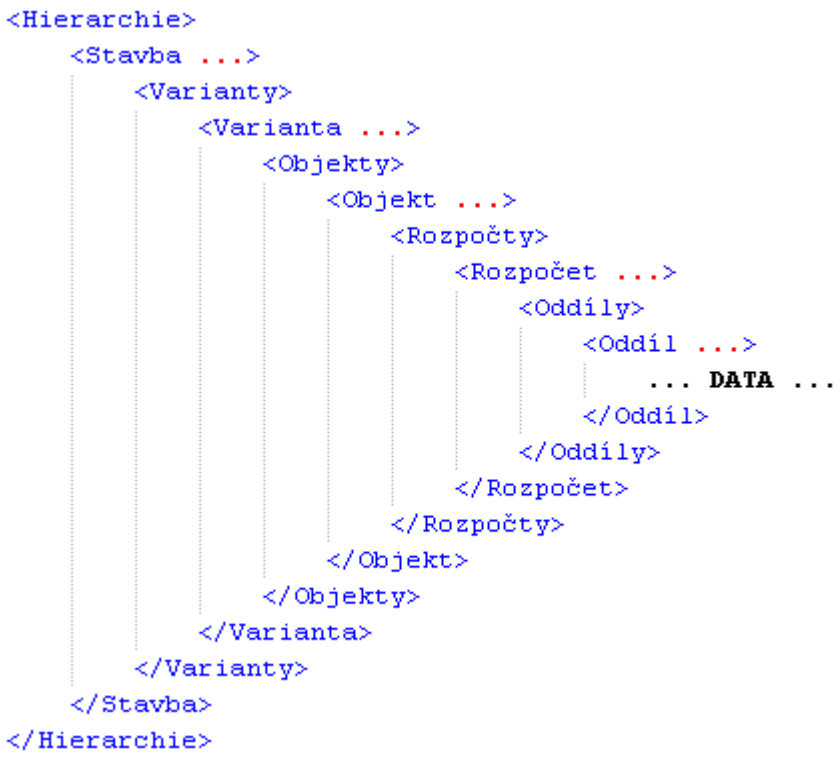
2.4 Ukládání a načítání Staveb

Data reprezentující *Stavby*, se kterými pracuje uživatel, je potřeba ukládat tak, aby je bylo možné sdílet nejen mezi jednotlivými uživateli, ale i mezi jednotlivými uživatelskými stanicemi – například pro práci z kanceláře a domova zároveň. Jedním z řešení je ukládat *Stavby* do databáze, kde jsou již obsaženy položky. To by bylo výhodné zejména pro centralizovanou databázi na firemním serveru. Všichni uživatelé by tak měli aktuální data, musela by se ale vyřešit synchronizace úprav jednoho objektu více uživateli zároveň.

Aplikace KaRaFa předpokládá, že ji bude používat malá firma s jedním uživatelem pro tvorbu *Rozpočtu* a jedním pro správu *Harmonogramu* a fakturaci. Často bude mít tyto úlohy na starosti jen jeden uživatel se všemi právy zároveň. Proto je důležitější řešení, které počítá s více pracovními stanicemi, na kterých se nepracuje zároveň. Jednoduchá přenositelnost dat je cennější než řešení konfliktů při současné práci více uživatelů na jedné *Stavbě*.

I v případě malé firmy je možné ukládat data na lokální instanci MS SQL Serveru na uživatelském počítači. Velkou nevýhodou tohoto řešení je, že pokud uživatel bude potřebovat na jednom zpracování projektu pracovat na více počítačích (v kanceláři a poté z domova), jedinou možností jak přenést data z jednoho pracovního počítače na druhý je provést zálohu databáze na jednom počítači, přenést zálohu na druhý, kde ji ze zálohy obnovit. Toto může být pro uživatele příliš technicky náročné.

Formát *Hierarchie Stavby*, kdy každá *Stavba* má své *Varianty*, tyto *Varianty* se dělí na menší *Objekty* a *Objekty* se skládají z *Rozpočtů* a *Rozpočty* se dělí na *Oddíly*, které obsahují položky pro výpočet *Celkové ceny Stavby*, se dá jednoduše znázornit jako hierarchie pomocí formátu XML – jak ukazuje obrázek číslo 4. Ukládání *Staveb* do XML souborů umožňuje jednodušší přenos mezi více počítači. Zůstává tak potřeba přenášet nějaký soubor mezi počítači, ale jeho použití je o mnoho jednodušší, protože odpadá složitý proces obnovy a zálohy databáze. Pro ukládání a načítání *Staveb* pomocí XML dokumentů je vhodné využít XML serializaci.



Obr. 4 – Zjednodušené XML schéma *Hierarchie stavby*

Využití XML serializace [5], která je určena jmenným prostorem `System.XML.Serialization`, pro ukládání *Staveb* do XML [2] dokumentů umožňuje uložit nejen celou *Stavbu*, ale odděleně i jednotlivé její části, kterými jsou *Varianty*, *Objekty* nebo *Rozpočty*. Toto řešení je možné využít pro „šablonování" výpočtu. Uživatel stavební firmy může vytvořit hrubý návrh *Stavby* pomocí položek potřebných pro každou *Stavbu*, tento návrh si zazálohovat a tuto šablonu může použít jako základ pro každou novou *Stavbu*. Tímto si může ulehčit práci s nahráváním položek, které vyskytují v každé stavbě. Pokud by se tato data *Staveb* ukládala do databáze namísto XML dokumentů, ukládání a nahrávání zvolených částí stavby by bylo složité, protože by byla potřeba vyvinout způsob, jak zálohovat, přenést a obnovit pouze část dat. Díky faktu, že formát XML dokumentů podléhá normě a XML serializace ji dodržuje, je zaručeno validní uložení a načtení dat.

XML serializace není jediným způsobem serializace, který by je možné pro ukládání a zpětné načítání dat použít. Dalším způsobem serializace je binární serializace [6] ze jmenného prostoru `System.Runtime.Serialization`, která serializuje všechny datové položky třídy včetně privátních. Výběr serializovaných položek je možné omezit atributem `[NonSerialized]`, kterým se musí označit všechny datové položky, které není potřeba serializovat. XML serializace serializuje pouze veřejné datové položky třídy, serializace nabízí možnost určit pomocí atributu `[XmlIgnore()]` datové položky třídy, které se nemají serializovat.

Binární serializace sice pracuje rychleji a výsledná data zaberou méně místa oproti XML serializaci, ale výstupem binární serializace jsou binární data, která jsou vhodná pro přenos, ale nejsou pro člověka čitelná. Oproti tomu, výstupem XML serializace je XML soubor, který je pro člověka čitelný. Tato vlastnost se dá

využít k tomu, aby si uživatelé pomocí XML souborů budou moci aplikaci KaRaFa uzpůsobit podle svých potřeb.

2.5 Hluboká kopie

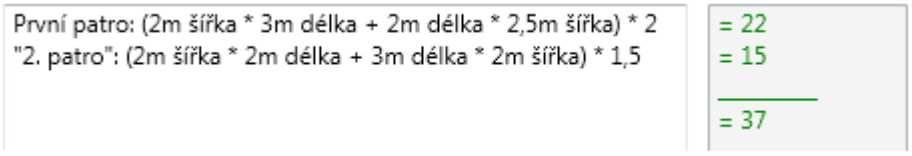
Jeden z požadavků na aplikaci byl, že musí umět vytvořit část stavby (například *Objektu Stavby*) jako kopii již existující části *Hierarchie Stavby*, přičemž změna v jakékoliv zkopírované části se nesmí projevit v původní části a obráceně. Tedy nestačí udělat tzv. mělkou kopii [7], ale je potřeba vytvořit tzv. hlubokou kopii [8] vybrané části *Stavby*. Hluboká kopie zaručuje, že nová část *Stavby* nevznikne pouze zkopírováním ukazatelů na obsažená data, ale že se vytvoří nová data podle předlohy. Změny nové části *Stavby* neovlivní původní část a naopak.

Jedna z možností realizace hluboké kopie je, že v každé třídě reprezentující část *Hierarchie Stavby*, kterou je potřeba takto zkopírovat, bude implementována metoda pro vytvoření své kopie. To není dobré řešení, protože jakákoliv změna v definici položek třídy vyžaduje úpravu kopírovací metody, to je velmi náročné na údržbu.

Aplikace KaRaFa využívá toho, že pro části *Stavby*, které vyžadují hlubokou kopii, je již využita XML serializace. Hluboká kopie objektu se dá jednoduše realizovat vytvořením kopie části *Stavby* pomocí XML serializace do dočasného umístění, poté část *Stavby* z něj zpět deserializovat do nově vytvořené kopie. Toto řešení je nezávislé na změnách v datových položkách. Pokud se změní definice nějaké položky, která patří do *Hierarchie Stavby*, například přidáním nové vlastnosti třídy, musí se nastavit i její serializace, aby se celá třída správně ukládala. Proto vytvoření hluboké kopie tímto způsobem bude vždy fungovat s minimální údržbou.

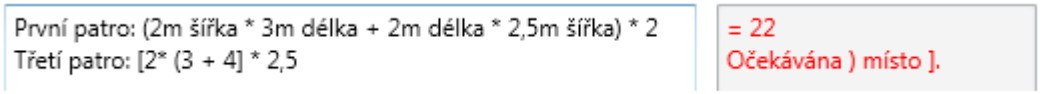
2.6 Výpočet množství

Důležitou součástí aplikace je výpočet množství položky, které je potřeba při stavbě použít. Požadavky na vyhodnocení výrazu jsou takové, že každý řádek výrazu bude vyhodnocen extra jako mezivýsledek. Mezivýsledky se mají zobrazovat na řádcích vedle výrazu. Na konci se mezivýsledky sečtou a dají dohromady hodnotu celkového výrazu. Do pole pro zadávání výrazu musí být možné psát i doprovodný text pro dobrou orientaci ve výrazu, jak ukazuje obrázek číslo 5:



Obr. 5 – Správný výraz

Mechanismus pro výpočet celkového množství z aritmetického výrazu by měl kontrolovat správnost syntaxe zapsaného výrazu a na případné chyby upozornit. Příklad takového výrazu je na následujícím obrázku číslo 6:



Obr. 6 – Špatný výraz

Vyhodnocování aritmetického výrazu se bude řídit podle rekurzivní definice syntaxe aritmetického výrazu, která zachovává prioritu násobení a dělení před sčítáním a odečítáním:

```
Výraz = Člen [ ± Člen ± Člen … ]
Člen = Faktor [ */ Faktor */ Faktor …]
Faktor = Číslo | (Výraz)
```

K vyhodnocení výrazu tímto způsobem stačí přečíst celý text s výrazem zleva doprava pouze jedním průchodem. Na konci čtení vstupu je hodnota výrazu již spočítaná. Pro složité a dlouhé výrazy roste časová a paměťová náročnost způsobená rekurzivní definicí výrazu. Díky vyhodnocování výrazu odděleně po řádcích se dá předpokládat, že výrazy budou spíše jednodušší a kratší, takže rekurze bude způsobovat pouze malou zátěž. Vzhledem k povaze dat, která se budou počítat, lze předpokládat, že nebude potřeba uvažovat složitější matematické operace jako mocniny či logaritmy. Rovněž se dá předpokládat, že ve většině případů bude použito uzávorkování jen do tří úrovní. Uzávorkovanými výrazy lze určit výšku, šířku či délku, které se mezi sebou násobí. Tedy tento jednoduchý algoritmus pro vyhodnocení výpočtu splňuje požadavky aplikace.

Dalšími možnostmi je vybudovat nad výrazem vhodnou stromovou strukturu - například postavit binární strom, který má ve vnitřních uzlech operátory či závorky a v listech číselné hodnoty. Binární strom je vhodnější pro obecnější výrazy s více operátory. Postavení stromu zabere jedno přečtení textu s výrazem a pro vyhodnocení výrazu je potřeba projít celý strom. Výraz je zadán v infixové notaci, k jeho vyhodnocení je možné jej převést na výraz v postfixové notaci. Výraz v postfixové notaci lze vyhodnotit pomocí zásobníku.

2.7 Výpočet celkové ceny

Aplikace KaRaFa musí umět vypočítat *Celkovou cenu Varianty Stavby*. Postup pro vypočítání *Celkové ceny* je přímočarý díky struktuře *Hierarchie Staveb*. *Celková cena Varianty* se počítá jako součet cen všech jejích *Objektů*. *Celková cena* jednoho *Objektu* se počítá jako součet cen všech jeho *Rozpočtů*. *Celková cena* jednoho *Rozpočtu* se počítá jako součet cen všech jeho *Oddílů*. *Celková cena* jednoho *Oddílu* se počítá jako součet cen všech jeho naimportovaných *Rozpočtových položek*. Cena *Rozpočtové položky* se počítá jako *Jednotková cena* položky vynásobená její *Výměrou*. *Jednotková cena Rozpočtové položky* se počítá jako součet *Jednotkových cen* všech jejích *Rozborových položek* - *Dodávka*, *Materiál*, *Stroj*, *Práce*, jak ukazuje následující tabulka:

Jedním řešením jak vypočítat *Celkovou cenu Varianty* je provést výpočet na místě, kde je *Celková cena* potřeba. Tedy provést průchod celou hierarchií pomocí mnoha vnořených `foreach` cyklů. Toto řešení bude fungovat a bude nejjednodušší na implementaci. Zároveň je nejméně vhodné, protože je toto řešení použité jen na jednom místě, což není užitečné pro případná pozdější rozšíření.

Celkovou cenu je vhodné počítat nejen pro *Variantu*, ale i pro ostatní části *Hierarchie Stavby*. Vzhledem ke struktuře *Hierarchie* a logice výpočtu se jako řešení tohoto problému nabízí použití návrhového vzoru Visitor [9]. V každé

třídě reprezentující část *Stavby* bude implementována metoda pro získání její *Celkové ceny*. *Celkovou cenu* tak bude možné zjistit nejen pro *Variantu*, ale i pro *Rozpočet* nebo jen pro importovanou *Rozpočtovou položku* pouhým zavoláním metody pro výpočet *Celkové ceny*. Použití tohoto návrhového vzoru výrazně zpřehlední kód, a umožní použít výpočet *Celkové ceny* z mnoha míst aplikace například v přehledech.

Cenu je potřeba vypočítat a ukládat přesně, nesmí dojít ke ztrátám z důvodu nepřesné reprezentace čísel, proto bude cena reprezentována datovým typem `Decimal`, ne `Double`. `Decimal` ukládá desetinná čísla přesně, `Double` desetinná čísla pouze aproximuje, proto je nutné použít `Decimal`.

2.8 Harmonogram

Požadavkem na aplikaci KaRaFa je vytvořit jednoduchý *Harmonogram*. *Harmonogram* bude sdružovat naimportované *Rozpočtové položky* do *Etap*. *Etapy* pak bude možné organizovat podle potřeby. Navrhované řešení ze strany stavební firmy bylo, že *Rozpočtář* předá stavbyvedoucímu hotový *Rozpočet* s naimportovanými *Rozpočtovými položkami*. *Stavbyvedoucí* každou položku ručně zařadí do *Etapy*, do které by měla patřit. Toto řešení je nevhodné, protože hotové *Rozpočty* mohou obsahovat i tisíce položek a projít ruční řazení každé položky bude trvat příliš dlouho. Zadavatelé navrhli toto řešení, protože předpokládali, že *Harmonogram* nebude používán pro každou *Stavbu*. To neřeší problém procházení každé položky.

Lepší řešení je, že *Harmonogram* se bude konstruovat průběžně s vytvářením *Rozpočtu*. Pokaždé když *Rozpočtář* naimportuje *Rozpočtovou položku* do nějakého *Oddílu*, tak se té položce přiřadí výchozí *Etapa* daného *Oddílu*. *Rozpočtář* v tento moment může i nemusí *Etapu* změnit.

Oddíly mají nastaveny takové *Etapy*, že většina položek, které se importují do daného *Oddílu*, patří i logicky do té *Etapy*, proto je konstrukce *Harmonogramu* při importu *Rozpočtových položek* do *Oddílů* časově nenáročné oproti původně navrhovanému řešení.

2.9 Ukládání hesel

Aplikace bude umožňovat přihlašování uživatelů, tabulka s uživateli a jejich hesly bude uložena v databázi. Kvůli bezpečnosti je vhodné využít nějaký algoritmus, který transformuje hesel na jejich otisky, které se uloží do databáze místo uživatelských hesel. Algoritmus pro transformaci by měl být deterministický, aby se jedno heslo vždy transformovalo na stejný otisk, jinak by hesla nešla porovnávat.

Uživatelský přístup je málo důležitá vlastnost aplikace, proto není potřeba použít sofistikované algoritmy pro transformaci, jako jsou například SHA-1 [10] nebo MD5 [11]. Autor aplikace tento fakt využil k tomu, aby navrhl vlastní algoritmus pro transformaci hesel, který deterministicky transformuje uživatelská hesla na jejich otisky.

2.10 Lokalizace

Aplikace KaRaFa je zaměřena primárně na české stavební firmy, proto bude její grafické rozhraní lokalizováno do češtiny. Pro lokalizované texty je výhodné

využít slovník `Resources`, na rozdíl od použití lokalizovaných textů přímo v kódu. Pro případ pozdějšího zavedení dalších lokalizací stačí v případě použití slovníku `Resources` vytvořit slovník s novou lokalizací. Ve druhém případě by byla potřeba vyhledat lokalizované texty ve zdrojovém kódu celé aplikace a přeložit je.

Pojmenování tabulek, sloupců a pohledů v databázi, rovněž i pojmenování elementů a atributů ve schématech XML souborů pro ukládání *Staveb* a *Nastavení* bude také zaměřeno pro české uživatele. Tyto soubory jsou dostupné všem uživatelům, každý je může editovat. Toto je výhodné pro soubor s nastavením. Každému instanci aplikace je potřeba nastavit připojení do databáze a cestu k výchozímu souboru s uloženými *Stavbami*, která se liší na každé pracovní stanici.

2.11 Výstup do PDF

Výstupy aplikace KaRaFa by měly být nejen faktury, ale i různé přehledy a další tiskové sestavy, které budou generovány do PDF [3] souborů. Formát PDF je softwarově podporován na všech platformách, proto při generování faktur pro *Investora Stavby* odpadá závislost na jeho operačním systému. Existuje několik knihoven ke stažení zdarma [12], která umožňují vytvářet PDF soubory. Výstupní faktury, přehledy a tiskové sestavy budou především textové dokumenty, každý dokument bude mít specifické nároky na rozvržení textu na stránce. Budoucí požadavky na aplikaci, by mohly spočívat v přidání loga firmy do faktury či doplnit zobrazit přehledy pomocí tabulky. Všechny tyto knihovny splňují současné i případné budoucí požadavky aplikace Karafa. PDF dokumenty se budou generovat pomocí knihovny PdfSharp [13], která rovněž splňuje všechny požadavky, navíc je doplněna o přehlednou dokumentaci včetně ukázkových příkladů použití knihovny.

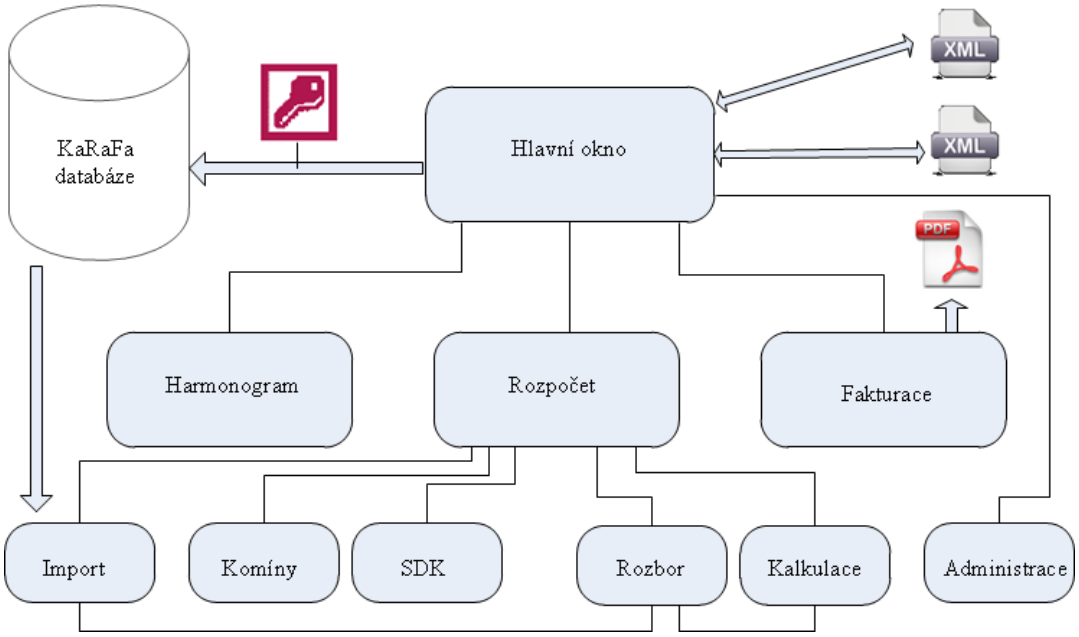
3. Implementace

3.1 Dokumentace

Dokumentaci vygenerovaná ze zdrojových kódů a Uživatelský manuál se nacházejí na přiloženém DVD [1].

3.2 Popis aplikace

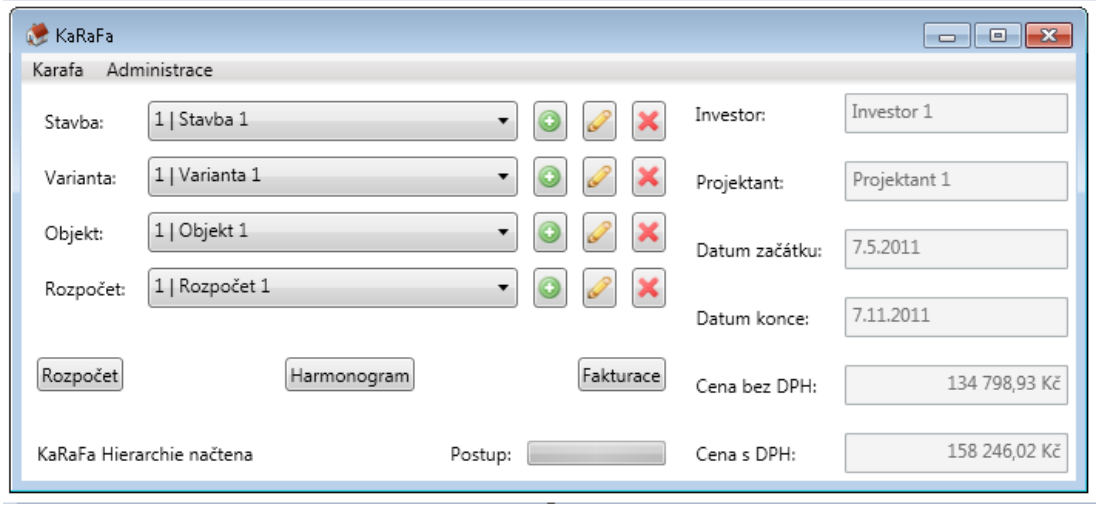
3.2.1 Základní schéma



Obr. 7 – Základní přehled aplikace

Schéma na obrázku číslo 7 znázorňuje základní interakce mezi moduly aplikace. Šipky znázorňují výměnu dat mezi aplikací a externími zdroji, spoje znázorňují interakci mezi moduly. Modulem rozumíme část aplikace, která implementuje část procesu realizace *Stavby*. Mezi výměnu dat patří ukládání a načítání *Staveb* a Nastavení z XML souborů, generování faktur do PDF souborů, aktualizaci databáze z MDB souboru a import položek z databáze do aplikace.

Každá třída daného modulu se nachází ve jmenném prostoru učeným modulem, do kterého třída patří. Každý modul má svůj vlastní jmenný prostor, kterému odpovídá adresářová struktura projektu celé aplikace. Výchozím jmenný prostor pro moduly je `Karafa.Modules`, který se dále větví podle modulu. Každý modul je ovládán ze svého grafického okna, všechna okna se nachází ve jmenném prostoru `Karafa.GUI`.



Obr. 8 – Hlavní okno

Obrázek číslo 8 zobrazuje Hlavní okno reprezentované třídou `Karafa.KarafaMainWindow`, které tvoří vstupní bod aplikace. Během jeho inicializace se načte Nastavení a uložené *Stavby* z XML souborů. Toto načítání dat probíhá na pozadí načítání Hlavního okna. Hlavní okno umožňuje uživateli vybrat *Stavbu*, její *Variantu*, její *Objekt* a jeho *Rozpočet*. Rovněž zobrazuje informace k vybrané *Stavbě* o jejím *Investorovi* či *Projektantovi*, datech začátku a konce *Stavby* a *Celkovou cenu* vybrané *Varianty* bez DPH a s DPH. Pokud je vybrán nějaký *Rozpočet*, může uživatel kliknout na tlačítko pro zobrazení grafického okna pro editaci vybraného *Rozpočtu*. Tlačítka Harmonogram a Fakturace jsou přístupná, pokud je vybrána nějaká *Stavba*.

Modul Rozpočet slouží uživateli k vytvoření a editaci *Rozpočtu*. Nový *Rozpočet* se vytvoří s výchozí sadou *Oddílů*, která se načte z databáze. Úprava a práce s *Rozpočtem* spočívá v úpravě seznamu *Oddílů*, dále importu nových *Rozpočtových položek* z databáze, úpravě jejich *Rozboru* a výpočtu jejich *Výměry*. V rámci modulu Rozpočet jsou rovněž přístupné *Automatické kalkulace*. Modul pro *Automatickou kalkulaci*, který zjednodušuje *Kalkulace* sádrokartonových a komínových konstrukcí, je unikátní mezi všemi ostatními aplikacemi zabývajícími se zpracováním stavebních *Rozpočtů*.

K úpravě *Rozboru Rozpočtových položek* slouží grafické okno, ve kterém aplikace zobrazí *Rozbor Rozpočtové položky* – seznam položek *Dodávka*, *Materiál*, *Stroj* nebo *Práce*, které jsou s *Rozpočtovou položkou* svázány. Okno pro *Rozbor Rozpočtové položky* umožňuje přidávat nové položky do jejího *Rozboru* importováním z databáze, nebo mazat existující položky z *Rozboru*. K úpravě vlastností vybrané položky - *Rozpočtové* nebo *Rozborové* - slouží grafické okno pro zobrazení detailu položky. U *Rozpočtových položek* se v tomto okně určuje její *Výměra*, a to buď přímo, nebo zadáním výrazu pro výpočet *Výměry*.

Modul Harmonogram slouží uživateli k úpravě a kontrole průběhu *Stavby*. *Harmonogram* je tvořen *Etapami*. Každý *Objekt* obsahuje sadu *Etap*. Každá *Rozpočtová položka* je přiřazena do nějaké *Etapy*. Rozdělení položek do *Etap* dává uživateli přehled o *Celkové ceně* jednotlivých *Etap*.

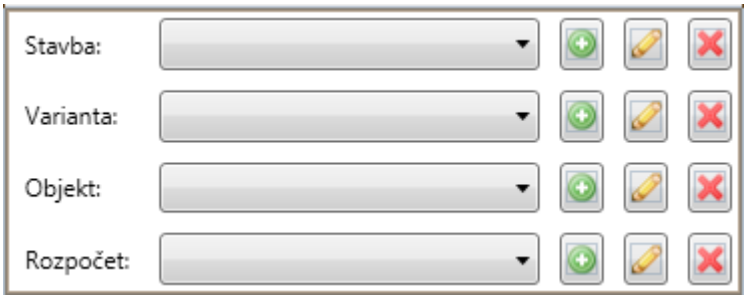
Modul Fakturace umožňuje uživateli vygenerovat faktury pro *Investora Stavby*. Uživatel vybere *Rozpočtové položky* a určí jejich množství, které bude stavební firma fakturovat. *Stavba* se jejímu *Investorovi* nefakturuje celá najednou, její fakturace probíhá průběžně, proto se vystavené faktury budou ukládat ke *Stavbám*.

3.2.2 Spojení s databází

Připojení k databázi zajišťuje třída `Karafa.Database.DatabaseConnection`, která otevírá databázové spojení dané, provádí dotazy do databáze a vrací jejich výsledky. Pokud se spojení nepodaří otevřít z daných přístupových údajů, uživatel je požádán zadat platné údaje. Třída `DatabaseConnection` umožňuje spustit libovolný SQL [4] dotaz, nebo využít metod pro vygenerování dotazů doplněním předdefinovaných šablon pro SLQ dotazy. Šablony jsou zobrazeny v následující tabulce. Pro `SELECT` dotaz platí, že pokud nejsou určeny sloupce, vyberou se všechny sloupce z dané tabulky nebo pohledu, klauzule `WHERE` a `ORDER BY` jsou nepovinné. `INSERT` dotaz vyžaduje, aby počet sloupců odpovídal počtu hodnot, aby dotaz mohl být úspěšně vytvořen. `UPDATE` dotaz vyžaduje seznam sloupců, kterým se mají změnit hodnoty a seznam nových hodnot. Z těchto dvou seznamů se vytvoří definice změn. Jednotlivé hodnoty v seznam hodnot v `INSERT` a `UPDATE` dotazech musí být ve tvaru, aby hodnotu bylo možné uložit do databáze. Tento tvar je dán datovým typem hodnoty.

`SELECT {sloupce} FROM {tabulka/pohled} [WHERE {podmínka}] [ORDER BY {sloupce}]`
`INSERT INTO {tabulka} ({sloupce}) VALUES ({hodnoty})`
`UPDATE {tabulka} SET {definice změn} [WHERE {podmínka}]`
`DELETE FROM {tabulka} [WHERE {podmínka}]`

3.2.3 Navigace stavbou

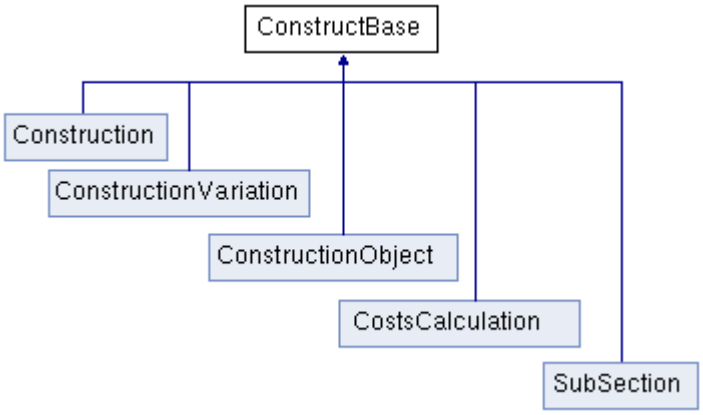


Obr. 9 – Kontrolka pro navigaci stavbou

Kontrolka na obrázku číslo 9, která se nachází v Hlavním okně a v oknech pro Harmonogram, Rozpočet a Fakturaci, umožňuje uživateli vybrat pomocí `ComboBoxů` aktuální *Stavbu*, její *Variantu*, její *Objekt* a jeho *Rozpočet*, zároveň si ukládá informace o naposledy vybraných hodnotách, aby uživatel při novém načtení aplikace nemusel hodnoty vybírat znovu. Tímto výběrem určuje uživatel kontext, ve kterém pracuje. Tlačítka vedle `ComboBoxů` slouží po řadě pro vytvoření nové, editaci či smazání vybrané části *Stavby*. Kontrolka obsahuje události, které jsou spuštěny pokaždé, když je vybrána nějaká položka, tím umožňuje na tyto události reagovat.

3.2.4 Hierarchie staveb

Třída `Karafa.ConstructionHierarchy.ConstructionSet` reprezentuje seznam všech *Staveb* a zároveň tvoří kořenový element v XML souboru, do kterého aplikace zálohuje celý seznam *Staveb* se všemi jejich daty. Hlavní okno má instanci této třídy obsahující všechny nahrané *Stavby* ve spojovém seznamu `ConstructionList`.



Obr. 10 – Schéma dědičnosti položek Hierarchie staveb

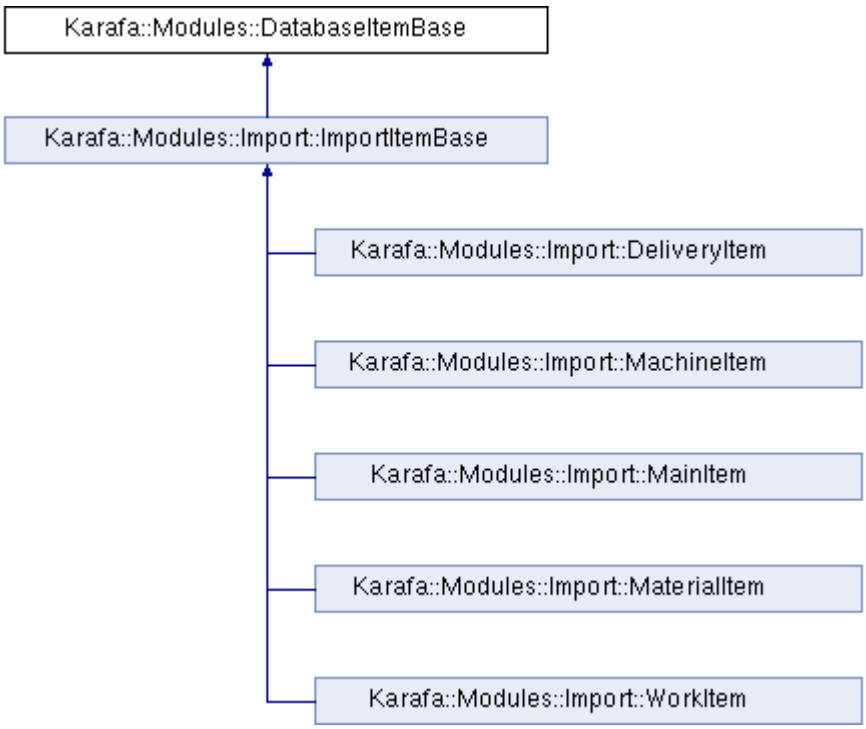
Obrázek číslo 10 ukazuje, reprezentaci *Hierarchie Staveb*. Všechny třídy pro reprezentaci části *Stavby* dědí od jedné obecné třídy `Karafa.ConstructionHierarchy.ConstructBase`. Tato obecná třída obsahuje vlastnosti společné všem částem *Stavby* – jmenovitě vlastnosti `ID`, `Name` a `Note` pro práci s *ID*, *Názvem* a *Poznámkou* částí *Staveb*. Rovněž předefinovaná `ToString()` metodu, aby vracela text tvaru „ID | Název“, který se zobrazí v `ComboBoxech` kontrolky pro navigaci *Stavbou* z obrázku číslo 8. Tato třída rovněž obsahuje virtuální metodu `CountPrices(…)`, kterou musí všechny položky *Hierarchie* předefinovat podle svých potřeb. Metoda slouží pro výpočet *Celkové ceny* s DPH a bez DPH dané části *Stavby*.

Částmi *Hierarchie Stavby* jsou *Stavba* (`Construction`), *Varianta* (`ConstructionVariation`), *Objekt* (`ConstructionObject`), *Rozpočet* (`CostsCalculation`) a *Oddíl* (`SubSection`). Všechny tyto třídy jsou definovány ve jmenném prostoru `Karafa.ConstructionHierarchy`. *Stavba* obsahuje data se začátkem a koncem, informace o *Investorovi* a *Projektantovi*, seznam faktur vytvořených k této *Stavbě* a seznam *Variant*. *Varianta* obsahuje ID nadřazené *Stavby*, seznam *Objektů Varianty*, a definuje metodu `CountPrices(…)` pro výpočet cen *Varianty* jako součet cen všech svých *Objektů*. *Objekt* obsahuje ID nadřazené *Varianty*, seznam *Rozpočtů* a *Etap* a definuje metodu pro výpočet cen *Objektu* jako součet cen všech svých *Rozpočtů*. *Rozpočet* obsahuje ID nadřazeného *Objektu* a seznam *Oddílů*. Cena Rozpočtu se počítá jako součet cen všech jeho *Oddílů*. Každý *Oddíl* obsahuje seznam všech *Rozpočtových položek* nahraných z databáze. Cena Oddílu se počítá jako součet *Celkových cen* všech jeho *Rozpočtových položek*.

3.2.5 Reprezentace položek nahraných z databáze

Nejobecnější třídou reprezentující položku načtenou z databáze je třída `Karafa.Modules.DatabaseItemBase`. Od této třídy dědí všechny konkrétní třídy pro reprezentaci položek nahraných z databáze. Má dva konstruktory. Bezparametrický konstruktor, který slouží při deserializaci položky z XML, a

konstruktor pro načítání položek imporováním z databáze s jedním parametrem `System.Data.DataRow` obsahujícím definici položky v databázi, ze které se načtou všechny vlastnosti položky. Tato obecná třída využívá faktu, že každá tabulka v databázi, ze které se načítají položky do aplikace, má sloupce `ID` a `Nazev`. Proto tato třída má vlastnosti *ID* (`ID`) a Název (`Name`).



Obr. 11 - Schéma dědičnosti databázových položek

Obecnou třídou pro reprezentaci položky, která se importuje do Rozpočtů v rámci modulu Import je třída `ImportItemBase`, která dědí od třídy `DatabaseItemBase`. Třída `ImportItemBase` obsahuje navíc vlastnosti *Výměra* (`Quantity`), Měrná jednotka (`MeasureUnit`), *Spotřeba* (`Usage`), Měrná jednotka spotřeby (`UsageMeasureUnit`), Cena za měrnou jednotku (`UnitPrice`) a Poznámka (`Note`). Od třídy `ImportItemBase` dědí třídy pro reprezentaci *Rozpočtové položky* (`MainItem`), která se nahrává z databáze do *Oddílů*, a třídy pro reprezentaci *Rozborových položek* typu *Materiál* (`MaterialItem`), *Práce* (`WorkItem`), *Stroj* (`MachineItem`) a *Dodávka* (`DeliveryItem`), jak je znázorněno na obrázku číslo 11.

Rozpočtová položka se nahrává z tabulky `Cenik`, její *Rozborové položky* z tabulek `Stroje`, `Material`, `Dodavky` a `Pracovnici`. Informace pro definici *Rozboru Rozpočtové položky* jsou uloženy v tabulkách `Cenik_Prace`, `Cenik_Dodavky`, `Cenik_Material` a `Cenik_Stroje`. Záznamy v těchto tabulkách určují, jaké *Rozborové položky* tvoří *Rozbor* jaké *Rozpočtové položky*. Primárním klíčem těchto vazebních tabulek je vždy dvojice (*Rozpočtová položka* – navázaná *Rozborová položka*). V případě *Práce* je ještě součástí primárního klíče Popis práce. V praxi to znamená, že v rámci jedné *Rozpočtové položky* – Výměna oken – může mít zedník více prací – Vysekat stará okna a Začistit omítku u nových oken. Vlastní výměnu oken provádí truhlář. Tyto 3 práce se objeví v *Rozboru Rozpočtové položky* – Výměna oken.

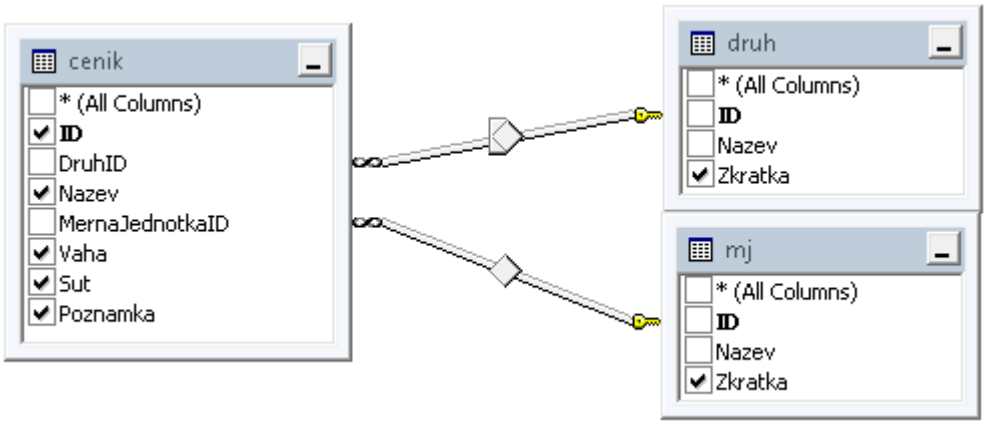
Celková cena *Rozpočtové položky* se počítá jako její *Jednotková cena* vynásobená její *Výměrou*. *Výměru* položky zadává uživatel ručně nebo pomocí výrazu v okně pro detail *Rozpočtové položky*. *Jednotková cena Rozpočtové položky* udává, kolik stojí realizace její jedné měrně jednotky a počítá se z jejího *Rozboru*. Každá *Rozborová položka* má určenou svou *Jednotkovou cenu* a *Spotřebu*, obě tyto hodnoty může uživatel měnit. *Jednotková cena Rozborové položky* udává cenu jedné její měrné jednotky, *Spotřeba* udává, jaké množství *Rozborové položky* realizuje jednu měrnou jednotku nadřazené *Rozpočtové položky*. *Jednotková cena Rozpočtové položky* se počítá jako součet součinů *Jednotkových cen* a *Spotřeb* všech svých *Rozborových položek*. Pokud je *Rozborovou položkou* položka typu *Stroj*, započítává se ještě cena za dopravy stroje – počet kilometrů * cena za kilometr * 2 (za cestu na stavbu a zpět) / Množství *Rozpočtové položky*. *Celková cena Rozpočtové položky* se počítá jako její Jednotková cena * Množství, proto se cena dopravy dělí množstvím, aby celková cena vyšla správně. Diagram výpočtu *Jednotkové ceny*:

Položka:	**Jednotková cena (určena v databázi)**	**Spotřeba (Určí uživatel)**	**Jednotková cena * Spotřeba**
Materiál	150,- Kč / měrnou jednotku	0,5	75,- Kč
Práce	250,- Kč / hod	0,8	200,- Kč
Dodávka	200,- Kč / měrnou jednotku	1	200,- Kč
Stroj	500,- Kč / hod	0,1	50,-Kč
Jednotková cena Rozpočtové položky:			525,- Kč + Doprava stroje

3.2.6 Databázové pohledy

Všechny položky jsou uloženy v příslušných tabulkách v databázi, přesto se definice položek nenačítají z tabulek přímo, ale z databázových pohledů usnadňující import. Pohledy nahrazují hodnoty odkázané cizími klíči jejich skutečnými hodnotami, tímto ulehčují nahrávání položek z databáze. Cizími klíči jsou navázány měrně jednotky položek. K nahrání stačí jen jeden jednoduchý SELECT dotaz z příslušného pohledu místo složitého dotazu ze spojení několika tabulek. O výsledek dotazu se stará definice pohledů, proto je vhodné importovat položky přes ně. Výhoda tohoto řešení je, že pokud se změní struktura databáze, stačí poupravit pohledy, aby stále vyhovovaly mechanismu pro import položek z databáze bez nutnosti úpravy kódu.

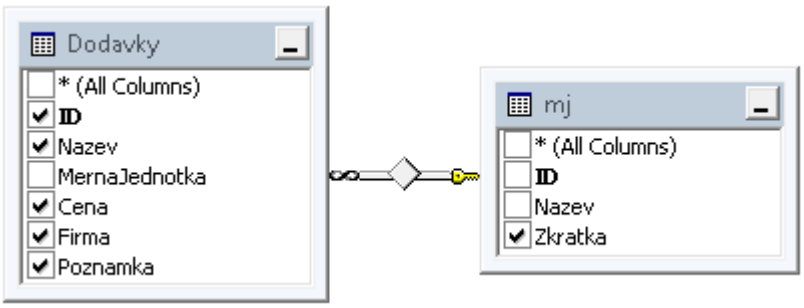
3.2.6.1 vw_Cenik



Obr. 12 – Pohled vw_Cenik

Pohled na obrázku číslo 12 nahrazuje v tabulce `Cenik` hodnotu měrné jednotky a druhu práce. Druh práce nemusí být určen, proto se tabulky propojují příkazem LEFT JOIN.

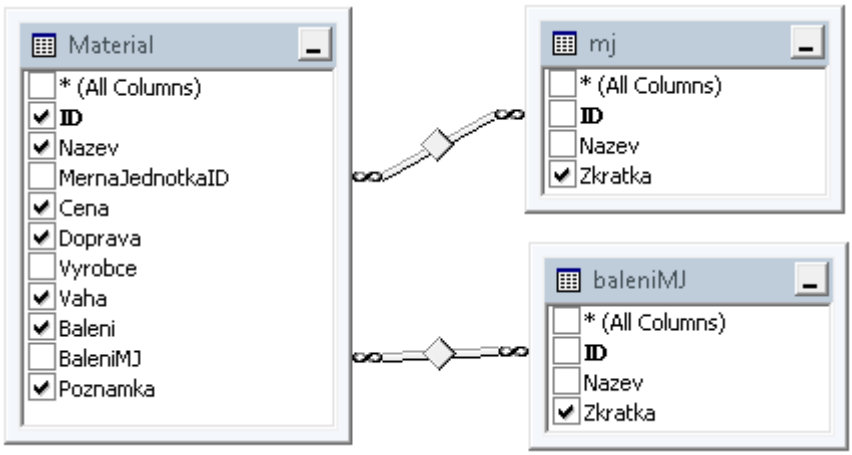
3.2.6.2 vw_Dodavky



Obr. 13 – Pohled vw_Dodavky

Pohled na obrázku číslo 13 nahrazuje v tabulce `Dodavky` hodnotu měrné jednotky.

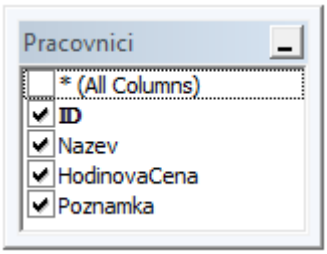
3.2.6.3 vw_Material



Obr. 14 – Pohled vw_Material

Pohled na obrázku číslo 14 nahrazuje v tabulce `Material` hodnoty měrných jednotek pro materiál i balení, ve kterém je materiál dodáván.

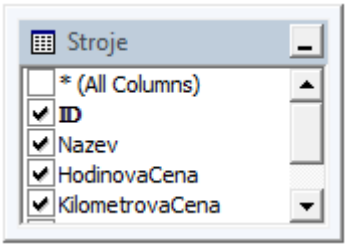
3.2.6.4 vw_Prace



Obr. 15 – Pohled vw_Prace

Pohled na obrázku číslo 15 obsahuje všechny sloupce tabulky `Prace` a nastavuje ‚hod' jako měrnou jednotku *Práce*.

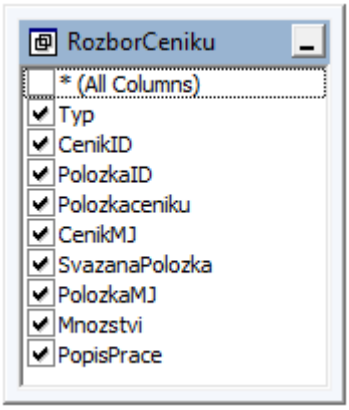
3.2.6.5 vw_Stroje



Obr. 16 – Pohled vw_Stroje

Pohled na obrázku číslo 16 obsahuje všechny sloupce tabulky Stroje a nastavuje ‚hod' jako měrnou jednotku a ‚km' jako dodatečnou měrnou jednotku.

3.2.6.6 vw_RozborCeniku



Obr. 17 – Pohled vw_RozborCeniku

Pohled na obrázku 17 obsahuje seznam všech *Rozpočtových položek* a jejich navázaných *Rozborových položek*. K získání *Rozboru* dané položky *Ceníku* (podle `ID` *Rozpočtové položky*) slouží SQL dotaz:

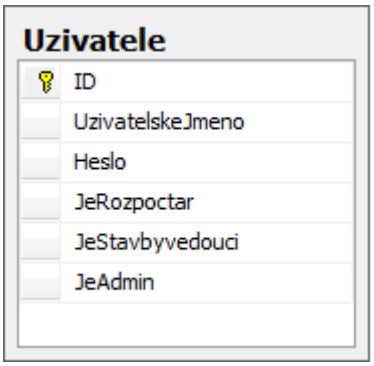
```
SELECT [Typ],[CenikID],[PolozkaID],[PolozkaCeniku],[CenikMJ]
      ,[SvazanaPolozka],[PolozkaMJ],[Mnozstvi],[PopisPrace]
FROM [dbo].[vw_RozborCeniku]
WHERE [CenikID] = {ID}
```

Dotaz vrací všechny položky tvořící *Rozbor Rozpočtové položky*. Hodnota sloupce `Typ` obsahuje jeden z textů: Ceník, Materiál, Práce, Stroj, Dodávka; podle kterého se při importu rozhoduje typ položky). Sloupec `PolozkaID` obsahuje

`ID` *Rozborové položky* svázané s *Rozpočtovou položkou*. Podle `ID` je pak možné příslušnou *Rozborovou položku* vyhledat a importovat.

3.2.7 Uživatelský přístup

Aplikace KaRaFa umožňuje přístup více uživatelům zároveň, rovněž lze pomocí uživatelských práv omezit přístup uživatelů k různým funkcionalitám aplikace – modulům. Uživatel s právem *Administrátor* má přístup k modulu Administrace a Aktualizace databáze. Uživatel s právem *Rozpočtář* má přístup k modulu Rozpočet, ke kterému patří import položek a jejich následná *Kalkulace*. Přes modul Rozpočet jsou rovněž dostupné *Automatické kalkulace*. Uživatel s právem *Stavbyvedoucí* má přístup k modulům Harmonogram a Fakturace.



Obr. 18- Tabulka pro uživatele

Uživatelé se budou ukládat do databázové tabulky `Uzivatele` s jejich hesly a nastavenými právy. Je to jediná výjimka porušující pravidlo, že aplikace KaRaFa nebude do databáze ukládat žádná data. Tabulka pro uživatele neexistovala v původní databázi, byla vytvořená nová tabulka, její schéma je na obrázku číslo 18. Nad touto tabulkou existuje i pohled vw_Uzivatele, který obsahuje všechny sloupce tabulky Uzivatele. Třída reprezentující uživatele je `Karafa.Login.User`.

Aplikace KaRaFa počítá s jedním uživatelem, který bude mít všechna práva, proto současná implementace uživatelských práv pouze simuluje přístup různých uživatelů. Odděluje přístup k modulům podle práv, ale uživatelé mohou editovat XML soubor s uloženými *Stavbami* a soubor s Nastavením. Pokud by uživatel editoval soubory ručně, riskoval by, že poruší jejich strukturu a tím by narušil chod aplikace KaRaFa. Pokud tedy uživatelé nebudou soubory editovat ručně, popřípadě je bude editovat uživatel, který dokáže soubory upravit správně, tak bude simulace uživatelských práv pracovat dobře.

Příkladem, kdy bude potřeba zasáhnout do XML souboru je, když bude aplikace KaRaFa zavedena na více stanicích s různými instancemi MSSQL Serveru. V souboru pro Nastavení bude potřeba nastavit správné přístupové údaje do databáze a cestu k výchozímu souboru s uloženými stavbami. Ruční nastavení údajů a cesty k souboru není bezpodmínečně nutné V případě špatných přístupových údajů je uživatel dotázán o správné, pokud jsou nové přístupové údaje správné, uloží se do nastavení.

3.2.8 Přihlašování uživatelů

Uživatelé se mohou přihlašovat a odhlašovat přes menu umístěném na Hlavním okně aplikace KaRaFa. O korektní přihlašování a odhlašování se stará třída `Karafa.Login.LoginProcess`, která se pokusí ověřit dané přihlašovací údaje oproti databázi. Pokud jsou údaje platné, třída vyvolá událost `userLoginSucces`, pokud údaje platné nejsou, vyvolá se událost `userLoginFail`. Na obě události reaguje Hlavní okno, které události obslouží a načte správná nastavení v závislosti na právech přihlášeného uživatele.

3.2.9 Administrace

Administrační nastavení programu je přístupné z menu Administrace, které je viditelné pouze pro uživatele s právem „Administrátor". Nastavení je reprezentováno třídou `Karafa.Settings.KarafaSettings` a ukládá se do souboru Nastaveni.xml, ze kterého se při spouštění aplikace rovněž načítá. Struktura souboru je naznačena na obrázku číslo 19. Nastavení ukládá databázový přístupové údaje k databázi na MSSQL serveru, výchozí hodnotu DPH, cestu k souboru s uloženými daty aplikace, údaje o stavební firmě potřebné pro fakturaci, patičku a výchozí dobu splatnosti faktury.

```
<KaRaFaNastaveni>
  <PřipojeníDoDatabáze />
  <DPH />
  <KaRaFaSoubor />
  <Společnost ... />
  <PatičkaFaktury />
  <DobaSplatnostiFaktury />
</KaRaFaNastaveni>
```

Obr. 19- XML schéma pro nastavení aplikace KaRaFa

Dále administrační aplikace umožňuje spravovat uživatele. Definice uživatelů se načtou z databáze, zobrazí se v aplikaci, kde je možné je mazat nebo editovat včetně změny hesla a nastavení práv, rovněž je možné přidat nové uživatele. Definice uživatelů se uloží zpět do tabulky `Uzivatele` v databázi. Pro použití nově nastavených práv je potřeba, aby se uživatel znovu přihlásil.

3.2.10 Aktualizace databáze

Tento modul je rovněž přístupný pouze uživateli s právem *Administrátor*. Před aktualizací je potřeba, aby uživatel vybral MDB soubor s novými daty pro aktualizaci databáze. Struktura MDB souboru odpovídá struktuře původní databáze v MDB souboru, kterou znázorňuje obrázek číslo 1. Data se aktualizují v závislosti na primárním klíči položek. Pokud v příslušné tabulce již existuje položka se shodným klíčem, proběhne její aktualizace, jinak bude nová položka vložena do databáze. Proces aktualizace pracuje s tabulkami `Cenik`, `Dodavky`, `Material`, `Pracovnici`, `Stroje`, `Cenik_Dodavky`, `Cenik_Material`, `Cenik_Prace`, `Cenik_Stroje` a `Vypocet_Materialy`. Proces počítá s tím, že aktualizační soubor může aktualizovat obsahovat nejen malé množství nových či pozměněných položek, ale i celou databázi najednou, proto běží na pozadí v novém vlákně, aby uživatel mohl pokračovat v práci, zatímco se databáze aktualizuje.

Proces aktualizace rovněž upravuje načítaná data, aby nedocházelo ke zbytečným chybám při vkládání dat. O zachování integrity dat se stará generická metoda `ProcessColumnValue<typ>(hodnota)`, která přijímá jako parametr hodnotu a její datový typ. Metoda se volá pro každou hodnotu z mdb souboru, která se bude ukládat do KaRaFa databáze, zpracuje každou hodnotu a jako výsledek vrátí správný výraz obsahující hodnotu, aby se hodnota uložila bezpečně do databáze. Výsledný výraz se liší podle typu hodnoty:

- `string` - `string.Format("RTRIM(LTRIM('{0}'))", hodnota);`
- `int` nebo `double` - pokud je hodnota platné číslo, tak se vrací hodnota, jinak se vrací `NULL`, jako desetinný oddělovač slouží tečka, ne čárka.
- Ostatní typy není potřeba uvažovat

3.2.11 Fakturace

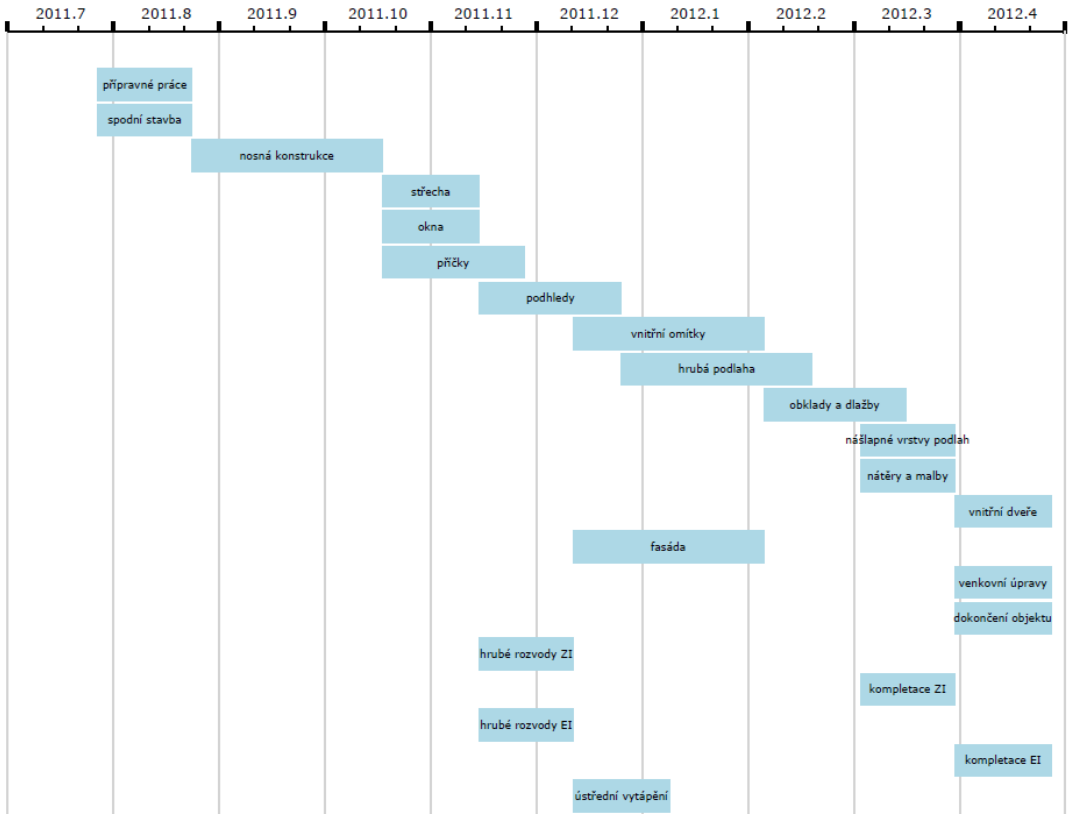
Modul se spouští kliknutím na tlačítko Fakturace na hlavním okně. Poté se uživateli zobrazí okno, kde může určit jaké *Rozpočtové položky* a v jakém množství se budou fakturovat. Pokud uživatel dokončí výběr položek a jejich množství, zadá ještě fakturační údaje investora, kterému fakturu vystavuje, číslo faktury a data vystavení a splatnosti, případně upraví její cenu. Po potvrzení se vygeneruje soubor PDF s fakturou. Pokud uživatel schválí vygenerovanou fakturu, faktura se uloží k příslušné *Stavbě*.

Pro vytváření souborů ve formátu PDF slouží externí nástroj *PdfSharp* [11]. Prvním krokem je vytvoření instance třídy `PdfSharp.Pdf.PdfDocument`, která reprezentuje PDF dokument. Do dokumentu se přidávají jednotlivé stránky, `PdfSharp.Pdf.PdfPage`. Pro zápis dat do stránky slouží instance třídy `PdfSharp.Drawing.XGraphics`, která má metody pro vkládání textu, obrázků na stránku, nebo do ní i kreslit. Pro zápis textu na stránku je potřeba znát jeho souřadnice, na které se má text na stránku napsat. K určení pozice textu lze použít instanci třídy `PdfSharp.Drawing.XRect`, která určuje obdélníkem vymezenou plochu, do které text napíše. Textu lze určit zarovnání v rámci daného obdélníku, jeho písmo, velikost, i barvu.

3.2.12 Harmonogram

Harmonogram je tvořen *Etapami*, seznam všech *Etap* se načítá z databáze pro každý nově vytvořený *Objekt Varianty*. Formulář pro *Harmonogram* umožňuje seznam *Etap* pro zvolený *Objekt* upravovat – přidat novou *Etapu*, upravit existující nebo smazat vybranou *Etapu*. *Etapy* jsou reprezentovány třídou `KaRaFa.Modules.Harmonogram.Etap`. Načtení *Etap* do nového *Objektu* proběhne zavoláním metody `LoadEtaps(…)` třídy `ConstructionObject`. Metoda přijímá 3 parametry – instanci třídy `DatabaseConnection` s otevřeným připojení do databáze, datum začátku a datum konce *Stavby*, do které *Objekt* patří. *Etapy* se načítají z tabulky `Etapy`, kde ve sloupcích `Zacatek` a `Konec` jsou uložena procentuální rozložení výchozího začátku a konce *Etapy* vzhledem k začátku a konci celé *Stavby*. Pokud je v databázi u *Etapy* uložena ve sloupci `Zacatek` hodnota 25 a ve sloupci `Konec` hodnota 50, přepočítají se tato procenta vzhledem k datům začátku a konci celé *Stavby* tak, že *Etapa* začne ve čtvrtině a skončí v polovině *Stavby*. Výchozí rozložení *Staveb* do *Etap* je znázorněno na obrázku číslo 20. Velká výhoda tohoto nastavení je, že uživatel nemusí výchozí rozdělení *Harmonogramu* dělat ručně, ale rozložení *Stavby* do Etap je určeno při každém vytvoření *Harmonogramu* automaticky. Uživatel

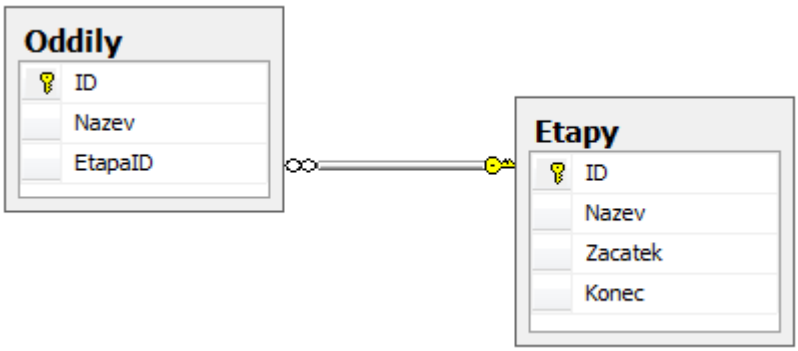
může libovolnou *Etapu* určit jako *Uzlový* bod (`IsImportant`), upravit Etapě datum začátku (`StartDate`) a konce (EndDate), nastavit její průběh (`Progress`).



Obr. 20 – Výchozí rozdělení *Stavby* do *Etap*

Výpočet začátku a konce etapy v závislosti na určených procentech provádí třídy `Karafa.Modules.Harmonogram.HarmonogramManager` zavoláním metody `CountEtapData(…)`. Třída `HarmonogramManager` rovněž upozorňuje na *Etapy*, které již měly být podle nastaveného data konce hotové, ale nejsou označeny jako hotové, k tomu slouží metoda `CheckOverduesEtaps(…)`. Mechanismus upozorňuje na *Etapy*, které jsou označené jako *Uzlový bod*, které již měly skončit, ale ještě nejsou dokončeny. Upozornění uživatelem probíhá zobrazením `MessageBoxu` se seznamem všech zpožděných *Etap* včetně jejich zpoždění.

Každá importovaná *Rozpočtová položka* bude mít určenou *Etapu*, do které je zařazena. Výchozí Etapa pro každou položku je dána tím, do jakého *Oddílu* ji uživatel naimportuje. Každý *Oddíl* má v databázi určeno, jaká výchozí *Etapa* se nastaví všem *Rozpočtovým položkám* při jejich importu do *Oddíu*. Určení výchozí *Etapy* pro *Oddíl* ukazuje obrázek číslo 21.



Obr. 21 – Vztah mezi *Oddíly* a *Etapami*

3.2.13 Rozpočet

Toto je nejdůležitější modul celé aplikace. Pro každý *Rozpočet* vytvořený v rámci *Objektu Stavby* se nahraje z databáze z tabulky `Oddily` výchozí sada *Oddílů*. *Oddíly* slouží k rozčlenění *Rozpočtu* do menších, logicky ucelených částí. Do *Oddílů* se importují *Rozpočtové položky*, u *Rozpočtových položek* lze upravovat jejich *Kalkulace* (*Jednotková cena* a *Výměra*) nebo *Rozbor*.

Editaci *Rozpočtové položky* umožňuje okno `Karafa.GUI.MainItemDetailForm`. Kromě úpravy vlastností *Rozpočtové položky* umožňuje toto editační okno přesunout Rozpočtovou položku do vybraného *Oddílu* nebo do vybrané *Etapy*. Dále okno umožnuje uživateli počítat *Výměru* položky zadáním výrazu. Poté, co je výraz vyhodnocen, automaticky se vyplní *Výměra* položky, kterou uživatel může ručně změnit podle potřeby. Tímto je splněn požadavek na možnost měnit hodnotu *Výměry* ručně bez potřeby měnit výraz pro výpočet.

Třída `Karafa.Modules.Calculation.ExpressionEvaluator` slouží pro vyhodnocení výrazu, kterým uživatel chce vypočítat *Výměru Rozpočtové položky*. Konstruktor třídy přijímá jako parametry textové pole, do kterého uživatel zadává výraz, a textové pole, do kterého má třída vypisovat mezivýsledky pro každý řádek zadaného výpočtu. Jakákoliv změna ve výpočtu přepíše uživatelem zadanou *Výměru* položky.

K editaci vlastností *Rozborových položek* typu *Materiál* (`MaterialItem`), *Práce* (`WorkItem`), *Stroj* (`MachineItem`), *Dodávka* (`DeliveryItem`) slouží okno `Karafa.GUI.AnalysisItemDetailForm`. Umožňuje editovat vlastnosti Název položky (`Name`), *Spotřebu* (`Usage`). Dále okno obsahuje tři volitelná textová pole, která se inicializují podle typu položky. Každá *Rozborová položka* má ve speciálním XML souboru, který je součástí aplikace jako `Embeded Resource`, určenou definici jednoho až tří těchto polí. Příklad takové definice:

```
<AdditionalFieldsDefinition>
  <AdditionalFieldList>
    <AdditionalField
      Label="Cena [{MeasureUnit}]:"
      PropertyName="Price"
      TextChange="RecalculateMainItemUnitPrice"
    />
    <AdditionalField
      Label="Dodavatel:"
      PropertyName="Company"
    />
  </AdditionalFieldList>
</AdditionalFieldsDefinition>
```

Obr. 22 – XML definice pro inicializaci volitelného textového pole

Tato definice říká, že se mají použít dvě prázdná textová pole. Atribut `Label` udává Popisek u textového pole. Pokud se v popisku objeví {Vlastnost}, tak se název vlastnosti položky nahradí hodnotou určené vlastností. Získání hodnoty z dané vlastnosti položky provádí následující kód, kde `detailedItem` je zobrazená *Rozborová položka* a `propertyName` název vlastnosti:

```
detailedItem.GetType().GetProperty(propertyName).GetValue(detailedItem, null)
```

Atribut `PropertyName` v definici pole říká, jaká vlastnost položky se edituje pomocí určeného pole. K načtení hodnoty vlastnosti slouží stejný příkaz jako na předchozí ukázce, k ukládání vlastnosti slouží obdobný kód, kde se místo metody `GetValue(…)` volá metoda `SetValue(…)`. Atribut TextChange v definici pole udává jméno veřejné bezparametrické metody, která se zavolá v reakci na změnu hodnoty vyplněné v textovém poli. Kód pro zaregistrování a vyvolání určené metody je na obrázku 23.

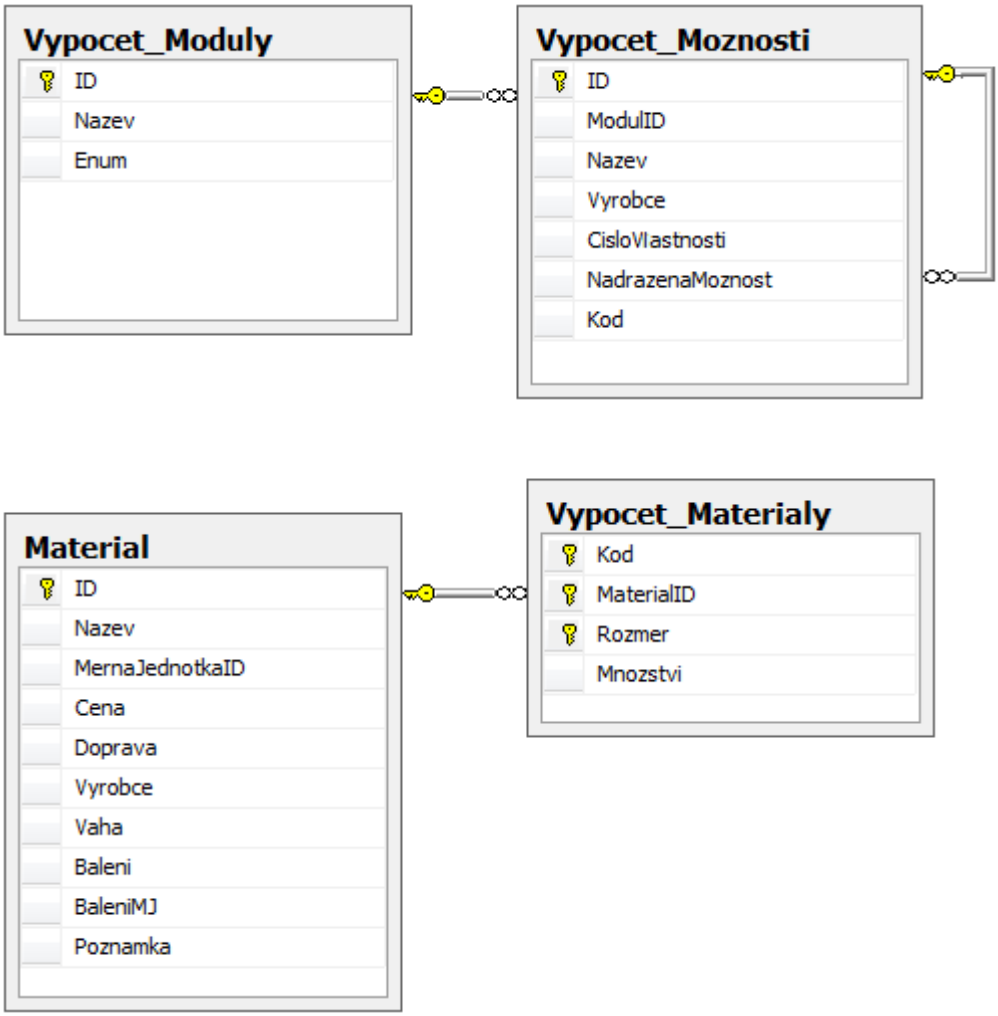
```
if (!string.IsNullOrEmpty(field.TextChangedMethod))
{
    // Get method
    MethodInfo method =
        GetMethodInfo(GetType().GetMethods(), field.TextChangedMethod);

    // If method found, register
    if (method != null)
        textField.TextChanged += delegate { method.Invoke(this, null); };
}
```

Obr. 23 –Volání metody určené názvem v XML atributu

3.2.14 Automatické kalkulace

Tento modul je svou funkcionalitou unikátní mezi všemi konkurenčními programy, v současné verzi aplikace modul ulehčuje uživatelům *Kalkulaci* pro sádrokartonové konstrukce a pro konstrukce komínů. Umožňuje uživateli zadat pouze rozměry nějaké konstrukce. *Kalkulace – Výměra* a *Spotřeba Materiálů* – konstrukcí se vypočítá automaticky. Modul na základně vstupních dat od uživatele vypočítá, jaké *Materiály* jsou potřebné na realizaci konstrukce a jejich množství.



Obrázek 24: Schéma tabulek pro Automatické výpočty

Všechny třídy realizující modul *Automatických kalkulací* se nachází ve jmenném prostoru `Karafa.Modules.AutomaticCalculation` (dále jen *KMA*) Datové podklady pro automatizované výpočty se budou načítat z databáze. Protože jde o novou funkcionalitu, byla potřeba vytvořit potřebné tabulky a naplnit je příslušnými daty. Schéma nových tabulek je zachyceno na obrázku číslo 24. Tabulka `Vypocet_Moduly` obsahuje seznam všech typů konstrukcí pro modul *Automatické konstrukce*, obsah tabulky odpovídá jednotlivým modulům v číselníku *KMA*`.Base.ModuleTypeEnum`, který slouží k určení typů konstrukcí v kódu. Typy konstrukcí jsou v tomto kontextu rovněž označeny jako moduly.

Tabulka `Vypocet_Moznosti` obsahuje seznam všech možností, pomocí kterých uživatel specifikuje vlastností konstrukce. Pro jednu vlastnost konstrukce existuje více možností, k jaké vlastnosti se možnost vztahuje, určuje sloupec `CisloVlastnosti`. Každá možnost má speciální Kód (sloupec `Kod`), který ji jednoznačně identifikuje mezi ostatními možnostmi pro jednu vlastnost dané konstrukce. Seznam možností pro nějakou vlastnost může záležet na vybrané možnosti jiné vlastnosti, k určení této závislosti slouží sloupec `NadrazenaMoznost`. Možnost reprezentuje třída *KMA*`.Base.ModuleOption`, která dědí od obecné třídy `Karafa.Modules.DatabaseItemBase` pro položku nahranou z databáze. Tabulka `Vypocet_Materialy` určuje, jaké *Materiály* je potřeba pro realizaci zadané konstrukce.

Možnosti z databázové tabulky Vypocet_Moznosti jsou reprezentovány třídou *KMA*.Base.ModuleOption. Každá vlastnost konstrukce je určena vybráním nějaké možnosti s jedinečným kódem (Code). Všechny třídy reprezentující konstrukce pro dědí od třídy *KMA*.Base.ConstructionBase, která reprezentuje obecnou konstrukci. Každá konstrukce obsahuje uživatelem zadané vstupní hodnoty a vybrané možnosti, aby bylo konstrukci možné zpětně editovat. Celá konstrukce (ConstructionBase) je jednoznačně identifikována unikátním kódem (ResultCode), který vznikne složením kódů všech jejích vybraných možností. Podle tohoto unikátního kódu (ResultCode) konstrukce se z tabulky Vypocet_Materialy nahrávají všechny *Materiály* (ResultCode je stejný jako hodnota sloupce Kod) pro danou konstrukci. Údaj rozměr (sloupec Rozmer) určuje, k jaké vlastnosti konstrukce *Materiál* patří. Sloupec Mnozstvi určuje *Spotřebu Materiálu* pro daný rozměr konstrukce.

Zavedením modulu pro výpočet nějakého typu konstrukce vyžaduje naplnění tabulky Vypocet_Materialy daty pro všechny možné konstrukce, které lze pomocí vlastností vytvořit. Tabulka musí obsahovat použití *Materiálů* pro všechny množné konstrukce podle jejich unikátního kódu, proto bude obsahovat řádově tisíce řádků. Tento krok není možné zautomatizovat, protože použití konkrétních *Materiálů* závisí výhradně na stavební firmě. Řešení pro odstranění tohoto kroku by mohlo spočívat v tom, že potřebné *Materiály* nebudou určeny přes databázi, ale uživatel si přímo zvolí, jaké *Materiály* chce pro konstrukci použít a jaké je jejich množství pro danou vlastnost. Tím by se odstranila potřeba tabulky Vypocet_Materialy. Ale toto řešení s sebou přináší dvě velké nevýhody. První nevýhodou je, že uživatel se může splést při určování materiálu pro danou vlastnost. Druhou nevýhodou je, že výpočet by přestal být automatický a doba potřebná k vytvoření jedné konstrukce by se výrazně prodloužila o hledání vhodných materiálů uživatelem. Smyslem modulu *Automatických kalkulací* je rychle a bezchybně realizovat konstrukce.

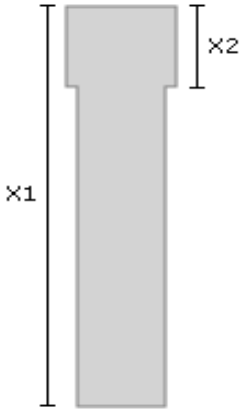
Obecnou třídou pro typ konstrukce je třída *KMA*.Base.ModuleItemBase, od které dědí každá konkrétní třída pro implementaci automatického výpočtu konstrukce. Výběr vlastností současných konstrukcí probíhá je realizován ComboBoxy, do kterých se nahrají možnosti (ModuleOption) z databáze pro jednotlivé vlastnosti, proto obecná třída pro modul obsahuje metodu LoadAndBindOptions(…). Metoda přijímá jako parametr otevřené připojení do databáze pro nahrání možností a seznam ComboBoxů, ve kterých uživatel určuje vlastnosti konstrukce právě pomocí načtených možností. Každá konkrétní třída – modul - dědící od této obecné třídy provádí výpočet množství *Materiál* podle svých specifikací.

Průběh realizace konstrukce začíná, když uživatel zadá vstupní rozměry a vybere možnosti pro příslušné vlastnosti konstrukce. Na základě těchto vstupních parametrů modul příslušné konstrukce vypočítá množství jednotlivých rozměrů konstrukce (sloupec Rozmer z tabulky Vypocet_Materialy). Každý *Materiál* je použit na určitý rozměr konstrukce, zároveň je určena *Spotřeba Materiálu* pro daný rozměr (sloupec Mnozstvi z tabulky Vypocet_Materialy). Celkové množství *Materiálu* pro daný rozměr určuje konkrétní implementace modulu pro výpočet konstrukce.

Výstupem výpočtu je seznam *Rozpočtových položek* (`Karafa.Modules.Import.MainItem`) pro každou konstrukci. *Materiály* načtené z databáze podle tabulky `Vypocet_Materialy` tvoří *Rozbor* příslušné *Rozpočtové položky*. Spojení seznamů *Rozpočtových položek* vytvořených pro jednotlivé konstrukce a *Rozpočtových položek* importovaných do *Oddílu* uživatelem umožňuje třída *KMA*`.Base.AutomaticConstructions`. Tato třída obsahuje oddělené seznamy všech konstrukcí rozdělených po jednotlivých modulech a všech *Rozpočtových položek*, jednu pro každou konstrukci. Všechny *Rozpočtové položky* jsou importované do *Oddílů*, proto každý *Oddíl* má instanci třídy `AutomaticConstructions`, která obsahuje konstrukce a jejich *Rozpočtové položky* jen pro daný *Oddíl*. *Rozpočtové položky* konstrukcí se zobrazí v seznamu všech *Rozpočtových položek* daného *Oddílu*. Protože položky mají vlastní *Rozbor* (seznam použitých *Materiálů* a jejich množství), uživatel může k těmto *Rozpočtovým položkám* přistupovat jako k importovaným *Rozpočtovým položkám* a *Rozbor* upravovat. Třída `AutomaticConstructions` se stará o správnou XML serializaci a deserializaci seznamu konstrukcí všech modulů, jejich *Rozpočtové položky* není potřeba ukládat, dají se opakovaně vygenerovat z uložených konstrukcí. Zároveň třída umožňuje přesun libovolných konstrukcí a jejich *Rozpočtových položek* mezi různé *Oddíly* či jejich úplné mazání. Celková průběh Automatické kalkulace znázorňuje následující obrázek:

Obr. 25 – Schéma průběhu výpočtu Automatických Kalkulací

3.2.14.1 Komíny



Obrázek 26 – Komínová konstrukce

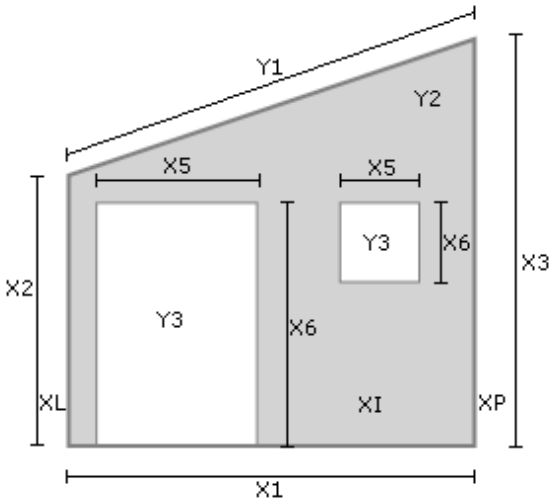
Obrázek číslo 26 zachycuje schéma komínu znázorněné v okně `Karafa.GUI.ChimneyModuleForm`, které slouží pro vytvoření a editaci komínových konstrukcí (*KMA*`.Chimney.ChimneyConstruction`). Uživatel určí vstupní rozměry komínu a jeho vlastnosti z možností načtených z databáze:

Označení na obrázku č. 23	**Popis**	**MJ**
X1	Celková výška komínu	[cm]
X2	Výška komínové hlavy	[cm]

Zadané vstupní rozměry komínu zpracuje třída *KMA*`.Chimney.ChimneyModule`, která vytvoří *Rozpočtovou položku* pro odpovídající komínovou konstrukci, vypočítá velikosti interních rozměrů (viz následující tabulka), nahraje *Materiály* podle tabulky `Vypocet_Materialy`, vypočítá jejich množství a přidá je do *Rozboru Rozpočtové položky* pro odpovídající komínovou konstrukci.

Rozměr	**Popis**	**Výpočet**
A	Počet komínu	Vždy 1
B	Redukovaná výška komínu	X1 – 1,333
C	Výška komínu bez hlavy	X1 – X2
D	Výška komínu	X1
E	Výška komínové hlavy	X2

3.2.14.2 Sádrokartonové konstrukce



Obr. 27 – Sádrokartonová konstrukce

Obrázek číslo 27 znázorňuje zjednodušené schéma jedné sádrokartonové desky. Šedá plocha značí sádrokarton, bílé plochy značí otvory v desce pro okno či dveře. Dveře či okna v desce být nemusí. Po obvodu desky se do zdi ukotví hliníkové příčky, ke kterým se přišroubuje sádrokarton. Kromě obvodových příček je konstrukce tvořena i vertikálními příčkami - stojkami – v daném rozestupu pro zpevnění konstrukce. Obecný tvar desky je pravoúhlý lichoběžník. Levá i pravá strana desky může mít rozdílnou velikost, proto při vhodném zvolení velikostí stran a délky desky lze získat trojúhelníkový, čtvercový či obdélníkový tvar. Pokud deska tvoří stěnu koupelny, je potřeba příslušnou stěnu desky impregnovat. Zakončení desky může být u zdi, v rohu nebo na desku může navazovat další deska. Jednotlivé desky lze skládat vedle sebe a tím uživatel může pomocí několika desek pokrýt i nepravidelný tvar celkové konstrukce. Přehled všech vstupních údajů ukazuje následující tabulka:

Označení na obrázku č. 24	**Popis**	**MJ**
X1	Délka	[cm]
X2	Výška levého konce konstrukce	[cm]
X3	Výška pravého konce konstrukce	[cm]
X5	Šírka dveří nebo okna	[cm]
X6	Výška dveří nebo okna	[cm]
XP	Pravé zakončení	číselník
XL	Levé zakončení	číselník
XI	Impregnace sádrokartonu	Číselník
XS	Mezery mezi vertikálními stojkami	[cm]

Potřebné výpočty pro realizaci sádrokartonových konstrukcí probíhají ve třídě *KMA*.SDK.SdkModule., která obsahuje nejen výpočty potřebných mezivýsledků, ale i vlastní výpočet velikosti interních rozměrů, které slouží pro výpočet množství *Materiálů*. Mezivýsledky shrnuje následující tabulka, jejich označení odpovídá obrázku číslo 27.

Označení	Popis	Výpočet
Y1	Délka zakončení u stropu	Pythagorova věta
Y2	Plocha konstrukce	Plocha lichoběžníku
Y3	Plocha dveří a oken	X5 * X6

Následující tabulka je přehled interních rozměrů, které korespondují s rozměry v databázové tabulce `Vypocet_Materialy` ze sloupce `Rozmer`. Každý *Materiál* má z databáze určen rozměr, pomocí kterého se počítá množství *Materiálu*.

Rozměr	Popis	Výpočet
A	Celková plocha sádrokartonu bez impregnace	Y2 – Y3
B	Celková plocha sádrokartonu s impregnací (jedno- i oboustranné)	(Y2 – Y3) * počet impregnovaných stěn
C	Zakládání u podlahy	X - X5 pouze pro dveře
D	Počet vertikálních stojek	X1 / XS
E	Zakládání u dveří	X5 jen pro dveře
F	Ukončení u stropu	Y1
G	Délka příček pro ukončení v rohu	X3 (při správném XP) + X2 (při správním XL)
H	Délka příček pro ukončení u zdi	X3 (při správném XP) + X2 (při správním XL)
I	Překlad nad dveřmi a okny [1]	X5 pro dveře + 2*X5 pro okna
J	Vertikální hrana u dveří [2]	X6

Poznámky:

1 - Délku překladu je násobek vzdálenosti vertikálních stojek - XS

2 – Hrana u dveří se dá nahradit vertikální spojkou, kterou dveře přeruší

Okno `Karafa.GUI.SdkModuleForm` slouží k vytváření a editaci sádrokartonových konstrukcí (*KMA*.SDK.`SdkConstruction`). Každá konstrukce se skládá z libovolného počtu sádrokartonových desek (*KMA*.`SDK`.SdkBoard). Zakončení sádrokartonových desek určuje číselník *KMA*.`SDK.SdkConstructionEndEnum`. Jeho hodnoty jsou Pokračování, Roh, Ukončení u zdi. Číselníkem obsahujícím hodnoty pro určení impregnace desek je *KMA*.`SDK.SdkImpregnationEnum`. Hodnoty pro impregnaci sádrokartonových desek jsou Žádná impregnace, Jednostranná impregnace, Oboustranná impregnace.

3.2.14.3 Implementace nového modulu Automatických kalkulací

Implementace *Automatické kalkulace* pro nový typ konstrukce - modulu - vyžaduje nejen vytvoření a nahrání podkladů do databáze, ale i zavedení mechanismu pro výpočet přímo v kódu aplikace.

Nový záznam pro nový modul musí být vložen do tabulky `Vypocet_Moduly`. Do tabulky `Vypocet-Moznosti` nahrát všechny nové možnosti pro vlastnosti konstrukce. Každá nová možnost musí mít určeno, k jaké vlastnosti konstrukce patří (`CisloVlastnosti`), svůj unikátní kód (`Kod`) a případně svou nadřazenou

možnost (NadrazenaMoznost). Pro každý interní rozměr (Rozmer) konstrukce, který vyžaduje použití Materiálu určit jeho Spotřebu na daný rozměr (Mnozstvi), vložit záznam do tabulky `Vypocet_Materialy` a to pro každou možnou kombinaci možností všech vlastností, které lze kombinovat. Každá kombinace možností pro všechny vlastnosti konstrukce je identifikována jednotlivými kódy vybraných možností. Celkový počet nových záznamů do této tabulky může dosahovat až řádu tisíců.

Každý nový modul konstrukce je vhodné umístit do nového podadresáře v adresáři Karafa.Modules.AutomaticCalculation, aby měl nový modul svůj jmenný prostor. Dále je potřeba vytvořit třídu reprezentující konstrukci, která bude dědit od třídy `KMA.Base.ConstructionBase`, třídu reprezentující modul, která bude dědit od třídy `KMA.Base.ModuleItemBase`, a okno ve jmenném prostoru `Karafa.GUI` pro vytváření a editaci konstrukcí. Poté vytvořit nový seznam konstrukcí a odpovídajících Rozpočtových položek ve třídě `KMA.Base.AutomatiConstructions` a určit jejich XML serializaci pro správné ukládání a načítání z XML souboru. Dále je potřeba zajistit, aby *Oddíly* dokázaly pracovat s tímto novým zdrojem *Rozpočtových položek* – implementovat stejně jako současné konstrukce.

Okno pro vytváření a editaci konstrukcí by mělo být voláno z okna `Karafa.GUI.CostsCalculationForm` modulu pro *Rozpočet*. V okně by se měly zobrazit vytvořené konstrukce ve zvoleném *Oddílu* v okně pro *Rozpočet*. Dále by nové okno mělo umožnit uživateli zadat rozměry konstrukce a určit její vlastnosti vybráním z možností, které jsou načteny z databáze. Třída pro modul musí umět z konstrukce určit seznam *Materiálů* a jejich množství pomocí tabulky `Vypocet_Materialy`.

3.2.15 Přehledy

Modul Rozpočet nabízí 3 přehledy aktuálně vybraného *Rozpočtu*, všechny jsou generovány pomocí PDF souborů. Nejzákladnější je přehled *Rozpočtu*, který ukazuje všechny *Oddíly Rozpočtu* a jejich ceny s daní a bez daně. Druhým přehledem je výkaz *Výměr*. Ten pro každý *Oddíl* vykazuje i seznam všech importovaných *Rozpočtových položek* s jejich výpočtem *Výměry*, pokud byl k výpočtu použit výraz. Třetím přehledem pro vybraný *Rozpočet* je přehled *Kalkulace*, který generuje seznam *Oddílů* a jejich importovaných *Rozpočtových položek*, pro každou importovanou položku zobrazuje *Rozbor*. Tyto rozbory slouží Rozpočtáři pro interní kontrolu výpočtu.

Další přehledy generované pomocí PDF souborů jsou přístupné z menu v Hlavním okně aplikace `Karafa.KarafaMainWindow`, pokud má uživatel právo *Stavbyvedoucí*. Prvním přehledem je přehled vybrané *Varianty*, který tiskne její *Objekty* včetně cen s daní a bez daně. Ke každému *Objektu* tiskne všechny jeho *Rozpočty* včetně jejich cen s daní a bez daně. Tento přehled bude sloužit pro předložení *Investorovi* pro schválení ceny *Stavby*.

Dalším přehledem je přehled *Harmonogramu* vybrané *Varianty*, který pro každý její *Objekt* vykreslí rozložení *Etap* v rámci celé *Stavby* včetně seznamu *Rozpočtových položek* podle jejich zařazení do *Etap*. Přehled *Harmonogramu*

jednotlivých *Objektů* je také dostupný z okna `Karafa.GUI.HarmonogramForm` pro modul Harmonogram.

Pro generování přehledů do PDF souborů slouží třídy ve jmenném prostoru `Karafa.Utilities.PDF`. Pro zápis do souborů PDF se využívá nástroje PdfSharp [11]. Obecnou třídou pro zápis do PDF souboru je třída `PdfDrawerBase`. Třída obsahuje instanci třídy `PdfSharp.Pdf.PdfDocument` reprezentující generovaný dokument, seznam výchozích štětců (`XBrush`), písem (`XFont`), metodu pro přidání nové stránky do PDF dokumentu `NewPage()`, metodu pro nepovinné očíslování stránek dokumentu `NumberPages()` a metodu pro uložení dokumentu `SaveDocument(…)`. Každá třída pro generování konkrétních souborů dědí od obecné třídy, kvůli jednotné správě dokumentu – přidávání stránek, číslování stránek a ukládání dokumentu. Zápis generovaných dat probíhá odděleně v každé konkrétní třídě stejným způsobem, jaký je popsán při popisu generování faktur do PDF souborů v modulu Fakturace.

Aplikace KaRaFa nabízí i přehledy za celou *Stavbu* (respektive její *Variantu*). Které se negenerují do PDF souborů, ale jsou zobrazeny pomocí okna `Karafa.GUI.OverviewForm`. Prvním přehledem je přehled veškerého *Materiálu*, *Práce*, *Stroje* a *Dodávek* pro z *Rozpočtových položek* všech *Oddílů* všech *Rozpočtů* všech *Objektů* vybrané *Varianty*. Položky se uloží do `HashTable`, kde klíčem je typ položky - Materiál, Práce, Stroj, Dodávka - a hodnotou je vždy seznam všech položek daného typu.

Druhým přehledem zobrazeným pomocí okna `Karafa.GUI.OverviewForm` je přehled faktur za celou *Stavbu*, který vykazuje všechny vydané faktury k dané *Stavbě*. Pro každou fakturu je zobrazeno datum vydání, datum splatnosti, cena s daní, cena bez daně. Seznam faktur je rovněž reprezentován pomocí `HashTable`. Jediným klíčem je typ položky – faktura – a hodnotu je seznam faktur vybrané *Stavby*.

Okno `Karafa.GUI.OverviewForm` v konstruktoru přijímá `HashTable` se zdrojovými daty pro zobrazení přehledu do tabulky a definici sloupců tabulky. Definici sloupce reprezentuje třída `Karafa.Utilities.ColumnDefinition`, která má vlastnosti pro určení názvu sloupce (`Header`), vlastnost zobrazené položky pro získání hodnoty do sloupce (`PropertyName`) a typ sloupce (`Column`). Definice sloupců je uzpůsobena zdroji dat uloženém v `HashTable`, kde klíčem je název seznamu zobrazených záznamů a hodnotou je samotný seznam záznamů, které se v tabulce zobrazí. Pokud `HashTable` obsahuje více klíčů a jejich příslušných seznamů, tak se pro každý klíč vytvoří `RadioButton`, jehož výběr vyplní tabulku záznamy z příslušeného seznamu. Proto definice sloupců tabulky musí obsahovat sloupce společné všem zobrazovaným typům záznamů.

3.2.16 Lokalizovaný MessageBox

Aplikace KaRaFa využívá `System.Windows.MessageBox`, pokud je potřeba od uživatele potvrdit jeho volbu například při mazání nějaké položky, nebo jen uživatele informovat o průběhu nějaké informace. Tento `MessageBox` nemá lokalizované popisy tlačítek, Aplikace KaRaFa proto pro lokalizaci popisů tlačítek využívá externí knihovnu *MessageBoxManager* [14], která umožňuje

změnit výchozí popis na tlačítkách okna MessageBox. Práci s externí knihovnou zajišťuje třída Karafa.Utilities.MessageBoxLocalizer. Její metoda ShowDialog(…) přijímá jako povinné parametry zprávu a titulek pro MessageBox. Nepovinnými parametry metody lze určit ikonu a tlačítka s jejich popisy určenými ve slovníku. Pokud nejsou parametry určeny, zobrazí se MessageBox s tlačítkem OK a informační ikonou. Příklad zobrazení MessageBoxu s lokalizovanými popisy tlačítek načtených pomocí slovníku Resources:

```
MessageBoxLocalizer.ShowDialog(
    message,
    title,
    MessageBoxButton.YesNo,
    MessageBoxImage.Question,
    new Dictionary<MessageBoxLocalizer.Buttons, string>()
    {
        {MessageBoxLocalizer.Buttons.YES, Properties.Resources.BUTTON_YES},
        {MessageBoxLocalizer.Buttons.NO, Properties.Resources.BUTTON_CANCEL}
    }
)
```

Obr. 25 – Zobrazení MessageBoxu s vlastním určením popisu tlačítek

4. Závěr

Aplikace byla v rámci práce úspěšně dokončena. V době dokončení práce byla aplikace ve stavební firmě H&Co ve finální fázi testování. Poslední ohlasy byly kladné. Poté by měla být nasazena do ostrého provozu. Mimo rámec Bakalářské práce proběhne konečná úprava podoby generované faktury a několika přehledů. Dále by se měl implementovat nový Automatických konstrukcí – Střechy.

4.1 Dokončené cíle

1. Práce s existující databází

    - Aplikace KaRaFa plně využívá rozdělení položek v databázi na Rozpočtové položky a Rozborové položky.

2. Ukládání a načítání dat

    - Data se načítají a ukládají do souborů XML, které jednotlivé uživatelé jednoduše mohou přenášet mezi svými pracovními stanicemi.

3. Jednoduché a přehledné grafické rozhraní

    - Pracovníci stavební firmy jsou s finálním grafickým rozhraním spokojeni.

4. Vylepšený výpočet výměr

    - Výpočet výměr umožňuje zadat výraz v požadovaném tvaru, včetně korektní detekce chyb a upozorňování na ně.

5. Automatické výpočty konstrukcí

    - Modul byl v rámci aplikace vytvořen pro sádrokartonové a komínové konstrukce, pracovníci stavební firmy jsou s ním spokojeni.

6. Možnost aktualizace databáze

    - Aplikace KaRaFa umožňuje aktualizovat existující databázi z externího souboru.

7. Vytváření nových částí *Staveb* zkopírováním již existující částmi *Stavby*

    - Nová část lze vytvořit kopií existující části *Stavby* tak, aby změny v nové (respektive původní) části neovlivnily původní (respektive novou) část *Stavby*.

8. Harmonogram

    - Modul Harmonogram umožnuje vytvářet, upravovat a mazat *Etapy*, případně upravovat organizovat rozdělení *Rozpočtových položek* do *Etap*.

9. Uživatelský přístup

    - Aplikace KaRaFa podporuje dostupnost funkcionalit v závislosti na právech přihlášeného uživatele.

10. Přehledy, Tiskové sestavy a Fakturace

    - Přehledy, tiskové sestavy a faktury se generují do PDF souborů.

4.2 Srovnání s podobnými aplikacemi

Stavební firma používá či používala programy Aspe [16] a Stavex [17], proto bude aplikace KaRaFa srovnána s těmito aplikacemi podle zkušeností pracovníků stavební firmy s ostatními aplikacemi. Výsledné hodnocení a srovnání shrnuje tabulka na konci této kapitoly 4.2.

Velkým přínosem aplikace KaRaFa je pří výpočtu *Výměr* je možnost zapisovat textové poznámky do výrazu pro výpočet *Výměry* položky, více úrovňové uzávorkování výrazu a kontrola syntaxe. Tyto vlastnosti ostatním aplikacím chybí.

Úpravy Kalkulace pro každou položku odděleně jsou velmi praktické a umožňují rychlé přepočítávání na jinou cenovou úroveň. Aplikace Stavex byla velmi složitá množstvím různých koeficientů, které se vztahovaly k množství různých základů. Aplikace Aspe umožňovala pouze všechny položky přenásobit stejným koeficientem. Další kladnou vlastností aplikace KaRaFa je nezávislost *Rozpočtu* na původní ceníkové databázi. V programu Stavex se například musela pracně udržovat oddělená ceníková databáze pro každou *Stavbu*, protože každá změna v základní databázi ovlivňovala všechny a tedy i již uzavřené rozpočty. V oblasti fakturace se všechny tři aplikace nijak zásadně neliší.

Modul Automatických kalkulací je zcela nový a u ostatních aplikací se neobjevuje. Přitom je zcela nenahraditelný pro předvýrobní přípravu k přesnému rozpisu jednotlivých potřebných materiálů. Tato práce se vždy dělala ručně velmi pracně, docházelo k omylům při počítání množství, navíc se nebylo možné tuto práci archivovat. Tento modul odstraňuje všechny tyto nevýhody.

Vlastnost \ *Program*	*Karafa*	*Aspe*	*Stavex*
Přehledný výpočet výměr včetně kontroly syntaxe a možnost vkládání textových komentářů	ANO	NE	NE
Jednoduchá úprava ceny položky ručně	ANO	NE	NE
Fakturace	ANO	ANO	ANO
Nezávislost databáze na rozpočtu	ANO	ANO	NE
Automatické kalkulace	ANO	NE	NE

4.2.1 Hodnocení aplikace KaRaFa stavební firmou

Aplikace KaRaFa za srovnávanými aplikaci v ničem nezaostává, naopak je vylepšena z hlediska praktičnosti (především ve vylepšeném zadávání výměrů položek). Jeho obrovským pozitivem je v jeho jedinečném modulu konstrukce. Aplikace KaRaFa je přehlednější a rychlejší, což se často ocení v časovém presu před odevzdávání nabídek do soutěže.

4.3 Známé nedostatky

- Současná verze aplikace neumožňuje zcela bezpečný a bezproblémový centralizovaný běh, protože je uzpůsobena pro malou stavební firmu s jedním uživatelem. Implementace centralizace – sdílení souboru s uloženými daty aplikace - by nebyla náročná, ale byla nad rámec zadání.

- Uživatelský přístup je sice naimplementován, ale ukládání dat aplikace do XML souboru, který je veřejně dostupný, popírá princip uživatelského přístupu. Implementace korektního uživatelského přístupu souvisí s centralizací databázi.

- Aplikace je primárně zaměřena na výpočet *Rozpočtu*. Moduly Harmonogram a Fakturace nejsou využity plnou měrou.

- Místo původně zamýšlené automatického výpočtu pro střešní konstrukce byl implementován výpočet pro komíny. A to kvůli velkému množství záznamů do databáze, která se musí ve firmě ručně vytvořit. Z časových důvodů byla implementace odložena.

4.4 Možnosti budoucího vylepšení

- Implementace centralizovaného přístupu různých uživatelů, kdy MSSQL Server s databází bude instalován na jednom firemním serveru, všechny pracovní stanice budou tuto databázi sdílet. Díky centralizaci by bylo možné sdílet jeden soubor s nastavením aplikace pro všechny instance aplikace KaRaFa. Ukládání *Staveb* by již mohlo být realizováno v databázi.

- Možnost synchronizace s nějakým účetním systémem (např. Pohoda [15]), se kterým by se mohly synchronizovat faktury případně synchronizovat informace o stavu materiálu na skladě

- Kombinace aktualizace databáze z daného souboru s porovnáváním databází z různých on-line zdrojů

5. Zkratky

[1] MDB - typ souboru pro aplikaci Microsoft Access (verze 2003 a starší)

[2] XML - eXtensible Markup Language

[3] PDF - Portable Document Format

[4] SQL – Structured Query Lnaguage

6. Reference

[1] První normální forma:
http://siret.ms.mff.cuni.cz/skopal/databaze/slidy/DBI025_03.pdf

[2] Druhá normální forma:
http://siret.ms.mff.cuni.cz/skopal/databaze/slidy/DBI025_03.pdf

[3] Třetí normální forma:
http://siret.ms.mff.cuni.cz/skopal/databaze/slidy/DBI025_03.pdf

[4] Boyce-Coddova normální forma:
http://siret.ms.mff.cuni.cz/skopal/databaze/slidy/DBI025_03.pdf

[5] XML serializace
http://msdn.microsoft.com/cs-cz/library/system.xml.serialization.aspx

[6] Binární serializace
http://msdn.microsoft.com/cs-cz/library/kd1dc9w5.aspx

[7] Mělká kopie:
http://www.codeproject.com/KB/cs/ShallowVsDeepCopy.aspx

[8] Hluboká kopie:
http://www.codeproject.com/KB/cs/ShallowVsDeepCopy.aspx

[9] Návrhový vzor Visitor:
http://ulita.ms.mff.cuni.cz/pub/predn/NPRG024/23-Visitor.ppt

[10] SHA – 1
http://en.wikipedia.org/wiki/SHA-1

[11] MD5
http://en.wikipedia.org/wiki/MD5

[12] Seznam knihoven pro C# umožňující vytvoření PDF:
http://csharp-source.net/open-source/pdf-libraries

[13] PdfSharp
http://pdfsharp.com/PDFsharp

[14] MessageBoxManager
http://www.codeproject.com/KB/miscctrl/Localizing_MessageBox.aspx

[15] Učetní system Pohoda
http://www.ucto-pohoda.cz/

[16] Aspe
http://www.aspe.cz/cs/

[17] Stavex
http://www.stavex.info/

7. Přílohy

[1] Příloha obsahuje DVD, na kterém se nachází

- Zdrojové kódy aplikace KaRaFa
- Celá aplikace KaRaFa
- Uživatelská dokumentace
  - Včetně prvotní zálohy databáze a návodu pro instalaci
- Vygenerovaná programátorská dokumentace
- Tento text Bakalářské práce