VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
KATEDRA INFORMATIKY

Editor postupu měření pro program UniSave

Editor of Method of Measurement for Program UniSave

2009 Marek Procházka

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. 5. 2009

Děkuji Ing. Davidu Ježkovi, Ph.D. za odborné metodické vedení a cenné rady při zpracování mé práce.

Abstrakt

Cílem práce je navrhnout a rozšířit stávající program UniSave. Program UniSave je implementován v jazyce Java a slouží pro načítání hodnot z různých měřidel. Naměřené hodnoty se dále zpracovávají a vytváří se z nich protokol o měření.

Protože program UniSave je implementován v jazyce Java, je i jeho rozšíření implementováno v tomto jazyce. První část je věnována analýze stávajícího programu UniSave, která je nezbytná pro jeho rozšíření o editor postupu měření. V druhé části jsou popsány použité technologie. Třetí a čtvrtou část tvoří analýza a návrh editoru postupu měření a měření podle postupu s implementacemi.

Klíčová slova: UniSave, Java, měření, postup měření, editor, třída, třídní diagram, XML, UML, GUI

Abstract

The aim of this thesis is to design and to extend existing UniSave application. UniSave application is implemented in Java language and it is used for loading values from measuring instruments. Then measured values are processed and the protocol of measurement is created.

Due to the fact that the UniSave is implemented in Java programming language, the extension will be created in this language too. The first part is devoted to the analysis of existing UniSave application, which is necessary for the extension of the editor of measurement method. The second part describes used technologies. The third and the fourth part consist of the analysis and designing the editor of measurement method and measurement from method and their implementations.

Keywords: UniSave, Java, measurement, measurement method, editor, class, class diagram, XML, UML, GUI

Seznam použitých zkratek a symbolů

API	– Application programming interface
CD	– Compact Disc
DOM	– Dokument Object Model
GUI	– Grafical User Interface
HTML	– HyperText Markup Language
JVM	– Java Virtual Machine
Mac	– Macintosh
MS	– Microsoft
OS	– Operating System
PC	– Personal Computer
PDF	– Portable Dokument Format
SAX	– Simple API for XML
UML	– Unified Modeling Language
USB	– Universal Serial Bus
W3C	– World Wide Web Consortium
WWW	– World Wide Web
XML	– EXtensible Markup Language

Obsah

Úvod

Svět se neustále vyvíjí, tempo růstu nových výrobních technologií se zvyšuje. S růstem počtu obyvatel na naší planetě rostou i nároky pro uspokojování jejich potřeb. Moderní technologie člověku pomáhají v každodenním životě. Podíl fyzické práce se snižuje na minimum, výroba se zrychluje a zefektivňuje.

Téměř žádný výrobní podnik se neobejde bez měření. Vytvoření přesného kvalitního výrobku bez vad vyžaduje neustálou kontrolu jak při jeho tvorbě, tak i kontrole jakosti. Se vznikem moderních digitálních měřidel spolu s vyžitím počítačové techniky vznikla potřeba měřené údaje přímo načítat do elektronické podoby. Moderní digitální měřidlo, které je propojeno s PC, práci ulehčuje a urychluje. Odpadá pak pracné ruční přepisování naměřených veličin. Tím se eliminuje počet chyb a zvyšuje přesnost prováděného měření. Následné zpracování údajů v elektronické podobě má opět své výhody. Archivace, filtrování, třídění, tvorba grafů a protokolů je poté již jen otázkou několika kliknutí tlačítkem myši.

Program UniSave je jedním z těch, které to umožňují. Měřidla připojená k PC s nainstalovaným programem UniSave načítají požadovaná data, jednotlivé údaje jsou pak zapisovány do přehledné tabulky. Tato data se mohou následně archivovat, vytvářet z nich grafy a protokoly.

Ve své diplomové práci se zabývám rozšířením programu UniSave 2006 naimplementovaném Ing. Davidem Ježkem, Ph.D., který je současně zadavatelem této práce. Hlavním cílem je vytvoření editoru postupu měření. Implementací editoru postupu měření se uživateli umožní nadefinovat jednotlivé kroky, definice jejich opakování, výpočet hodnot. Program je napsán v rozšířeném platformově nezávislém programovacím jazyku Java, proto bude i jeho rozšíření využívat tento jazyk.

Diplomová práce je rozdělena do čtyř kapitol. Po seznámení s programem UniSave následuje jeho analýza nezbytná k samotnému rozšíření programu a napojení na jeho funkce. Pro zpřehlednění rozsáhlé struktury jsou v implementaci využity návrhové vzory. Data jsou ukládána do strukturovaného formátu XML. Následující, druhá část, je věnována použitým technologiím, které jsou využívány v dalších kapitolách i implementaci.

Před implementací editoru postupu měření a měření podle postupu je nezbytná specifikace požadavků, analýza, návrh a řešení. Právě těmto částem vývoje softwaru se zabývá třetí a čtvrtá kapitola. Pro popis struktury využívají diagramy standardizovaného jazyka UML, použitého již při popisu samotného programu. Využití technologie vytvoření překladače je prostředkem k vyhodnocování vzorců užívaných při postupu měření.

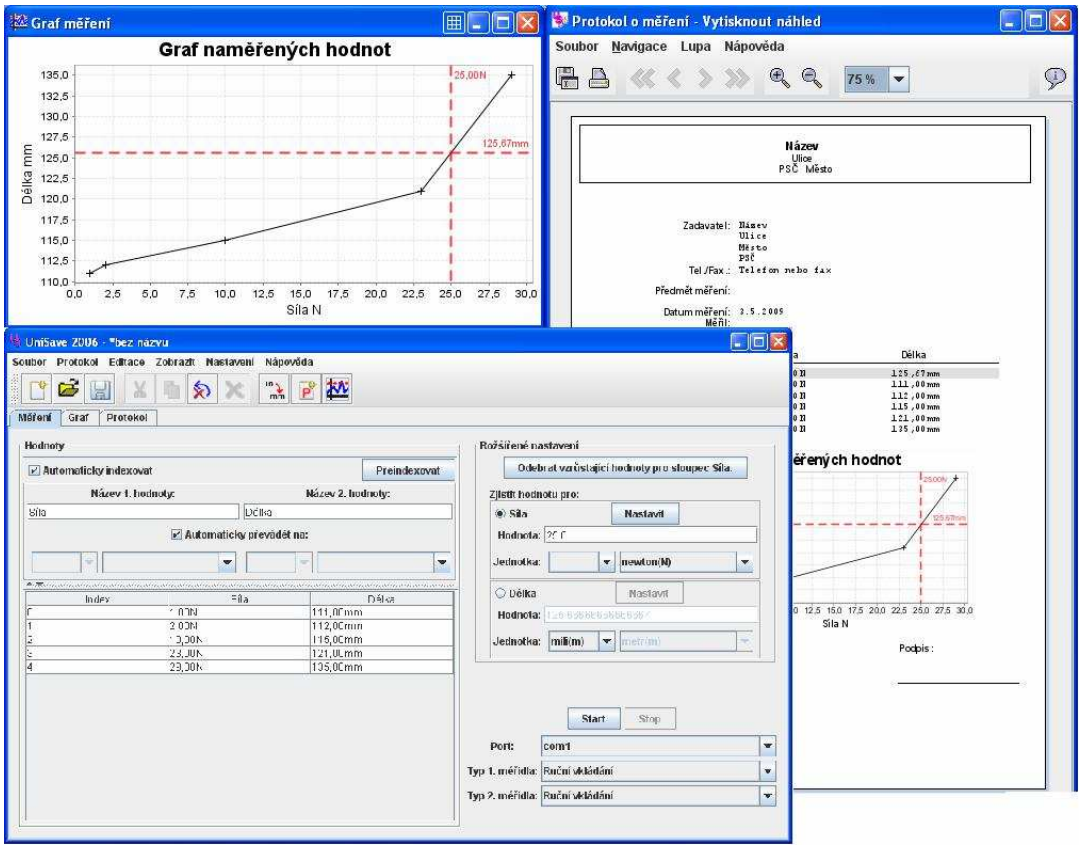
1 Program UniSave

Tato kapitola se zabývá popisem aplikace, dále její analýzou nezbytnou k jejímu rozšíření a pochopení struktury programu UniSave.[8]

1.1 Popis

Program UniSave (obrázek 1) slouží pro načítání naměřených hodnot z různých typů měřících zařízení. Ta jsou propojena s počítačem přes infračervený nebo sériový port. Pro využití USB portu je možné použití redukce ze sériového portu na USB. Pokud provádíme klasické ruční měření nebo nelze měřidlo s počítačem propojit, využijeme funkci ručního zadávání hodnot do programu.

Jednotlivé naměřené hodnoty, včetně jednotky a veličiny, se postupně načítají do zobrazené tabulky, kterou lze následně i editovat či konvertovat. Ze získaných údajů může být vytvořen graf dovolující odečtení hodnoty z vykreslené křivky. Data se mohou uložit do strukturovaného formátu XML. Po vyplněním formuláře se správnými údaji o zadavateli a laboratoři můžeme vygenerovat protokol měření v PDF formátu nebo jej vytisknout. Současná verze programu podporuje měření hodnot z dvou měřidel zároveň. V grafu jsou pak zobrazeny odděleně, data z prvního měřidla na ose *x* a z druhého na ose *y*.



Obrázek 1: Program UniSave

1.2 Analýza

Celkový náhled na samotnou funkčnost implementovaných částí programu UniSave vytváří tato podkapitola. Popisuje rozsáhlou strukturu, základní principy chodu programu a provázání jednotlivých částí.

1.2.1 Struktura programu

Zdrojové kódy programu se skládají z naimplementovaných tříd a externích knihoven. Pro lepší orientaci a přehlednost jsou související třídy společných vlastností, schopností a funkčnosti sloučeny do společných balíčků. Jednotlivé balíčky a jejich užití jsou podrobněji popsány v následující části.

Při popisu balíčků je pro zobrazení statické struktury, větší názornost a pochopení využíváno třídních diagramů popsaných v kapitole 2.2. Základní hierarchickou strukturu všech devíti balíčků obsahujících celkově 111 tříd zachycuje následující strom (obrázek 2).

Obrázek 2: Struktura balíčků programu UniSave

1.2.2 Balíček UniSave2006

Tento hlavní balíček zahrnuje všechny ostatní balíčky. Uloženy jsou v něm také třídy sloužící pro nastavení globálních a uživatelských vlastností programu.

Pomocí třídy *GlobalSetting* nastavujeme globální vlastnosti. Ta pro jejich uchovávání používá XML soubor. Data se pak načítají pomocí *GlobalSettingXMLLoader* a obsahují adresářovou cestu k souboru *CommBridge.exe* a PDF šabloně protokolu, nastavení vlastností programu a nápovědě.

1.2.3 Balíček data

Obsah balíčku data je složen z tříd reprezentujících různé typy měření a podbalíčků, obsahujících různé datové typy.

Pro vytváření instancí měření (*XYMesurment, PlanedMesurment* a *StatisticMesurment*) je využíván návrhový vzor *Factory* (třída *MesurmentFactory*). Současná verze podporuje pouze *XYMesurment*.

Jeho součástí jsou významné třídy:

- *XYMesurment*
- *PlanedMesurment*
- *StatisticMesurment*
- *MesurmentFactory*
- *MesurmentXMLLoader*
- *MesurmentSAXHandler*
- *MesurmentEntryFactory*
- *Mesurment*
- *MesurmentEntry*

1.2.4 Balíček data.value

Jeho obsahem jsou nadefinované datové typy, které reprezentují hodnoty získané z různých typů měření. Pro tvorbu instancí slouží třída *MesurmentEntryFactory*, využívající návrhový vzor *Factory metod pattern*.

Hlavní datové typy jsou:

- *Value*
- *XYValue*
- *Statistic*
- *NamedValue*

1.2.5 Balíček device

Součástí tohoto balíčku jsou třídy, které souvisejí s konkrétními měřidly a třídy pro různá nastavení komunikace daných měřidel se sériovými porty. Třída *CommDeviceSettingFactory* vytváří příslušné instance nastavení připojení jednotlivých měřidel. Pro obecné nastavení připojení slouží třída *CommDeviceSetting*, která je rodičovskou třídou nastavení sériového, infračerveného a nulového portu, užívaného k ručnímu vkládání.

1.2.6 Balíček grabber

Protože různé měřicí přístroje mají rozdílný formát naměřených hodnot, je ztíženo jejich následné zpracování. Řešením je parsování měřidly posílaných dat. Instance každého grabberu, implementujícího rozhraní *GrabberInterface*, je vytvářena třídou *GrabberFactory* což se týká těchto tříd:

- *AsciiGrabber*
- *DoubleGrabber*
- *ElectroPhysicAsciiGrabber*
- *MahrAsciiGrabber*
- *ManualGrabber*
- *MitutoyoAsciiGrabber*

1.2.7 Balíček gui

Tento nejrozsáhlejší balíček zahrnuje všechny třídy spojené s grafickým prostředím aplikace (GUI). Vnořeny jsou v něm další dva podbalíčky *action* a *value*., blíže popisované v následujících oddílech. Třída *MainFrame* tvoří hlavní okno aplikace, do něhož jsou vkládány panely definované daným typem měření a zároveň vytvářené třídou *MesurmentPanelFactory*.

Mezi ně patří:

- *NullMesurmentPanel* – úvodní obrazovka, bez zvoleného měření
- *XYMesurmentPanel* – panel měření pomocí dvou měřidel

Informace programu UniSave poskytuje třída *About*, nastavení je přístupné třídou *SettingDialog*. Pro zobrazení hodnot v tabulce a nastavení souvisejících funkcí je prováděno třídou *XYMesurmentValuePanel*.

1.2.8 Balíček gui.value

Slouží pro vkládání a editaci všech datových typů. O vytváření jednotlivých panelů se stará třída *MesurmentEntityEditPanelFactory*. Dále obsahuje:

- *EditDialog* – edituje naměřené hodnoty
- *XYValuePanel* – vkládá, edituje hodnotu *XYValue*
- *ValuePanel* – vkládá, edituje hodnotu Value
- *UnitSelectionPanel* – slouží pro výběr jednotky
- *MesurmentEntityEditor* – je vyvářen příslušnou factory pro úpravu konkrétní jednotky
- *MesurmentEntityConstructionException* – výjimka vyvolaná při chybě převodu řetězce na číslo

1.2.9 Balíček units

Obsahem tohoto balíčku jsou následující třídy, zastupující jednotky užívané programem UniSave. Každá naměřená číselná hodnota je charakterizována svou jednotkou (*Unit*). Ta je tvořena předponou (*UnitPrefix*) a typem jednotky (*UnitDescription*). Jednotky a předpony jsou

uloženy v souboru. Jednotky je možné navzájem konvertovat podle pravidel definovaných ve třídě *UnitConversion*.

2 Použité technologie

V této kapitole jsou stručně popsány technologie využívané při tvorbě vlastní práce. Úvodní část je věnována historii a vlastnostem programovacího jazyka Java, na ni navazuje charakteristika jazyka UML s diagramy, které jsou využívány v analýze a návrhu rozšíření programu. Dále je popsána technologie a rozdělení návrhových vzorů, neboť některé z nich jsou v práci využity. Protože data programu UniSave jsou ukládány do souboru ve formátu XML, je jedna podkapitola věnována právě tomuto jazyku. Při vyhodnocování vzorců použitých v editoru postupu měření je využíváno jednoduchého překladače. Technologií překladačů a jejich tvorbou se zabývá další podkapitola.

2.1 Programovací jazyk Java

2.1.1 Stručná historie

Jazyk Java [džava] (v americkém slangu to znamená kafe) je dílem společnosti Sun, konkrétně její dceřiné společnosti JavaSoft, která jej neustále vyvíjí. Její počátek sahá až do počátku 90. let 20. století. Tehdy se firma Sun rozhodla vyvinout jazyk pro výrobky spotřební elektroniky (původně se jmenoval Oak). Okolo roku 1994 se dostala až do sféry počítačů a internetu.

Kolem roku 1995 statické webové stránky přestávaly vyhovovat potřebám stále se rozrůstající internetové sítě. V Oaku byl vyvinut nový webový prohlížeč *Web Runner* s možností interpretace kódu v jazyce Oak. Přinesl tak dosud nerealizované dynamické a interaktivní možnosti pro *World Wide Web*. Důvodem přejmenování na jazyk Java byla již registrovaná ochranná známka Oak. Prohlížeč *Web Runner* byl přejmenován na *HotJava*. 23. května 1995 byla Java i prohlížeč *HotJava* oficiálně představeni na konferenci SunWorld. [6]

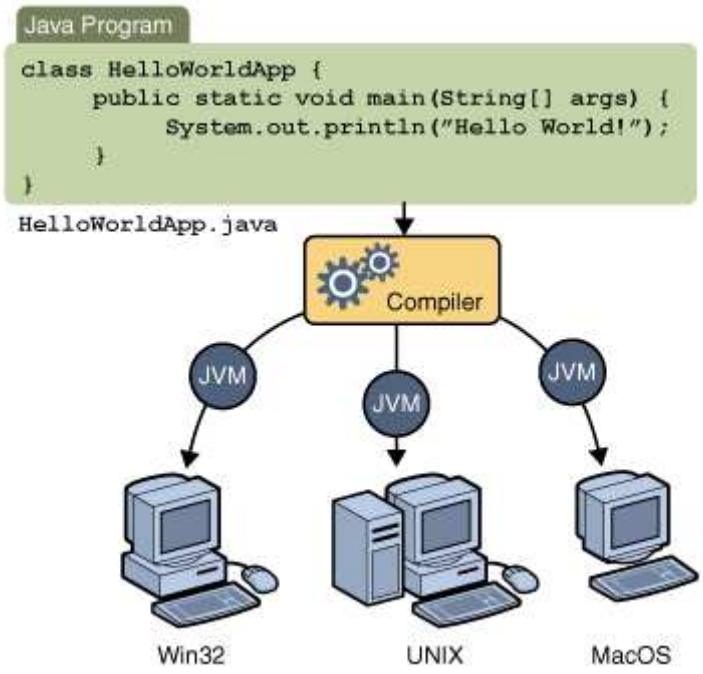
2.1.2 Základní vlastnosti

Programovací jazyk Java je:

- *jednoduchý* – jeho syntaxe je zjednodušenou verzí jazyka C a C++, zůstal stejný základ, problémová konstrukce byly odstraněny a nahrazeny dalšími užitečnými funkcemi
- *objektově orientovaný* – s výjimkou osmi datových typů jsou ostatní datové typy objektové
- d*istribuovaný* – je navržen pro síťové prostředí
- *interpretovaný* – místo strojového kódu se vytváří mezikód, který je zpracován až na koncovém zařízení
- *robustní* – neumožňuje konstrukce, které způsobovaly často chyby, používá silnou typovou kontrolu, o automatickou správu paměti se stará Garbage collector, který nepoužívané části v paměti odstraňuje
- *bezpečný* – chrání počítač v síťovém prostředí

- *nezávislý na architektuře* – aplikace lze spustit na libovolném operačním systému nebo architektuře
- *přenositelný* – nezávislý na architektuře
- *výkonný* – přestože je interpretovaný, ztráta výkonu není významná,
- *víceúlohový* – podporuje vícevláknové aplikace
- *dynamický* – navržen pro nasazení ve vývojovém prostředí, knihovnu lze dynamicky za chodu rozšířit
- *elegantní* – je čitelný

Soubor s příponou *java* se pomocí překladače zkompiluje do souboru s příponou *class*, ten obsahuje bajtový kód s instrukcemi pro virtuální stroj Javy (Java Virtual Machine). Po spuštění programu interpret zkontroluje, rozšifruje bajtový kód a provede jeho akce. JVM je dostupná na mnoha operačních systémech, proto lze stejné *class* soubory spustit v MS Windows, Solária, Linux i Mac OS (obrázek 3).



Obrázek 3: Schéma zkompilování a spuštění aplikace pod různými OS

2.1.3 Java platforma

Od ostatních platforem se Java platforma liší tím, že je pouze softwarová a běží nad jinými hardwarově založenými platformami. Skládá se ze dvou částí, *JVM* a *API* (Aplication Programming Interface).

Dílčí platformy jsou:

- *Java SE* – aplikace pro stolní počítače
- *Java EE* – aplikace pro rozsáhlé informační a podnikové prostředí

- *Java ME* – aplikace pro mobilní zařízení (mobilní telefony, PDA, atp.)
- *Java Card* – aplikace pro tzv. chytré karty (platební, kreditní, atp.)

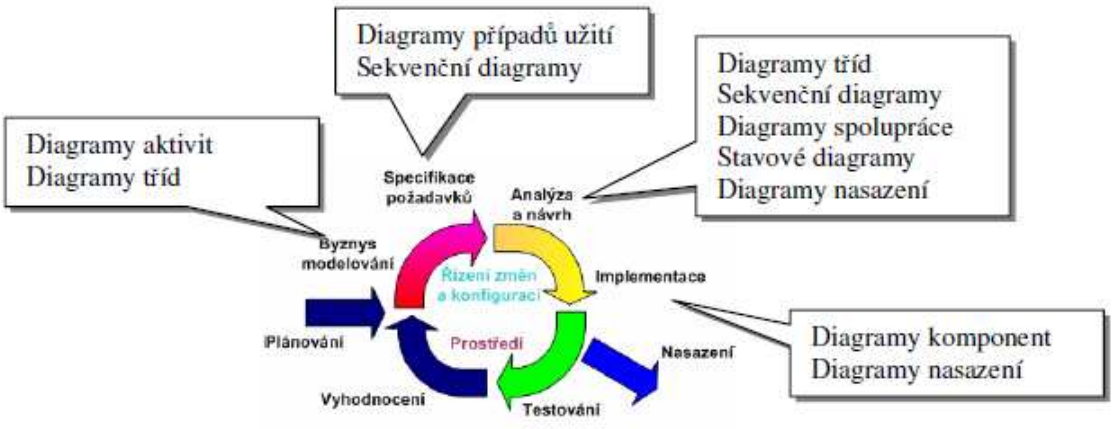
2.2 Jazyk UML

UML, Unified Modeling Language je grafický jazyk používaný v softwarovém inženýrství. Byl vytvořen pro sjednocení různých přístupů při vytváření specifikací softwarového produktu. Dnes je již universálním standardizovaným jazykem, který slouží k návrhu, vizualizaci, specifikaci a dokumentaci programových systémů. Specifikace jazyka zahrnuje řadu diagramů popisující systém z hlediska různých náhledů a abstrakcí. [2, 3]

Základní náhledy procesu modelování se svými diagramy:

- *Funkční náhled*
  - Diagram případu užití (Use Case diagram)
  - Logický náhled
  - Diagram tříd
  - Objektový diagram
- *Dynamický náhled* – popisuje chování
  - Stavový diagram
  - Diagram aktivit
  - Interakční diagramy (sekvenční a spolupráce)
- *Implementační náhled*
  - Diagram komponent
  - Diagram nasazení

Jednotlivé etapy vývoje softwarového systému a přiřazení souvisejících diagramů znázorňuje následující schéma (obrázek 4).

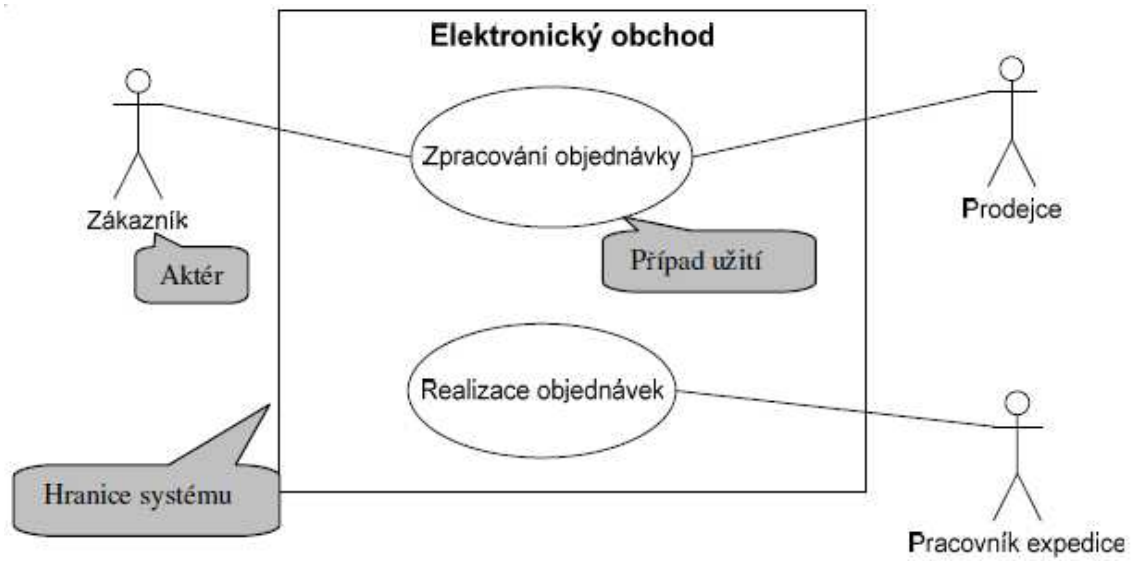


Obrázek 4: Přiřazení diagramů jednotlivým etapám vývoje softwarového produktu

Při popisu analýzy programu UniSave a vytváření funkčního a logického náhledu budou využívány diagramy případu užití a diagramy tříd. Proto je následují část věnována jim.

2.2.1 Diagram případu užití

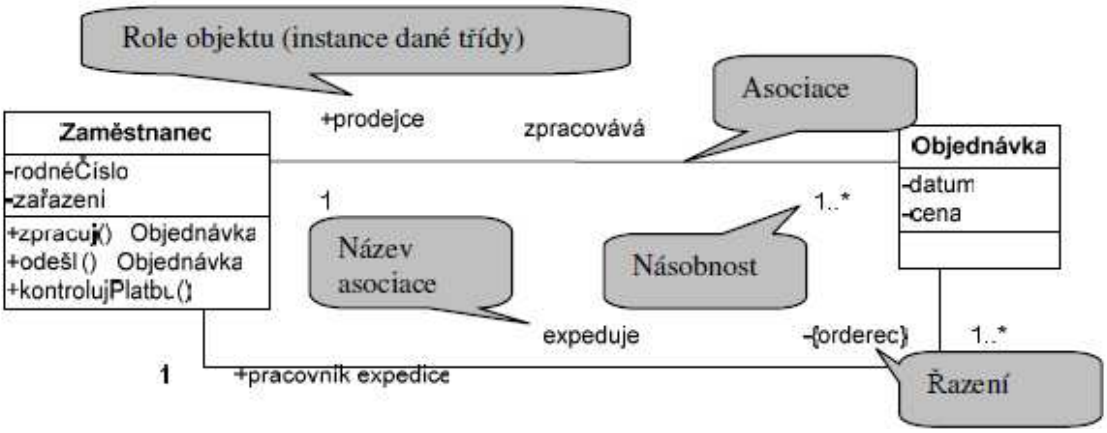
Funkční specifikace je v jazyce UML řešena prostřednictvím diagramů případu užití (Use Case Diagram). Pomocí nich se popisují vztahy mezi případy užití systému a aktéry, stojícími vně systému. Případ užití je posloupnost vzájemně navazujících transakcí mezi aktérem a vlastním softwarovým systémem. Aktéři jsou uživatelé nebo jiné systémy komunikující s vyvíjeným softwarovým systémem (obrázek 5).



Obrázek 5: Funkční specifikace systému

2.2.2 Diagram tříd

Diagram tříd popisuje obecně platné třídy a typy s jejich vnitřní strukturou a vzájemnými relacemi nezávislými na čase. Statická struktura systému specifikuje jeho prvky, vnitřní strukturu těchto prvků a vztahy mezi nimi a je popsána třídními diagramy. Vztahy mezi třídami jsou znázorněny v diagramu (obrázek 6).



Obrázek 6: Třídní diagram s popisem asociací mezi třídami

2.3 Návrhové vzory

Návrhové vzory jsou abstrakcí užitečných částí softwarových produktů. Po nalezení úspěšného řešení problému, který se opakuje v různých produktech různých doménových oblastí, lze toto řešení zobecnit a toto se stává návrhovým vzorem. Pro popis vzorů se používá množina objektů a jejich tříd, přizpůsobených k řešení obecného problému. [2]

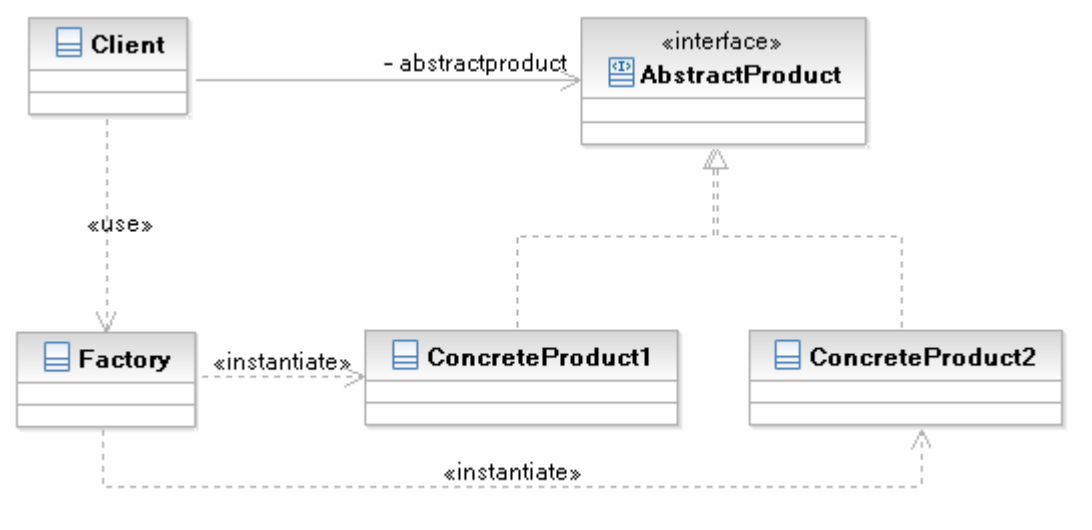
Rozdělení nejpoužívanějších návrhových vzorů podle způsobu jejich použití:

- *Návrhové vzory tvořící (Creational Patterns)* jsou určeny k řešení problému vytváření instancí tříd. Například:
  - Factory Metod Pattern
  - Builder Pattern
  - Prototype Pattern
  - Singleton Pattern
- *Návrhové vzory strukturální (Structural Patterns)* řeší problémy strukturování objektů a jejich tříd. Například:
  - Adapter Pattern
  - Bridge Pattern
  - Composite Pattern
  - Decorator Pattern
  - Farade Pattern
  - Flyweight Pattern
  - Proxy Pattern
- *Návrhové vzory chování (Behavioral Patterns)* popisují algoritmy a spolupráci objektů. Například:
  - Chain of Responsibility Pattern
  - Command Pattern
  - Interpreter Pattern
  - Iterator Pattern

- Mediator Pattern
- Memento Pattern
- Observer Pattern
- State Pattern
- Strategy Pattern
- Template Pattern
- Visitor Pattern

2.3.1 Factory Method Pattern

Tento návrhový vzor poskytuje rozhraní pro vytváření skupin příbuzných objektů bez nutnosti specifikovat jejich třídy. Klient je oddělen od logiky výběru správné třídy a obdrží požadovanou instanci. Jeho obecnou strukturu přibližuje následující třídní diagram (obrázek 7).



Obrázek 7: Struktura návrhového vzoru Factory

2.4 XML

XML (eXtensible Markup Language) je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Slouží ke snadnému vytváření konkrétních značkovacích jazyků k různým účelům a pro různé typy dat. [6]

Jeho úlohou je především výměna dat mezi aplikacemi a publikování dokumentů. Umožňuje popsat strukturu dokumentu, ale nezabývá se vzhledem dokumentu nebo jeho částí, nýbrž jeho věcným obsahem. Vzhled dokumentu se pak definuje pomocí kaskádových stylů. Je umožněna i transformace do jiného typu dokumentu, nebo do jiné struktury XML.

Jazyk XML nemá žádné předdefinované značky (tagy, názvy jednotlivých elementů) a také jeho syntaxe je podstatně přísnější, než syntaxe jazyka HTML. Oproti HTML má vysoký

informační obsah. Pomocí XML značek (tagů) se vyznačuje v dokumentu význam jednotlivých částí textu. Dokumenty obsahují více informací, než při použití značkování zaměřeného na prezentaci (vzhled), tj. definici písma, odsazení apod. Proto jsou XML dokumenty informačně bohatší, čehož se využívá v mnoha oblastech, např. při hledání v dokumentu, kdy lze určit i význam hledaného textu.

Existují dva nejčastější přístupy zpracování XML dokumentu:

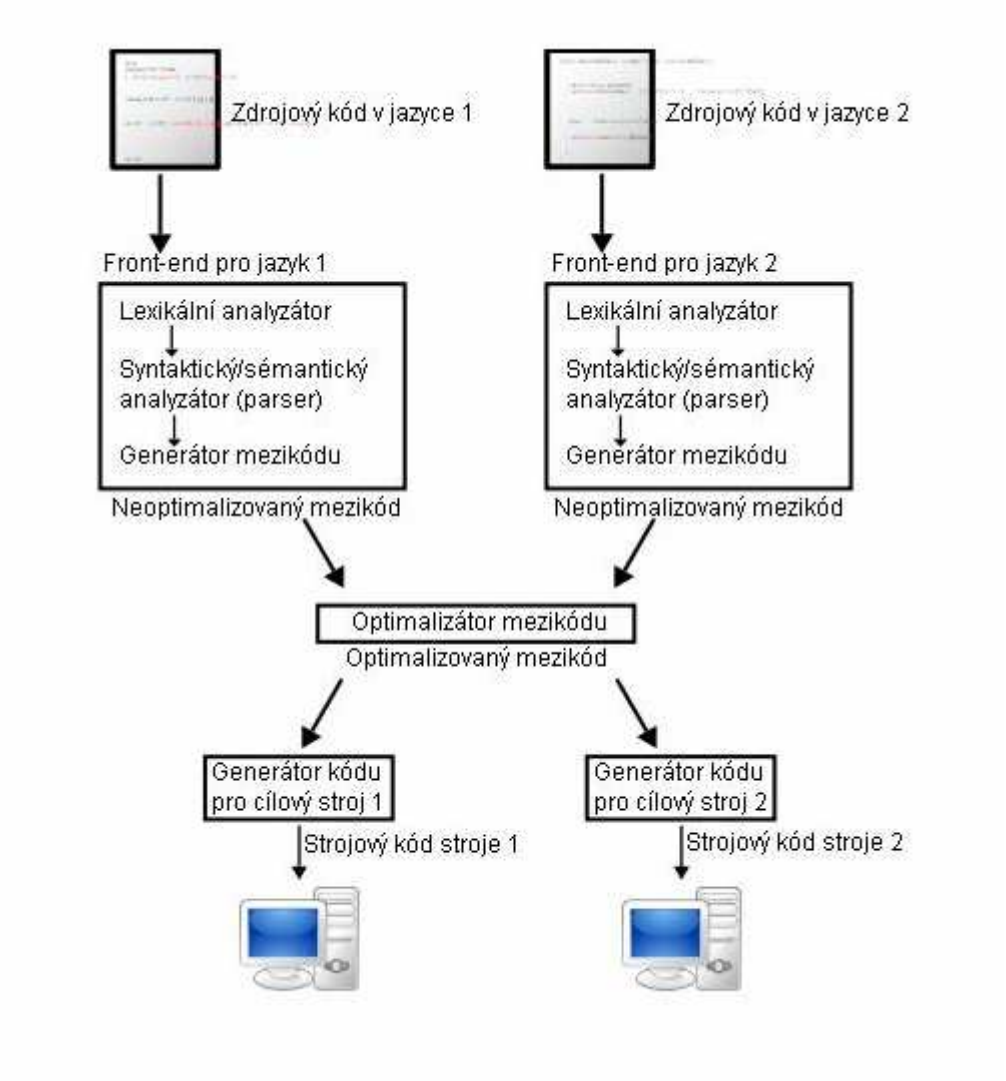
- *DOM parser (DOM = Document Object Model)* vezme XML dokument, ze kterého vytvoří strom v paměti.
- *SAX parser (SAX = Simple API for XML)* při postupném procházení dokumentu se vyvolávají události, které programátor zpracovává. Tohoto přístupu využívá i program UniSave.

Příklad struktury receptu zapsaného v XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Poznamka je nutné přidat více receptů. -->
<recept jméno="chleba" čas_přípravy="5 minut" čas_vaření="3 hodiny">
    <titulek>Jednoduchý chleba</titulek>
    <přísada množství="3" jednotka="šálky">Mouka</přísada>
    <přísada množství="0,25" jednotka="unce">Kvasnice</přísada>
    <přísada množství="1,5" jednotka="šálku">Horká voda</přísada>
    <přísada množství="1" jednotka="kávová lžička">Sůl</přísada>
    <instrukce>
        <krok>
            Smíchejte všechny přísady dohromady a dobře prohněťte.
        </krok>
        <krok>Zakryjte tkaninou a nechejte hodinu v teplé místnosti.</krok>
        <krok>Znovu prohněťte, umístěte na plech a pečte v troubě.</krok>
    </instrukce>
</recept>
```

2.5 Překladače

Překladač (kompilátor) je nástrojem používaným programátory pro vývoj softwaru. Kompilátor slouží k překládání algoritmů, které jsou zapsány ve vyšším programovacím jazyce, do strojového kódu (strojového jazyka). Z obecného hlediska je kompilátor stroj (program), který provádí překlad ze vstupního jazyka do jazyka výstupního a mapuje jeden nebo více zdrojových kódů podle překladových parametrů na kód výstupního jazyka (obrázek 8). Důležitou součástí procesu překladu jsou diagnostické zprávy, které uživatele informují o přítomnosti chyb ve zdrojovém programu. [5, 7, 9]



Obrázek 8: Struktura překladače podporujícího dva vstupní jazyky i dvě cílové architektury

Překladač lze navrhnout mnoha způsoby, často bývá rozdělen na dvě části. První je závislá na vstupním jazyce (tzv. *front-end*) a druhá závisí na cílové architektuře (tzv. *back-end*). Překlad nemusí probíhat přímo, může využívat tzv. mezikód (*bytekód*), tzn. že kód je nejprve překládán do mezikódu a v druhém kroku z mezikódu do strojového jazyka. Tím se rozdělí jeden kompilátor na dva jednodušší kompilátory, přičemž překlad ze zdrojového jazyka do mezikódu může být společný pro více architektur. Rovněž je výhodné, že pokud je potřeba vyrobit překladač pro další jazyk, stačí vytvořit pouze část pro překlad do mezikódu.

2.5.1 Části překladače

Překladač je možné rozdělit na následující části, které nemusí být vždy striktně odděleny (např. lexikální a syntaktická analýza může splynout):

- lexikální analyzátor
- syntaktická analýza
- sémantický analyzátor
- optimalizace kódu
- generování cílového kódu

Lexikální analyzátor

První jednotkou překladače je lexikální analyzátor, který získává ze vstupního zdrojového textu tzv. *lexému* (základní prvek, klíčové slovo), kterou pak zasílá syntaktickému analyzátoru spolu s dalšími souvisejícími údaji (operátor, identifikátor aj.). Všechny možné lexémy jsou ve vstupním jazyce popsány pomocí regulárních výrazů. Všechny údaje předávané syntaktickému analyzátoru se označují výrazem *token*.

Syntaktický analyzátor

Hlavní částí překladače je syntaktický analyzátor (parser) provádějící vlastní analýzu vstupního jazyka. Rozpoznává, zda je program zapsán správným způsobem, určuje pořadí prováděných jednotlivých částí příkazů (např.u „*x +y*z*“ rozpozná a určí prioritu operace násobení před sčítáním) U vnořených příkazů upřednostní vyhodnocení parametrů funkce před jejím voláním. Konečným výsledkem práce tohoto analyzátoru je tzv. syntaktický strom (parse tree) popisující strukturu vstupního programu.

Běžným způsobem konstrukce syntaktického analyzátoru je metoda rekurzivního sestupu, kde se pomocí rekurzivních volání konstruuje syntaktický strom. Celková struktura implementace překladače tak odráží strukturu příslušné gramatiky. Neterminálnímu symbolu gramatiky odpovídá v překladači rekurzivní volání funkce, která zastupuje příslušný neterminál. Při dosažení terminálních symbolů gramatiky (lexémů jazyka) se rekurze ukončí přečtením a uložením těchto symbolů do syntaktického stromu.

Sémantický analyzátor

Syntaktický strom je zpracováván sémantickým analyzátorem, který zároveň provádí analýzu významu a korektnosti jednotlivých operací a kontroluje správnost struktury programu. Probíhá zde veškerá typová kontrola a konverze. Příkladem je analýza typů výrazů na levé a pravé straně příkazu přiřazení a zabránění přiřazení hodnoty s plovoucí řádovou čárkou do proměnné s pevnou řádovou čárkou. Vstupní syntaktický strom je v této fázi doplněn dodatečnými informacemi a provedenými konverzemi. Tedy výstupem je opět syntaktický strom.

Optimalizátor

Optimalizátor zlepšuje vlastnosti výsledného kódu pomocí různých transformací mezikódu. Tím dochází.k zrychlení běhu a zmenšení velikosti kódu. Například může dojít k odstranění cyklického přiřazení.

Generátor kódu

Pro vygenerování programu v cílovém jazyce z mezikódu slouží generátor kódu. Nejčastěji je cílovým jazykem strojový kód.

2.5.2 Typy překladačů

Klasický překladač

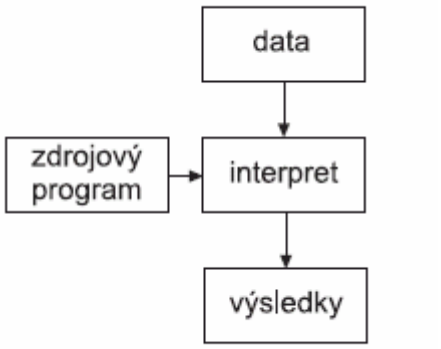
Nejběžnějším typem překladače je klasický překladač. Na vstup překladače je poslán celý program ve zdrojovém tvaru (text či struktura).V průběhu překladu nelze do činnosti překladače zasahovat. Samotný překlad je proveden jedním nebo několika průchody s postupnou lexikální, syntaktickou a sémantickou analýzou, optimalizací a generováním cílového kódu.

Konverzační překladač

Během překladu je umožněna uživateli komunikace s konverzačním (interaktivním) překladačem. Řádky zdrojového kódu jsou ihned po jejich dokončení postupně posílány překladači ke zpracování. Pomocí příkazů metajazyka určených překladači se zjednoduší analýza vstupu, překladač nemusí rozlišovat vstupní jazyk a metajazyk. Jeho výhodou je rozšířená možnost ladění.

Konverzační interpretační překladač

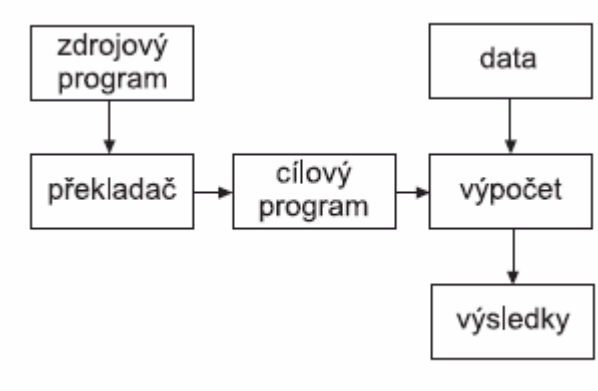
Tento překladač postupně přijímá jednotlivé příkazy, následně kontroluje jejich správnost, uloží je a interpretuje. Používá se především pro implementaci jednoduchých grafických a výukových programů (obrázek 9).



Obrázek 9: Interpretační překladač

Konverzační kompilační překladač

Tento překladač načtené příkazy uloží do zdrojového kódu a následně je zpracovává a generuje cílový nebo intermediální kód. Komplikace při zpracování kontextových závislostí způsobuje přístup pouze k příkazu, který je právě zpracováván. Díky této nevýhodě je používán jen u nejjednodušších jazyků (obrázek 10).



Obrázek 10: Kompilační překladač

Jednoprůchodový překladač

Překladač je jednodušší, má nižší režii, neboť během jeho činnosti nedochází ke vzniku mezikódu.

Víceprůchodový překladač

Překladač je složitější, ale jeho vytvoření a údržba je jednoduchá. V důsledku oddělení jednotlivých částí jsou nároky na hardware nižší a zpracování probíhá postupně v samostatných průchodech.

2.5.3 Gramatika překladače

Gramatiky programovacích jazyků patří podle hierarchie *Chomského* do třídy bezkontextových jazyků. Zahrnují parsovací algoritmy. Proces parsování, neboli syntaktická analýza, jsou postupné derivace řídící gramatiky (tzv.tvorba derivačního stromu) a ověřování, zda jí lexémy odpovídají. Existují dvě parsovací strategie, shora dolů (*top-down parsing*) nebo zdola nahoru (*bottom-up parsing*).

Top down parser vychází ze startovacího symbolu gramatiky a postupně hledá derivace. K jeho implementaci se používá rekurzivní sestup, u něhož se pro každý neterminální symbol implementuje obslužná funkce, volaná při parsování.

Bottom-up parser vykonává postupně operace shift-reduce a postupuje od nejnižších terminálních symbolů gramatiky ke startovacímu neterminálu.

Gramatika v *Chomského normální formě* musí splňovat následující derivační pravidla:

- $A \rightarrow BC$ (A, B, C jsou neterminální symboly)
- $A \rightarrow \alpha$ (α je terminální symbol)
- $S \rightarrow \varepsilon$ (volitelné pravidlo, S je startovací symbol)

LL (left-to-right, leftmost derivation) parsery používají parsování shora dolů, jejich vstup je zpracováván zleva doprava a vytvářejí levé derivace. Mohou být také označovány LL(k), kde *k* je počet tokenů potřebný k rozhodnutí o průběhu další analýzy.

LR (left-to-right, rightmost derivation) parsery vytvářejí nejpravější derivace, přičemž vstup je zpracováván zleva doprava. Využívají parsování zdola nahoru, lze je implementovat efektivně. LR gramatiku má většina programovacích jazyků.

3 Editor postupu měření

Cílem této diplomové práce je, jak již bylo zmiňováno, rozšíření další funkčnosti programu UniSave. Hlavní částí je doplnění programu o editor postupu měření, pomocí kterého si uživatel nadefinuje plán měření. Vytvořený plán definuje jednotlivé sousledné kroky měření s postupem, ke každému kroku určuje související atributy sloužící pro další výpočty a vlastnosti napomáhající uživateli orientaci při samotném měření. Samozřejmostí je ukládaní, editace a načítání plánu pomocí souboru.

S vytvořeným plánem musí být vytvořeno i nové, s ním související měření podle plánu, které jej doplňuje o další položky hodnot. Protože s jedná o poměrně obsáhlý logický celek navazující na vytvořený plán, je podrobněji popsán v následující kapitole.

3.1 Specifikace požadavků

Postup měření definuje plán, pomocí kterého bude samotné měření probíhat. Skládá se z jednotlivých, na sebe navazujících kroků. Pomocí editoru je můžeme vytvářet, upravovat a mazat. Tyto specifikované údaje pak ukládat, načítat a editovat.

Každý krok postupu by měl mít následující hodnoty:

- *Index* – slouží pro orientaci v tabulce, jeho hodnota se automaticky zvyšuje při přidávání dalších položek
- *Jméno* – popisuje měřenou hodnotu, případně určuje její výpočet pomocí vzorce
- *ID* – identifikuje hodnotu, užívá se ve vzorci jako odkaz na měřenou hodnotu
- *Jmenovitá hodnota* – číselná hodnota s jednotkou (včetně předpony) určuje ideální měření
- *Tolerance* – povolená odchylka (včetně jednotky s předponou) měření od jmenovité hodnoty
- *Opakování* – počet opakování měření stejné položky v postupu
- *Popis* – napomáhá uživateli při měření hodnoty a popisuje ji
- *Obrázek* – napomáhá uživateli v orientaci při měření

Místo měřené hodnoty můžeme přidat vzorec, který údaj vypočítá. Ve vzorci lze použít jako proměnné *ID* předchozích naměřených hodnot, také se mohou používat základní matematické operace a funkce. Výpočet hodnot definovaných vzorci proběhne při měření podle postupu.

Operace, operátory, funkce používané ve vzorci:

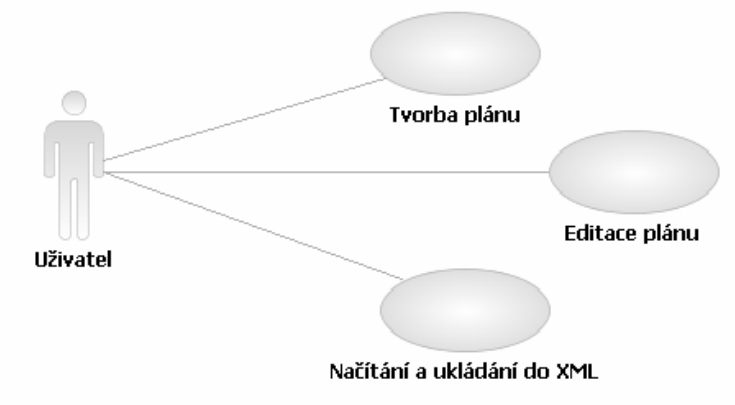
- *[ID]* – odkaz na naměřenou proměnnou s identifikátorem ID
- „+“ – operátor sčítání
- „-“ – operátor odčítání
- „*“ – operátor násobení
- „/“ – operátor dělení
- *„( )“* – závorky

Další operace, operátory, funkce používané ve vzorci (možné rozšíření programu):

- *„abs([hodnota])“* – funkce absolutní hodnoty
- *„exp([hodnota], [exponent])“* – hodnota umocněná na daný exponent
- *„sqr([hodnota])“* – druhá mocnina hodnoty
- *„sqrt([hodnota])“* – druhá odmocnina hodnoty
- *[hodnota]E[x]* – hodnota krát 10 na x-tou
- *„sin([hodnota])“* – goniometrická funkce sinus hodnoty
- *„cos([hodnota])“* – goniometrická funkce cosinus hodnoty
- *„tg([hodnota])“* – goniometrická funkce tangens hodnoty
- *„cotg([hodnota])“* – goniometrická funkce cotg hodnoty

Všechna zadaná data je možno uložit do souboru, který lze později načíst a upravit dle potřeby. Program UniSave využívá k ukládání již naimplementovaného měření strukturovaného formátu XML. Proto i pro uložení postupu měření je zvolen stejný formát. Takový soubor si lze i navíc prohlédnout či upravit v jakémkoli textovém editoru. Editorem vytvořený plán se poté využije při samotném měření podle postupu.

Celkový náhled hlavních funkcí editoru znázorňuje diagram případu užití (obrázek 11). Popisuje vztah mezi uživatelem a editorem. Uživatel vytváří nový postup měření nebo upravuje již uložený. Přidává nové hodnoty a data, upravuje je, zadává vzorce pro výpočet dalších hodnot. Nakonec takto upravený postup uloží do souboru.



Obrázek 11: Use Case editoru postupu měření

3.2 Analýza, návrh a řešení

Celkový plán měření je složen z několika dílčích kroků. U každé naměřené hodnoty dle postupu musíme uložit několik dalších údajů (viz specifikace), proto bude pro zobrazení nejpřehlednější tabulka. Protože popis a obrázek pro vložení do tabulky je příliš velký, bude zobrazován jen při vkládání či editaci. Vkládané údaje budou v programu reprezentovány samostatnou třídou, která k nim bude poskytovat příslušné funkce.

Pro vytvoření celkového náhledu a pochopení řešení editoru jsou v této podkapitole podrobněji rozebrány důležité třídy implementující editor postupu měření, jsou zde také znázorněny vazby těchto nových tříd na ostatní třídy stávajícího řešení.

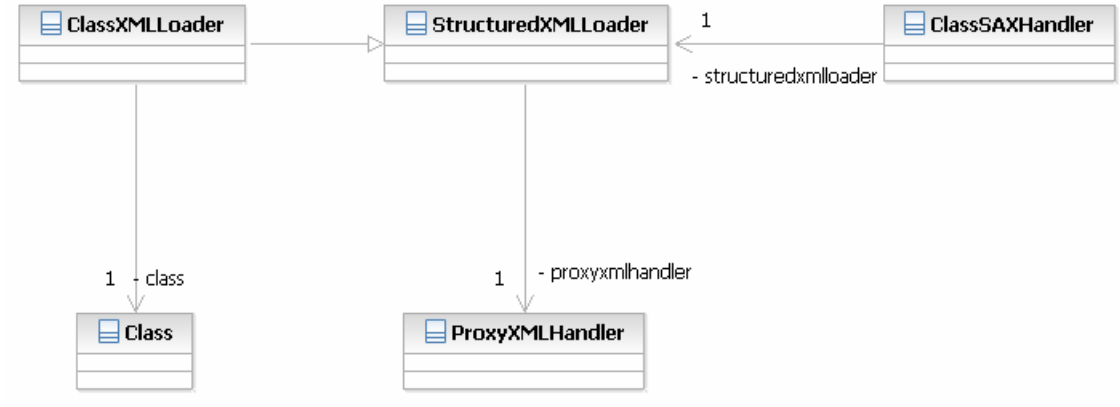
Ukládané položky:

- *Index* – označuje hodnoty v tabulce, je reprezentován celým číslem, jehož hodnota se bude zvyšovat při každém vložení záznamu
- *Jméno* – textový řetězec pro výstižný popis hodnoty, případně určuje její výpočet pomocí vzorce
- *ID* – identifikátor, z důvodu použití ve vzorci to bude textový řetězec, identifikuje hodnotu, užívá se ve vzorci jako odkaz na měřenou hodnotu
- *Jmenovitá hodnota* – číselná hodnota s jednotkou (včetně předpony) určuje ideální měření, v měření je již naimplementována třída *Unit* skládající se z číselné hodnoty s možností zvolit jednotku s předponou
- *Tolerance* – povolená odchylka měření od jmenovité hodnoty, zároveň číselná hodnota s jednotkou (včetně předpony) využívající třídu *Unit*
- *Opakování* – počet opakování měření stejné položky v postupu daný celým kladným číslem
- *Popis* – řetězec znaků, který napomáhá uživateli při měření hodnoty a popisuje ji
- *Obrázek* – napomáhá uživateli v orientaci při měření, je reprezentován cestou v adresářové struktuře

3.2.1 Práce s XML souborem

Pro uchovávání všech výše zmíněných údajů slouží strukturovaný soubor XML, který má jednoznačný formát a jednoduše se parsuje. V současné verzi UniSave je tento soubor využíván k uložení hodnot získaných měřením, a také k uložení globálního nastavení programu, šablony protokolu, dostupných měřidel a jednotek s jejich vzájemnou konverzí. Většina XML souborů je uložena v adresáři *resource*, měření a uživatelská nastavení programu v domovském adresáři. Proto jsou při uložení a zpětném načítání postupu měření využívány již naimplementované metody přístupu k souboru. Nynější struktura XML (viz příloha A) vychází z původní struktury měření.

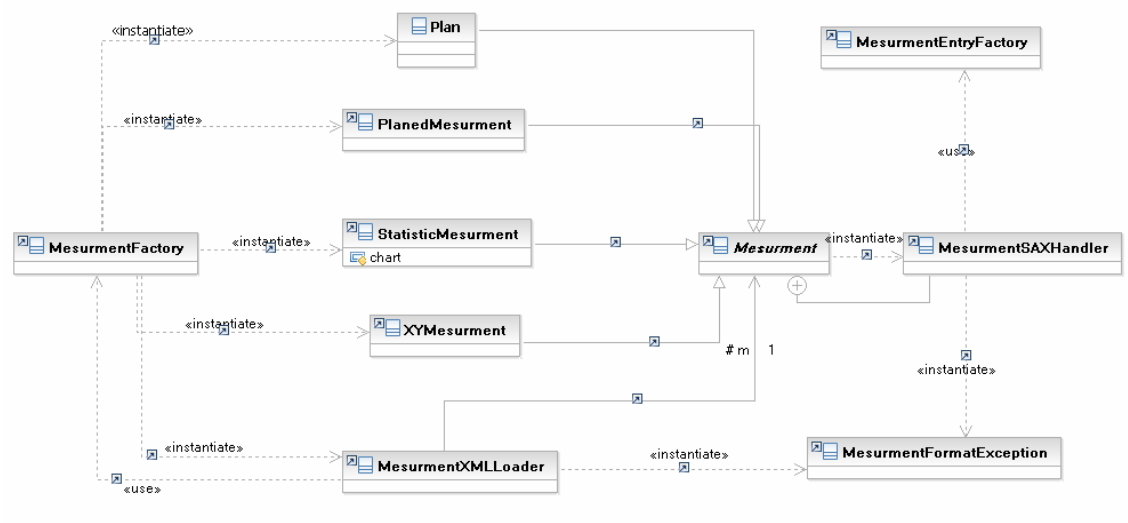
Statickou strukturu využití jednotlivých tříd při práci s XML souborem popisuje třídní diagram(obrázek 12). Hlavní třídou, obsluhující logiku práce s daty, je *StructuredXMLLoader* specializovaná do třídy *ClassXMLLoader*, jejíž úkolem je údaje načítat. V této třídě jsou překryty metody *startElement( )* a *endElement( )* volané při počátečním a koncovém tagu. Třída *Class*, načítající data, je předána jako odkaz nebo vytvořena pomocí *ClassXMLLoaderu*. Obsahuje vnitřní třídu *ClassSaxHandler*, které je předáno *parsování*. Jako vnitřní třída má přístup k privátním atributům a udržuje si odkaz na předchozí *parser* atributem *loader*, aby mohla načítání při koncovém tagu předat zpět.



Obrázek 12: Třídní diagram ClassXMLLoader

3.2.2 Plán měření - Plan

Třída *Plan* je podobného charakteru jako ostatní třídy reprezentující různé typy měření. Je ji možno považovat za hlavní třídu plánu měření, která poskytuje potřebné metody pro jeho nastavení a správu. O vytvoření instance třídy *Plan* i jednotlivých typů měření se stará třída *MesurmentFactory*. Použitím návrhového vzoru *Factory Metod pattern* (viz kapitola Návrhové vzory) dochází k dynamickému vytváření potřebné instance a při tom je klient odstíněn od výběru správné třídy. Výhodou je taktéž jednoduché přidání dalšího typu měření. Třída *MesurmentFactory* je podrobněji rozebrána v analýze programu UniSave, oddíl 1.2. Vztahy mezi popisovanými třídami znázorňuje následující třídní diagram (obrázek 13). Součástí třídy *Plan* je třída *PlanTableModel* představující model tabulky v GUI, pomocí níž je možno vkládat, editovat a odstraňovat záznamy v tabulce.



Obrázek 13: Třídní diagram Plan

Třída *Plan* rozšiřuje abstraktní třídu *Mesurement*, stejně jako každý typ měření. Představuje hlavní třídu editoru měření. K vytvoření její instance dochází prostřednictvím třídy *MesurmentFactory*. Hodnoty získané měřením nebo načtením z XML souboru jsou uloženy ve vektoru *Vector<MesurmentEntry> elements*. Ten je součástí třídy *Mesurement*, spolu s metodami přístupu k nim.

Plán měření obsahuje:

- *public void reindex( )* – přeindexuje všechny záznamy měření uložené ve vektoru *elements*
- *public void addEntry(MesurmentEntry e)* – vloží záznam měření do vektoru *elements*, v případě nastavení automatické konverze využije k převodu jednotky metodu *convert* datového typu *Value* a následně uloží. Převod se týká jmenovité hodnoty a tolerance.
- *public void addEntries(Collection<MesurmentEntry> e)* – vloží kolekci měřených záznamů do vektoru *elements*.
- *class PlanXMLWriter* – je vnitřní třídou, rozšiřuje třídu *MesurmentXMLWriter*, její metoda *public void storeMesurmentBody( )* ukládá atributy do XML souboru
- *class PlanSAXHandler* – tato vnitřní třída rozšiřuje třídu *MesurmentSAXHandler*, slouží pro načítání atributů z XML, obsahuje metodu *public void startElement (String uri, String localName, String qName, Attributes attributes)* definující načítané atributy, po načtení je přiřadí do příslušných proměnných
- *class PlanTableModel* – vnitřní třída, vytváří model tabulky pro hodnoty získané vytvářením plánu měření či načtením plánu ze souboru, tuto třídu využívá panel *PlaningValuePanel*. Obsahuje metody:
    - *public int getRowCount( )* – vrátí počet řádků tabulky
    - *public int getColumnCount( )* – vrátí počet sloupců tabulky, jejich počet je pevně nastaven v měření
    - *public Object getValueAt(int rowIndex, int columnIndex)* – získá hodnotu na požadované pozici v tabulce, vrácený objekt je typu *PlanValue* z vektoru *elements*

- *public String getColumnName(int column)* – vrací jméno sloupce tabulky s daným indexem, názvy sloupců jsou zde definovány
- metody pro vložení, editaci a smazání záznamů v tabulce

3.2.3 Datový typ - PlanValue

Datový typ *PlanValue* implementující rozhraní *MesurmentEntry* se nachází v balíčku *data.value* který zároveň obsahuje i další datové typy využívané pro zvolená měření. Při vytváření instancí těchto datových typů třídou *MesurmentEntryFactory* je opět použit návrhový vzor *Factory metod pattern* (viz kapitola Návrhové vzory). Obsahem datového typu jsou index, jméno, identifikátor, jmenovitá hodnota, tolerance, opakování, popis a obrázek, blíže popsané již na začátku této kapitoly.
Datový typ obsahuje:

- metody pro nastavení a získání indexu, jména, identifikátoru, jmenovité hodnoty, tolerance, počtu opakování. Hodnota jmenovité hodnoty a tolerance je číslo s jednotkou a předponou, proto jsou typu *Value*
- *public MesurmentEntry cloneInstance( )* – vytvoří kopii *PlanValue*
- *public void storeToXML(BufferedWriter w, int offset)* – slouží pro ukládání tohoto typu do XML souboru, definuje názvy jednotlivých značek XML
- *class PlanValueSAXHandler* – vnitřní třída pro načítání datového typu z XML souboru
  - *public void endElement(String uri, String localName, String qName)* – koncový element datového typu v XML souboru
  - *public void startElement(String uri, String localName, String qName, Attributes attributes)* – načítá postupně všechny elementy datového typu a nastavuje je

3.2.4 Hlavní okno - MainFrame

Třída *MainFrame* přestavuje hlavní okno aplikace. Obsah tohoto okna závisí na zvoleném typu měření. K vytváření jednotlivých instancí se používá návrhový vzor *Factory metod pattern* pomocí třídy *MesurmentPanelFactory* (viz kapitola Návrhové vzory). Po spuštění programu se zobrazí původní okno s logem aplikace bez zvoleného měření *NullMesurmentPanel*. Pomocí třídy *MesurmentFactory* je vytvořeno nové měření a podle jeho typu je přizpůsoben obsah hlavního okna. Třída *SettingDialog* poskytuje možnost k nastavení programu – přidávání a odebírání portů. Tato uživatelská nastavení se ukládají pomocí *GlobalSetting*. K zobrazení informací o programu se využívá třídy *About*.

Pro vytvoření nového plánu měření je nutné vyvolat akci příslušného tlačítka umístěného v panelu nástrojů nebo v menu třídy *MainFrame*. *NullMesurment*, které je zobrazeno po spuštění programu, je nahrazeno editorem *Plan*. Následně se uživateli zobrazí okno editoru postupu měření reprezentované třídou *MainFrame* s vloženým panelem editoru *PlanPanel*. Tato třída GUI zahrnuje všechny grafické komponenty editoru a je blíže popsaná v následující části. Panel *PlaningValues* obsahuje tabulku s hodnotami plánu měření a tlačítka obsluhující funkce spojené s touto tabulkou.

Akcí tlačítka *Otevřít* načteme soubor s plánem měření, zobrazí se okno dialogu, ve kterém je aplikován filtr pro zobrazení pouze souborů s příponou *unisave*. Pak dochází k načtení obsahu XML souboru s využitím třídy *SAXLoader*, ve kterém je i uložen typ měřeni, v našem případe *Plan*. Podle načteného typu se pomocí třídy *MesurmentFactory* vytvoří správný typ měření. S využitím datového typu *PlanValue*, reprezentujícího záznam v tabulce, dojde k načtení uložených hodnot do vektoru měření umístěného v třídě *Mesurment*, ze které každý typ měření dědí.

Hlavní okno obsahuje:

- Metody pro přístup a vytváření grafických komponent (menu, panel nástrojů, vnořené panely, atd.)
- Vnitřní třídy obsluhující akce načtení, otevření a uložení plánu i měření

3.2.5 Panel plánu měření – PlanPanel

Tento panel je hlavním panelem GUI plánu měření, je vložen do hlavního panelu měření *MainFrame*. Do tohoto panelu je umístěn *GrabberPanel*, který slouží pro načítání hodnot do plánu.

Panel plánu měření obsahuje:

- Metody pro vytváření všech komponent GUI tohoto panelu
- Vnitřní třídy pro obsluhu akcí tlačítek a dalších komponent panelu *PlanPanel*

3.2.6 Panel hodnot – PlaningValuePanel

Naměřené hodnoty se zobrazují v tabulce panelu *PlaningValuePlan*, který reaguje na změny nastavení měření. Po kliknutí do tabulky hodnot se zobrazí okno editace (*EditDialog*). Při načítání hodnot z měřidel je umožněno jejich automatické nebo manuální převádění. K výběru požadované jednotky slouží třída *UnitSelektionPanel*. Po zvolení ručního převodu jednotek se zobrazí dialog *UnitConversionDialog*, kterým můžeme převést jen vybrané nebo i všechny jednotky. Pro zjištění všech různých povolených konverzí se využívá metoda z třídy *UnitSet*.

Panel hodnot obsahuje metody:

- Metody pro vytváření jednotlivých panelů a komponent GUI zobrazované uvnitř panelu
- *public void editSelectedItem( )* – metoda pro editaci řádku v tabulce, edituje se datový typ PlanValue, k provedení změny údajů v tabulce využívá třídu *EditDialog* zobrazující editační okno

3.2.7 Panel hodnoty plánu – PlanValuePanel

PlanValuePanel poskytuje grafické komponenty hodnoty plánu, zobrazí se jako formulář s položkami pro vyplnění atributů datového typu *PlanValue*. Je využíván při zadávání

hodnot do plánu, a také při jejich editaci v tabulce. Implementuje rozhraní *MesurementEntityEditor* a k vytváření jeho instance je využívána třída *MesurementEntityEditorPanelFactory*.

Panel hodnoty plánu obsahuje metody:

- Metody vytvářející komponenty formuláře a poskytující přístup k nim
- Metody pro vytváření jednotlivých panelů

4 Měření podle postupu

Po nadefinovaném plánu vytvořeném editorem popsaným v předcházející kapitole následuje etapa měření dle postupu. Toto měření přidává k hodnotám nadefinovaným v postupu další položku – měřenou hodnotu. Zároveň také dopočítává odchylku od jmenovité hodnoty a zjišťuje, zda je měřená hodnota v toleranci. Při vkládání vypočítá hodnotu nadefinovanou vzorcem.

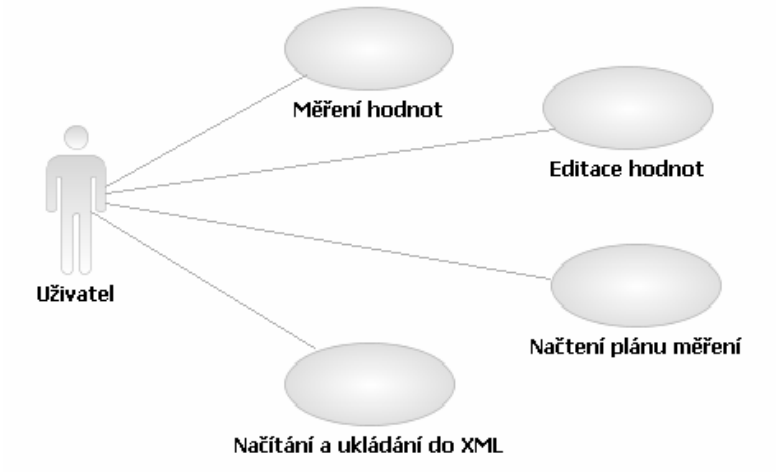
4.1 Specifikace požadavků

Pro měření podle plánu musíme mít tento plán vytvořen a uložen pomocí editoru. Když dojde k načtení souboru do samostatné tabulky, můžeme přistoupit k samotnému měření. Hodnoty lze zadávat buď manuálně vyplněním formuláře nebo přímo z připojeného vybraného měřidla. Pak dochází k načítání automaticky. Tyto hodnoty jsou zobrazovány v další tabulce, kterou můžeme po ukončení měření editovat. Při vkládání se naskýtá volba automatické indexace a převodu na zvolený typ jednotky

Měřená hodnota může mít svou jednotku s předponou. Odchylka od jmenovité hodnoty se dopočte automaticky, zároveň se zjišťuje, zda splňuje měřená hodnota danou toleranci. Při vložení vzorce proběhne jeho vyhodnocení výpočtem.

4.2 Analýza, návrh a řešení

Data, načtená z plánu měření, jsou doplněna o měřenou hodnotu. Je využita opět třída *Unit*, která obsahuje číselnou hodnotu s jednotkou a předponou. Po vložení hodnoty do formuláře se vzorcem dojde k výpočtu. Jednotlivé případy užití znázorňuje následující *use case* diagram (obrázek 14).



Obrázek 14:Use Case měření podle postupu

4.2.1 Měření podle plánu - PlanedMesurment

Podobnou strukturu jako ostatní třídy měření má třída *PlanedMesurment*. Rozšiřuje abstraktní třídu *Mesurment* a překrývá některé její metody. Vytvoření instance *PlanedMesurment* zajišťuje třída *MesurmentFactory*. V *PlanedMesurment* je vnořena vnitřní třída *PlanedTableModel*, které je využíváno pro zobrazení tabulky naměřených hodnot panelem *MesurmentValuePanelPlan*. Další vnitřní třídou modelující tabulku je *PlanTableModel*, ta slouží pro zobrazení hodnot načtených z plánu měření.

Tato třída, která dědí z abstraktní třídy *Mesurement*, reprezentuje hlavní třídu plánovaného měření. K vytvoření její instance je využíváno, jako i u ostatních typů měření, třídy *MesurmentFactory*. Hodnoty načtené z plánu měření jsou uloženy do vektoru *Vector<MesurmentEntry> planElements*. Záznamy získané měřením jsou uloženy ve vektoru *Vector<MesurmentEntry> elements*, který je definován ve třídě *Mesurement* spolu s metodami přístupu.

Měření podle plánu obsahuje:

- *Vector<MesurmentEntry> planElements* – vektor obsahující hodnoty typu *PlanValue*, načtené z plánu měření. Pro přístup a nastavení jsou vytvořeny *get* a *set* metody
- *MesurmentEntry getPlanElement(int index)* – získá hodnotu elementu zadaného indexu z vektoru *planElement*s
- *public void reindex( )* – přeindexuje všechny záznamy měření uložené ve vektoru *elements*
- *public void addEntry(MesurmentEntry e)* – vloží záznam měření do vektoru *elements*, v případě nastavení automatické konverze využije k převodu jednotky metodu *convert* datového typu *Value* a následně uloží
- *public void addEntries(Collection<MesurmentEntry> e)* – vloží kolekci měřených záznamů do vektoru *elements*
- *class PlanedMesurmentXMLWriter* – je vnitřní třídou, rozšiřuje třídu *MesurmentXMLWriter*, poskytuje metody:
  - *public void storeMesurmentBody( )* – ukládá atributy do XML souboru
- *class PlanedMesurmentSAXHandler* – tato vnitřní třída rozšiřuje třídu *MesurmentSAXHandler*, slouží pro načítání atributů z XML, poskytuje metody:
  - *public void startElement (String uri, String localName, String qName, Attributes attributes)* – metoda definující načítané atributy, po načtení je přiřadí do příslušných proměnných
- *class PlanedTableModel* – vnitřní třída, vytváří model tabulky pro hodnoty získané během samotného měření, zobrazení v GUI je prováděno třídou *MesurmentValuePanelPlan*, obsahuje metody:
  - *public int getRowCount( )* – vrátí počet řádků tabulky
  - *public int getColumnCount( )* – vrátí počet sloupců tabulky, jejich počet je pevně nastaven v měření

    - *public Object getValueAt(int rowIndex, int columnIndex)* – získá hodnotu na požadované pozici v tabulce, vrácený objekt je typu *NamedValue* z vektoru *elements*
    - *public String getColumnName(int column)* – vrací jméno sloupce tabulky s daným indexem, názvy sloupců jsou zde definovány
    - metody pro vložení, editaci a smazání záznamů v tabulce
- *class PlanTableModel* – vnitřní třída, která definuje model tabulky pro hodnoty načtené z plánu měření, její metody jsou obdobné třídě *PlanedTableModel*, k zobrazení dochází pomocí panelu *MesurmentPanelPlan*

4.2.2 Datový typ - NamedValue

V balíčku *data.value* je umístěn datový typ *NamedValue* implementující rozhraní *MesurmentEntry*. Reprezentuje údaje získané v měření podle plánu. Je hlavním datovým typem třídy *PlanedMesurment*. K vytvoření jeho instance slouží třída *MesurmentEntryFactory*. Obsahem tohoto datového typu jsou index, jméno, identifikátor, jmenovitá hodnota, měřená hodnota, tolerance, popis a obrázek, které jsou blíže popsané již na začátku této kapitoly.

Jeho součástí je vnitřní třída *PlanValueSAXHandler*, která slouží pro načítání tohoto typu z XML. Pro ukládání využíváme metodu *storeToXML*.

Datový typ obsahuje:

- metody pro nastavení a získání indexu, jména, identifikátoru, měřené hodnoty, jmenovité hodnoty a tolerance. Hodnota měřené hodnoty, jmenovité hodnoty a tolerance je číslo s jednotkou a předponou, proto jsou typu *Value*
- *public MesurmentEntry cloneInstance( )* – vytvoří kopii *NamedValue*
- *public void storeToXML(BufferedWriter w, int offset)* – slouží pro ukládání tohoto typu do XML souboru, definuje názvy jednotlivých značek XML
- *class PlanValueSAXHandler* – vnitřní třída pro načítání datového typu z XML souboru s metodami:
    - *public void endElement(String uri, String localName, String qName)* – koncový element datového typu v XML souboru
    - *public void startElement(String uri, String localName, String qName, Attributes attributes)* – načítá postupně všechny elementy datového typu a nastavuje je

4.2.3 Panel měření podle postupu – MesurmentPanelPlan

Po vytvoření nového měření je původní *NullMesurmentPanel* nahrazen panelem *MesurmentPanelPlan*. Funkce jednotlivých položek menu (protokol, editace a zobrazit) jsou definovány ve vnitřních třídách ovládacích prvků. Panel je tvořen záložkou měření a v dalším rozšíření přibudou další záložky nazvané graf a protokol. Záložka měření je záložka s nejvyšším počtem funkcí. Skládá se z panelu hodnot *MesurmentValuePanelPlan*, panelu *DoubleDeviceGrabberPanel* výběru měřidla a portu.

MesurementPanelPlan je hlavním panelem GUI měření podle postupu, je vložen do hlavního panelu měření *MainFrame*. Do tohoto panelu je umístěn *GrabberPanel*, který slouží pro načítání měřených hodnot, poskytuje možnost volby portu a typu měřidla, včetně ručního zadávání hodnot. Pro načtení hodnot z plánu měření slouží akce, která po zmáčknutí tlačítka *jBLoadPlanValues* pomocí *FileDialogu* načte data do tabulky plánovaných hodnot.

Panel měření obsahuje:

- Metody pro vytváření všech komponent GUI tohoto panelu
- Vnitřní třídy pro obsluhu akcí tlačítek a dalších komponent

4.2.4 Panel hodnot – MesurmentValuePanelPlan

Panel *MesurmentValuePanelPlan* je vytvářen hlavním panelem měření podle plánu třídou *MesurmentPanelPlan*. Obsahuje tabulku měřených hodnot, kterou získává z modelu vnitřní třídy *PlanTableModel* umístěné ve třídě *PlanedMesurment*. Zde jsou načítány a zobrazovány naměřené hodnoty. Po načtení nové hodnoty z měřidla nebo ručním zadáním je umožněna automatická konverze na jednotku zvolenou z nabídky tohoto panelu pomocí třídy *UnitSelektionPanel*. Pro zjištění všech různých povolených konverzí se využívá metoda z třídy *UnitSet*. V panelu je možná volba automatické indexace při vkládání záznamu a stejně tak volba reindexace.

Panel hodnot obsahuje metody:

- Metody pro vytváření jednotlivých panelů a komponent GUI zobrazované uvnitř panelu
- *public void editSelectedItem( )* – metodu pro editaci řádku v tabulce, edituje se datový typ *NamedValue*, využívá třídu *EditDialog* zobrazující editační okno

4.2.5 Panel hodnoty měření podle plánu – NamedValuePanel

NamedValuePanel poskytuje grafické komponenty hodnoty měřené podle plánu, vzhledem připomíná formulář, kdy jeho část (položka pro vyplnění) odpovídá atributu datového typu *NamedValue*. Je využíván při ručním vkládání hodnot při měření podle plánu, a také při editaci naměřených hodnot v tabulce. Implementuje rozhraní *MesurementEntityEditor* a k vytváření jeho instance je využívána třída *MesurementEntityEditorPanelFactory*.

Panel hodnoty měření podle plánu obsahuje metody:

- Metody vytvářející komponenty formuláře a poskytující přístup k nim
- Metody pro vytváření jednotlivých panelů

4.2.6 Překladač

Pro vyhodnocení zadaného vzorce načteného z plánu měření je nutné zkonstruovat překladač, který vzorec vyhodnotí dosazením hodnot za jednotlivé proměnné a vypočítá výsledek. Při tomto výpočtu jsou uplatňovány matematické zákonitosti.

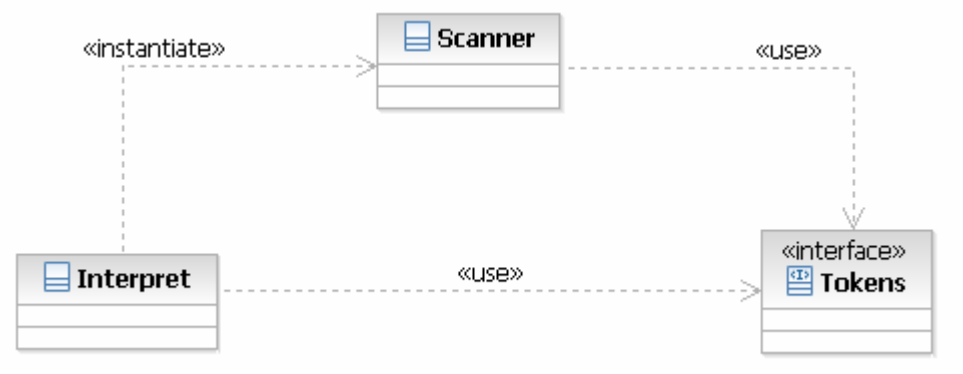
Při konstrukci překladače je nezbytné nejprve správně navrhnout jeho regulární bezkontextovou gramatiku, kterou bude vyhodnocen požadovaný vzorec.

Gramatika vzorce:

1. E : T E1;
2. E1 : '+' T E1
3. | '-' T E1
4. | {e};
5. T : F T1;
6. T1 : '*' F T1
7. | '/' F T1
8. | {e}
9. F : '(' E ')
10. | num | variable

Tato gramatika začíná startovacím neterminálem E, upřednostňuje operaci násobení a dělení před sčítáním a odčítáním, vyhodnocuje nejprve výrazy v závorkách. Jejím koncovým terminálem je číslo nebo proměnná zastupující naměřenou hodnotu.

Statickou strukturu překladače zachycuje následující třídní diagram (obrázek 15). V samotné implementaci překladače jsou vytvořeny třídy *Interpret* , *Scanne*r a rozhraní *Tokens* a využívá se metody rekurzivního sestupu. Rozhraní *Tokens* definuje všechny terminální symboly jako konstanty. Třída *Scanne*r provádí lexikální analýzu načteného vzorce a produkuje *tokeny*. Její metody implementují jednotlivá pravidla gramatiky. Třída *Interpret* vykonává syntaktickou analýzu s překladem a interpretací.



Obrázek 15: Třídní diagram překladače

Závěr

Výsledkem této práce, nazvané Editor postupu měření, je rozšíření programu UniSave o další funkce. Vytvořil jsem editor postupu, který umožňuje uživateli nadefinovat jednotlivé kroky postupu měření výrobku, což zahrnuje možnost definice požadovaných hodnot, opakování a možnost vložení vzorce. Vytvořený plán definuje jednotlivé sousledné kroky měření s postupem, ke každému kroku určuje související atributy sloužící pro další možné výpočty a vlastnosti napomáhající uživateli orientaci při samotném měření. Samozřejmostí je ukládaní, editace a načítání plánu pomocí souboru XML.

Po nadefinovaném plánu vytvořeném editorem popsaném ve třetí kapitole následuje etapa měření dle postupu. Toto měření přidává k hodnotám nadefinovaným v postupu další položku – měřenou hodnotu. Zároveň také dopočítává odchylku od jmenovité hodnoty a zjišťuje, zda je měřená hodnota v toleranci.

Protože zdrojový kód programu UniSave nemá téměř žádné komentáře, byla orientace v jeho rozsáhlé struktuře ztížena. K některým používaným částem kódu jsou v této práci proto komentáře k možnosti dalšího případného rozšíření tohoto programu doplněny.

Naskýtá se možnost rozšíření programu např. o zobrazování grafu a vytváření protokolu v měření podle plánu nebo nahrazení přístupu k XML souboru pomocí SAX dokonalejší technologií XStreme, která již pracuje s objekty.

Literatura

1. Vondrák Ivo, Metody byznys modelování, Ostrava, 2004
2. Vondrák Ivo, Kožušník Jan, Ochodková Eliška, Metody specifikace softwarových systémů, Ostrava, 2006
3. Vondrák Ivo, Úvod do softwarového inženýrství, Ostrava, 2002
4. Sun, Java, http://www.sun.com
5. Wikipedia: Překladač, http://cs.wikipedia.org/wiki/Překladač
6. Wikipedia: XML, http://cs.wikipedia.org/wiki/XML
7. ABClinuxu: Jazyky a překladače, http://www.abclinuxu.cz/serialy/jazyky-a-prekladace
8. Baranec Radovan, Refaktoring a testovanie aplikácie UniSave, bakalářská práce, Ostrava, 2008
9. Beneš Miroslav, Překladače, skripta

A Schémata

Tato část přílohy znázorňuje strukturu XML souboru plánu měření a také souboru užívaném v měření podle postupu.

A.1 Struktura XML souboru postupu měření

Po vytvoření postupu měření se hodnoty a zvolení akce uložení, hodnoty obsažené ve vektoru *elements* vloží do XML souboru. Pro náhled na strukturu jednoho záznamu plánu slouží následující část XML souboru.

Příklad struktury XML souboru s vytvořeným postupem:

```
<mesurmentEntries>
    <mesurmentEntry type="7">
        <index value="0"/>
        <name value="3"/>
        <id value=""/>
        <repeat value="1"/>
        <nominalValue>
            <mesurmentEntry type="0">
                <index value="0"/>
                <value value="3.0"/>
                <unit prefix="0" description="0"/>
            </mesurmentEntry>
        </nominalValue>
        <toleranc>
            <mesurmentEntry type="0">
                <index value="0"/>
                <value value="3.0"/>
                <unit prefix="0" description="0"/>
            </mesurmentEntry>
        </toleranc>
        <descr value="3"/>
        <picture value="33"/>
    </mesurmentEntry>
</mesurmentEntries>
```

A.2 Struktura XML souboru měření podle postupu

K uložení hodnot získaných v měření podle postupu je využíván XML soubor, který obsahuje jednotlivé záznamy měření. Strukturu tohoto záznamu zachycuje následující fragment XML souboru.

Struktura XML souboru s vytvořeným postupem:

```
<mesurmentEntries>
    <mesurmentEntry type="1">
        <index value="1"/>
        <name value="delka a"/>
        <id value=""/>
        <mesurmentValue>
            <mesurmentEntry type="0">
                <index value="0"/>
                <value value="10.0"/>
                <unit prefix="-2" description="2"/>
            </mesurmentEntry>
        </mesurmentValue>
        <nominalValue>
            <mesurmentEntry type="0">
                <index value="0"/>
                <value value="11.0"/>
                <unit prefix="-2" description="2"/>
            </mesurmentEntry>
        </nominalValue>
        <toleranc>
            <mesurmentEntry type="0">
                <index value="0"/>
                <value value="10.0"/>
                <unit prefix="-3" description="2"/>
            </mesurmentEntry>
        </toleranc>
        <descr value="Mereni strany a kvadru."/>
        <picture value=""/>
    </mesurmentEntry>
</mesurmentEntries>
```

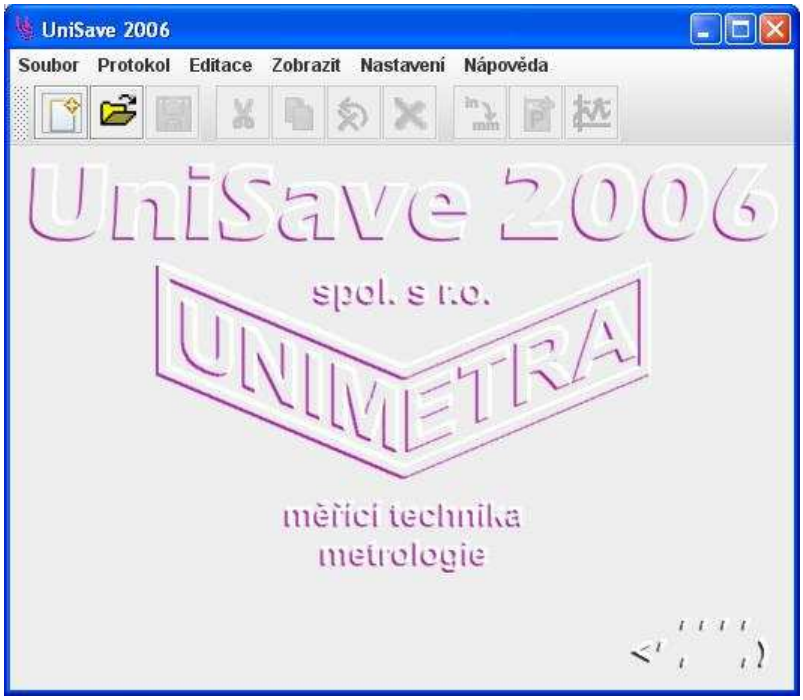
B Uživatelská příručka

Nezbytné kroky instalace a spuštění programu jsou popsány v první části uživatelské příručky. Následuje přehledná grafická příručka vytvořeného editoru postupu měření a měření podle postupu.

B.1 Instalace a spuštění programu UniSave

Z CD přiloženého k této práci zkopírujeme obsah složky *UniSave*. Jestliže ještě nemáme nainstalováno JRE (Java Runtime Enviroment) verze 1.6, je volně k dispozici na stránkách Sun[4]. V proměnných prostředí je pak nutné nastavit cestu k adresáři nainstalovaných souborů. Nastavení v OS MS Windows provedeme přes *ovládací panel*, dále zvolíme položku *systém* a v ní záložku *upřesnit*.

Samotné spuštění programu UniSave lze provést z příkazové řádky zadáním příkazu *java –jar [cesta k souboru] /UniSave2006.jar* nebo kliknutím myši na tento soubor. Poté je zobrazena úvodní obrazovka (obrázek 16).

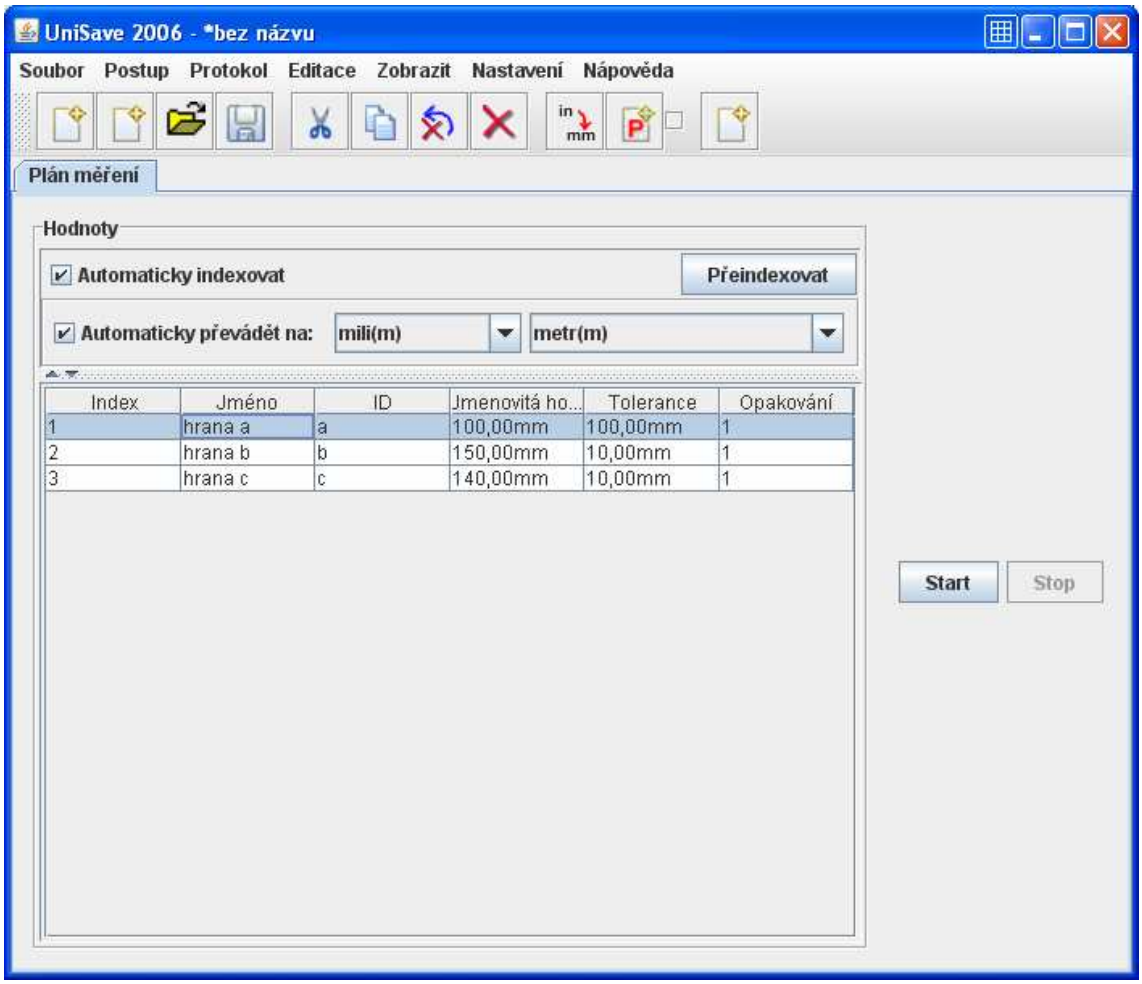


Obrázek 16: Úvodní obrazovka programu UniSave

B.2 Editor postupu měření

B.2.1 Nový postup měření a vkládání hodnot

Po spuštění programu UniSave a zobrazení úvodní obrazovky (obrázek 16), zvolíme tlačítko vytvoření nového postupu v panelu nástrojů. Také můžeme volbu provést pomocí položky *Nový postup* v menu Postup. V okně se zobrazí nový panel postupu (obrázek 17), ve kterém nejsou žádné položky.



Obrázek 17: Editor postupu měření

Pokud nechceme, aby se nám index při vkládání hodnot automaticky zvyšoval, odtrhneme políčko *Automaticky indexovat*. Pro zadání nových hodnot postupu zvolíme tlačítko *Start*, následně se objeví dialogové okno (obrázek 18). Vyplníme jednotlivé položky formuláře, zvolíme tlačítko *Přidat* a následně se záznam přidá do tabulky v původním okně. Formulář byl vynulován a můžeme vkládat další záznam. Jakmile zvolíme tlačítko *Stop*, dialogové okno se

zavře. Tlačítkem *Reindex* lze přeindexovat všechny hodnoty indexu v tabulce. Dále můžeme provádět akce popsané v následující části.

Obrázek 18: Okno vkládání hodnot plánu

B.2.2 Uložení, otevření, editace postupu

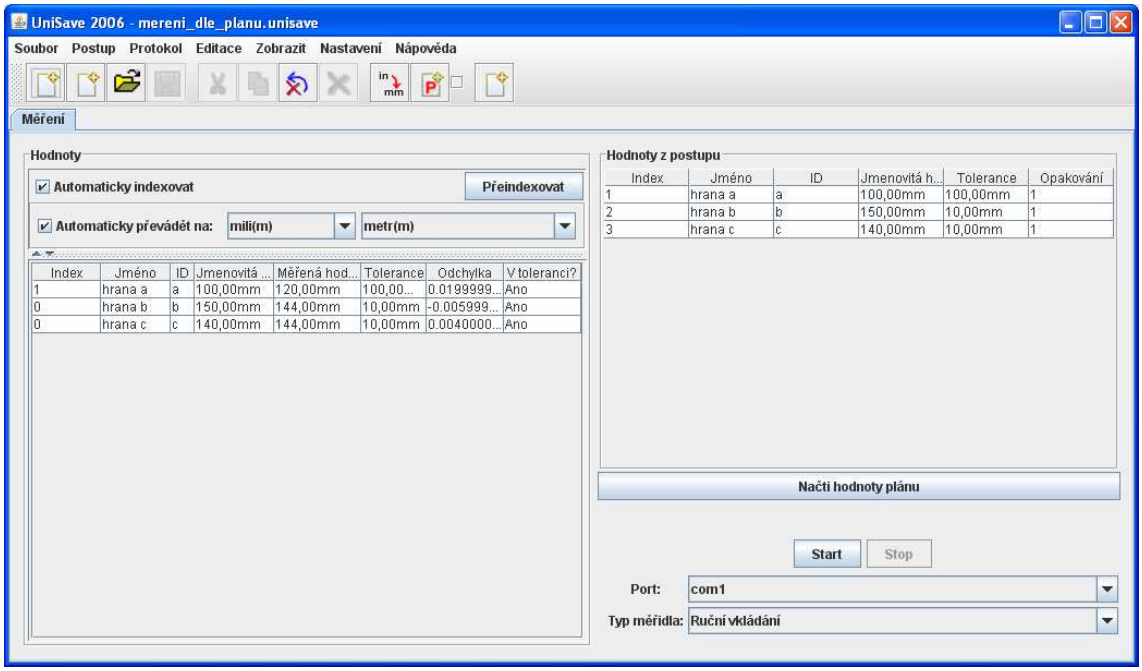
Vytvořený plán měření můžeme uložit do XML souboru, automaticky je zvolena přípona *unisave*, použitá již při ukládání měření. Tuto akci provedeme tlačítkem panelu nástrojů nebo pomocí položky v menu. Vyplněním klasického dialogu uložíme plán do souboru. Načtení souboru provedeme tlačítkem *Otevřít* a vybráním vhodného souboru.

Hodnoty v tabulce lze editovat poklepáním na editovaný záznam, následně je otevřeno dialogové okno podobné dialogu vkládání (obrázek 18). Řádky v tabulce lze mazat a kopírovat vybráním řádku a volbou příslušných tlačítek v panelu nástrojů.

B.3 Měření podle postupu

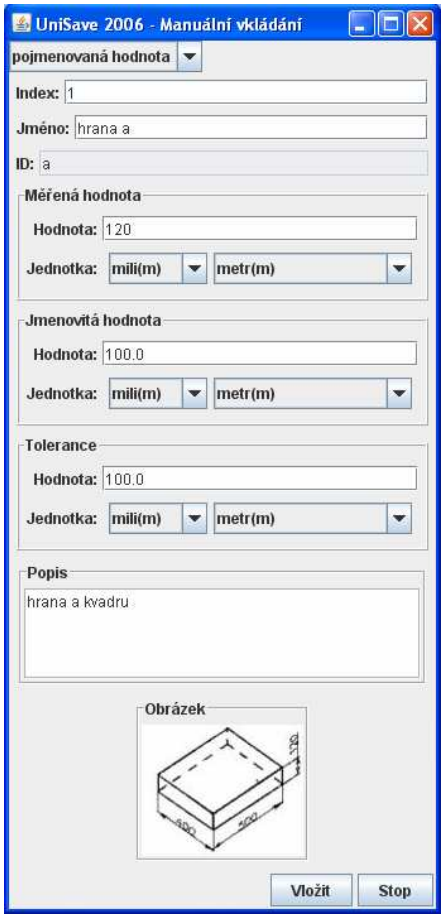
B.2.1 Nové měření podle postupu

Chceme-li vytvořit nové měření podle plánu, zvolíme tlačítko vytvoření nového postupu v panelu nástrojů. Také můžeme volbu provést pomocí položky *Měření podle postupu* v menu *Postup*. V okně se zobrazí nový panel postupu (obrázek 19), ve kterém nejsou žádné položky. Pro načtení plánu, vytvořeného pomocí editoru postupu měření, slouží tlačítko *Načtení plánu měření*. Zobrazí se dialogové okno, ve kterém zvolíme vhodný plán měření. Následně ze data načtou do připravené tabulky.



Obrázek 19: Měření podle postupu

Jestliže je nežádoucí, aby se nám index při vkládání hodnot automaticky zvyšoval, odtrhneme políčko *Automaticky indexovat*. Pak vybereme vhodný typ měřidla s příslušným portem a klikneme na tlačítko *Start*. V případě načítání z měřidla dochází k vkládání do tabulky automaticky akcí měřidla. Pokud jsme zvolili ruční zadávání hodnot, následně se objeví dialogové okno (obrázek 20), jeho hodnoty jsou již předem vyplněny hodnotami z plánu. Doplníme pouze měřenou hodnotu, zvolíme tlačítko *Přidat* a následně se záznam přidá do tabulky v původním okně. Formulář byl obnoven další položkou plánu a můžeme vkládat další záznam. Jakmile zvolíme tlačítko *Stop*, dialogové okno se zavře. Tlačítkem *Reindex* lze přeindexovat všechny hodnoty indexu v tabulce. Dále můžeme provádět akce popsané v následující části.



Obrázek 20: Okno vložení měřené hodnoty

B.2.1 Uložení, otevření, editace měření podle postupu

Naměřené hodnoty zobrazené v tabulce můžeme uložit do XML souboru, tak jako u ostatních typů měření. Uložení provedeme tlačítkem panelu nástrojů nebo pomocí položky v menu. Vyplněním dialogu uložíme plán do souboru. Automaticky je zvolena přípona *unisave*, užívaná programem UniSave. Načtení souboru provedeme tlačítkem *Otevřít* a vybráním vhodného souboru s měřením podle postupu.

Pro editaci hodnot v tabulce poklepeme na editovaný řádek tabulky, následně je otevřeno dialogové okno podobné dialogu vkládání (obrázek 20). Funkce pro práci s tabulkou jsou dostupné v panelu nástrojů nebo v menu *Edit*.

C Obsah CD

Přílohou této práce je CD s textem diplomové práce, zdrojovými kódy aplikace a softwarem potřebným pro spuštění programu UniSave a načtení projektu s implementací.

Struktura CD:

- /text – text a zadání diplomové práce
- /install – instalační soubory
- /UniSave – zdrojové kódy aplikace UniSave
- /ukazkova_data – vzorová data měření a postupu měření