Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Programový systém pro sledování cen komodit

Plzeň, 2012 Bc. Miroslav Hauser

ZADÁNÍ

Poděkování

Na tomto místě bych rád poděkoval zejména vedoucímu své diplomové práce Prof. Ing. Karlu Ježkovi, CSc. za jeho podporu, trpělivost a mnoho cenných rad při vedení diplomové práce. Dále bych rád poděkoval svým rodičům za morální a finanční podporu v době studia a partnerce Karolínce za vše (obzvláště vydatné snídaně).

Prohlašuji, že jsem diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne

Abstract

Nowadays customers are forced to pay attention to costs they pay and to be careful about their investments also. This thesis describes the way how to inform customer about changing prices. The main aim is to implement programmatic commodity market checker that offers users (customers) to define about which commodity they want to be informed and from which particular source. The implementation will be generic enough to offer users possibility to check any volatile source defined by price (currency pairs, stocks, etc.).

Obsah

1 Úvod

V dnešní době je, více než kdy jindy, důležité mít správné informace ve správný čas. Informace nabývají na důležitosti, pokud jejich hodnotu vyvažujeme zdravím, ale také penězi. Již čtyři roky se zajímám o akciové a devizové trhy, jakožto rizikovou část portfolia a z tohoto důvodu mě dané téma zaujalo a chtěl jsem jej realizovat.

Cílem práce je implementovat robota pro sledování cen komodit.

Pro udržení logické návaznosti je práce rozdělena na část teoretickou (Teoretická část), poskytující teoretický základ pro pochopení problematiky. A část realizační, pojmenovanou Programový systém pro sledování cen komodit, pojednávající o samotné realizaci s cílem splnění zadání.

2 Teoretická část

2.1 Trhy komodit

2.1.1 Komodita a komoditní trh

„Komodity jsou suroviny, jako například oves, kukuřice, vepřové maso, ale i zlato, bavlna, ropa. Při komoditním obchodování se, na rozdíl od obchodování akciového, neobchoduje s podíly určité společnosti, ale právě s těmito surovinami. Každá ze surovin se pak obchoduje s určitým termínem dodání. Můžeme si proto již nyní koupit například kukuřici s dodáním v září příštího roku. Při obchodu je uzavřena smlouva - takzvaný „futures kontrakt", který v případě nákupu zavazuje prodávajícího dodat kupujícímu po vypršení termínu smlouvou dané množství kukuřice. Kupující je naopak povinen tuto kukuřici odebrat za cenu stanovenou při uzavření smlouvy. „ [4].

Toto „ujednání“ formou futures kontraktu je výhodné jak pro obchodníka, tak pro kupujícího. Obchodníkovi zajišťuje odbyt za minimální cenu a kupujícímu zajišťuje nákup za maximální cenu.

Dalším prostorem k využití kontraktů je jejich obchodování. V případě, že již není kontrakt pro jednu se stran výhodný, lze jej přeprodat jinému subjektu, který z něj může mít užitek.

Tento princip přeprodání, dává přirozeně majiteli kontraktu prostor k zisku v případě, že

cena komodity stoupne. Kontrakt lze poté přeprodat za vyšší cenu. Prostor k zisku nabízí kontrakty i v případě, že cena komodity klesá. Je možné nabídnout kontrakt na prodej komodity za cenu X a po poklesu ceny komodity na $Y<X$ lze koupit kontrakt za cenu Y, kterým uspokojíme potřebu v rozsahu původního kontraktu za cenu X. Zisk je poté,

$$ZISK = X - Y - poplatky$$

, kde poplatky zahrnují jak poplatky za provedení jednotlivých operací na trhu, tak i daně.

2.1.2 Práce s trhem

Základem rozhodnutí zda-li koupit, či prodat, komoditu (devizu, akcii, …) je buďto její potřeba, nebo spekulace s vývojem její ceny. V případě spekulace potřebujeme informace, na základě kterých budeme schopní provést dané rozhodnutí s určitou, námi požadovanou, mírou pravděpodobnosti.

Při práci s trhem existují dva základní přístupy jeho analýzy a to,

- fundamentální analýza a
- technická analýza.

Fundamentální analýza

Cílem fundamentální analýzy je sledovat a následně správně odhadnout pohyb hodnoty komodit (deviz, akcií, …), dle reakcí trhu na nová finanční a ekonomická data. Různé trhy jsou ovlivněné různými faktory s různými návaznostmi.

Cílem fundamentální analýzy je stanovit takovou cenu (tzv. vnitřní cena), za kterou by se daná komodita měla prodávat. Ke stanovení této ceny jsou využity veřejně přístupné informace.

Základními daty určujícími cenu jsou:

- ekonomická data,
- politická data,

- historická data,
- ale také živelné pohromy, či jiné nahodilé události.

Kupujícím (investorem) určená cena je poté porovnána s aktuální cenou a dle srovnání jsou přijata odpovídající opatření.

Technická analýza

Technická analýza se používá pro určení hodnoty komodity (akcie, devizy, …) na základě systematického zkoumání a analýzy historických a aktuálních dat.

Technická analýza staví na předpokladu, oproti fundamentální analýze, že veškeré vlivy, ve fundamentální analýze uvažované, jsou již zahrnuté v ceně. V technické analýze není cílem stanovení vnitřní ceny a její srovnání s aktuální cenou ale predikce chování trhu na základě vzorců a historických dat.

Technická analýza staví následujících předpokladech,

- veškerá data a informace jsou obsažené v ceně,
- historie se opakuje,
- ceny se pohybují v trendech.

V technické analýze se využívá mnoha nástrojů poskytujících různé informace. Základem je však samotný graf vývoje ceny. Na obrázku Obr. 2.1.2.1 vidíme příklad grafu devizového páru USD/EUR s aplikovanými ukazateli technické analýzy.

Konkrétními ukazateli mohou být[7]:

- klouzavé průměry,
- MACD,
- index relativní síly,
- hladina podpory,
- hladina odporu,

- atd

Konkrétních metod je nepřeberné množství a další metody stále přibývají[7].

Obr. 2.1.2.1: Graf komodity (zlato) s MACD

2.1.3 Softwarové nástroje

Práci s trhem nám usnadňují nástroje, poskytující jak informace o stavu trhu, tak základní i pokročilé analýzy.

Nejpoužívanější nástroje současné doby na českém trhu jsou [16][17][18],

- MetaTrader,
- NinjaTrader a
- e-Brooker.

První dva nástroje (MetaTrader a NinjaTrader) jsou dostupné jako tlustý klient instalovatelný na PC a poslední nástroj (e-Brooker) je tenký klient přístupný z internetu. Všechny nástroje umožňují fiktivní obchodování na zkoušku v demo účtu.

2.2 Dolování dat (data-mining)

„Dobývání znalostí z databází (KDD)[14] lze definovat jako netriviální získávání implicitních, dříve neznámých a potenciálně užitečných, informací z dat.“[11]. Pojem „Data mining“ přišel okolo roku 1990[12] i když oblast výběru informací z rozsáhlých bází dat má daleko delší historii. Tímto rokem počal rozsáhlý rozvoj metod a metodik dolování dat. Hnacím motorem byl a stále je komerční sektor[14].

V dnešní době se dolování dat využívá v oblastech jako jsou,

- marketing,
- managerské informační systémy (MIS)
- business inteligence (BI),
- analýze genetické informace,
- bezpečnostní analýza v počítačové síti,
- monitorování aktivit na internetu,
- atd.

Každá z výše uvedených oblastí vyžaduje unikátní přístup ke získání informací. Historicky se dolováním dat zabývalo více subjektů, akademického i komerčního charakteru, čímž vznikla sada přístupů (metodik), z nichž nejdůležitější v dnešní době jsou,

- KDD,
- SEMMA a
- CRISP-DM.

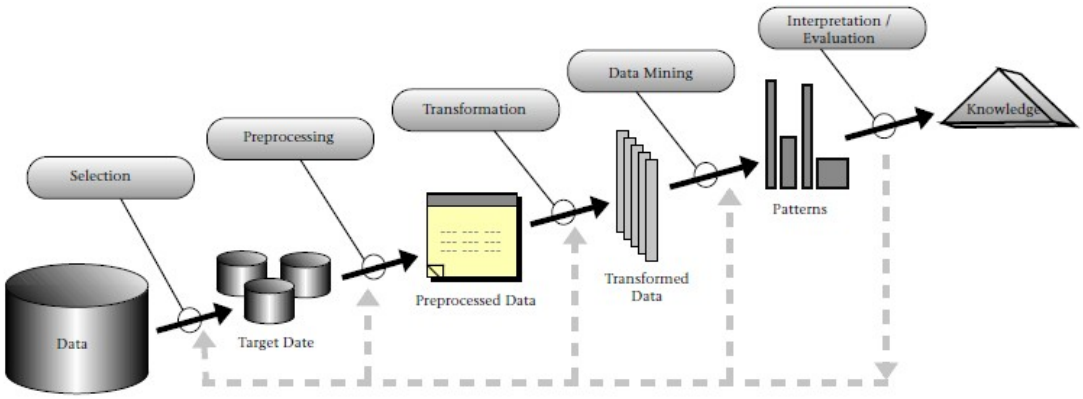
Většina metodik však využívá základních kroků vedoucích k získání informace[13][14]. Tyto kroky metodiky dále rozšiřují.

2.2.1 Základní kroky

Základní kroky (procesy) většiny metodologií jsou[13][14][11],

1. selekce (selection)
2. předzpracování (preprosessing)
3. transformace (transformation)
4. dolování dat (data mining)
5. interpretace a zhodnocení (interpretation / evaluation)

Posloupnost procesů je znázorněna na obrázku Obr. 2.2.1.1.



Obr. 2.2.1.1: Posloupnost procesů vedoucích ke získání informace

Úkolem procesu selekce vyhledat a připravit data vhodná pro následnou analýzu.

Proces předzpracování, zajišťuje úpravu dat s cílem získat významová konzistentní data.

Transformace zajišťuje úpravu dat s cílem snížit jejich rozsah, za pomoci redukce dimenzí[15], nebo transformačních metod.

Proces dolování dat zajišťuje samotné vyhledání informace v datech. K vyhledání informace lze použít řadu metod jako jsou statistické modely, genetické programování, logistická regrese atp.

Proces interpretace a zhodnocení je zodpovědný za uvedení informací do praxe, ať už se jedná o jejich zobrazení uživateli na portálu MIS, nebo jejich přímé uvedené do praxe. Součástí procesu je i měření relevantnosti a přesnosti získaných informací.

2.2.2 Dolování dat z WWW

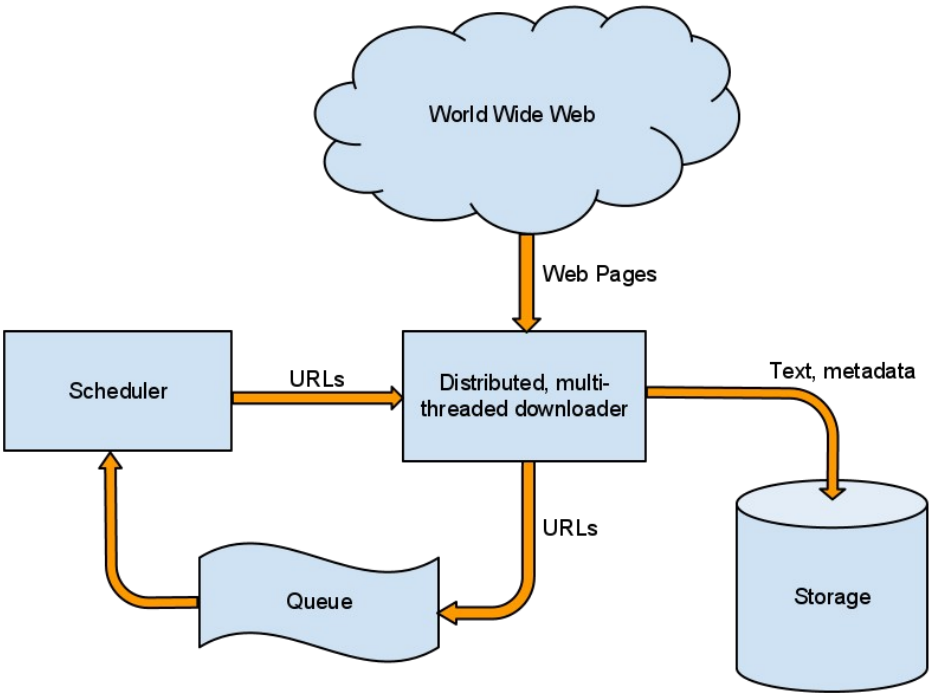
Dolování dat z WWW stránek (někdy také nazývané „web scraping“, „web harvesting“, „web data extraction“ a někdy i „web wrapping“, nebo „web crawling“) je souhrn technik používaných pro získání informací z webových stránek[6].

Pro automatizované stahování (dolování dat) z internetu se používají tzv. roboty. Jedná se o specializované programy, procházející WWW stránky, stahující z nich data a uchovávající informace pro navigaci mezi WWW stránkami.

Na obrázku Obr. 2.2.2.1 lze vidět schéma standardního distribuovaného robota. Standardní robot obsahuje následující komponenty,

- frontu úloh,
- plánovač,
- mnoho-vláknový stahovač dat a
- úložiště dat.

Tyto prvky robota jsou základními stavebními prvky. Dalšími komponentami, které může robot obsahovat jsou komponenta pro analýzu získaných dat, zpětnovazební komponenta pro nastavení robota, atp.



Obr. 2.2.2.1: Schéma robota (web-crawler)

Každý robot využívá konkrétní prohledávací strategii. Ze studie[10] vyplynulo, že nejpoužívanější jsou,

- **Prohledání primárně do šířky**. Robot prohledávající primárně do šířky upřednostňuje získávání nových zdrojů informací, před získáním informce samotné.
- **Opětovné prohledání zdrojů.** Některé roboty opětovně prohledávají již prohledané zdroje což jim umožňuje aktualizovat zastaralé informace. Tato vlastnost je realizována na základě heuristického algoritmu, který robotovi pomůže rozhodnout, které stránky je relevantní opětovně prohledat.
- **Zaměřené prohledávání.** Některé roboty si vybírají, které WWW stránky jsou pro ně relevantní. Stejně jako v předchozím bodě i zde se využívá heuristických algoritmů.
- **Náhodné procházení**. Opakem předchozího bodu je náhodné procházení po

grafu WWW stránek.

- **Procházení „skytého webu“**. Tato strategie prohledávání se opírá o fakt, že „zajímavá“ data nemusí být prezentována přímo na WWW stránce, ale mohou se nacházet např. v databázi za WWW stránkou. Tyto data lze získat například dotazy nad danou stránkou (vyhledávací formuláře, dialogy, atp.).

Kromě dobré vyhledávací strategie musí mít kvalitní robot odpovídající vlastnosti z pohledu softwarové architektury. Dle studie[10] jsou těmito vlastnostmi,

- **Flexibilita**, tedy schopnost býti použit ve více možných scénářích.
- **Nízká cena s vysokým výkonem**. Tento požadavek lze zjednodušit termínem **efektivita**. Robot musí být schopný prohledat maximální množství stránek za minimální čas s minimálním vytížením systému.
- **Robustnost** je standardním požadavkem na software. Tím, že robot pracuje s různými WWW stránkami v různých kódováních, formátech, provozovaných na různých systémech s rozdílnými implementacemi a chováním, je nezbytné aby byl dosti robustní a poradil si i s neočekávaným.
- **Etiketa a rychlost stahování** je komplexní požadavek na slušné chování a ohleduplnost. Při HTML dotazu se klient identifikuje v softwarem, který používá (který se dotazuje serveru). Je slušnost uvádět v tomto elementu název robota. Webové stránky tak mohou pomocí „Robots.txt“ řídit chování robota. Stejně tak není vhodné dotazovat se opakovaně a rychle jednoho serveru na data. Může takto dojít k zakázání přístupu k serveru (známé jako „ban“), což v době sdílených IP adres a internetových bran není moc ohleduplné.
- **Řiditelnost a konfigurovatelnost** je přirozeným požadavkem na možnost řídit chování robota. Uživatel (administrátor) robota musí mít možnost řídit jeho rychlost, kontrolovat stav prohledávání, povolovat, nebo zakazovat prohledávání konkrétních zdrojů informace.

3 Programový systém pro sledování cen komodit

Následující sekce popisuje praktické řešení teoretického problému.

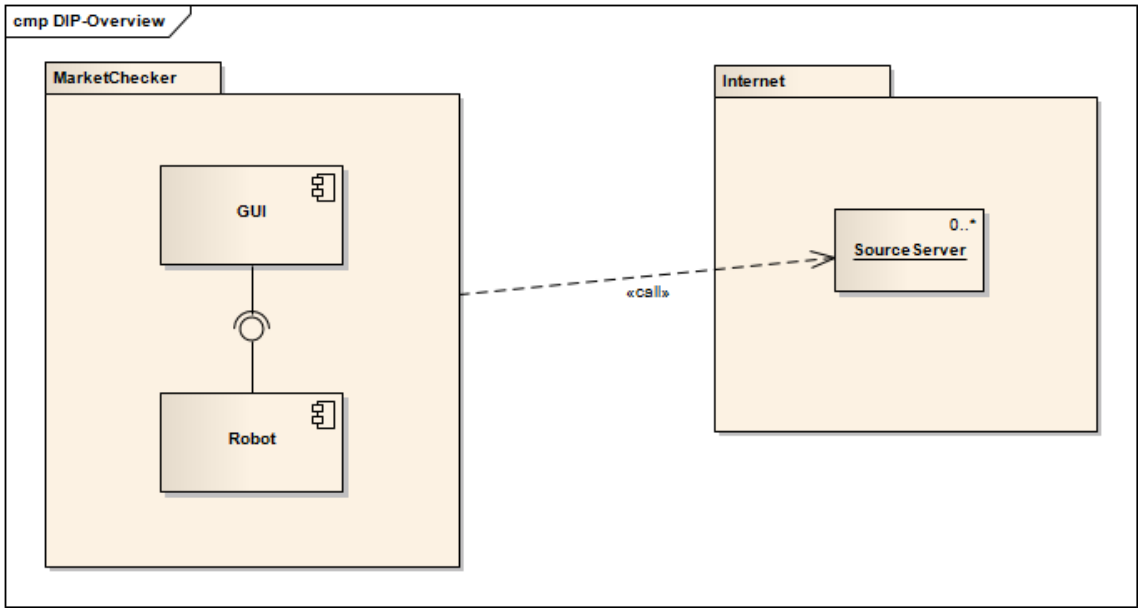
Sekce byla rozdělena do dvou sekcí, kde sekce 3.1 Funkční design popisuje aplikaci na úrovni funkčních bloků a business požadavků. A 3.2 Aplikační design popisuje aplikaci na úrovni softwarové architektury.

3.1 Funkční design

Aplikace je z pohledu funkčního designu rozdělena do dvou high-level celků a to,

- robot a
- uživatelské rozhraní.

Robot je realizován jako knihovna, což umožňuje jeho znovupoužití v libovolné aplikaci. Stará o získávání informací z internetu (ceny komodit) a uživatelské rozhraní je interface mezi uživatelem a robotem, které slouží pro zadávání úkolu robotovi a prezentaci zjištěných výsledků uživateli. Tento vztah je popsán obrázkem Obr. 3.1.1: Architektura aplikace



Obr. 3.1.1: Architektura aplikace

Přes uživatelské rozhraní (GUI) zadává uživatel úlohy robotu a zpět jsou mu

propagovány informace o stavu úloh. Forma propagace a informování uživatele bude popsána v sekci 3.1.3 Notifikace stavu uživateli.

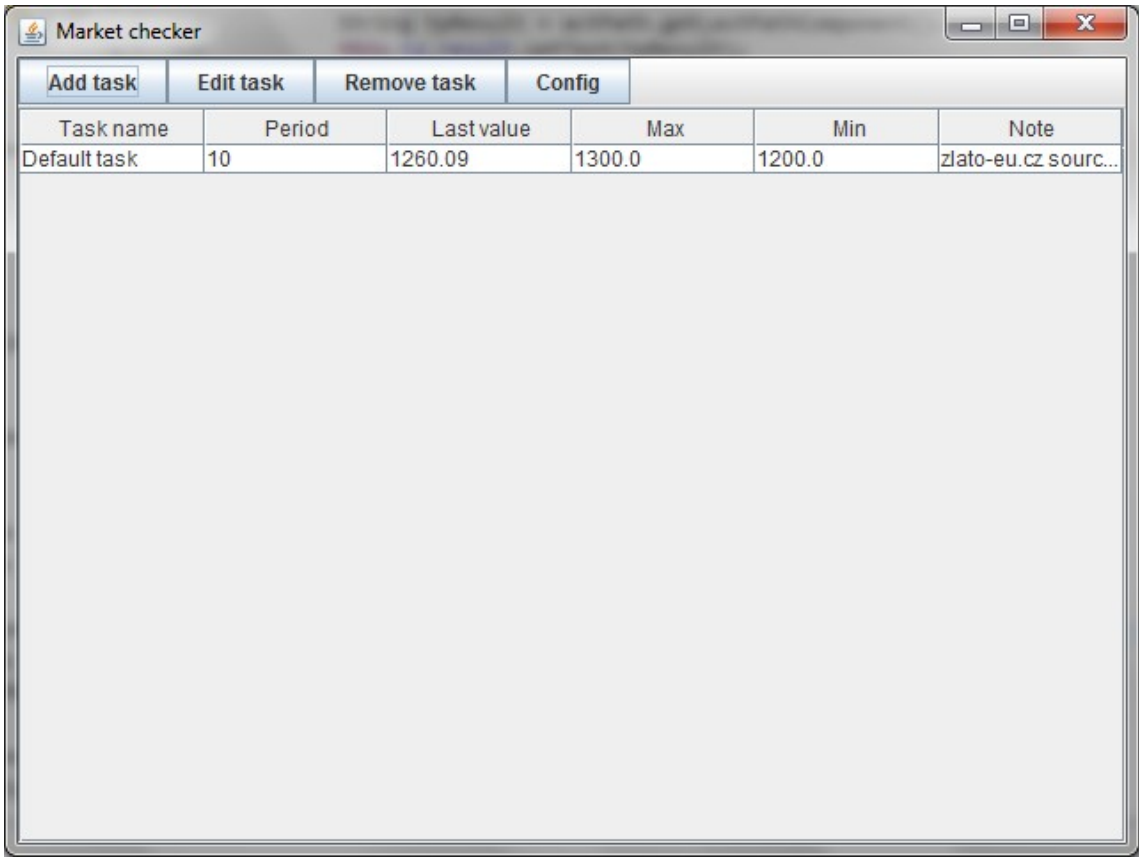
3.1.1 Uživatelské rozhraní

Aplikace se skládá z následujících oken,

- list úloh (hlavní okno aplikace),
- okno pro zdávání nových, nebo editaci existujících úloh a
- okno konfigurace aplikace.

List úloh

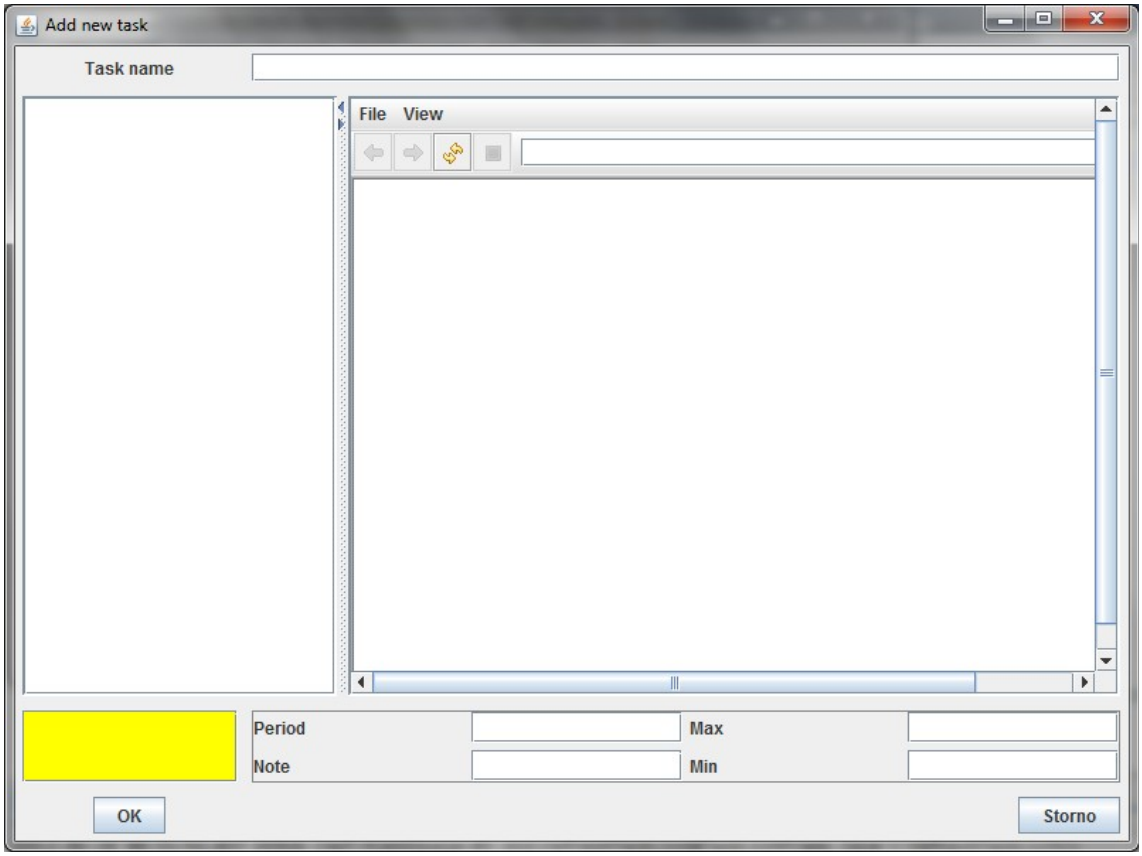
Pomocí tohoto dialogu lze získat přehled o běžících úlohách a spouštět základní případy užití (bude dále popsáno v sekci 3.1.2 Případy užití). Náhled daného okna následuje na o obrázku Obr. 3.1.1.1. Dialog poskytuje základní informace o každé z běžících úloh aplikace.



Obr. 3.1.1.1: Hlavní okno se seznamem úloh

Zadávání nových úloh / editace existujících úloh

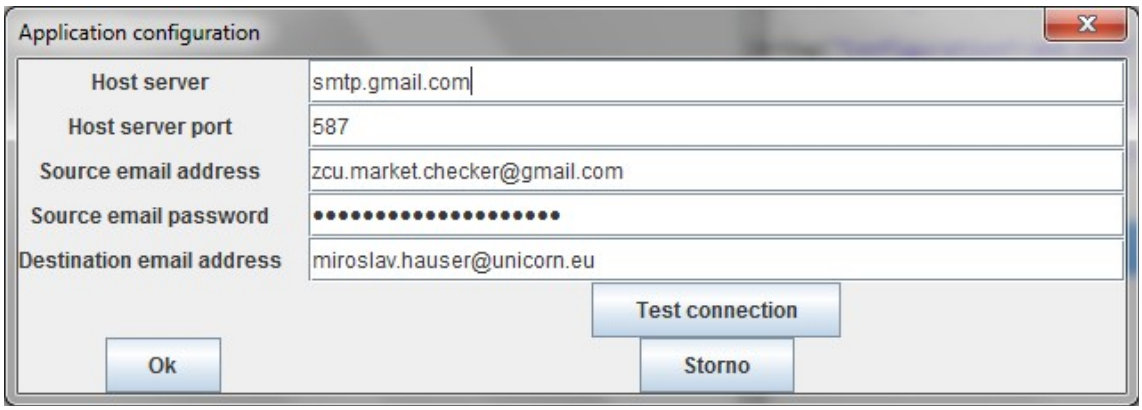
Tento dialog se stane aktivním pokud uživatel projde případy užití Vytvoření nového úkolu nebo Úprava existujícího úkolu. Vizualizace daného okna je na obrázku Obr. 3.1.1.2.



Obr. 3.1.1.2: Okno pro zadávání a úpravu úloh

Konfigurace aplikace

Dialog pro nastavení vlastností aplikace lze aktivovat průchodem případu užití Změna nastavení aplikace. Vizualizace konfiguračního dialogu následuje na Obr. 3.1.1.3.



Obr. 3.1.1.3: Dialog konfigurace aplikace

3.1.2 Případy užití

Realizovaná aplikace poskytuje základní funkčnosti (use-case) pro ověření návrhu samotného robota.

Jsou jimi,

- vytvoření nového úkolu,
- úprava existujícího úkolu,
- odebrání úkolu a
- změna nastavení aplikace.

Vytvoření nového úkolu

Prerekvizity:

- Uživatel má zapnutou aplikaci Market Checker.
- Aplikace Market Checker má přístup k internetu.

Primární scénář užití:

1. Uživatel klikne na tlačítko pro přidání nového úkolu
2. Aplikace Market Checker zobrazí dialog pro zadání informací uživatelem
3. Uživatel zadá potřebné informace
4. Uživatel potvrdí zadání údajů stisknutím tlačítka potvrzení

5. Aplikace Market Checker validuje vstupy uživatele
6. Aplikace Market Checker vytvoří v systému novou úlohu a spustí ji
7. Aplikace Market Checker zavře dialog pro přidání nového úkolu

Alternativní scénáře užití:

UC-VNU1:

4.1 Uživatel stiskne tlačítko pro zrušení zadávání údajů

4.2 Aplikace Market Checker zavře dialog pro přidání nového úkolu

UC-VNU2:

5.1 Uživatelem zadané údaje jsou špatné

5.2 Aplikace Market Checker upozorní uživatele na špatně zadaná data

Use-case UC-VNU2 pokračuje návratem na bod 4.

Úprava existujícího úkolu

Prerekvizity:

- Uživatel má zapnutou aplikaci Market Checker.
- Aplikace Market Checker má přístup k internetu.
- Uživatel má označenu úlohu, pro kterou si přeje změnit parametry

Primární scénář užití:

1. Uživatel klikne na tlačítko pro úpravu úlohy
2. Aplikace Market Checker zobrazí dialog pro zadání nového úkolu
3. Aplikace Market Checker předvyplní dialog daty upravované úlohy
4. Uživatel upraví informace
5. Uživatel potvrdí zadání údajů stisknutím tlačítka potvrzení
6. Aplikace Market Checker validuje vstupy uživatele
7. Aplikace Market Checker vytvoří v systému novou úlohu a spustí ji
8. Aplikace Market Checker zavře dialog pro přidání nového úkolu

Alternativní scénáře užití:

UC-UEU1:

4.3 Uživatel stiskne tlačítko pro zrušení zadávání údajů

4.4 Aplikace Market Checker zavře dialog pro přidání nového úkolu

UC-UEU2:

5.1 Uživatelem zadané údaje jsou špatné

5.2 Aplikace Market Checker upozorní uživatele na špatně zadaná data

Use-case UC-EUE2 pokračuje návratem na bod 4.

Odebrání úkolu

Prerekvizity:

- Uživatel má zapnutou aplikaci Market Checker.

Primární scénář užití:

1. Uživatel zvolí úlohu, kterou si přeje odebrat
2. Uživatel klikne na tlačítko pro odebrání úlohy z aplikace
3. Aplikace Market Checker odebere úlohu ze systému

Změna nastavení aplikace

Prerekvizity:

- Uživatel má zapnutou aplikaci Market Checker.

Primární scénář užití:

1. Uživatel klikne na tlačítko konfigurace aplikace
2. Aplikace zobrazí dialog a předvyplní hodnoty dle známé konfigurace
3. Uživatel zadá nové údaje
4. Uživatel klikne na tlačítko pro kontrolu nastavení
5. Aplikace informuje uživatele o stavu kontroly nastavení
6. Uživatel klikne na tlačítko pro potvrzení

7. Aplikace uzavře dialog a uloží nastavení do konfiguračního souboru

3.1.3 Notifikace stavu uživateli

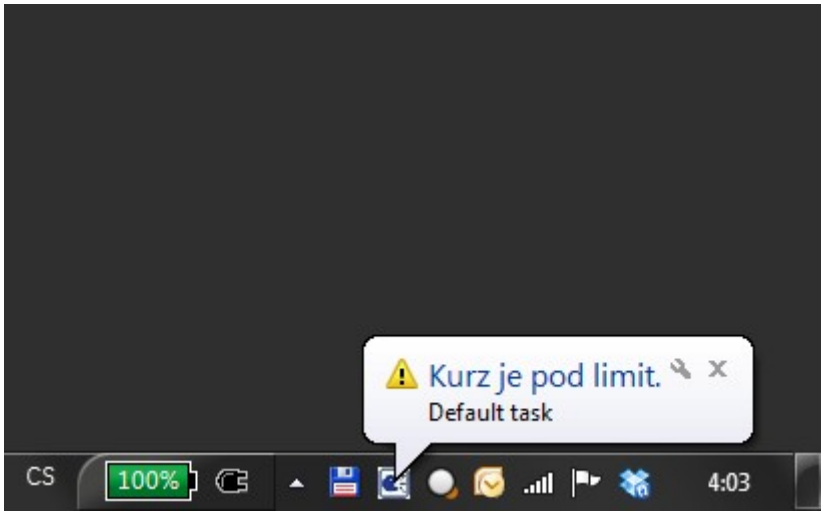
Aplikace informuje uživatele na dvou úrovních a to,

- bublinkovou nápovědou na lokálním PC,
- zasíláním informačních emailů.

Bublinková nápověda

Uživatel je informován standardní bublinkovou nápovědou o stavech, kdy je hodnota mimo definované meze. Uživatel dostane základní informace na základě kterých provede další operace, aplikace jej do ničeho nenutí.

Vzor bublinkové nápovědy, viz obrázek Obr. 3.1.3.1.



Obr. 3.1.3.1: Vzor bublinkové nápovědy

Zasílání informačních emailů

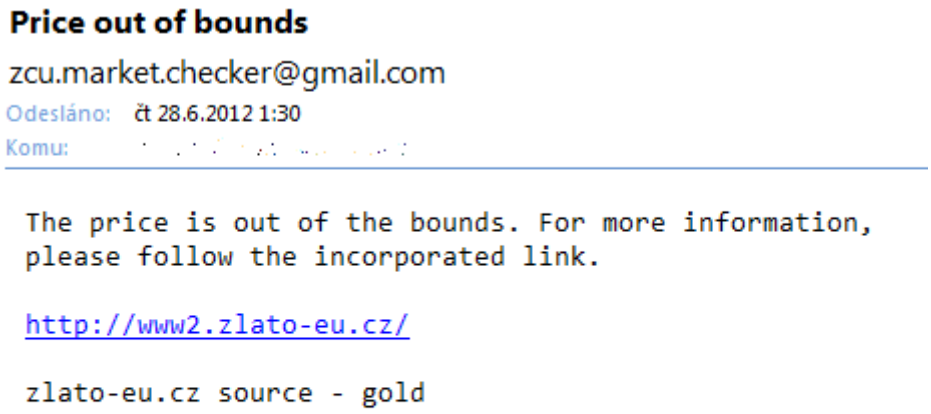
Email obdrží uživatel na základě konfigurace aplikace. V případě nekorektní konfigurace nebude uživatel nijak upozorněn, daný stav bude reportován do logu.

Emaily budou zasílané vždy, když dojde k průrazu definovaných mezí a to maximálně jednou za 24 hodin.

Vzor emailu viz obrázek Obr. 3.1.3.2. Rozsah informací poskytnutých v emailu je přímo

závislý na množství informací, které uživatel zadá při vytváření úlohy (popis úlohy).

Email se skládá z informace o změně, které úlohy se změna týká (odkaz na danou WWW stránku) a popisu úlohy.



Obr. 3.1.3.2: Vzor informačního emailu

3.1.4 Robot

Do návrhu robota vstupují následující požadavky, které dále probereme podrobněji. Konkrétní realizací se bude zabývat sekce 3.2.3 Robot.

Požadavky[10][6],

- obecnost návrhu,
- optimalizace využití systému a
- znovupoužitelnost.

Obecnost návrhu

Tento požadavek se doplňuje s požadavkem na znovupoužitelnost. Snaha je, aby byl návrh na tolik obecný, aby bylo možné aplikací sledovat nejen trhy komodit, ale např. elektronické obchody, aukční portály nebo jakékoliv jiné internetové stránky, jejíž

primární informací, kterou hledáme, je číslo (cena).

Optimalizace využití systému

Základem tohoto požadavku je snaha o rovnoměrné využití prostředků, které nám systém nabízí (CPU, paměť). U aplikací tohoto typu (aplikace na straně klienta) není žádoucí, aby aplikace nárazově čerpala velké množství zdrojů (CPU, paměť).

Znovupoužitelnost

Součástí tohoto požadavku není jen, „zabalení“ aplikace jako knihovny, ale primárně architektonický návrh a realizace na takové úrovni, aby byl další rozvoj maximálně usnadněn. Jak bude vidět v sekci 3.2.3 Robot, i na tento požadavek byl kladený velký důraz.

3.1.5 Ukládání/načítání úloh

Aplikace poskytuje možnost uložení a načtení běžících úloh. Tyto funkcionality nejsou poskytovány na požadavek uživatele (on-demand), ale provádí se při startu a vypínání aplikace automaticky.

Při startu aplikace dojde k prohledání aplikačního adresáře pro standardní soubor uložených úloh a v případě, že je soubor nalezen, dojde k jeho načtení a okamžitému naplánování všech úloh v něm obsažených.

Bližší popis realizace viz sekce 3.2.4 Ukládání/načítání úloh.

3.2 Aplikační design

3.2.1 Úvod do realizace

Tato sekce popisuje z širšího pohledu realizaci problému sledování trhu komodit (obecně jakéhokoliv trhu).

Aplikace byla realizována v programovacím jazyce Java za použití knihoven třetích stran pod otevřenými licencemi. Aplikace je distribuována pod licencí GNU/GPL v3.0.

Seznam knihoven realizovaných v rámci projektu,

- Robot

Seznam knihoven třetích stran,

- *log4j* v.1.6.1
- *JSoup* v.1.2.16 a
- *DJNativeSwing* v.1.0.2

Log4J

Log4J je v dnešní době standardem pro zaznamenávání událostí v Java aplikacích. Knihovna je vydávána pod Apache licencí což umožňuje její široké použití.

Základem knihovny je konfigurační soubor, definující tzv. appendery, co, kdy a jak má knihovna zaznamenat.

Příklad konfiguračního souboru následuje (soubor použitý této implementaci)

```
#
# our log4j properties / configuration file
#
# STDOUT appender
log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender
log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout
log4j.appender.STDOUT.layout.ConversionPattern=%d %p [%t] %L:%M@%F -
      %m\n

# use the STDOUT appender. set the level to INFO.
log4j.rootLogger=INFO, STDOUT
```

Následné použití komponenty může vypadat takto,

```
try {

      ...

} catch (ClassNotFoundException cnfe) {
      Logger.getLogger(ConfigManager.class).error(cnfe);
} catch (Exception e) {
      Logger.getLogger(ConfigManager.class).error(e);
}
```

JSoup

JSoup je knihovnou pro zpracování HTML obsahu. Jedná se knihovnu vydanou pod

MIT licencí, která nás v současné implementaci nijak nelimituje.

JSoup poskytuje rozsáhlé api pro dolování, manipulaci a tvorbu HTML obsahu s použitím standardního DOM modelu, používaného současnými prohlížeči. V současné době je *JSoup* schopen pracovat i s HTML5 technologií. Této vlastnosti však nebylo využito.

S knihovnou lze [1],

- číst a parsovat HTML data z URL, souboru, nebo textového řetězce,
- hledat a extrahovat data s použitím průchodu DOM, nebo CSS selekce,
- upravovat HTML elementy, atributy a text,
- upravit data zadaná uživatelem s cílem zabránit XSS útokům,
- tvořit čisté HTML.

Jednou z hlavních výhod knihovny je, stabilita v případě nevalidních dat na vstupu. *JSoup* je schopný parsovat i data, která se nedaří parsovat jinými nástroji.

Příklad použití s textovým řetězcem,

```
String html = "";
html += "<?xml version='1.0' encoding='utf-8' ?>";
html += ...
html += "</html>";
Document doc = Jsoup.parse(html);
```

Příkald použití s on-line zdrojem dat,

```
int timeout = 1000;
URL siteUrl = new URL("http://google.com/");
Document doc = Jsoup.parse(siteUrl, timeout);
```

Knihovna obsahuje několik drobných nedostatků, kde nejzásadnější z nich je nemožnost serializace jejích datových struktur. Datové struktury (*JSoup Element*) je nutné ukládat spolu s ukládáním a načítáním úloh (viz kapitola 3.2.4 Ukládání/načítání úloh). Komponenta byla analyzována a byly upraveny potřebné datové struktury.

Úpravy JSoup

Po analýza bylo zjištěno, že je nutné upravit následující datové struktury,

- *org.jsoup.nodes.Node* a všechny její potomky,
- *org.jsoup.nodes.Entities,*
- *org.jsoup.nodes.Attributes,*
- *org.jsoup.nodes.Attribute,*
- *org.jsoup.parser.Tag* a
- *org.jsoup.parser.Token.*

U všech výše uvedených tříd bylo implementováno rozhraní java.io.Serializable a byly jim vygenerovány unikátní identifikátory pro serializaci.

DJNativeSwing

Native Swing (dále jen *DJNS*) je knihovna umožňující snadnou integraci nativních komponent do *Java Swing* aplikací [2].

DJNS se skládá z knihovny frameworků a sady komplexních komponent založených na *SWT* (Standard Widget Toolkit).

Základními komponentami *DJNS* jsou,

- web browser,
- flash player,
- media player a
- HTML editor se schopností zabarvování zdrojového kódu (syntax highlighting).

Komponenta je uvolněna pod GNU LGPL licencí, která nám umožňuje širší možnosti použití, než GNU GPL (umožňuje použití i pro proprietární software).

Použití komponenty je velice snadné, viz citace zdrojového kódu aplikace,

```
// browser initialisation
NativeSwing.initialize();
NativeInterface.open();

. . .


this.jwb_window = new JWebBrowser(
            NSComponentOptions.destroyOnFinalization());
this.jwb_window.setBarsVisible(true);
this.jwb_window.setStatusBarVisible(false);
this.jwb_window.addWebBrowserListener(new WebBrowserAdapter() {
      @Override
      public void locationChanged(WebBrowserNavigationEvent e){
            if (flagUpdate) {
                  flagUpdate = false;
                  return;
            }
            // about:blank fix
            // the browser calls about:blank page right before
            // page load
            if (e.getNewResourceLocation().startsWith("about"))
                  return;

            try {
                  doc = Jsoup.parse(
                        jwb_window.getHTMLContent());

                  myModel = new DocumentTreeModel(doc);
                  tree_pageStruct.setModel(myModel);
            } catch (Exception ex) {
                  Logger.getLogger(this.getClass()).error(ex);
                  JOptionPane.showMessageDialog(
                        null,
                  Messages.getString("AddTaskFrame.wrontNet"));
            }
      }
});


. . .


this.add(new JScrollPane(this.jwb_window), c);
```

DJNS je možno použít jako prohlížeč a dotazovat se jej na kontext, se kterým aktuálně pracuje, nebo je možné jej použít pro zobrazení kontextu, který si sami přejeme (metoda *setHTMLContent(String)*). V této implementaci je *DJNS* použit jako plnohodnotný prohlížeč, který je poté dotazován na kontext, viz úryvek kódu výše, ze kterého je vidět, že dojde-li ke změně adresy na které se prohlížeč nachází, aplikace načte HTML nové

stránky a zobrazí jej v navigačním stromu.

JavaMail API

JavaMail API (dále jen JavaMail) je knihovna umožňující posílání elektronické pošty přímo z programu psaného v jazyce *Java*. Knihovna je vydaná pod *Oracle Binary Code* licencí pro *Java EE* technologie.

Použití komponenty je intuitivní a integrace velice jednoduchá,

```
// Create a mail session
props = new java.util.Properties();
props = System.getProperties();
props.put("mail.smtp.host", smtpHost);
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtps.auth", "true");

this.content = content;
this.subject = subject;
this.smtpPort = smtpPort;
this.to = to;
this.urPassword = urPassword;
this.from = from;

. . .

try {
	Session session = Session.getInstance(props, null);
	session.setDebug(true);

	// Construct the message
	Message msg = new MimeMessage(session);
	msg.setFrom(new InternetAddress(from));
	msg.setRecipient(Message.RecipientType.TO,
		new InternetAddress(to));
	msg.setSubject(subject);
	msg.setText(content);
	msg.setSentDate(new Date());
	Transport tr = null;
	tr = session.getTransport("smtp");
	tr.connect(props.getProperty("mail.smtp.host"),
		smtpPort,
		from,
		urPassword);
	tr.sendMessage(msg, msg.getAllRecipients());

	this.fSuccess = true;
} catch (Exception e) {
	Logger.getLogger(getClass()).error(e);
	this.fSuccess = false;
```

```
}
```

Samotné zasílání emailů je vhodné implementovat v odděleném vlákně, neboť odeslání emailu trvá řádově vteřiny. Po tuto dobu je vlákno vlající odesílání emailu sekvenčně blokováno. Čas blokace se může protáhnout až na 30 vteřin v případě nedostupného serveru.

3.2.2 Uživatelské rozhraní

List úloh

Okno list úloh (*TaskListFrame*) obsahuje následující základní komponenty,

- *javax.swing.JTable* pro zobrazení seznamu běžících úloh a
- *javax.swing.JMenuBar* pro volbu operací s běžícími úlohami.

K zobrazení seznamu běžících úloh bylo nutné implementovat vlastní *javax.swing.table.AbstractTableModel*, konkrétně *TaskTableModel*. Model byl implementován za účelem zobrazeí informací o běžících úlohách a umožnění jejich rychlé editace (pro konkrétní implementaci, viz Přílohy).

TaskTableModel si drží referenci na aktuální instanci plánovače (*Scheduler*) a zobrazuje přímo jeho frontu úkolů.

TaskListFrame je vstupním bodem grafického uživatelského prostředí a „běží“ tedy po celou dobu běhu programu. V případě minimalizace je okno minimalizováno do oblasti systémových ikon (systray).

Implementaci minimalizace do oblasti systémových ikon popisuje následující blok kódu,

```
// minimize to systray
if (SystemTray.isSupported()) {
	// systray pop-up menu
	PopupMenu popup = new PopupMenu();
	this.exit = new MenuItem("Exit");
	this.exit.addActionListener(this);
	popup.add(this.exit);

	// create a systray icon
```

```
		try {
			Image image = Toolkit.getDefaultToolkit().getImage(
				(String) ConfigManager.getProperty(
				ConfigManager.PROPERTY.TRAY_ICON_IMAGE));
			trayIcon = new TrayIcon(
					image,
					"Market checker",
					popup);
			trayIcon.addActionListener(this);
		} catch (Exception e) {
			Logger.getLogger(getClass()).error(e);
		}
	} else {
		Logger.getLogger(getClass()).warn(
				"Running without tray icon.");
	}


	. . .


	@Override
	public void windowDeiconified(WindowEvent arg0) {
		SystemTray.getSystemTray().remove(this.trayIcon);
		this.setVisible(true);
	}

	@Override
	public void windowIconified(WindowEvent arg0) {
		try {
			SystemTray.getSystemTray().add(this.trayIcon);
		} catch (Exception e) {
			Logger.getLogger(getClass()).error(e);
		}
		this.setVisible(false);
	}
```

Zadávání nových úloh / editace existujících úloh

Dialog pro zadávání nových úloh a editaci existujících obsahuje následující prvky,

- *JWebBrowser (DJ Native Swing Web Browser)* jako prohlížeč webu,
- *JTree* pro zobrazení struktury webu,
- *JTextField* pro zadání parametrů úlohy a
- *JTextArea* pro zobrazení aktuálního zadání.

JWebBrowser je, jak bylo zmíněno, použit pro výběr webu, na kterém chceme hodnotu kontrolovat. Dále se v označuje element, který jsme v *JTree* zvolili jako zdrojový.

Jakmile *JWebBrowser* načte novou stránku, je struktura této stránky načtena do *JTree* komponenty, kde nám slouží pro výběr cílového elementu. Konkrétní implementace je citována v kapitole 3.2.1 Úvod do realizace, sekce DJNativeSwing.

Pro *JTree* komponentu byl upraven *TreeModel* a to tak, aby byl schopen zobrazit *JSoup* document model ve stromové struktuře. Tato komponenta tedy zobrazuje HTML strom webu, který je načtený *JWebBrowserem*.

Pokud v *JTree* panelu zvolíme element dojde k jeho zabarvení v *JWebBrowseru* žlutou barvou (obdobný mechanismus používá vývojový modul pro Mozilla Firefoz Firebug). K zabarvení konkrétního zvoleného elementu je využito *JavaScriptu*, který je možný spouštět nad *JWebBrowserem* pomocí metody *highlightHTMLSection(),* viz

```
protected void highlightHTMLSection() {
	TreePath actPath = this.tree_pageStruct.getSelectionPath();
	if (actPath == null)
		return;

	this.js = "document";
	Object[] path = actPath.getPath();
	Node act = doc;
	Element	lastE = doc;
	int indexOfElement = 0;
	for (int i = 1; i < path.length; i++) {
		act = act.childNode(((Node)path[i]).siblingIndex());
		if (act.nodeName().equals("#text"))
			continue;

		Elements e = null;
		if (lastE != null) {
			e = lastE.getElementsByTag(act.nodeName());
			indexOfElement = e.indexOf(act);
		}

		this.js += ".getElementsByTagName(\""
				+ act.nodeName()
				+ "\")["
				+ indexOfElement
				+ "]";

		lastE = e.get(indexOfElement);
	}

	this.js += ".style.backgroundColor = \"yellow\"";
	this.jwb_window.executeJavascript(this.js);
	Logger.getLogger(this.getClass()).debug(this.js);
}
```

Původně zvažovaným postupem pro zvýraznění elementu byla změna HTML reprezentace stránky na straně klienta. Tento postup nebyl funkční, neb z

bezpečnostních důvodů není možná změna HTML bez ztráty kontextu stránky. Jakmile tedy jakkoliv změníme browseru HTML „pod rukou“, přestanou fungovat relativní linky, média atd.

Konfigurace aplikace

Dialog obsahuje následující prvky,

- *JTextField* pro zadání parametrů konfigurace,
- *JButton* pro potvrzení zadaných hodnot, či test zadaných hodnot.

Dialog při svém zobrazení načte aktuální konfiguraci aplikace. A dále umožní uživateli tuto konfiguraci měnit.

Uživatel má možnost provést test nastavení emailové notifikace stiskem odpovídajícího tlačítka. Implementace emailové notifikace je provedena dvakrát a to jednou neblokující a podruhé blokující. Viz následující metody (komentáře byly záměrně odstraněny),

```
public static void quickSend(String subject, String body) { }

public static void send(String smtpHost, int smtpPort,
            String from, String urPassword, String to,
            String subject, String content) {}

public static boolean blockingSend(String smtpHost, int smtpPort,
            String from, String urPassword, String to,
            String subject, String content) {}

public static boolean blockingQuickSend(String subject,
            String body) {}
```

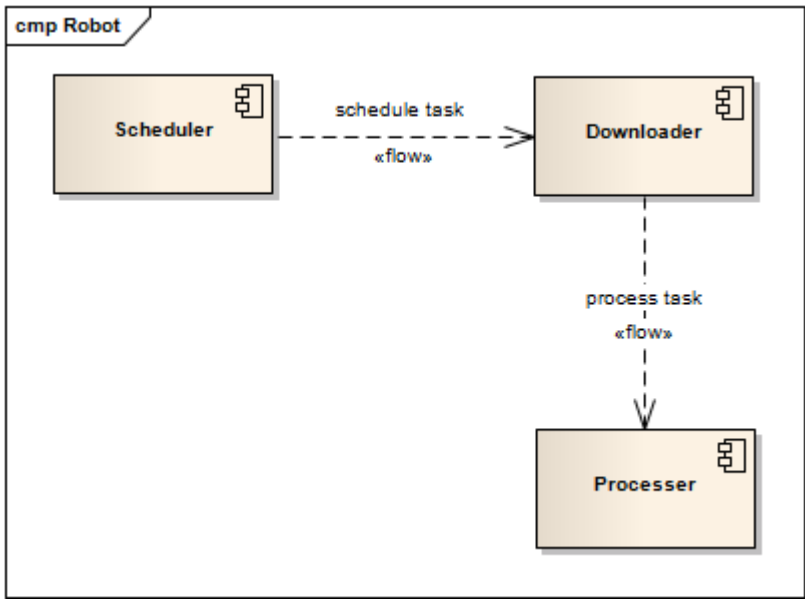
Zde uvedené metody reálně neposílají email. Volají vlákno, které se stará o zasílání emailů.

Neblokující implementace zasílání emailu nečeká na výsledek prováděné operace. Případná chyba bude zaznamenána v logu aplikace, nebude však propagována v rámci aplikace.

Blokující implementace naopak čeká na výsledek odesílání emailu a tato hodnota je dále propagována do uživatelského rozhraní. V konfiguraci aplikace je využito tohoto přístupu, uživatel je tedy informován, zda-li se podařilo email odeslat.

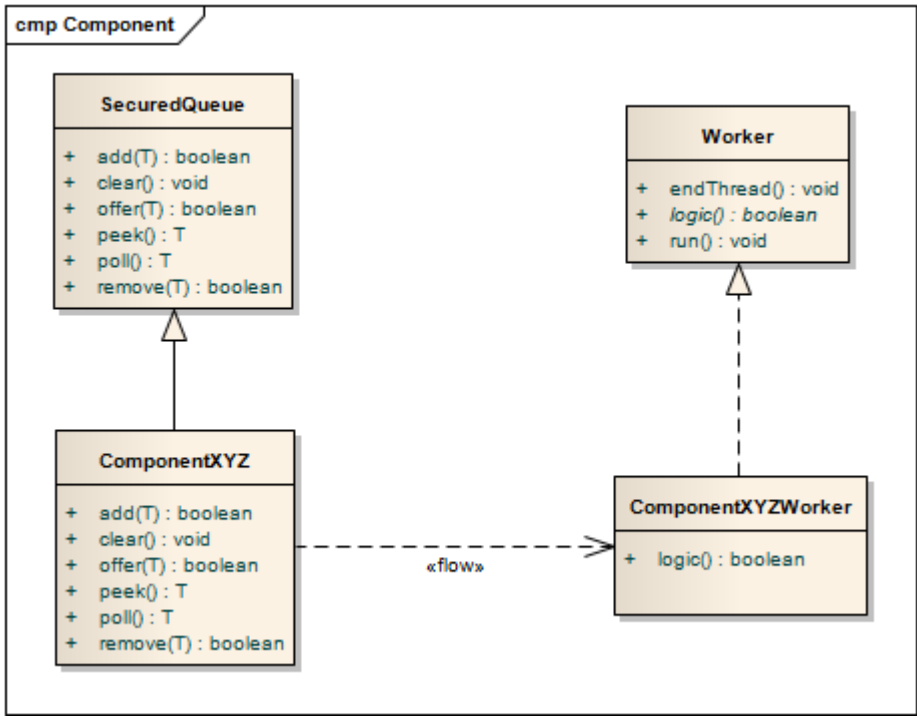
3.2.3 Robot

Architektura robota (Obr. 3.2.3.1) podléhá, jak bylo zmíněno v sekci Optimalizace využití systému, optimalizaci pro využití systému, konkrétně stabilní zatížení procesoru v čase, na úkor doby obsluhy.



Obr. 3.2.3.1: Schéma robota

Když do robota vstoupí požadavek, je periodicky plánován plánovačem (*Scheduler*) a postupuje systémem od komponenty ke komponentě, dokud nedojde cílové komponenty, která jej zpracuje a nepošle dále.



Obr. 3.2.3.2: Schéma komponenty

Jednotlivé prvky robota jsou navržený totožně a to na následující tři části (Obr. 3.2.3.2),

- interface (na obrázku reprezentován *ComponentXYZ*),
- worker (na obrázku reprezentován *ComponentXYZWorker*),
- obslužné třídy.

Interface definuje způsob kooperace s komponentou. Všechny komponenty použité v systému obsahují vstupní frontu požadavků. Všechny komponenty mají tedy základ interface stejný, může být jen rozšířen o specifické metody.

Worker je třída, provádějící funkcionality požadované po komponentě. Vlastní výpočetní vlákna, která, dle definované disciplíny, vyjímají požadavky ze vstupní fronty a obsluhují je. Po skončení obsluhy je buďto vrátí zpět do fronty (případ komponenty scheduler), nebo vloží do fronty jiné komponentě (případ komponent downloader, processer).

Obslužné třídy poskytují své služby (metody) pro podporu zpracování požadavku workerem.

Výše popsaná architektura zjednodušeně kopíruje komponentové frameworky (OSGI,

Spring, …), které by pro takovýto problém nebylo efektivní používat. Nenabízí takové možnosti jako zmíněné frameworky, ale umožňuje udržovat čistou, funkční a rozšiřitelnou architekturu programu.

Interface

Interface obsahuje prioritní frontu upravenou pro problém producent/konzument. Tato fronta je základem každé komponenty, a slouží pro zadávání úloh komponentě.

Interface každé komponenty kopíruje interface fronty, kterou implementuje, viz příklad.

```
public class SecuredQueue<T> extends PriorityQueue<T> {

	/**
	 * Creates queue with limited size.
	 *
	 * @param capacity of queue
	 */
	public SecuredQueue(int capacity) {}

	/**
	 * Creates priority queue.
	 */
	public SecuredQueue() {}

	@Override
	public synchronized boolean add(T arg0) {}

	@Override
	public synchronized boolean offer(T arg0) {}

	@Override
	public synchronized T peek() {}

	@Override
	public synchronized T poll() {}

	@Override
	public synchronized boolean remove(Object arg0) {}

	@Override
	public synchronized void clear() {}
}
```

Komponenta může interface rozšířit o specifické metody.

Worker

Worker je pracovní vlákno každé komponenty. Vyjímá úlohy z interface, zpracovává je

za pomoci obslužných metod a předává je dalším komponentám ke zpracování. V této implementaci obsahuje worker právě jedno vlákno. Malým zásahem do kódu by bylo možné upravit aplikaci tak, aby byl počet vláken jednotlivých workerů konfigurovatelný. Tím by se zvýšila propustnost celého systému, ale také by se zvýšilo špičkové zatížení.

Worker standardně rozšiřuje abstraktní třídu *Worker* mající následující implementaci,

```
public abstract class Worker implements Runnable {

	private boolean alive = true;

	/**
	 * Indicates thread that it should end.
	 */
	public void endThread() {
		this.alive = false;
	}

	/**
	 * Main thread logic.<br />
	 * This method will be <b>periodically executed</b> till the
	 * thread ends.
	 *
	 * @return true if thread should go on. False for thread end.
	 */
	public abstract boolean logic();


	@Override
	public final void run() {
		while (alive && logic()) {

		}
	}
}
```

Konkrétní implementace workera pro *Downloader* by mohla vypadat takto,

```
public class DwnWorker extends Worker {

	private Downloader  parent = null;
	private boolean alive = true;

	/**
	 * Download worker thread.
	 *
	 * @param parent downloader
	 */
```

```
	public DwnWorker(Downloader parent) {
		this.parent= parent;
	}

	@Override
	public boolean logic() {
		Task nextTask = this.parent.getTask();
		String page = "";

		if (nextTask != null) {
			page = processTask(nextTask);

			if (page != null)
				nextTask.setHtml(page);
			this.parent.getProcessor().queueTask(nextTask);
		}

		return true;
	}

	/**
	 * Processes the task
	 *
	 * @param t task to be processed
	 */
	protected String processTask(BasicTask t) {
		InputStream is = null;
		DataInputStream dis;
		String line;
		StringBuffer sb = new StringBuffer();

		try {
		    is = t.getAddress().openStream();
		    dis = new DataInputStream(new
		            BufferedInputStream(is));

		    while ((line = dis.readLine()) != null) {
		        sb.append(line + "\n");
		    }
		} catch (Exception e) {
			ConfigManager.log.error(e);
		} finally {
		    try {
			if (is != null) is.close();
		    } catch (IOException ioe) {
		        // intentionally left blank - nothing to see here
		    }
		}

		return sb.toString();
	}
}
```

Metoda *logic()* implementuje základní logiku komponenty. V tomto konkrétním případě dojde ke stažení obsahu HTML stránky a následné uložení do objektu úkolu. Poté je

úkol předán další komponentě ke zpracování.

Obslužné třídy

Obslužné třídy jsou implementovány v balíčku *cz.zcu.fav.hauser.dip.api* a v současné implementaci jsou to,

SecuredQueue.java slouží jako základní interface pro komponenty a zároveň implementuje základní logiku pro práci s komponentou.

Worker.java je abstraktní třída, sloužící jako předpis pro implementaci workerů jednotlivých komponent.

BasicTask.java je POJO pro uchování stavu úkolu s minimální množinou informací.

Task.java je POJO rozšiřující *BasicTask.java* pro uchování kompletní sady informací k úkolu. Instance tohoto objektu následně putují systémem.

3.2.4 Ukládání/načítání úloh

Ukládání a načítání úloh jsou automatizované operace zajišťující uchování úloh i po vypnutí a zapnuití programu.

Ukládání úloh

Probíhá při vypínání aplikace. Je využito mechanizmu serializace objektu do streamu (zde do souboru). Pro funkčnost daného mechanizmu bylo nutné upravit implementaci komponentu třetí strany *JSoup*.

Metoda použitá pro ukládání kontextu aplikace je následující,

```
public static void saveTaskQueueState(SecuredQueue<Task> queue) {

	FileOutputStream fos = null;
	ObjectOutputStream oos = null;

	try {
		fos = new FileOutputStream(new File(
				(String) getProperty(
					PROPERTY.APPLICATION_STATE)));
		oos = new ObjectOutputStream(fos);
```

```
                oos.writeObject(queue);
                oos.flush();

        } catch (Exception e) {
                Logger.getLogger(ConfigManager.class).error(e);
        } finally {
                try {
                        if (oos != null) oos.close();
                        if (fos != null) fos.close();
                } catch (Exception e) {
                        /* no need to catch anything */
                }
        }
}
```

Načítání úloh

Načítání úloh probíhá automaticky při startu aplikace. V návaznosti na implementaci ukládání úloh je realizace provedena deserializací objektu z binárního streamu (souboru).

Metoda použitá pro binární deserializaci objektu je následující,

```
public static SecuredQueue<Task> loadTaskQueueState() {

        FileInputStream fis = null;
        ObjectInputStream ois = null;
        SecuredQueue<Task> queue = null;

        try {
                fis = new FileInputStream(new File(
                                (String) getProperty(
                                        PROPERTY.APPLICATION_STATE)));
                ois = new ObjectInputStream(fis);

                queue = (SecuredQueue<Task>) ois.readObject();

        } catch (ClassNotFoundException cnfe) {
                Logger.getLogger(ConfigManager.class).error(cnfe);
        } catch (Exception e) {
                Logger.getLogger(ConfigManager.class).error(e);
        } finally {
                try {
                        if (ois != null) ois.close();
                        if (fis != null) fis.close();
                } catch (Exception e) {
                        /* no need to catch anything */
                }
        }

        return queue;
}
```

Problémy

V době implementace byla zvažována implementace pomocí *XMLEncoderu*, což je standardní metoda serializace objektu do XML souboru. Výhodou této realizace je snadní čitelnost uložených dat pro uživatele a přenositelnost takto uložených dat. Serializaci jednoduchého objektu ukazuje následující příklad [3],

```
public class Person {
	private String firstName;
	private String lastName;

	public String getFirstName() {
		return firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public void setFirstName(String str) {
		firstName = str;
	}

	public void setLastName(String str) {
		lastName = str;
	}
}

. . .

FileOutputStream os = new FileOutputStream("C:/cust.xml");
XMLEncoder encoder = new XMLEncoder(os);
Person p = new Person();
p.setFirstName("John");
encoder.writeObject(p);
encoder.close();
```

Daná realizace se ukázala jako problémová z důvodu ukládání datových struktur komponenty třetích stran *JSoup*. Pro serializaci je nutné, aby serializovaný objekt implementoval konstruktor bez parametru, v opačném případě je nutné definovat delegát pro daný konstruktor s parametrem, viz následující příklad [3]

```
public Person(String aFirstName, String aLastName){
	firstName = aFirstName;
	firstName = aLastName;
}

XMLEncoder e = new XMLEncoder(os);
Person p = new Person("John","Smith");
```

```
e.setPersistenceDelegate(
            Person.class,
            new DefaultPersistenceDelegate(
                        new String[] {
                              "firstName","lastName"}));
e.writeObject(person);
```

Vzhledem k těmto požadavkům na použití, které se jevily jako problémové, byla použita binární serializace objektů, ve které postačuje, aby serializovaný objekt implementoval rozhraní *Serializable*.

4 Zhodnocení a další postup

Obsahem této práce byla realizace programového systému pro sledování cen komodit. Celkový její přínos lze shrnout do následujících bodů,

- realizace robustního a znovupoužitelného robota a
- prověření současných knihoven a modulů pro práci s HTML.

Realizovaný robot lze použít pro sledování jakéhokoliv číselného údaje a lze jej použít nejen pro potřeby sledování, ale také přímé reakce na změnu (internetové aukce, atp.).

Knihovny použité pro realizaci *DJ Project Native Swing*[2] a *JSoup*[1] se ukázaly jako velice schopné nástroje hodné dalšího použití[8][9].

Práce naplnila cíle zadání, bylo by ovšem vhodné na ni navázat a pokračovat v dalším rozvoji. Úlohy hodné realizace jsou,

- zvýšení odolnosti oproti změně struktury zdrojové web stránky,
- implementace historie získaných hodnot pro statistické účely,
- implementace algoritmů predikce nad získanými daty.

Přehled zkratek

Zkratka	Kompletní název	Popis
DJNS	DJ Native Swing	Knihovna nativních Swing komponent (browser, …).
SWT	Standard Widget Toolkit	Open-source knihovna widgetů pro grafická rozhraní založená na programovacím jazyku Java.
DOM	Document Object Model	Strom objektů reprezentující dokument.
MACD	Moving Average Convergence/Divergence	Metoda technické analýzy[7].
KDD	Knowledge Discovery in Databases	Koncept prohledávání velkého množství dat (báze dat) s cílem vyhledání informací.
CRISP-DM	Cross Industry Standard Process for Data Mining	Metodika dolování dat, jejíž vývoj byl zaštítěn společnostmi SPSS, Teradata, Daimler_AG a OHRA
SEMMA	Sample, Explore, Modify, Model and Assess	Metodika dolování dat, jejíž vývoj byl zaštítěn společností SAS Institute Inc.
MIS	Management Information System	Informační systém poskytující relevantní informace pro řídící pracovníky za účelem rychlého a přesného rozhodování.
BI	Business Inteligence	Systém pro analýzu rozsáhlých dat pro podporu rozhodování.
WWW	World Wide Web	Systém provázaných hypertextových dokumentů na internetu.

Reference

1) Hedley, Jonathan. *JSoup: Java HTML Parser* [online]. 2009 [cit. 2012-06-02]. URL: <http://jsoup.org/>.

2) Deckers, Christopher. *The DJ Project:Native Swing* [online]. 2009 [cit. 2012-06-02].URL:<http://djproject.sourceforge.net/ns/>.

3) WINCHESTER, Joe a MILNE Philip. *XML Serialization of Java Objects* [online]. 2003 [cit. 2012-06-01]. URL: <http://java.sys-con.com/node/37550>.

4) ŠIRC, Matěj. *Předpověď vývoje trhu komodit pomocí autokonstruktivní evoluce*. Brno, 2008. URL: <http://is.muni.cz/th/98693/fi_b/bcp_xsirc-print.pdf>. Bakalářská práce. Masarykova univerzita. Vedoucí práce Ing. Aleš Horák, Ph.D.

5) Technická analýza. *Forex-zone* [online]. 2012 [cit. 2012-05-29]. URL: <http://www.forex-zone.cz/p/technicka-analyza>.

6) SAHUGUET, Arnaud a AZAVANT, Fabien. *WysiWyg Web Wrapper Factory (W4F)* [online]. 1998 [cit. 2012-06-03]. URL: <http://db.cis.upenn.edu/DL/WWW8/index.html>

7) ACHELIS, Steven B. *Technical analysis from A to Z*. 2nd ed. New York: McGraw Hill, c2001, s. 2-23. ISBN 978-0071363488.

8) JUAIN, Anuja, GHALIB, Rukunuddin a SWARNALATHA, P. *GAT Framework To Perform Automation On Web-Based Application*. [online]. 2012, s. 123-126 [cit. 2012-05-29]. ISSN 2248-9037. URL: <http://rjpa.info/index.php/rjpa/article/view/97/80>.

9) ŠIMANSKÝ, David. *Nástroj na sledovanie zmien obsahu webov*. Brno, 2012. URL: <http://is.muni.cz/th/359267/fi_b/bp.pdf>. Bakalářská práce. Masarykova univerzita. Vedoucí práce RNDr. Jaroslav Ráček, Ph.D.

10) SHKAPENYUK, Vladislav a SUEL, Torsten. *Design and Implementation of a High-Performance Distributed Web Crawler* [online]. Polytechnic university, Westchester, 2001 [cit. 2012-06-08]. URL: <http://cis.poly.edu/tr/tr-cis-2001-03.pdf>. Technická zpráva. Polytechnic University Westchester.

11) BERKA, Petr. *Dobývání znalostí z databází*. Vyd. 1. Praha: Academia, 2003, s. 15-19. ISBN 80-200-1062-9.

12) THE UNIVERSITY OF NORTH CAROLINA. *Data mining* [online]. 2003 [cit. 2012-06-10]. URL: <http://www.unc.edu/~xluan/258/datamining.html>.

13) FAYYAD, Usama, PIATETSKY-SHAPIRO, Gregory a SMYTH, Padhraic. *From Data Mining to Knowledge Discovery in Databases*. [online]. 1996, s. 37-43 [cit. 2012-06-12]. URL: <http://www.kdnuggets.com/gpspubs/aimag-kdd-overview-1996-Fayyad.pdf>.

14) AZEVEDO, Ana a FILIPE, Santos, Manuel. *KDD, SEMMA and CRISP-DM*: *A parallel overview*. 2008. ISBN 978-972-8924-63-8. URL: <http://www.iadis.net/dl/final_uploads/200812P033.pdf>.

15) YU, Lei, YE, Jieping a LIU, Huan. *Dimensionality Reduction for Data Mining*: *Techniques, Applications and Trends* [online]. 2007, s. 1-14 [cit. 2012-05-28]. URL: <http://www.cs.binghamton.edu/~lyu/SDM07/DR-SDM07.pdf>.

16) FOREX-ZONE.CZ. *Software a hardware* [online]. 2012 [cit. 2012-06-25]. URL: <http://www.forex-zone.cz/software-a-hardware>.

17) XTB online trading. XT BROOKERS. *XT Brookers* [online]. 2012 [cit. 2012-06-25]. URL: <http://www.xtb.cz/>.

Přílohy