VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

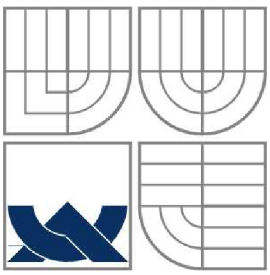
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

EVOLUČNÍ ALGORITMY V ÚLOZE PREDIKCE ČASOVÝCH ŘAD

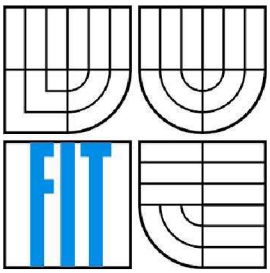
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc, Jan Křivánek
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

EVOLUČNÍ PREDIKCE ČASOVÝCH ŘAD

EVOLUTIONARY PREDICTION OF TIME SERIES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Jan Křivánek

VEDOUCÍ PRÁCE
SUPERVISOR

Lukáš Sekanina

BRNO 2008

Abstrakt

Náplní předkládané práce je sumarizace znalostí v oblasti teorie časových řad, jejich analýzy a aplikace na finančních trzích. Dále práce podává přehled evolučních algoritmů, jejich klasifikaci a použití. Jádrem práce je propojení těchto znalostí a vytvoření systému, který využívá evoluční algoritmy k optimalizaci predikčních modelů finančních časových řad. Při vývoji byly použity techniky softwarového inženýrství (automatická kontinuální integrace, automatizované kontrolování kvality produktu apod.) nutné pro snadnou udržovatelnost a rozšiřovatelnost projektu více vývojáři.

Klíčová slova

časové řady, modely časových řad, predikce časových řad, evoluční algoritmy, genetické algoritmy, genetické programování, evoluční strategie, evoluční programování

Abstract

This thesis summarizes knowledge in the field of time series theory, method for time series analysis and applications in financial modeling. It also resumes the area of evolutionary algorithms, their classification and applications. The core of this work combines these knowledges in order to build a system utilizing evolutionary algorithms for financial time series forecasting models optimization. Various software engineering techniques were used during the implementation phase (ACI – autonomous continual integration, autonomous quality control etc.) to ensure easy maintainability and extendibility of project by more developers.

Keywords

time series, time series modeling, time series forecasting, evolutionary algorithms, genetic algorithms, genetic programming, evolution strategy, evolutionary programming

Citace

Jan Křivánek: Evoluční predikce časových řad, diplomová práce, Brno, FIT VUT v Brně, 2009

Evoluční predikce časových řad

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Doc. Lukáše Sekaniny Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Křivánek
25.5.2009

Poděkování

Rád bych poděkoval Doc. Ing. Lukáši Sekaninovi, Ph.D. za laskavé vedení této práce, věnovaný čas a podnětné náměty na zaměření a cíl práce. Dále pak Michalu Kreslíkovi za zajímavý námět k práci a podporu při jeho realizaci.

Obsah

1 Úvod

V současnosti se s různými podobami časových řad setkáváme prakticky ve všech oborech lidské činnosti – od předpovědi počasí, přes moderní medicínu až po finančnictví a ekonomii. Porozumění vnitřním mechanizmům těchto jevů skrze časové řady, jež je popisují, nám tak může pomáhat usnadňovat běžný život, vydělávat peníze, zkvalitnit péči o zdraví či dokonce zachraňovat lidské životy. Analýza časových řad je tedy oborem s rozsáhlými aplikacemi a tedy i oborem neustále se vyvíjejícím a značně propracovaným. Analýza časových řad založená čistě na matematických modelech je tak velmi složitá a pro neznalého uživatele aplikovatelná jen na omezeně náročnou problematiku. Jinou možností je pokusit se inženýrským přístupem hledat vhodný model (případně vhodné nastavení zvoleného matematického modelu) s minimálními znalostmi, ale s o to vyšším výpočetním nasazením. Jeden z možných takovýchto inženýrských přístupů je použití evolučních algoritmů a tímto směrem se právě vydává tato práce.

Cílem práce je implementace modulární systému schopného využívat různé optimalizační moduly pro optimalizace různých predikčních systémů v individuálně ohraničených stavových prostorech, bez nutnosti přepracovávání jiných modulů než těch, které právě potřebujeme v systému zaměnit.

V druhé kapitole, *Časové řady*, je předkládán stručný úvod do problematiky časových řad, jejich matematického modelování a formální analýzy. Čtenář je seznámen se základními složkami časových řad a jejich definicí, dále s pojmem stacionárních řad a nejčastějšími matematickými modely stacionárních řad. V závěru podkapitoly *Analýza časových řad* je nabídnut náhled do alternativní možnosti modelování časových řad, které je inspirované biologií. Dále je druhá kapitola věnována problematice *Aplikace časových řad na finančních trzích*. Tato podkapitola uvádí provázanost teorie časových řad a ekonomických procesů a shrnuje ekonomické teorie s různým náhledem na tuto problematiku.

Třetí kapitola, *Evoluční algoritmy,* sumarizuje informace o jednotlivých skupinách evolučních algoritmů, jejich znacích, využití a nedostatcích. Na konci kapitoly jsou zhodnoceny možnosti použitelnosti jednotlivých uvedených skupin evolučních algoritmů v navrhovaném systému.

Vlastní *Návrh systému optimalizace predikcí* následuje ve čtvrté kapitole. Jsou zde rozvedeny všechny kroky návrhu systému, který může používat různé formy optimalizačních technik (především evolučních algoritmů) k hledání vhodných kalibračních parametrů modelu pro predikci finančních časových řad. Systém je navrhován pro možnost neomezené škálovatelnosti paralelizací atomických výpočtů.

Postup implementace a využití podpůrných technik a nástrojů je popsán v páté kapitole s názvem *Implementace systému optimalizace predikcí.* V kapitole jsou naznačeny postupy, pomocí kterých bylo dosahováno úspěšného vývoje tohoto poměrně rozsáhlého projektu se splněním

požadavků snadné udržovatelnosti a kontinuální dostupnosti průběžných verzí řešení a dokumentací. Vzhledem k faktu, že popisovaný projekt vznikal jako týmový projekt řešený na Fakultě informačních technologií VUT v Brně pod vedením autora, je v této kapitole předložen podrobnější popis podílu autora práce na celém projektu.

Zhodnocení dosažených výsledků předkládá kapitola šestá, *Vyhodnocení*. V této kapitole je uvedena především případová studie optimalizace jednoho predikčního modelu na dostupné datové řadě. Kapitola předkládá vyhodnocení této studie a posouzené obhájení hlavního zkoumaného problému – tedy optimalizovatelnosti predikčních modelů pomocí evolučních algoritmů.

Shrnutí přínosu práce, její úspěšnosti a posouzení stavu splnění úvodních požadavků a myšlenek, lze nalézt v poslední kapitole - *Závěr*.

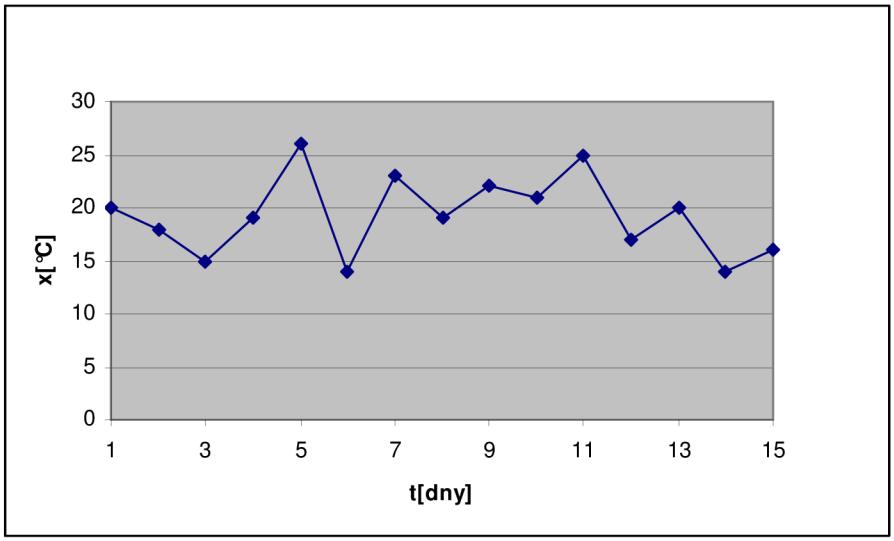
2 Časové řady

V běžném životě se setkáváme s řadou jevů, jež jsou závislé na čase. S přirozenou lidskou vlastností popisovat jevy kolem sebe a snažit se případně i předpovídat jejich vývoj, vznikl myšlenkový model těchto jevů a to *časové řady*.

Formálně bychom časovou řadu mohli definovat jako množinu pozorování x_t veličiny x, každé zaznamenané v určitém čase t. Dále rozlišujeme *časové řady diskrétního času*, kdy množina T_0 všech časových okamžiků pozorování je spočetná (v této práci se budeme zabývat především tímto typem časových řad) a *kontinuální časové řady*, když jsou pozorování v určitém časovém intervalu zaznamenávány kontinuálně[1].

2.1 Příklad časové řady

Způsob zápisu a zobrazení dat časové řady můžeme uvést na smyšleném příkladu měření denní teploty v průběhu poloviny měsíce. Soubor pozorování náhodné (respektive u většiny časových řad se snažíme o následnou analýzu a nalezení jistých závislostí, do té doby je ovšem pro nás takováto časová řada náhodná) veličiny $\{x_t, t \in T\}$, kde T je množina časových okamžiků $T = \{1, 2, \ldots, 15\}$. Časovou řadu můžeme také znázornit formou grafu (tedy ve formě závislosti pozorované veličiny na čase) – příklad takového znázornění naší smyšlené časové řady znázorňuje Obr. 2.1.



Obr. 2.1.: Graf časové řady – teplota vzduchu v Brně v jednotlivých dnech září 2008

2.2 Analýza časových řad

Jak již bylo zmíněno v předchozí kapitole, v době zaznamenávání hodnot časové řady se nám tyto mohou jevit jako naprosto náhodné. To by ovšem výrazně ztížilo další praktické využití časové řady, proto se snažíme časové řady analyzovat a tím si umožnit:

- *porozumění mechanizmu* generujícímu hodnoty časové řady (včetně pochopení podmínek a vazeb působících na vznik těchto hodnot),
- *konstrukce modelu* mechanizmu, který jsme odhalili (příkladem může být odvození matematického popis – „vzorečku“ – fyzikálních dějů) a
- *simulace* pomocí tohoto modelu (například za účelem doplnění chybějících hodnot, předpovědi budoucích hodnot atp.).

Hlavním cílem analýzy časových řad je tedy hledání jejich modelu. ***Model časové řady*** pro pozorovaná data $\{x_t\}$ bychom si formálně mohli definovat jako specifikaci rozmístění bodů časové řady (případně i pouze střední hodnoty a rozptylu) náhodných proměnných $\{X_t\}$, kde řadu $\{x_t\}$ nazýváme jako *realizaci* [1].

Pokud se nám podaří odvodit přesný model časové řady tak, že jsme pomocí něho schopni přesně vypočítat jakoukoliv (i budoucí) hodnotu řady, pak tento model nazýváme *deterministickým modelem časové řady*. Typickým příkladem je právě výše zmiňovaný matematický popis ideálního fyzikálního děje („vzoreček“). Takovéto modely jsou ovšem pro řadu reálných řad prakticky nezjistitelné – pak vytváříme takzvané *stochastické modely časových řad*. V této práci se budeme dále zabývat hledáním takovýchto modelů. Typický postup zahrnuje statistickou analýzu získaných hodnot tak, jako si to ukážeme v následující kapitole.

2.2.1 Postup analýzy časových řad

Jádrem analýz časové řady je její dekompozice na systematické složky (respektive nalezení případné existence těchto složek), a to:

- trendové složky T_t,
- sezónní složky S_t,
- cyklické složky C_t *a*
- náhodné složky ε_t.

Výsledné hodnoty zkoumané časové řady jsou pak součtem hodnot těchto složek v daném časovém bodě *t* (*aditivní model*), popřípadě součinem těchto složek (*multiplikativní model*).

Dále si popíšeme hlavní vlastnosti těchto složek (viz [4]).

2.2.1.1 Trendová složka

Někdy také zkráceně označovaná jako trend zachycuje dlouhodobé změny v chování časové řady. Nejde tedy o krátkodobé změny v průběhu časové řady, ale o to, jaké má vývoj řady tendence z dlouhodobého hlediska. Většinou je možné trendovou složku popsat jedinou matematicko funkcí v celém průběhu časové řady.

2.2.1.2 Sezónní složka

Sezónní složka popisuje periodické změny v časové řadě, které se odehrávají v rámci jednoho časového období a každý další časové období se opakují. Příkladem může být opakování jistého vývoje v časové řadě pro jednotlivá roční období po každý kalendářní rok (odtud také název složky). Přestože se tato složka pravidelně v časové řadě opakuje, může v průběhu jednotlivých časových období měnit svůj charakter.

2.2.1.3 Cyklická složka

Popisuje dlouhodobé fluktuace kolem trendu. Zachycuje tedy dlouhodobou fázi poklesu či růstu, která je mnohem větší než jedno časové období u sezónní složky. U ekonomických řad je cyklická složka často spojována se střídáním hospodářských cyklů. Protože působí dlouhodobě, je velmi obtížné ji vysledovat a popsat. Perioda cyklické složky se může být příliš dlouhá na to, abychom ji u krátké časové řady rozeznaly. Navíc se charakter této složky může v čase měnit.

2.2.1.4 Náhodná složka

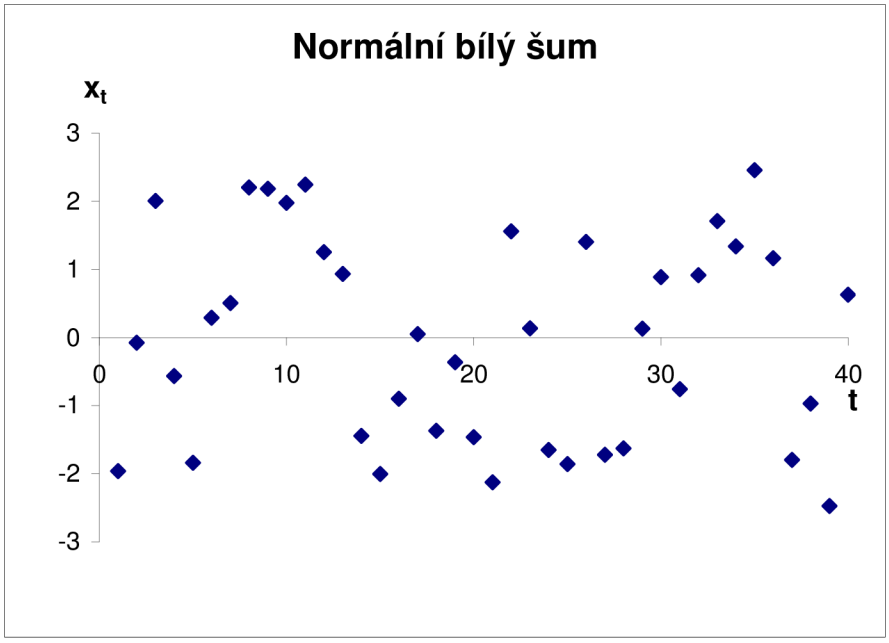
Náhodná složka je nesystematická (na rozdíl od předchozích tří složek) a je tvořena náhodnými výkyvy časové řady. Do této složku můžeme zařadit všechny vlivy, které na časovou řadu působí a které nedokážeme systematicky podchytit a popsat. Protože je to složka, která nám zůstane po vyloučení předchozích složek, je také někdy nazývána jako *reziduální složka* (či *rezidua modelu*).

Toto složka musí splňovat následující vlastnosti [4]:

- $E(\varepsilon_t) = 0, \ \forall\, t \in T$ (střední hodnota je nulová), tedy náhodná složka neovlivní ostatní složky – *nevychýlenost náhodné složky.*
- $D(\varepsilon_t) = \sigma^2, \ \forall\, t \in T$ (rozptyl je konstantní), tedy variabilita náhodné složky nezávisí na hodnotách systematických složek – *homoskedasticita náhodné složky.*
- $Cov(\varepsilon_t, \varepsilon_{t'}) = 0, \ t \neq t' \ \wedge \forall\, t, t' \in T$ (kovariance je nulová), tedy hodnoty náhodní složky jsou navzájem nezávislé (nekorelované) – *nekorelovanost náhodné složky.*

Časová řada splňující výše zmíněné podmínky se označuje jako tzv. *bílý šum*. Pokud je navíc splněna podmínka normálního rozdělení, pak jde o tzv. *normální bílý šum* (příklad viz Obr. 2.2).

Význam této složky spočívá především ve statistickém testování správnosti modelu – různými testy ověřujeme splnění uvedených kritérií, pokud tato kritéria splněna nejsou, značí to špatné matematické popsání ostatních (systematických) složek časové řady.



Obr. 2.2.: Graf normálního bílého šumu – hodnoty generovány pomocí generátoru pseudonáhodných čísel s potřebnými parametry nástroje MS Excel

2.2.2 Postup modelování časových řad

Cílem konstrukce modelu časové řady je nalezení takových modelů systematických složek (trendové, sezónní, cyklické), aby výsledná residuální složka (tedy nesystematická náhodná složka) měla právě vlastnosti bílého šumu. Pro komplexní časové řady je na místě tento postup rozšířit. Dle [1] je pak postup vytvoření modelu časové řady následující:

- Nalezení trendové a sezónní složky řady.
- Odstranění trendové a sezónní složky řady – existují různé metody jak tohoto dosáhnout. Můžeme se například nalézt modely těchto složek a poté je od hodnot řady odečíst. Nebo můžeme provést diferenci řady – nahrazení řady $\{X_t\}$ řadou $\{Y_t := X_t - X_{t-d}\}$ pro nějaké $d \in N$. Cílem těchto postupů je nalézt *stacionární* (viz dále) řadu označovanou jako *rezidua*.
- Nalezení modelu pro popis reziduí.
- Simulace modelu pak obnáší simulace modelu popisujícího rezidua a invertování transformací popsaných výše a tím získání modelových hodnot $\{X_t\}$.

Nyní si popíšeme jednotlivé aspekty tohoto postupu.

2.2.2.1 Stacionární řady a modely

Stacionární řada je taková řada $\{X_t\}$ jejíž statistické vlastnosti se neliší od libovolné časově posunuté řady $\{X_{t+d}\}$ pro libovolně zvolené celé d. Formálně bychom pak stacionaritu řady mohli definovat následovně [1]:

$\{X_t\}$ je stacionární pokud

- $E(X_t)$ (střední hodnota řady) je nezávislá na t.
- $Cov(X_t, X_{t+h})$ (kovariance) je nezávislá na t pro libovolné h, kde
  - $Cov(X_r, X_s) = E[(X_r - E(X_r))(X_s - E(X_s))]$

Stacionární model je pak takový model, který generuje stacionární časové řady. Nejčastěji používané stacionární modely si uvedeme v následující kapitole.

2.2.2.2 Stacionární modely reziduí

Asi nejjednodušším příkladem stacionárního modelu je **bílý šum**. Z jeho definice v kapitole 2.2.1.4 je zřejmé, že podmínky stacionarity splňuje. Pokud jsou rezidua naší časové řady bílým šumem, pak jsme v této fázi modelování hotovi – stačí rezidua namodelovat jako zdroj bílého šumu a poté provést zpětné transformace pro aplikaci trendové, sezónní a cyklické složky.

Jedním z nejdůležitějších modelů procesů je **lineární model**. Lineární model vzniká sumací potřebného počtu signálů s vlastnostmi bílého šumu, formálně

$$X_t = \sum_{j=-\infty}^{\infty} \psi_j Z_{t-j} \text{ [1]}, \qquad (2.1.)$$

kde $\{Z_t\} \sim WN(0,\sigma^2)$ (bílý šum s nulovou střední hodnotou a rozptylem σ^2) a $\{\psi_j\}$ je množina konstant takových, že $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$.

Všechny následující modely, které si zde popíšeme, splňují tyto podmínky, tedy jsou speciálními případy lineárního modelu.

Dalším významným modelem je **autoregresní model**. Obvykle jej označujeme jako **AR(*p*)**, kde p je řád autoregresního modelu. Formálně tento model popisuje vzorec (2.2) [4]

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t , \qquad (2.2.)$$

kde c je konstanta, φ_i jsou parametry modelu a ε_t je bílý šum.

Klouzavý průměr řádu q, schématicky značený jako **MA(*q*)**, je další ze stacionárních lineárních modelů. Tento model je definován vzorcem (2.3) [4]

$$X_t = \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} , \qquad (2.3.)$$

kde θ_i jsou parametry modelu a ε_t je bílý šum.

Integrovaný model časové řady je speciálním případem, který nám umožní pracovat s některými nestacionárními řadami jako se stacionárními (často je aplikovatelný na řadu procesů v ekonomice). Řada $\{x_t\}$ je integrovaná řádu d, jestliže se stane stacionární po d-násobné diferenciaci. Příkladem může být integrovaná řada prvního řádu $\{x_t\}$, pro níž zavedeme řadu $\{v_t\}$ definovanou tak, že $v_t = \Delta x_t$, pak platí [5]:

$$v_t = \sum_{k=0}^{\infty} \Phi_k \varepsilon_{t-k} \,, \qquad (2.4.)$$

kde Φ_k jsou parametry modelu a ε_t je bílý šum.

tedy řada $\{v_t\}$ je stacionární a navíc i lineární.

Komplexnější modely získáme kombinací výše uvedených, označujeme je pak zkratkami těchto základních modelů (AR pro autoregresní model, MA pro klouzavý průměr a I pro integrovaný model). Vznikají tak modely **ARMA, AIRMA** a další.

2.2.3 Alternativní - biologií inspirované - modelování časových řad

V předchozí kapitole jsme si uvedli některé základní matematické modely časových řad. Někdy jsou ovšem tyto modely, i modely vzniklé jejich kombinacemi, nedostatečné pro požadovaný popis určitých časových řad. Pokud by pak vytvoření vyhovujícího komplexního modelu bylo neúměrně náročné na naše schopnosti anebo zdroje, můžeme využít alternativní – především biologií inspirované – postupy.

2.2.3.1 Klasifikace biologií inspirovaných algoritmů

Tvorba počítačových algoritmů prošla za svoji existenci rozsáhlým vývojem, který mimo jiné potvrdil možnost praktického využití netradičních přístupů ke tvorbě a návrhu algoritmů a později dokonce celých výpočetních strojů realizujících tyto výpočty. Jednu z nejrozšířeněji využívaných skupin algoritmů této kategorie tvoří *biologií inspirovaná algoritmy* (další skupiny tvoří například algoritmy inspirované fyzikálními ději, kvantovými principy apod.). Podrobnějším popisem algoritmů této skupiny se zabývá například literatura [4], my si zde uvedeme pouze stručnou taxonomii biologií inspirovaných algoritmů dle tohoto zdroje:

Algoritmy inspirované fylogenezí – Tedy evoluční algoritmy (fylogeneze označuje průběh evoluce biologických druhů, pro naše účely jsou tyto dva pojmy v podstatě zaměnitelné). Algoritmy této skupiny napodobují proces fylogeneze pomocí ohodnocování kvality jednotlivých řešení - jedinců a následně selekce určitého procenta těchto řešení na základě jejích ohodnocení, případně zároveň aplikují určité procento rekombinací a náhodných úprav (mutací) na tato řešení – tento proces se iterativně opakuje dokud nedosáhne uspokojivých výsledků (anebo provedeme určitý počet iterací).

Algoritmy inspirované ontogenezí – algoritmy této skupiny se inspirují procesem vývoje organizovaného mnohobuněčného organismu z jednobuněčného zárodku. Algoritmy tento proces v podstatě velmi zjednodušují zavedením určité sady pravidel aplikované na každou atomickou částici řešení. Do této skupiny řadíme například *L-systémy* nebo *celulární automaty* nacházející uplatnění v teoretických matematických a fyzikálních oborech, počítačové grafice apod.

Algoritmy inspirované epigenezí – Epigenezí zde míníme teorii vývoje chování jedince (tedy vlastně jeho učení). V biologii jde o změnu struktury jedince vyvolanou podněty okolí – v biologické struktuře se tak projevuje paměťový efekt. Určité paměti, či zpětné vazby, se pak snažíme dosáhnout v algoritmech inspirovaných těmito biologickými procesy – algoritmy se takzvaně *učí* na základě vnějších impulsů. Do této kategorie řadíme algoritmy umělých neuronových sítí, *umělých imunitních systémů* apod.

Speciální skupina těchto algoritmů vznikla snahou inspirovat se chováním jedinců ve skupinách – a to jak chováním kooperačním, tak soupeřícím. Do této skupiny řadíme například tzv. *agentní a multiagentní systémy*, *optimalizace na bázi kolonií mravenců* apod.

2.2.3.2 Role biologií inspirovaných algoritmů v predikci časových řad

Jak bylo naznačeno výše, je možné úspěšně použít biologií inspirované algoritmy v úloze modelování a predikce časových řad. Různé algoritmy ovšem mají ovšem v modelování odlišné role.

Neuronové sítě – Neuronová síť vlastně provádí postupnou aproximaci neznámé hledané funkce transformující vstupní vektor délky *m* na výstupní vektor délky *n*. Více o technickém pozadí tohoto typu algoritmů podává literatura (např. [7]). Vlastní natrénovaná neuronová síť se tak často používá jako model časové řady, který pro *m* vstupních historických bodů časové řady předpoví jeden výstupní – budoucí – bod časové řady.

Genetické algoritmy, optimalizace na bázi kolonie mravenců a další – Tyto algoritmy provádí hledání optimálního řešení úlohy specializovaným prohledáváním *m*-rozměrného stavového prostoru. Jejich využití v modelování časových řad tak spočívá spíše v *optimalizaci* existujícího typu modelu (tj. hledání vhodné kombinace parametrů modelu). Postupem by bylo například zvolit jistý typ modelu časových řad, tak jak jsme je uvedly v kapitole 2.2.2.2, a hledat nejvhodnější kombinaci parametrů (viz vzorce (2.1) až (2.4)).

2.3 Aplikace časových řad na finančních trzích

Jednou ze zajímavých aplikací časových řad je snaha modelovat – a tím i předpovídat – chování procesů ve finančnictví. Aplikovatelnost těchto teorií předpokládá existenci jistých vzorů ve vývoji cen aktiv. Formálně řečeno, ceny aktiv nesledují takzvanou *náhodnou procházku* (*random walk*) a tudíž existují *neefektivity finančních trhů*. Co tyto pojmy znamenají a jaké je možné praktické využití těchto znalostí si naznačíme v následujících podkapitolách.

2.3.1 Náhodná procházka

V kapitole 2.2.1.4 jsme se seznámili s náhodným procesem nazvaným *bílý šum*. Kumulativní sumací (řekněme integrováním časové řady) bílého šumu získáme jiný náhodný proces – takzvanou

náhodnou procházku. Příklad náhodné procházky získané tímto způsobem z konkrétní realizace procesu bílý šum na Obr. 2.2 vidíme na Obr. 2.3.

Obr. 2.3.: Graf náhodné procházky – hodnoty generovány pomocí generátoru pseudonáhodných čísel s potřebnými parametry nástroje MS Excel

Formálně je časová řada normální $\{S_t\}$ procházky definována vzorcem (2.5)

$$S_t = S_0 + \sum_{i=1}^{t} X_i \,, \qquad (2.5.)$$

kde $\{X_t\}$ je normální bílý šum [8]. Pokud položíme S_0 rovno 0, pak získáme náhodnou procházku s nulovou střední hodnotou.

Významnou vlastností náhodné procházky je to, že libovolná hodnota v čase *t*, je nezávislá na předchozích hodnotách této časové řady. V analogii u finančních trhů by toto znamenalo, že budoucí vývoj cen aktiv je absolutně nezávislý na historickém vývoji těchto cen – formálněji řečeno, že v časových řadách vývoje cen aktiv neexistuje žádná *autokorelace*.

2.3.2 Efektivita finančních trhů

Teorie, že ceny aktiv nesledují náhodnou procházku, úzce souvisí s popíráním *teorie efektivity trhů* (*EMT – Efficent market theory*) a její konkurenční *behaviorální teorií financí (BFT – behavioral finance theory*). Teorie efektivity finančních trhů předpokládá, že všichni investoři investují racionálně (vliv všech případných iracionálních investorů je naprosto náhodný a tedy se navzájem vyruší) a tedy, že ve svém investování vždy využijí všechny své znalosti. Trh je pak takzvaně *informačně efektivní* a tudíž není možné existující informace použít pro předpověď budoucího vývoje cen, protože tyto informace ovlivnily ceny okamžitě jakmile vznikly. A s náhodnou povahou všech

informací, které mohou ovlivnit vývoj cen, je pak i samotný vývoj cen naprosto náhodný. Důsledkem této teorie je tvrzení, že není možné dlouhodobě profitovat z obchodování s aktivy finančního trhu, aniž by takovýto zisk nebyl dílem absolutní náhody (jinými slovy – aby nebyl stoprocentně vyvážen rizikem investic).

V literatuře zabývající se teorií finančních trhů můžeme nalézt tři formy *informační efektivity trhu* podle typu informací, k nimž mají investoři přístup (viz např. [9]):

- *Slabá efektivita trhu* – při této formě informační efektivity trhu investoři znají (a tedy ceny aktiv automaticky reflektují) všechny historické informace vývoje cen a výnosů daného aktiva.
- *Polosilná efektivita trhu* – investoři znají všechny veřejně dostupné informace v daný časový okamžik.
- *Silná efektivita trhu* – investoři mají přístup ke všem existujícím informacím – a to jak veřejným tak neveřejným.

Přestože má tato teorie řadu příznivců i mezi držiteli Nobelových cen za ekonomii, řada praktických zkušeností i teoretických závěrů naznačují, že nemusí být zcela správná. Jedna z nejúspěšnějších teorií oponující teorii efektivního trhu je *Behavioral finance theory*, což bychom do češtiny mohli přeložit jako behaviorální teorie financí. Hlavní myšlenkou této teorie je, že je rozdíl mezi dostupností informací a jejich interpretováním a emočním obohacení investory. Praktickým dopadem je vysledovatelnost jistých pravidelností ve finančních časových řadách.

2.3.2.1 Korelační vzory finančních časových řad

Jedním z nejzajímavějších anomalit v časových řadách finančních trhů, které behaviorální teorie vysledovala, je existence tří následujících vzorů [2]:

- *krátkodobé zvraty* – významná změna cen aktiv bývá následována další významnou změnou aktiv, aniž by tato musela mít důvod v dostupných informacích (směr této změny je ovšem těžké predikovat). Jinými slovy: vývoj cen aktiv bude pravděpodobně nestálý, pokud je v současnosti nestálý. Zdroj této anomálie můžeme vidět v lidských emocích jako jsou panika či euforie, které mohou ovlivnit investory k investování nezaloženém čistě na racionální interpretaci dostupných informací.
- *střednědobá setrvačnost* – relativně stálý vývoj ceny aktiv v délce přibližně 3 až 12 měsíců často indikuje setrvačnost orientace trendu cen těchto aktiv (ve smyslu růstu/klesání cen) v blízkém budoucím časovém horizontu.
- *dlouhodobé zvraty* – naopak dlouhodobý vývoj cen jedním směrem (ve smyslu růstu/poklesu) má tendenci k objevení se negativní autokorelace – tedy návratu ceny k nižší/vyšší hodnotě.

2.3.3 Praktické využití neefektivity finančních trhů

Předchozí kapitola naznačila reálnou možnost existence neefektivit trhu způsobující možnost přechodných výskytů autokorelací v časových řadách vývoje cen aktiv na finančních trzích. Tyto neefektivity plynou ze skutečnosti, že v pozadí finančních procesů můžeme vždy vysledovat lidský faktor investorů. Dle literatury (viz např. [2], [9]) tvoří v naprosté většině odvětví obchodování na finančních trzích nadpoloviční objem transakcí spekulativní obchodování za účelem zisku z tohoto obchodování (na mezinárodním devizovém trhu, *FOREX – Foreign Exchange*, tvoří údajně objem spekulativního obchodování až 95% objemu veškerého obchodování na tomto trhu). Tím spíše je lidský faktor vysledovatelný a myšlenka absolutně efektivního trhu vyvíjejícího se absolutně bez emocí a jakékoliv závislosti se zdá být příliš nadhodnocená.

Pokud bychom byly ochotni uvěřit těmto argumentům a připustili možnost smysluplného obchodování na finančních trzích, pak si musíme uvědomit, že možnost profitování z neefektivit trhu existuje jen díky lidskému faktoru obchodování, který na finanční trhy vnáší investoři. Obchodování za účelem zisku je pak tedy ve své podstatě soupeřením mezi jednotlivými investory a jejich schopnostmi vypozorovat navzájem své obchodní záměry a taktiky.

Úspěšnější pak bude jednoznačně ten investor, který nejrychleji vypozoruje vyskytnuvší se anomálie a navíc sám bude jednat co nejracionálněji. Toto se investorovi podaří nejlépe, pokud své obchodování sofistikovaně zautomatizuje za pomocí výpočetní techniky – to mu umožní maximální rychlost a neemociálnost reakcí na tržní změny.

Pro zautomatizování obchodování je ovšem nutné přesně formalizovat obchodní taktiku (a tu následně vhodně implementovat). To může být ovšem velmi složité – investor pravděpodobně jedná částečně podle své intuice a také své taktiky obměňuje podle jejich úspěšnosti a podle vývoje trhu. Takto pružně nemůže žádný formalismus fungovat – naskýtá se tedy možnost použít *biologií inspirované algoritmy*, které mají adaptabilitu ve své podstatě. Jejich použití jsme si naznačili v kapitole 2.2.3.2. Nyní se zaměříme na speciální kategorii těchto algoritmů – *evoluční algoritmy*.

3 Evoluční algoritmy

Evoluční algoritmy jsou výpočetním modelem patřícím do třídy algoritmů umělé inteligence (někdy též nazývaných jako soft computing). Evoluční algoritmy používají některé principy biologické evoluce (reprodukce, mutace, rekombinace a selekce) k realizaci metaheuristické optimalizace [11]. Aplikací těchto principů na množiny kandidátních řešení umožňuje vyhledávat nejlepší řešení (globální extrémy prohledávaného prostoru) s nutností otestovat pouze malé množství ze všech teoreticky možných kandidátních řešení.

3.1 Dělení evolučních algoritmů

Podle způsobu implementace a zaměření na konkrétní třídu problémů rozlišujeme následující typy evolučních algoritmů, které je možné stručně charakterizovat takto [11]:

- *Genetické algoritmy* – nejpopulárnější typ evolučních algoritmů. Kandidátní řešení zakódujeme do řetězce čísel (obvykle do binárního řetězce) a aplikováním operátorů rekombinace a mutace na jednotlivé generace kandidátních řešení postupně iterujeme ke globálnímu optimu dané úlohy (respektive k dostatečně kvalitnímu lokálnímu extrému obecně vícerozměrné funkce).
- *Genetické programování* – Zde jednotlivé jedince evoluce představují celé programy řešící danou úlohu a dochází ke hledání cílové podoby (struktury) programu.
- *Evoluční strategie* – jsou podobné genetickým algoritmům, ovšem kandidátní řešení kódujeme pomocí řetězce reálných čísel. Každý rodič (skupina rodičů) produkuje skupinu potomků, k soupeření poté dochází v rámci této skupiny (rodiče a jejich potomci).
- *Evoluční programování* – podobné jako genetické programování, ovšem struktura programu je daná a vyvíjejí se pouze jeho číselné parametry. Hlavním operátorem bývá mutace (jedinci se berou jako zástupci různých druhů a k mezidruhovému křížení nedochází).

3.1.1 Genetické algoritmy

Jsou nejčastěji používanou a také nejznámější skupinou evolučních algoritmů. Nejčastějším uplatněním je (optimalizační) úloha hledání globálního extrému vícerozměrné funkce. Používají se především v jejich tradiční podobě označované jako *jednoduchý genetický algoritmus* (simple genethic algorithm - *SGA*) [12] v české literatuře též jako kanonický genetický algoritmus [10]. Pro tyto algoritmy nejčastěji platí [10],[11],[12]:

- Jedinec (kandidátní řešení, či chceme-li, fenotyp) je reprezentován binárním řetězcem x délky n (tedy svým genotypem)

$$x \in \{0,1\}^n \qquad (3.1.)$$

- Používá se proporcionální výběr rodičů z populace jedinců na základě jejich hodnoty *fitness* funkce f nad binárními řetězci x viz vzorec (3.2). Způsob výběru rodičů ovlivňuje rychlost konvergence k řešení, ale také pravděpodobnost uváznutí v lokálním extrému funkce.

$$f: \{0,1\}^n \rightarrow R \qquad (3.2.)$$

- *Rekombinace* rodičů je prováděna jednobodovým (i vícebodovým, případně jiným) křížením – tj. vzájemnou záměnou definovaných částí binárního řetězce.
- Používá se *Mutace* jednotlivých bitů binárních reprezentací jedinců. Nejčastějším způsobem mutace je bitová negace vybraných bitů. Pravděpodobnost vybrání bitu pro mutaci je pro každý bit každého řetězce celé generace rovna stejné hodnotě p_m, kde p_m je většinou v intervalu *<1/(bitů v generaci); 1/(bitů v chromozomovém řetězci)>*.

Tyto *jednoduché genetické algoritmy* mají však řadu nedostatků [12]:

- *Omezenost prostoru kandidátních řešení* – omezení je dáno délkou řetězce n (ze vzorce (3.1) vyplývá, že celkový počet možných kódování a tím i kandidátních řešení je 2^n). Pro větší přesnost je tedy třeba prodloužit délku chromozómů a tím zpomalit evoluci.
- *Proporcionální výběr jedinců* je náchylný pro vybírání jedinců s podobnou hodnotou *fintness* funkce (a tedy pravděpodobně méně variabilních jedinců).

Řešením těchto problémů je propracovanější výběr jedinců a způsobu jejich kódování – především pomocí řetězců přirozených anebo reálných čísel (a tím už se blížíme k evolučním strategiím).

3.1.2 Genetické programování

Tato technika se používá ke generování kódů řešících zadanou úlohu. Základními znaky odlišující genetické programování od ostatních technik evolučních algoritmů jsou [10],[11],[12]:

- Reprezentace jednotlivých jedinců populace je nejčastější v podobě stromových struktur představujících spustitelné kódy. Stromové výrazy jsou typicky netypované, vycházející z množiny terminálů T (vstupy do programu, konstanta a funkce bez argumentů s vedlejším účinkem) a množiny funkcí F s definovanými aritami dle následující rekurzivní definice [10]:
  1. Každé $t \in T$ je korektní výraz.
  2. $f(e_1,e_2,\ldots,e_n)$ je korektní výraz právě tehdy když $f \in F \wedge$ $\text{arity}(f) = n \wedge e_i$ je korektní výraz $\forall i \in \{1,2,\ldots,n\}$ (3.3.)
  3. Žádná jiná forma korektních výrazů neexistuje.

- Výběr jedinců do další generace probíhá proporcionálně na základě hodnoty funkce *fitness* (jejímž oborem hodnot je dle různých definic celé *R*, popř. interval <0;1>).
- K vyčíslení funkce dochází spouštěním vzniklého programu na *množině trénovacích vstupů* a následně vyhodnocením odchylky od požadovaných výstupů. Na konci evoluce je ovšem potřeba výsledného jedince zkontrolovat pomocí *množiny testovacích vstupů* (pro ověření dostatečné generalizační schopnosti vzniklého jedince).
- *Rekombinace* probíhá pro vybrané rodičovské dvojice jedinců výběrem náhodných podstromů v obou jedincích a jejich vzájemným prohozením (vzniknou tak dva potomci).
- *Mutace* probíhá pro zvoleného jedince náhodným výběrem podstromu a jeho nahrazením korektním náhodně vygenerovaným podstromem.
- Aplikace operátorů rekombinace a mutace je pro všechny evoluční algoritmy esenciální, neméně důležitá je volba jejich aplikace. Proto je třeba poznamenat, že dle některých pramenů ([12]) se v genetickém programování striktně volí buď rekombinace dvou rodičovských jedinců anebo (s řádově nižší až nulovou pravděpodobností) mutace jednoho rodičovského jedince, zatímco v jiných zdrojích ([10]) je možnost aplikovat oba operátory současně ponechána na uživateli.

Také *genetické programování* má své nedostatky – některé z nich jsou odstranitelné, jiné v současnosti nikoliv:

- *Rozrůstání se jednotlivých jedinců* – a to jak vznikem úseků kódu neovlivňující výsledek výpočtů (takzvaných *intronů*), tak vznikem správně počítajících jedinců, které mají ovšem přemrštěnou velikost (takzvaných *bloatů*). Tento nedostatek je možné poměrně efektivně řešit vhodně zvoleným penalizováním velikosti jedinců při vyčíslování jejich *fitness* funkce.
- *Oblast použitelnosti genetického programování* – v současnosti je nemyslitelné, aby se touto technikou běžně navrhovaly rozsáhlejší programy. Technika je vhodná pro vývoj drobnějších dobře definovatelných programů, jejichž struktura se konvenčními technikami hledá obtížně (například různé prediktory nebo klasifikátory dat apod.).
- *Nízká generalizace výsledného programu* – postupnou evolucí vytvoříme program dobře zpracovávající problémy z *trénovaní množiny*, to ovšem nemusí implikovat optimální řešení všech problémů z daného prostoru. Tento problém je třeba alespoň detekovat pomocí tzv. *testovací množiny*. Ta ovšem neumožňuje vylepšit samotné řešení – pouze identifikovat evoluci na jejíž výsledky se není možné spoléhat.
- *Rychlost konvergence* – bývá extrémně pomalá. Jednak z důvodu časové náročnosti výpočtu fitness funkce (spouštěním všech generovaných programů na všechny vstupy trénovaní množiny, kdy navíc generované programy se mohou velmi lišit svojí

výpočetní náročností), tak z důvodu potřeby velikého množství jedinců v každé generaci. K částečnému řešení tohoto problému existuje řada sofistikovaných pokročilých technik (viz například zdroj [10]).

3.1.3 Evoluční strategie

Tento typ evolučních algoritmů tak může dosahovat lepších výsledků v kratším čase pro úlohy hledající optimum reálné funkce n proměnných (její úspěšnost v těchto úlohách je ovšem, jako u ostatních uvedených algoritmů, nezaručitelná a závislá na konkrétním problému a volbě parametrů algoritmu). Pro evoluční strategie je typické [12]:

- *Reprezentace* jedinců je v podobě vektoru reálných (či přirozených) čísel představujících kandidátní řešení. Často jsou ovšem součástí kódování jedince i řídící parametry evoluce v podobě hodnot mutačních kroků (rozptylů gaussovských šumů) a rotačních úhlů:

$$\boldsymbol{X} = <x_1,x_2,\ldots,x_n,\sigma_1,\sigma_2,\ldots,\sigma_n,\alpha_1,\alpha_2,\ldots,\alpha_k>, \qquad (3.4.)$$

  kde x_i jsou hodnoty hledaných parametrů, σ_i hodnoty mutačních kroků $\forall i \in \{1,2,\ldots,n\}$

  α_i jsou hodnoty rotačních úhlů $\forall i \in \{1,2,\ldots,k\}$

  $k = n(n\text{-}1)/2$ (odpovídá počtu prvků kovariační matice n mutačních kroků)

- *Mutace* je založena na upravení každého parametru každého jedince přičtením náhodné hodnoty gausovského šumu (se střední hodnotou 0 a rozptylem σ_i pro dané x_i). Nejdříve se pomocí mutace vygenerují nové hodnoty rozptylů σ_i a s těmito hodnotami se pak provádí mutace parametrů x_i (viz vzorec (3.5)).

$$\sigma_i' = \sigma_i e^{\tau.N(0,1)}$$

$$x_i' = x_i + \sigma_i'.N(0,1) \qquad (3.5.)$$

  kde $\tau \approx \frac{1}{\sqrt{n}}$, je tzv. *úroveň učení* a

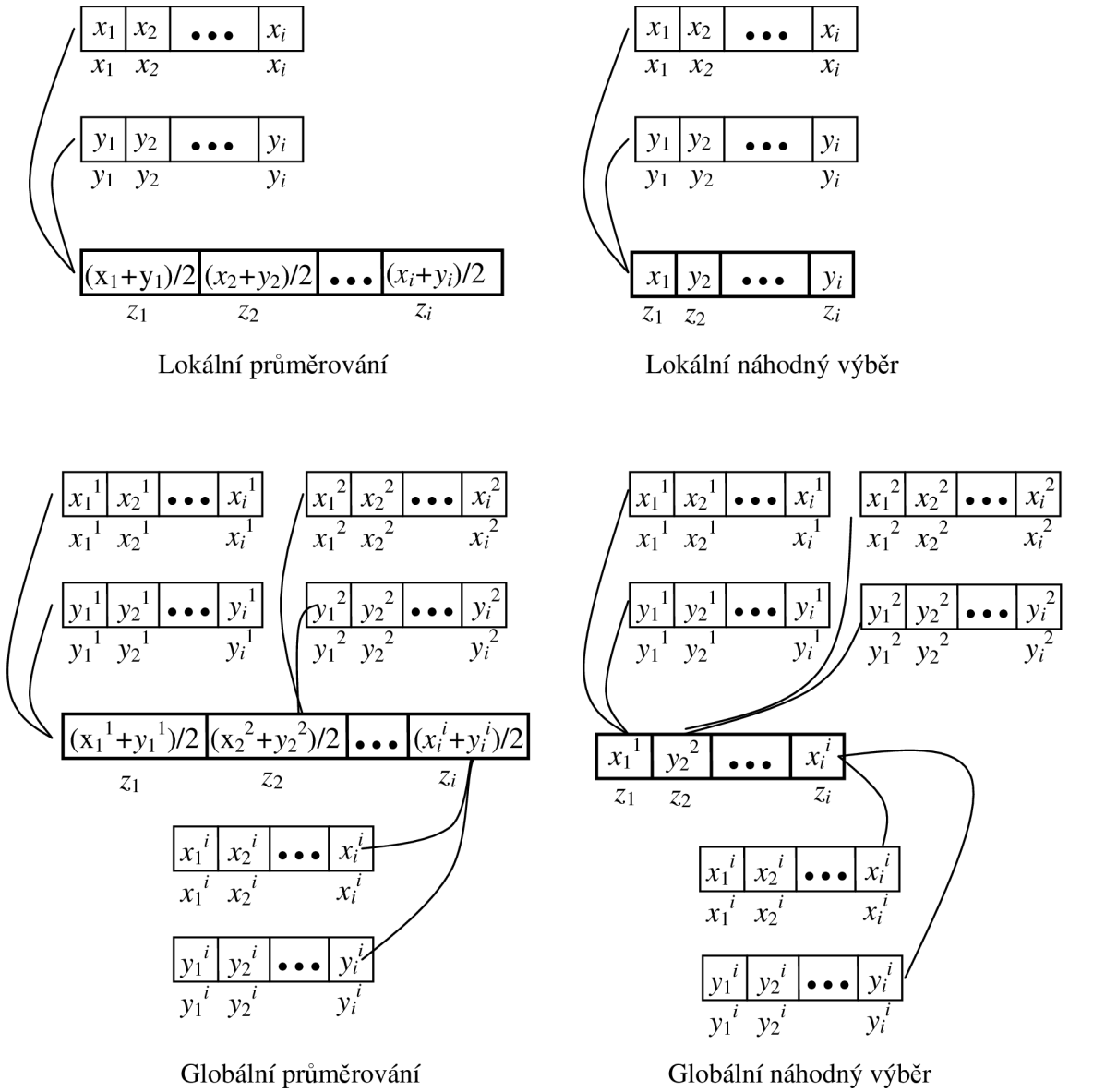
  N(0,1) je náhodná proměnná s normálním rozložením, nulovou střední hodnotou a rozptylem 1.

- *Rekombinace* v případě evoluční strategie může probíhat například těmito způsoby: průměrováním hodnot rodičů ($z_i = (x_i + y_i)/2$), náhodným výběrem z hodnot rodičů (z_i je náhodně vybraná hodnota z $\{x_i, y_i\}$), a u obou těchto způsobů se mohou rodičovské dvojce volit pro každý parametr zvlášť anebo jedna pro celého jedince (viz Obr. 3.1.). Zajímavým rozdílem oproti genetickým algoritmům je uniformní výběr jedinců ke křížení (tedy rovná šance všech jedinců na křížení bez ohledu na jejich *fitness*).

- *Výběr jedinců* do další generace probíhá po aplikaci rekombinace a mutace na základě *fitness* funkce. Rozeznáváme dva druhy výběru potomstva – (μ, λ) selekce, přežívají pouze nejlepší potomci a $(\mu + \lambda)$ selekce, přežívají nejlepší jedinci ze skupiny potomků a rodičů. Ve výše

uvedených modelech μ označuje počet rodičů a λ počet potomků, čím je násobnost λ oproti μ vyšší, tím je vyšší i tzv. *selektivní tlak* (vnější působení na změny jedince).

Nejzajímavější vlastností právě zmíněných algoritmů je právě schopnost autonomní dynamické změny probíhající evoluce (a to za pomocí výše zmíněné evoluce řídicích parametrů). Postupným snižováním mutačních kroků se algoritmus postupně přibližuje k optimu (tak jak je tomu například u metody simulovaného žíhání), to však ale může být pouze lokální, proto algoritmus pravidelně (obvykle po 200 krocích [12]) změní oblast prohledávání.



Obr. 3.1.: Způsoby rekombinace jedinců u evolučních strategií (čerpáno z [12])

3.1.4 Evoluční programování

Na svém počátku byly tyto algoritmy používány k učení konečných stavových automatů. Jako jedna z prvních úloh byla snaha pomocí evolučního programování navrhnout konečný automat schopný předpovídat binární časové řady. V dnešní době se používá k širšímu okruhu optimalizačních úloh a stírají se rozdíly oproti evolučním strategiím. V zásadě je však evoluční programování používáno k hledání optimálních parametrů existující struktury výpočetního modelu (typicky například vah neuronové sítě). Hlavními znaky této skupiny algoritmů jsou [12]:

- *Reprezentace* jedinců je typicky v podobě vektoru reálných čísel. Modernější nekanonické verze evolučního programování používají pokročilejší schémata reprezentace pro účely auto-adaptace mutačních kroků (obdobně jako tomu je u evolučních strategií, viz vzorec (3.4)).
- K *rekombinaci* v klasickém evolučním programování nedochází. Ke vztahu mezi jedinci prohledávajícími stavový prostor kandidátních řešení bývá dávána paralela se vztahem mezi zástupci různých živočišných druhů v přírodě – zde také nedochází k mezidruhovému křížení ale k vývoji v rámci druhů a soupeření mezi nimi navzájem.
- Operátor *mutace* má prakticky stejný význam jako u evolučních strategií – dochází k součtu hodnot parametrů s náhodným gausovským šumem.
- *Výběr jedinců* do další generace probíhá zpravidla turnajovým způsobem. Každé řešení ze skupiny μ rodičů a jejich λ potomků je porovnáván s náhodně vybranou skupinou q jedinců z celé generace. Do nové generace je vybráno μ jedinců s největším počtem „vítězství“. Pomocí parametru q můžeme obvykle měnit sílu selekčního tlaku.

V současnosti je pojem evolučního programování vymezen mnohem vágněji. Tyto algoritmy mohou používat prakticky libovolné kódování dat, variační operátory i selekční procedury (viz [13]). Tato skupina algoritmů tak splývá s ostatními skupinami, někteří autoři dokonce jako evoluční algoritmy označují celou skupinu genetické programování, evoluční strategie, evoluční programování tak, jak jsou vymezeny v této práci (viz [13]).

3.2 Evoluční algoritmy pro optimalizaci predikcí

Praktickým výstupem této práce bude systém schopný pomocí různých optimalizačních technik optimalizovat modely pro predikci finančních časových řad. Součástí bude také vytvoření a odzkoušení několika optimalizačních technik, především pak ze skupiny evolučních algoritmů. Ještě před samotnou realizací se v této podkapitole pokusíme analyzovat, na které ze skupin algoritmů uvedených v předchozích kapitolách by bylo vhodné se zaměřit více.

První z aplikací *evolučního programování* byla právě predikce (binárních) časových řad. Z tohoto pohledu se zdá být tato skupina algoritmů vhodným kandidátem. Další vhodnou vlastností

je, že se ve své klasické podobě používají především ke hledání vhodných kalibračních parametrů existujícího modelu – což bude prakticky náš případ, naší snahou totiž bude optimalizace existujících predikčních modelů.

Ve svém moderním pojetí evoluční programování téměř splývá s *evolučními strategiemi*, získáme tak v optimalizaci další potencionálně hodnotné aspekty jako *rekombinace*, více řídících parametrů schopných autoevoluce v kódování jedince apod.

Naproti tomu *genetické programování* je technika, při které dochází k vytváření nového výpočetního modelu řešícího zadanou třídu úloh. Z prvních fází naší práce, kdy budeme hledat vhodné algoritmy optimalizující existující predikční modely, tedy můžeme tuto techniku vyloučit.

Poslední nezmíněnou skupinou jsou *genetické algoritmy*. Ty se pro úlohu optimalizace predikčního modelu potencionálně hodí, mají ovšem řadu omezení vycházejících z používaného binárního kódování kandidátních řešení, které není pro optimalizaci vektoru reálných parametrů příliš vhodné. Této technice bude tedy pravděpodobně věnována menší pozornost případně bude použita v kombinaci s vlastnostmi ostatních technik (například kódováním kandidátních řešení pomocí reálných čísel).

4 Návrh systému optimalizace predikcí

Praktickým cílem této práce je návrh a implementace systému využívajícího evoluční algoritmus pro optimalizaci predikčního modelu. Detailněji by mělo jít o modulární systém schopný nalézt optimální vstupní parametry modulu predikčního modelu. Navíc by tento systém měl umožňovat škálovatelnost prováděných výpočtů – tedy schopnost provádět paralelní výpočty (jak na úrovni více jader či procesorů jednoho stroje, tak na úrovní více fyzických strojů).

Nejdříve si navrhneme předběžnou podobu architektury obecného systému splňujícího výše uvedený popis. Dále se pak zaměříme na detailnější návrh takovéhoto systému schopného optimalizovat moduly pro automatizované obchodování na finančních trzích. Při tom požijeme základní techniky objektově orientovaného návrhu.

4.1 Modulárnost a škálovatelnost systému

Aby byl systém co nejpružněji schopný optimalizovat různé implementace predikčního modelu, je nutné celý systém dekomponovat do co nejvíce nezávislých podsystémů. Pokud budou tyto podsystémy pro větší část svého výpočtu nezávislé na ostatních částech systému, umožníme takovou dekompozicí i snadnou škálovatelnost výkonu (prostým rozdělením celkové dostupné výpočetní kapacity mezi jednotlivé podsystémy). Další aspekt, který musíme uvážit v případě optimalizování predikčních modelů, je vysoká výpočetní náročnost výpočtů samotných predikcí – tedy z pohledu evolučních algoritmů vlastně simulace „života“ jedince za účelem vyčíslení jeho *fitness*. Bylo by tedy vhodné oddělit především generování nových jedinců od ohodnocování jejich fitness funkce.
Z tohoto pohledu bychom mohli v našem systému rozeznat tři hlavní komponenty:

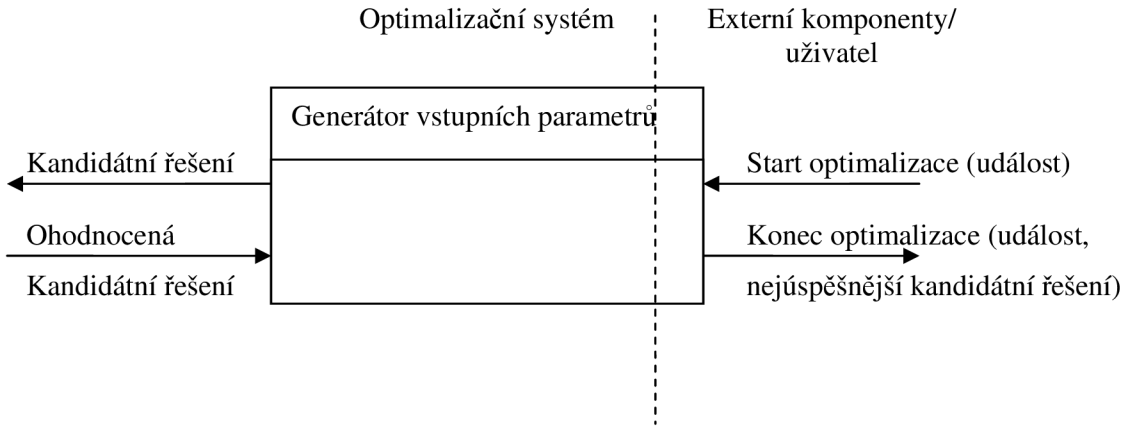
- komponenta generující a spravující jedince evolučního procesu (pro nás tedy vstupní parametry optimalizovaného predikčního modelu),
- komponenta(y) vyčíslující fitness pro jednotlivé jedince (provedení výpočtu pomocí hodnoceného predikčního modelu s vygenerovanými parametry a následné vyhodnocení úspěšnosti tohoto modelu) a
- komponenta rozdělující a spravující výpočty nezávislých komponent vyčíslujících fitness.

V těchto třech komponentách můžeme vidět základní stavební bloky našeho systému. Při vhodně navržené komunikaci můžeme tyto komponenty realizovat jako samostatné procesy potenciálně rozmístitelné na různé výpočetní stroje. Škálovatelnost výkonu je zajištěna možností přidávat další komponenty vyčíslující fitness generovaných jedinců (vstupních parametrů predikčního modelu) a tím zvýšení stupně paralelizace výpočtu. Zaměříme se tedy na paralelizaci navrženého

řešení ne na paralelizaci evolučního algoritmu samotného (a to především z důvodu velké časové a paměťové náročnosti ohodnocování jedinců – vzhledem k níž je režie ostatních komponent systému, a tedy i evolučního algoritmu, v podstatě zanedbatelná)

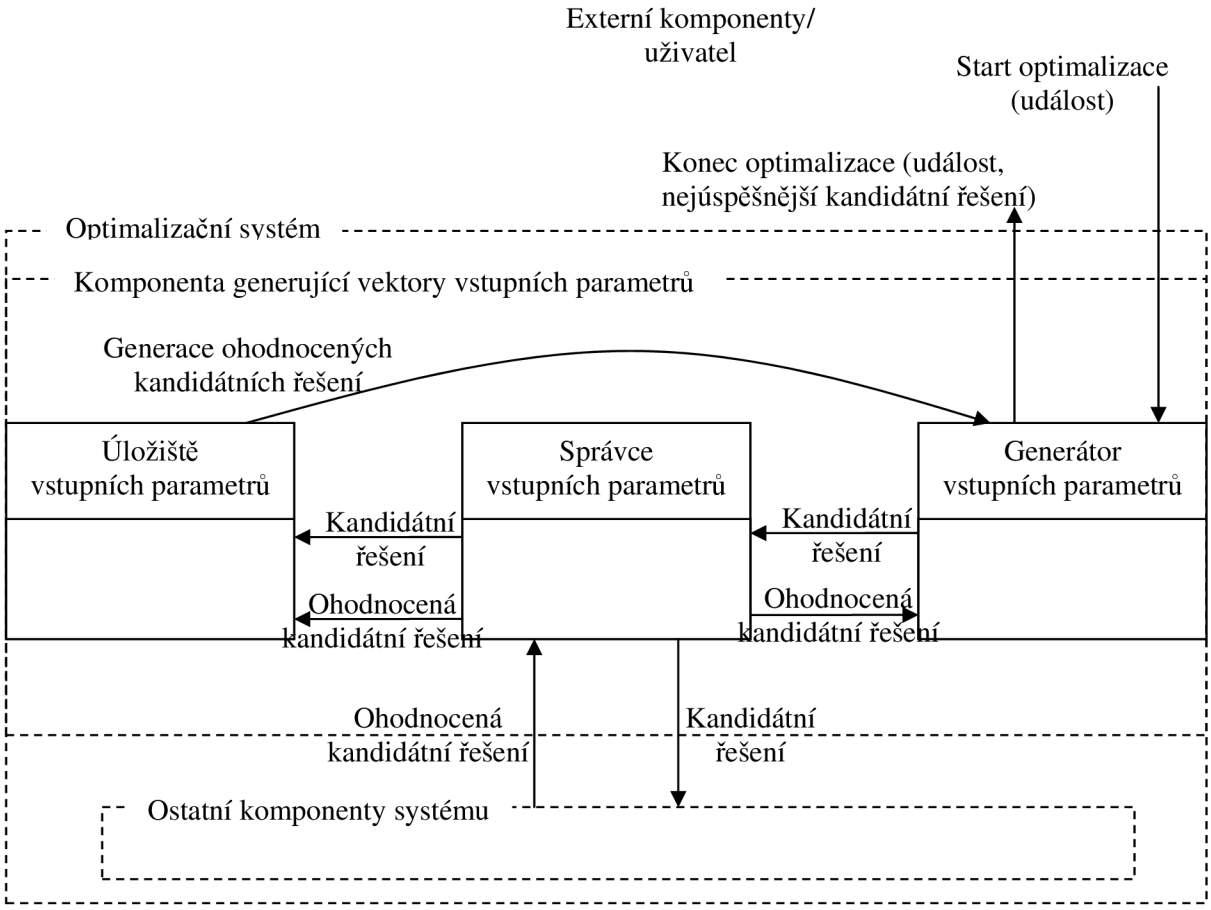
4.1.1 Komponenta generující vektory vstupních parametrů

Tato komponenta bude provádět hlavní část evoluční optimalizace vstupních parametrů predikčního modelu. Z principů evolučních algoritmů (viz kapitolu 3 Evoluční algoritmy) vyplývá, že pro optimalizaci je třeba generovat skupiny (tzv. generace) kandidátních řešení (tzv. jedinců), dále mít dostupné všechny ohodnocené kandidátní řešení z takovéto skupiny pro možnost vygenerování nové skupiny. Po nastaveném počtu kroků anebo dosažení požadované úspěšnosti kandidátního řešení algoritmus končí a jeho výstupem je nejúspěšnější řešení. Schéma rozhraní takovéto komponenty vidíme na Obr. 4.1.



Obr. 4.1.: Konceptuální schéma komponenty generující vstupní parametry predikčního modelu

Musíme ovšem vzít v úvahu, že tato komponenta potřebuje velmi pravděpodobně více než poslední ohodnocené kandidátní řešení – spíše celou populaci ohodnocených řešení. Abychom tohoto dosáhli, musíme někde ohodnocená kandidátní řešení uchovávat. Aby byl systém co nejmodulárnější a zároveň, aby byl snadno měnitelná konkrétní implementace evoluční optimalizace, měla by tento úkol plnit zvláštní podkomponenta. Navíc je potřeba zajistit komunikaci s ostatními částmi systému (provádějícími hodnocení kandidátních řešení), tento úkol by měla plnit další podkomponenta. Schéma rozhraní takto zjištěných komponent je znázorněno na Obr. 4.2.



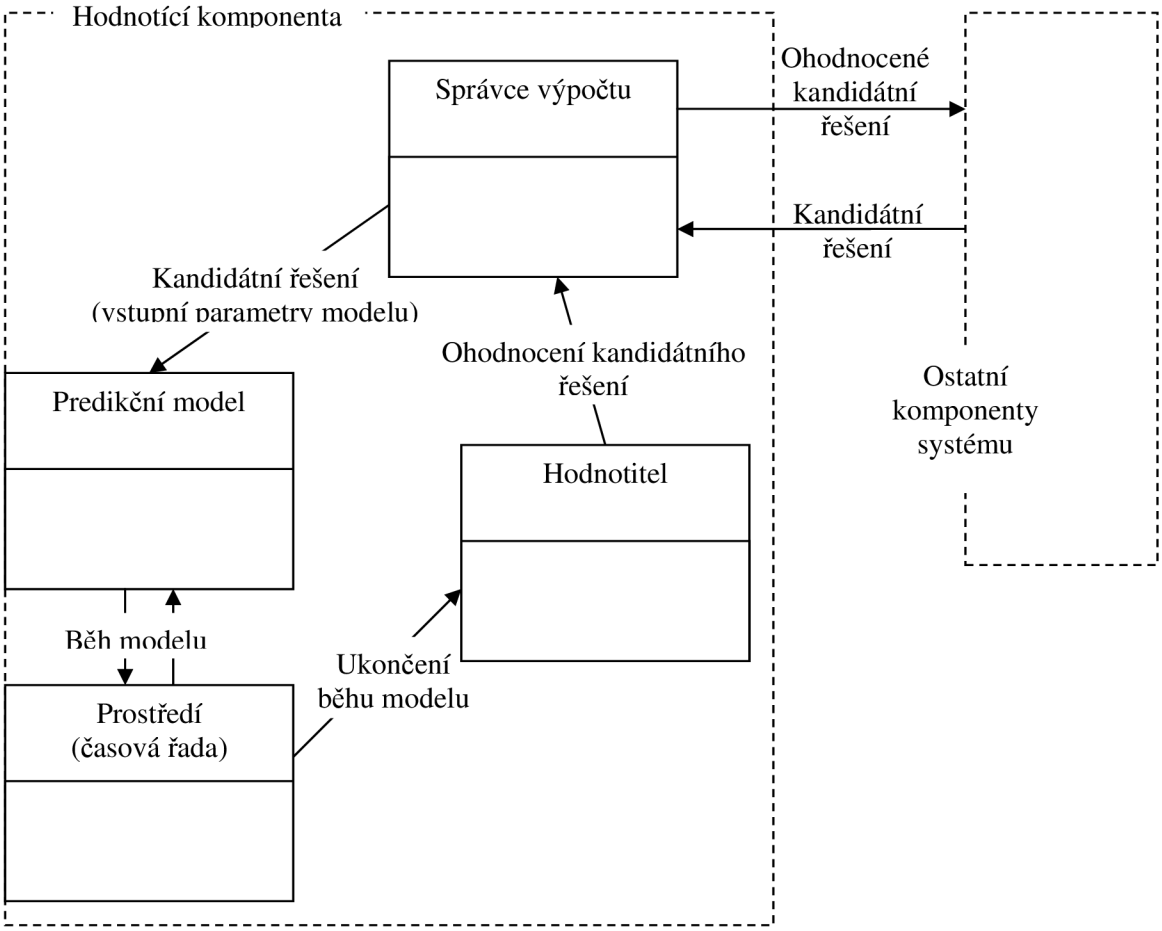
Obr. 4.2.: Konceptuální schéma komponenty generující vstupní parametry predikčního modelu, jejích podkomponent a navázání na další komponenty systému a externí rozhraní

4.1.2 Komponenta hodnotící vektory vstupních parametrů

Komponenta hodnotící vektory vstupních parametrů by měla na vstupu obdržet vektor vstupních parametrů predikčního modelu, odsimulovat běh predikčního modelu s těmito vstupními parametry v daném prostředí (tj. provést predikci na připravených datech časové řady), ohodnotit úspěšnost takto parametrizovaného predikčního modelu a toto ohodnocení vrátit zpět. V tomto popise můžeme rozeznat tři hlavní podkomponenty:

- predikční model,
- prostředí a
- hodnotitel úspěšnosti modelu.

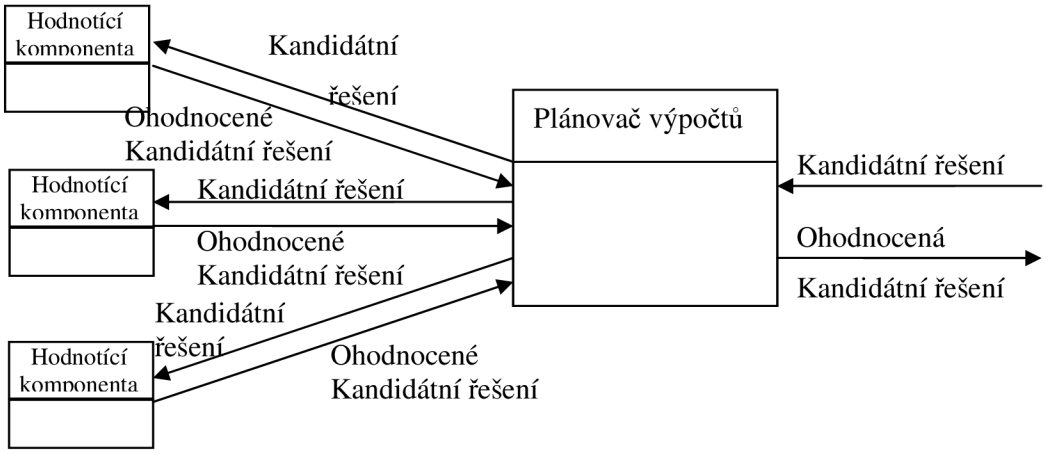
Pro praktické aspekty realizace hodnotící komponenty (možnost spouštět paralelně více výpočtů ve více vláknech, ohlašování volné výpočetní kapacity, automatizované stahování aktualizovaných knihoven apod.) bude ještě vhodné přidat řídicí komponentu, spravující běžící výpočty. Pokud bychom schématicky znázornili komunikaci těchto podkomponent a jejich spojení s ostatními částmi systému, získali bychom schéma Obr. 4.3.



Obr. 4.3.: Konceptuální schéma komponenty hodnotící vstupní parametry predikčního modelu

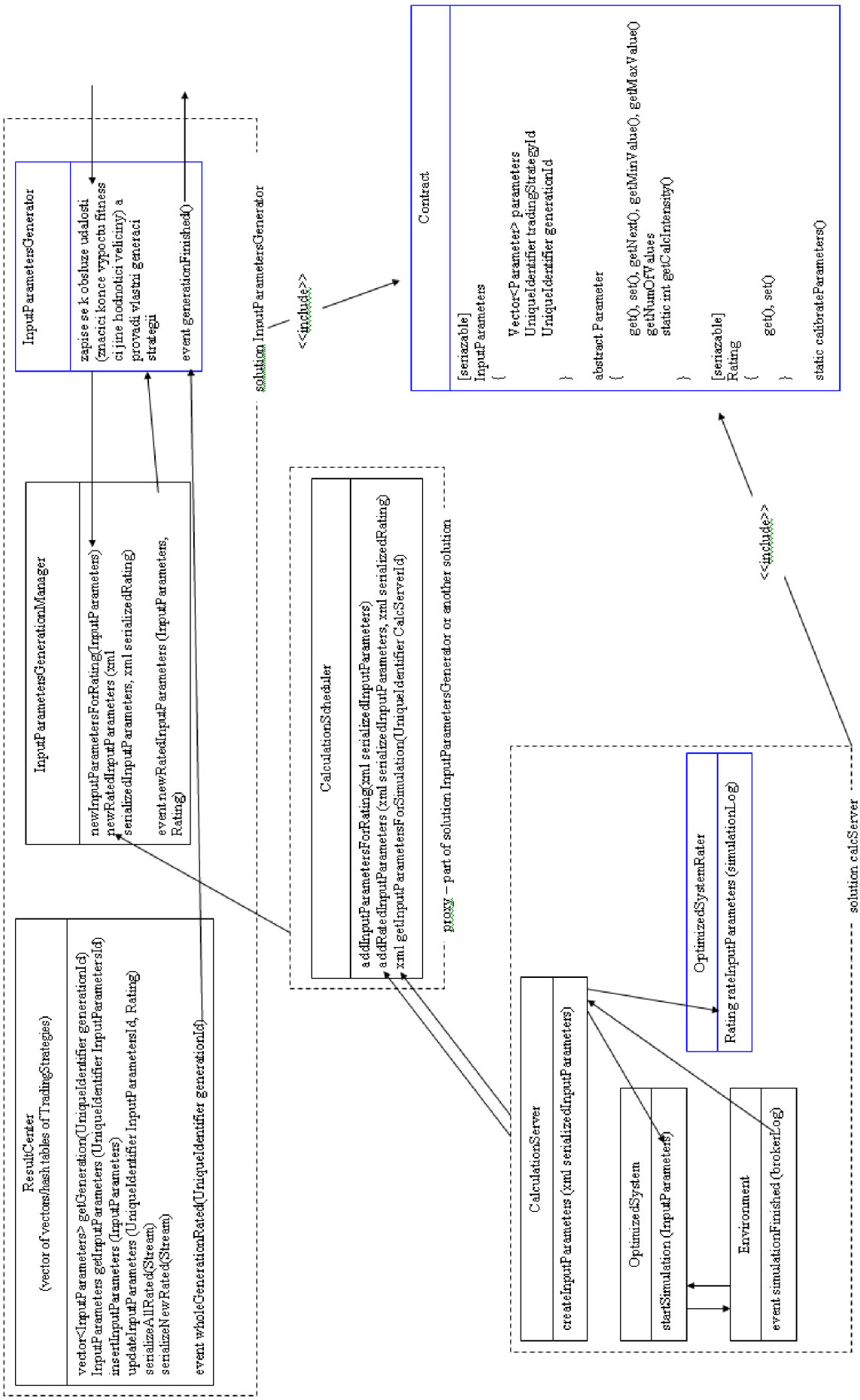
4.1.3 Komponenta rozdělující a spravující atomické výpočty

Rozhraní této komponenty je velice jednoduché – pouze přebírá kandidátní řešení od generující komponenty a předává je dále k ohodnocení a poté zpět předává výsledky hodnocení. Přítomnost této komponenty je ovšem nezbytná pro sofistikované řízení paralelního provádění výpočtů více hodnotících komponent, které musejí být na sobě zcela nezávislé. Konceptuální schéma této komponenty znázorňující její rozhraní je na Obr. 4.4.



Obr. 4.4.: Konceptuální schéma komponenty rozdělující a spravující atomické výpočty

Celkový diagram všech uvedených komponent a jejich podkomponent i s naznačením možného implementačního rozhraní jednotlivých částí je naznačeno v diagramu Obr. 4.5. Pro modře označené komponenty je vhodné umožnit jejich snadnou nahraditelnost bez nutnosti znovu sestavovat kód (zbylá část kódu tak bude zcela nezávislá na konkrétní instanci problému a způsobu jeho řešení).



Obr. 4.5.: Konceptuální schéma komponent a komunikace systému optimalizace predikčního modelu s naznačeným návrhem implementace.

4.2 Třídní návrh systému

V předchozí kapitole jsme provedli návrh základních komponent modelovaného systému. Toto je první krok k rozpoznání jednotlivých tříd. Dále je třeba konkrétní podobu těchto komponent přizpůsobit požadované funkcionalitě systému (komponenty jsme navrhovali spíše pro optimalizaci vstupních parametrů obecného modulu, v návrhu tříd je již třeba se zaměřit na konkrétní instanci problému a to optimalizaci vstupních parametrů obchodního modulu, schopného automatizovaně obchodovat s virtuální zprostředkovatelem finančního trhu – tzv. brokerem). A v neposlední řadě je potřeba do třídního návrhu zahrnout komunikační rozhraní jednotlivých komponent – třídy realizující předávání dat a třídy představující datový kontrakt komunikace. V této kapitole se tedy budeme zabývat podrobným návrhem jednotlivých balíčků a tříd. Výsledný diagram návrhových tříd s naznačeném rozdělení do balíčků sestavení je znázorněn v části Příloha 1.

4.2.1 Balíček s datovým kontraktem

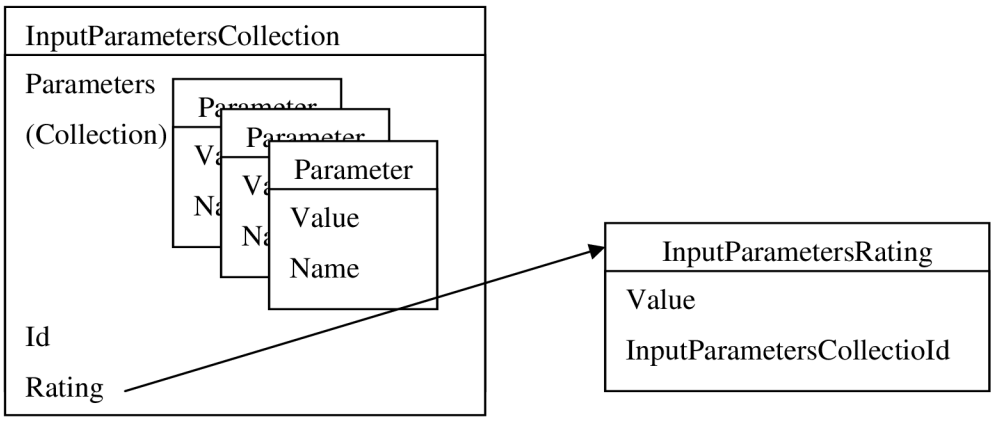
Z předchozích podkapitol vyplývá, že celý systém optimalizace predikcí časových finančních řad bude sestaven z více spolupracujících aplikací, umístěných na obecně různých výpočetních strojích. Obvyklý správně dokomponovaný objektový návrh obsahující datový a komunikační kontrakt definující provázání objektů je v tomto případě o stupeň důležitější – budou totiž použity nejen pro implementaci komunikace objektů v rámci jedné aplikační domény, ale i mezi aplikačními doménami na různých strojích.

Základem datového kontraktu bude vektor parametrů modelu časové řady tak, jak jsme si jej definovali v kapitole 2.2.2. Množinu možných parametrů bude možné rozšířit pro účely konkrétního řešeného problému (tedy pro účely modelování finančních časových řad). Pro jednotlivé hodnoty vektoru vstupních parametrů tedy definujeme vlastní typ.

Dalším významným datovým prvkem v systému, jehož instance bude nutné předávat, je ohodnocení úspěšnosti vygenerovaného vektoru parametrů – tedy datový typ hodnot fitness funkce. Pro snadnou možnost v budoucnu měnit obor hodnot této proměnné (jež je pravděpodobně celé $\Re$, výsledná implementace může ovšem obsáhnout pouze diskrétní spočitatelnou podmnožinu $\Re$ dle možností rozsahu a přesnosti použitého typu).

Ohodnocení a vektor parametrů jsou vzájemně svázány (jedno ohodnocení patří k jednomu konkrétnímu vektoru a naopak). Předchozí informace tedy vedou na realizaci objektu vektoru vstupních parametrů v podobě uspořádané n-tice, z níž jedna položka je ohodnocení vektoru parametrů a druhá položka je právě posloupnost (kolekce, vektor) objektů definujících jeden parametr. Pro účely identifikace objektů napříč aplikačními doménami a možnost persistentního uložen objektů i s jejich vztahy je v této n-tici ještě položka jednoznačného identifikace (OID – object

identification). Pro bližší vysvětlení pojmů uspořádaná n-tice, posloupnost, OID v oboru modelování datových vrstev systémů, viz [14].



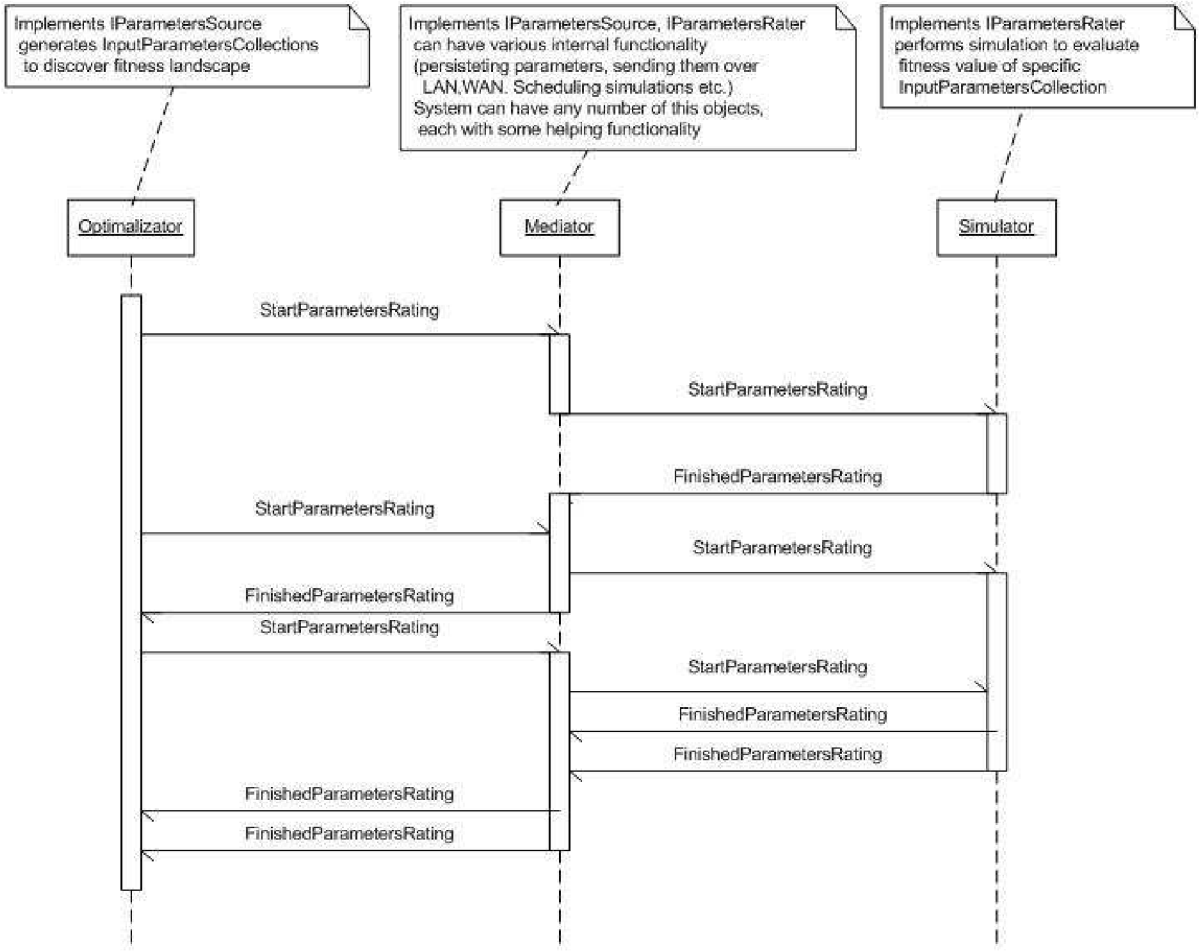
Obr. 4.6: Konceptuální schéma datového kontraktu navrhovaného systému

4.2.2 Balíček s komunikačním kontraktem

Jak bylo zmíněno výše, komunikační kontrakt bude (spolu s datovým kontraktem) základem návrhu celého systému. U návrhu meziobjektové komunikace je dobré dodržovat zásadu minimálního možného provázání objektů. Pro účely jednoduchého a univerzálního komunikačního rozhraní se můžeme na komunikaci v navrhovaném systému dívat tak, že zde vystupují objekty předávající vektory parametrů k ohodnocení a očekávají, že bude asynchronně vráceno ohodnocení, a prvky, které přijímají vektory a asynchronně zasílají zpět jejich ohodnocení (bez ohledu na to jestli sami toto ohodnocené vyčíslují nebo jen delegují výpočet na další prvky).

Zjednodušený sekvenční diagram komunikace těchto prvků je znázorněn na diagramu Obr. 4.7. Z principu asynchronní povahy zasílaných zpráv vyplývá nutnost explicitní vazby mezi ohodnocovanými vektory parametrů a jejich ohodnoceními (Na konceptuálním schématu Obr. 4.7 je tato vazba vidět jednak jak referenční vazba a jednak jako explicitní vazba pomocí jednoznačného objektového identifikátoru – tato bude zejména použitelná pro asociaci objektů napříč aplikačními doménami nebo tam, kde nemá prostá referenční programová vazba dosah).

Dále je z diagramu Obr. 4.7 zřejmá možnost rozšiřování funkcionality celého systému zapojením „přeposílacích“ prvků, bez nutnosti rozšiřovat rozhraní nebo měnit vnitřní funkcionalitu již existujících prvků (tedy splnění úvodního požadavku modularity). Prvek je možné po implementaci obou rozhraní zapojit do existujícího řetězu tak, aby se všemi zasílanými daty mohl provést potřebnou činnost (jejich uložení, naplánování jejich ohodnocení, odeslání na jiný fyzický stroj atd.) a aniž by se této skutečnosti museli přizpůsobovat koncové prvky.



Obr. 4.7: Diagram sekvence znázorňující zjednodušenou komunikaci navržených rozhranní systému

4.2.3 Balíček s definicí prohledávaného stavového prostoru

Třída generující jednotlivé kandidátní řešení (viz kapitolu 4.1.1 Komponenta generující vektory vstupních parametrů) bude vlastně provádět prohledávání stavového prostoru. K tomu je ovšem nutné tento stavový prostor ohraničit, případně definovat jeho vlastnosti (granularita jednotlivých dimenzí, požadovaný výchozí bod prohledávání a podobě) a definovat omezující podmínky.

K tomuto účelu bude sloužit balíček s definovanou šablonou vektoru parametrů. Tato šablona bude udávat dimenzi generovaných vektorů a pro jednotlivé prvky generovaných vektorů také požadovaný rozsah a přesnost hodnot, úroveň měřitelnosti proměnné (nominální, ordinální, intervalový, poměrný) atd.

Pro snazší a rychlejší změny definice prohledávaného prostoru bude vhodné, aby vnitřní implementace tohoto balíčku načítala tato nastavené z externího xml souboru (který bude zároveň validován oproti xsd schématu).

4.2.4 Balíček pro správu vektorů

Optimalizační evoluční algoritmy (viz kapitolu 3 Evoluční algoritmy) potřebují pro svoji činnost znát dříve vygenerovaná kandidátní řešení a jejich ohodnocení. Dále kandidátní řešení a jejich ohodnocení jsou hlavními vyprodukovanými artefakty celého systému, které by se proto hodilo persistentně ukládat pro možnosti pozdější analýzy. Jelikož tyto funkcionality budou společné pro jakýkoliv implementovaný optimalizační výpočet, bude vhodné je vyčlenit do zvláštního modulu, který právě bude moci být do systému začleněn jako výše zmínění „přeposílací objekt".

Hlavní funkcionalitou, kterou bude tedy tento prvek vystavovat bude práce s kandidátními řešeními (vektory parametrů) – jejich poskytování dle identifikátoru, případně členění do generací a poskytování celých generací a oznamování ohodnocení celé generace. Pro třídní diagram tohoto balíčku odkazujeme čtenáře na diagram Příloha 1.

4.2.5 Balíček pro optimalizační výpočty

Kromě případných podpůrných tříd bude tento balíček obsahovat jedinou třídu implementující rozhraní zdroje vektoru parametrů (viz 4.2.2 Balíček s komunikačním kontraktem). Vnitřní implementace bude na základě dostupných hodnocení již vygenerovaných vektorů parametrů (dostupných v úložišti vektorů parametrů v balíčku pro správu vektorů) generovat vektory nové s cílem co nejefektivněji prohledat stavový prostor a najít optimální popřípadě dostatečně vhodné suboptimální řešení.

4.2.6 Balíčky s komunikační funkcionalitou a správou běhu systému

Evoluční algoritmy obvykle k nalezení vhodného řešení potřebují dostatečně velký počet kroků stavovým prostorem. I když množina všech ohodnocených bodů prostoru, které algoritmus prohledal je malou podmnožinou celého stavového prostoru (jež by bylo bez optimalizačních či heuristických technik nutno prohledat celý), stále je běžné, že dochází k vyčíslování fitness funkce kandidátních jedinců v řádu stovek až tisíců bodů prostoru. Pro uspokojivou rychlost je tedy nutné použít rychlé ohodnocení kandidátního vektoru parametrů. V našem případě ovšem bude vyčíslování fitness probíhat formou simulace obchodování a to na datových řadách v řádu miliónu bodů (pro zaručení co nejpřesnějšího ohodnocení prediktivity optimalizovaného modulu).

Možností, jak urychlit celou optimalizaci, je zvýšení výpočetního výkonu, čehož v dnešní době nejlevněji dosáhneme využitím vícejádrových procesorů a navýšením počtu fyzických výpočetních strojů. Úkolem popisovaných balíčků bude umožnění běhu systému v libovolném stupni paralelizace a to jak na úrovni vláken v jedné aplikační doméně (pro využití vícejádrových procesorů), tak na úrovní různých aplikačních domén (běžících na oddělených fyzických strojích schopných

komunikovat pomocí protokolu TCP/IP) s možností libovolně měnit stupeň paralelizace za běhu (výpadky výpočetních uzlů neovlivní běh celého systému) a bez nutnosti na tuto funkcionalitu jakkoliv připravovat ostatní prvky systému.

Dalším úkolem tříd tohoto balíčku je správa výpočetních serverů (strojů vyčíslujících fitness funkci) – automatická distribuce aktuálních balíčků na každý nově přihlášený výpočetní server (případně při startu běhu systému) a správa a monitorování probíhajících výpočtů na výpočetních serverech (s možností využití jednoho výpočetního serveru dvěma různými instancemi optimalizačního systému).

První požadavek splníme pomocí návrhového vzoru *Proxy* (viz literaturu [15]). Kdy v každém komunikujícím prvku navážeme třídy systému na *Proxy* třídu implementující rozhraní třídy nacházející se na vzdáleném stroji. Vnitřní funkcionalitou této *Proxy* třídy pak bude převést volání na naplánování výpočtu nejvhodnějším strojem (který má například nejrelevantnější mezivýpočty v pomocné mezipaměti) a provedení volání na příslušném procesu na vzdáleném stroji. *Proxy* musí být schopné monitorovat dostupnost jednotlivých komunikačních bodů a operativně jejich seznam upravovat.

Druhý požadavek je možné implementovat jako vnitřní funkčnost výše zmíněných proxy tříd – v případě, že se vzdálený výpočetní server přihlásí jako schopný provádět výpočty, lokální proxy mu nejprve zašle aktuální verze balíčků pro provádění výpočtů, které si vzdálený prvek načte do paměti a bude volat jejich funkčnost při provádění výpočtů. Řešení tohoto požadavku tedy neovlivní rozhraní komunikujících tříd – z pohledu návrhu není důležité jej před samotnou implementací do detailu analyzovat (pokud se technikou prototypování potvrdí jeho implementovatelnost).

4.2.7 Balíček se správou simulací pro ohodnocení vektorů

V předchozí kapitole byl naznačen problém zvyšování výpočetního výkonu systému optimalizace predikcí. Jedna z navržených variant – a to paralelizace výpočtů v rámci jedné aplikační domény – nebyla postihnuta v balíčcích s komunikační funkcionalitou. Další velmi znatelný nárůst výpočetního výkonu totiž přinese sdílení simulačních dat (popřípadě pomocných mezivýpočtů) mezi různými instancemi výpočtů.

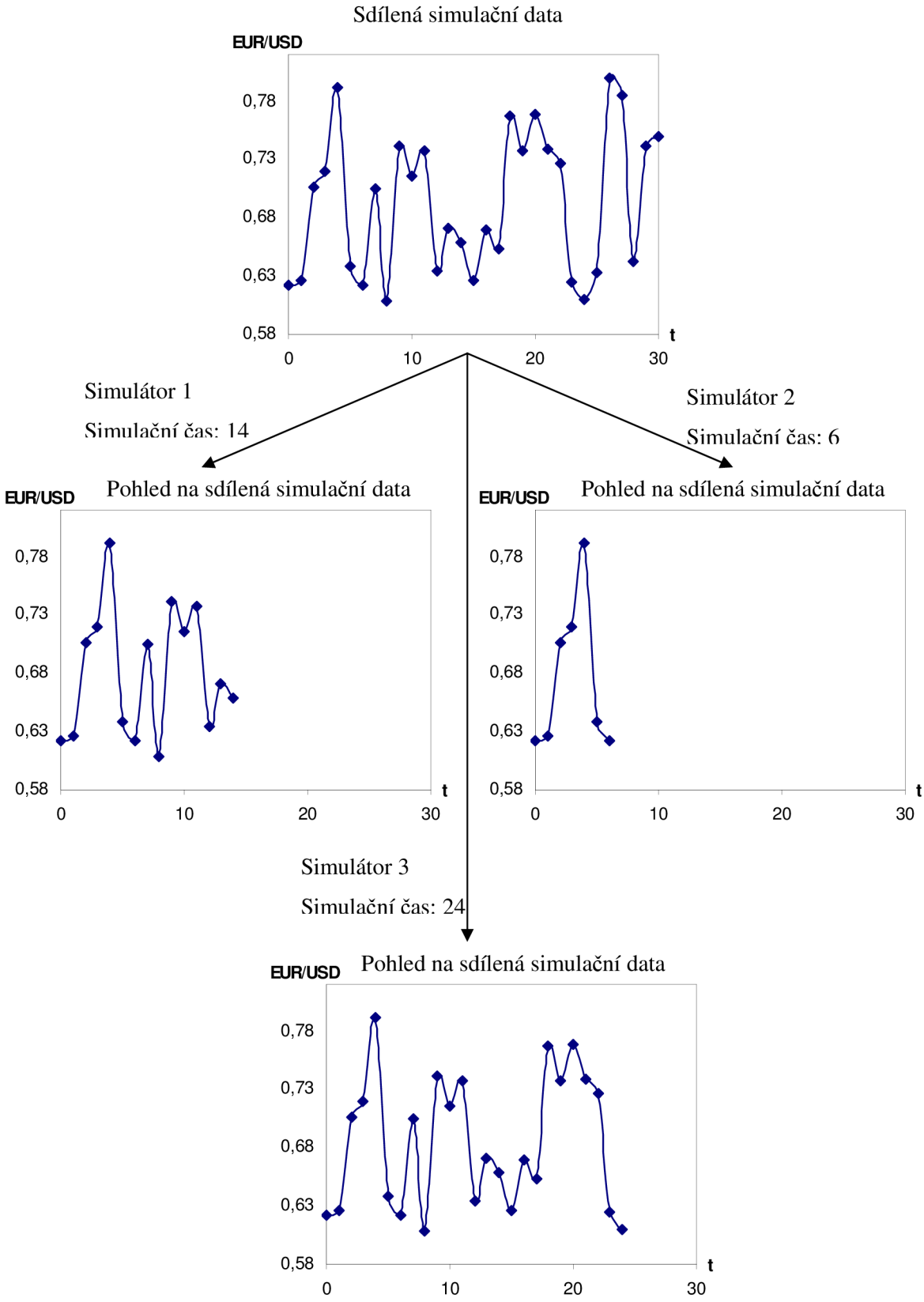
K tomuto bude ovšem nutné spravovat simulační data tak, aby různé instance výpočtů přistupovaly ke sdíleným datům různým způsobem (budou se nacházet v rozdílném simulačním čase a tedy by jim měla být přístupna různá část dat). Pro tento účel je třeba implementovat zjednodušený diskrétní simulátor splňující následující požadavky:

- V každém kroku simulace se střídají moduly obchodníka a brokera, model obchodníka zadává příkazy, čímž plánuje události, jejichž naplnění není ovšem vázáno pouze na simulační čas ale také na hodnoty dat příslušejících konkrétnímu časovému okamžiku.

- Simulátor musí být schopen zprostředkovávat data příslušející k danému času simulace a data historická, nikoliv však data příslušející nadcházejícím časům simulace.
- Fyzicky musí existovat jedna instance dat.

Více o implementaci simulátoru diskrétního času, především pomocí techniky „*activity scanning*“, kterou je třeba ve zjednodušené podobě použít pro implementaci našeho simulátoru viz literaturu [16].

Schematické znázornění existence více instancí simulací s různým časem simulací přistupující k jediné fyzické instanci dat je na Obr. 4.8. Této funkcionality je možné dosáhnout existencí jednoho objektu představujícího sdílená data a více objektů s různým vnitřním stavem, které zpřístupňují a procházejí data (tedy například v jazyce C++ pomocí techniky *kontejnerů* a *iterátorů*, v jazycích Java či C# pak pomocí techniky *kolekcí* a *enumerátorů*).



Obr. 4.8: Schéma poskytování unikátních pohledů paralelních simulací na sdílená data

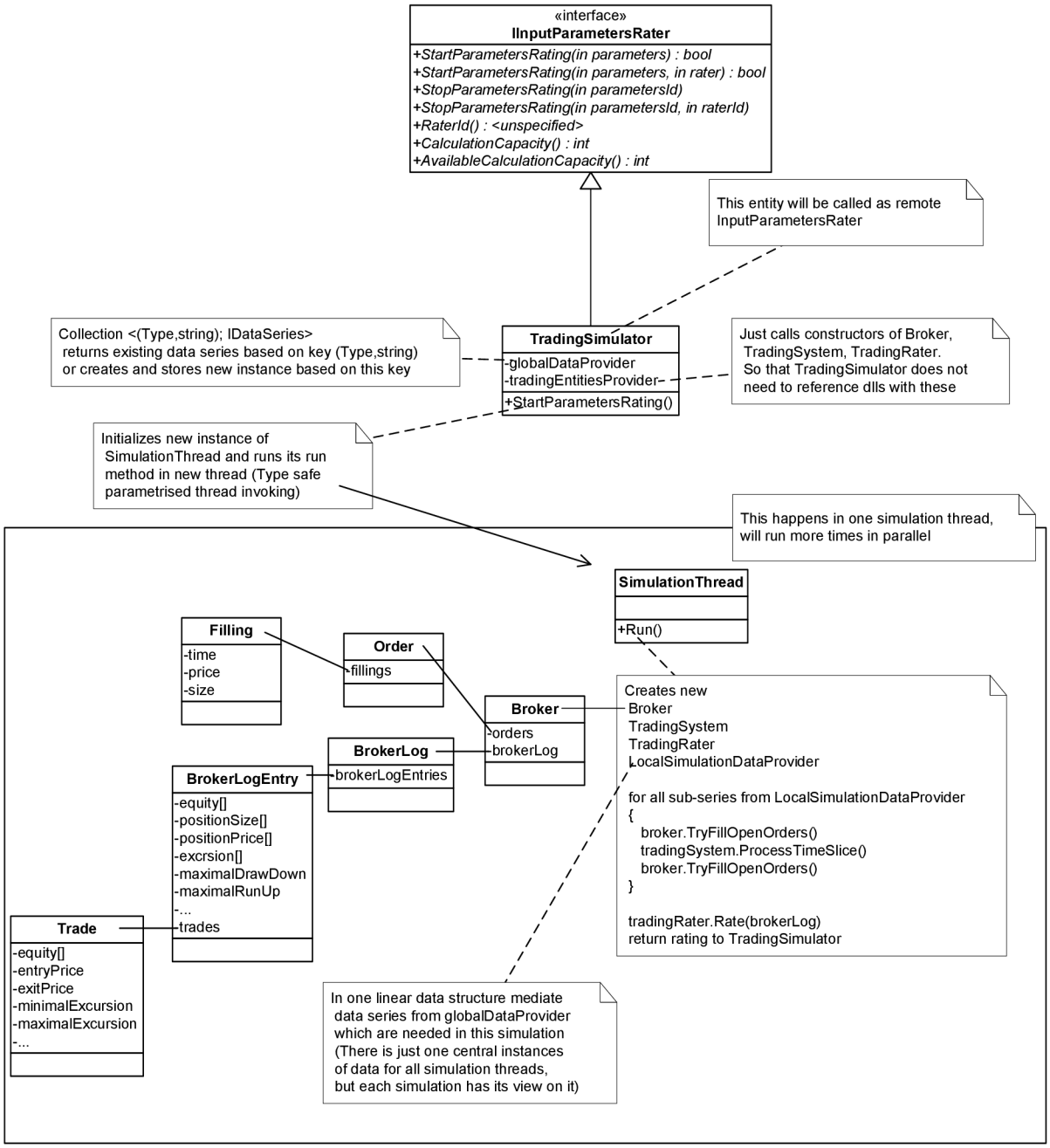
4.2.8 Balíček s definicí obchodního prostředí

Až do této chvíle je popisovaný optimalizační systém univerzální z hlediska úlohy, jež optimalizujeme. Optimalizační modul pouze prohledává stavový prostor, správce vektorů zaznamenává nalezená kandidátní řešení, komunikační modul spravuje výpočetní výkon paralelizovaného systému a správce simulací spouští jednotlivé simulace pro vyhodnocení fitness funkce („simulace života vygenerovaného jedince“) a poskytuje jim potřebná data.

Jednotlivé instance úloh se pak liší právě v samotných simulacích – v našem případě je to simulace obchodního prostředí, kde se optimalizovaný obchodní modul snaží s každým krokem simulace předikovat vývoj poskytnutých dat (ve formě časových datových řad). Nahrazením tohoto balíčku tedy můžeme provádět optimalizace naprosto odlišných problémů – toto ovšem přesahuje rozsah této práce a my se nadále budeme zabývat pouze simulací obchodního prostředí. Entity vystupující v tomto prostředí budou tyto:

- **Obchodník** – optimalizovaný modul, jež se pomocí predikcí finančních časových řad snaží dosahovat optimálního vývoje své *equity* – průběžného zisku.
- **Broker** – ve skutečném světě obchodů je to zprostředkovatel obchodních transakcí mezi obchodníky, zde je to koncový prvek, naplňující zadané příkazy obchodníka podle aktuálních dat pro daný simulační čas.
- **Objednávky** – tedy příkazy zadávané obchodníkem brokerovy (jaký obnos jakých aktiv a za jakých podmínek chce obchodovat)
- **Pozorovatel obchodování** – pasivní prvek zaznamenávající všechny události a jejich metriky spojené s obchodováním obchodníka a brokera, jež jsou relevantní pro vyhodnocení úspěšnosti tohoto obchodování – a tedy fitness kandidátního řešení.
- **Hodnotitel obchodování** – prvek aktivovaný po ukončení běhu simulace obchodování, který na základě zaznamenaných informací pozorovatelem vyhodnotí úspěšnost obchodování (například konečná hodnota equity – tedy celkový zisk, nebo počet změn equity – tedy nestabilita průběžného zisku a podobně).

Náznak rozčlenění těchto entit a jejich spolupráce je zachycena na schématu Obr. 4.9. Konkrétní podoba entit, nebo jejich rozhraní, je znázorněna v materiálu Příloha 1.



Obr. 4.9: Poloformální schéma části entit a jejich spolupráce při simulaci obchodních prostředí

4.2.9 Balíčky s implementací obchodního prostředí

Balíček pro definici obchodního prostředí popsaný v kapitole 4.2.8 udává pro třídy *obchodníka*, *brokera* a *hodnotitele obchodování* pouze jejich rozhraní. Je tak možné implementovat celou simulaci obchodování s využitím pouze těchto rozhraní a jejich konkrétní implementaci dodávat v separátních balíčcích (a tím měnit celou úlohu optimalizace a jednoduše zkoušet více variant obchodních prostředí různým způsobem aproximujících reálné obchodní prostředí).

A právě tyto balíčky bude nutné rozmisťovat na výpočetních strojích. Urychlení procesu a vyhnutí se chybě lidského faktoru dosáhneme zautomatizováním tohoto procesu popsaným v kapitole 4.2.6 Balíčky s komunikační funkcionalitou a správou běhu systému.

4.2.10 Balíček pro sdílení pomocných výpočetních dat

Při popisu simulátoru jsme zmínili potřebu sdílení dat mezi vlákny spuštěných simulací. Sdílením obchodních dat totiž ušetříme čas potřebný k jejich načtení z persistentního úložiště a předpřipravení pro použití. Pokud budeme sdílet i pomocné mezivýpočty (například klouzavý průměr finanční časové řady popřípadě jiné modely časových řad popsaných v kapitole 2.2.2 Postup modelování časových řad), ušetřeného strojového času bude ještě znatelně více.

V tomto případě se nám ale nabízí otázka sdílení dat i napříč paralelními stroji. Pokud připravení dat nebo pomocných výpočtů je operace časově náročnější než vyhledání již existujících dat v centrálním úložišti a jejich zaslání do potřebného výpočetního stroje, pak má smysl takovéto centrální úložiště implementovat. Důležitým prvním krokem je tedy analyzovat časovou náročnost výpočtů bez centrálního úložiště a až poté úložiště případně implementovat. Při jeho implementaci je pak nejdůležitější navrhnout, jakým způsobem se budou jednotlivé instance přepočítaných dat identifikovat – tedy navrhnout index dat.

4.3 Iterativní vývoj systému

Třídní návrh systému provedený v kapitole 4.2 popisuje vcelku rozsáhlý a komplexní systém. Nebylo by tedy příliš vhodné spoléhat na úplnou správnost tohoto návrhu a na to, že postihuje všechny případné problémy vyskytnuvší se v průběhu implementace. Z tohoto pohledu je jednoznačně nejvhodnější rozdělit vývoj systému do iterací a minimalizovat tak dopad chyb návrhu (více k rizikům a dopadům špatného návrhu a jejich minimalizování viz [17]).

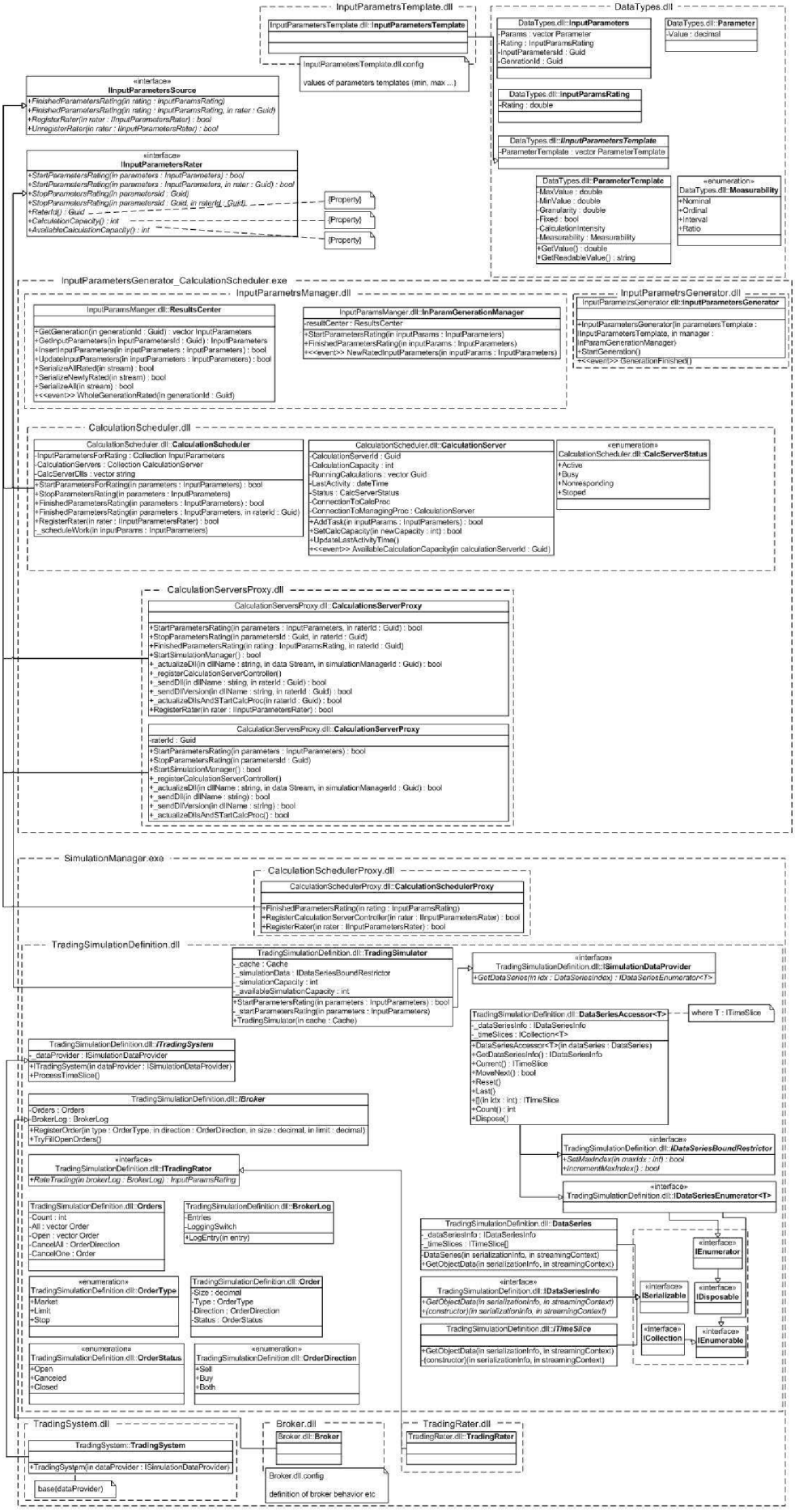
4.3.1 První iterace

Výstupem implementační fáze první iterace vývoje by měl být funkční systém potvrzující základní myšlenku systému (možnost evolučně optimalizovat predikční systém). První iterace tedy bude muset obsahovat následující balíčky:

- **balíčky s definicí datového a komunikačního kontraktu** – jsou základem celé implementace („páteří její kostry“), budou použity ve všech ostatních balíčcích.
- **balíček pro optimalizační výpočty** – pokud má první iterace ověřit smysluplnost myšlenky optimalizovatelnosti predikčních systému, musí obsahovat optimalizační modul.

- **balíček pro správu generovaných vektorů** – pro správnou funkčnost evolučních optimalizačních technik je nutné spravovat historické výsledky algoritmu. Toto je také nutné pro vyhodnocování úspěšnosti algoritmů.
- **balíček s definicí prohledávaného stavového prostoru** – optimalizačnímu systému musíme být schopni zadat definici a hranice prohledávaného stavového prostoru – k tomu je tedy potřeba implementovat tento balíček.
- **balíček s definicí obchodního prostředí** – samotný optimalizační balíček nemůže provádět optimalizace bez modulu vyhodnocujícího fitness funkci kandidátních řešení. Jak jsme zmínili výše, v této práci se budeme zabývat pouze optimalizací obchodních systémů. Pro implementaci výpočtu fitness funkce tedy potřebujeme definici obchodního prostředí a při běhu systému pak i konkrétní implementaci tohoto prostředí, již budou obsahovat.
- **balíčky s implementací obchodního prostředí**

I přes značné zjednodušení první iterace oproti celkovému návrhu je systém implementovaný v rámci této iterace poměrně komplexní – případné problémy nebo chybějící znalosti je velmi vhodné během vývoje odhalit pomocí přístupu „down-top“ za využití rychlého prototypování ([17]). O prototypování v rámci tohoto projektu se zmiňuje kapitola 5.4 Uspořádání zdrojových kódů.



Obr. 4.10: Diagram návrhových tříd první iterace navrhovaného systému

4.3.2 Následující iterace

Již v rámci počátečního návrhu je možné navrhnout postup vývoje v ostatních iteracích vývoje. Samozřejmě jejich konkrétní podoba pak bude upravována podle výstupů předchozích iterací.

4.3.2.1 Druhá iterace

V druhé iteraci bude vylepšován funkční systém z první iterace z hlediska výkonu. Zaměříme se na vylepšení výkonu paralelizací výpočtů. Z celkového návrhu systému, jehož konceptuální model je uveden v materiálu Příloha 1, tak bude v rámci druhé iterace přidána implementace následujících balíčků:

- **balíček s první iterací komunikační funkčnosti** – tedy bude implementována ta komunikační funkčnost, která umožní transparentní rozesílání výpočtů na vzdálené výpočetní stroje.
- **balíček pro správu simulací** – tento balíček bude implementován z důvodu umožnění paralelizace výpočtů v rámci jednoho výpočetního stroje spouštěním separátních výpočetních vláken.

4.3.2.2 Třetí iterace

Ve třetí iteraci bude stále hlavním objektivem vylepšování výpočetního výkonu systému. Tedy kromě optimalizování existujícího a podrobně profilovaného kódu z předchozích iterací se budeme zabývat funkčností centrálního sdílení dat. Přidán tak bude:

- **balíček pro sdílení dat**

4.3.2.3 Iterace nad rámec návrhu

I po dokončení vývoje celého systému podle komplexního návrhu v materiálu Příloha 1 bude možné tento systém dále vylepšovat a rozšiřovat. Další návrhy funkcionality, jež může být přidána, obsahuje následující výčet:

- **automatické vyhodnocování úspěšnosti a efektivnosti optimalizací** – navržení metrik vhodných k porovnání vlastností optimalizačních algoritmů a jejich automatické zaznamenávání.
- **tvorba GUI** – jednak jako rozhraní pro nastavování a ovládání systému, dále pak pro grafické výstupy systému (zobrazování průběhu *equity* nalezených řešení – tj. zobrazování grafů o řádu miliónů datových bodů).
- **změna definice úloh za běhu** – především přidáváním průběžně sbíraných dat a přiblížení se tak možnosti optimalizovat obchodní systém za běhu („*real time*“) a to použitím k tomu vhodných optimalizačních technik - například *ant colony* apod.

4.4 Zpětná vazba z implementační fáze

Jak bylo uvedeno výše, návrh systému a plán pro následující iterace bude nutné neustále upravovat podle aktuálních výstupů implementační fáze. Hlavními výstupy z implementační fáze budou především:

- **potvrzení smysluplnost myšlenky práce** – v průběhu první fáze je nutné potvrdit možnost optimalizování predikčního modelu schopného obchodování, a tak potvrzení smysluplnosti pokračování v implementaci dalších iterací.
- **dostatečnost a minimalista datového a komunikačního kontraktu** – pro snadnou implementaci a udržovatelnost vzniklého kódu je nutné navrhnout dostatečné ale zároveň minimální možné rozhraní (datové a komunikační) komunikujících komponent systému. Až samotná implementace prověří splnění takovéhoto požadavku.
- **výkonnostní analýzy** – ve výše uvedeném plánu pro druhou a třetí iteraci vývoje systému je hlavním objektivem zvyšování výpočetního výkonu – toto je možné smysluplně provést až po důkladné výkonnostní analýze (profilování), jinak bychom mohli velice úspěšně optimalizovat části systému, jež mají minimální dopad na celkový výpočetní výkon.

5 Implementace systému optimalizace predikcí

Analýza teoretického pozadí problému a návrh řešení systému byly provedeny v předchozích kapitolách. V této kapitole se budeme zabývat vlastní implementací systému optimalizace predikcí. Popíšeme implementační prostředí, podpůrné techniky a cílovou platformu. Dále uvedeme nejdůležitější aspekty vlastní implementace. Pro podrobný popis implementační části odkazujeme čtenáře na podrobnou programovou dokumentaci nacházející se na přiloženém CD (pro podrobnější popis tvorby této dokumentace viz kapitolu 5.5 Dokumentování).

5.1 Cílová platforma a použité technologie

Pro implementaci navrženého systému byl zvolen programovací jazyk C# (jeho aktuálně nejnovější verze 3.0). Systém tedy poběží na strojích s instalovaným běhovým systémem .NET framework (verze 3.5 popř. starší verze 2.0 s doinstalovanými knihovnami s funkcionalitou *lambda výrazů*, *LINQ* a *Windows Communication Foundation*). Nutnost přítomnosti *.NET frameworku* ovšem v žádném případě neznamená omezení ve výběru podkladového operačního systému (viz zdroje [18] a [19] pro podrobnosti spouštění *.NET* aplikací na *UNIX*ových a dalších platformách).

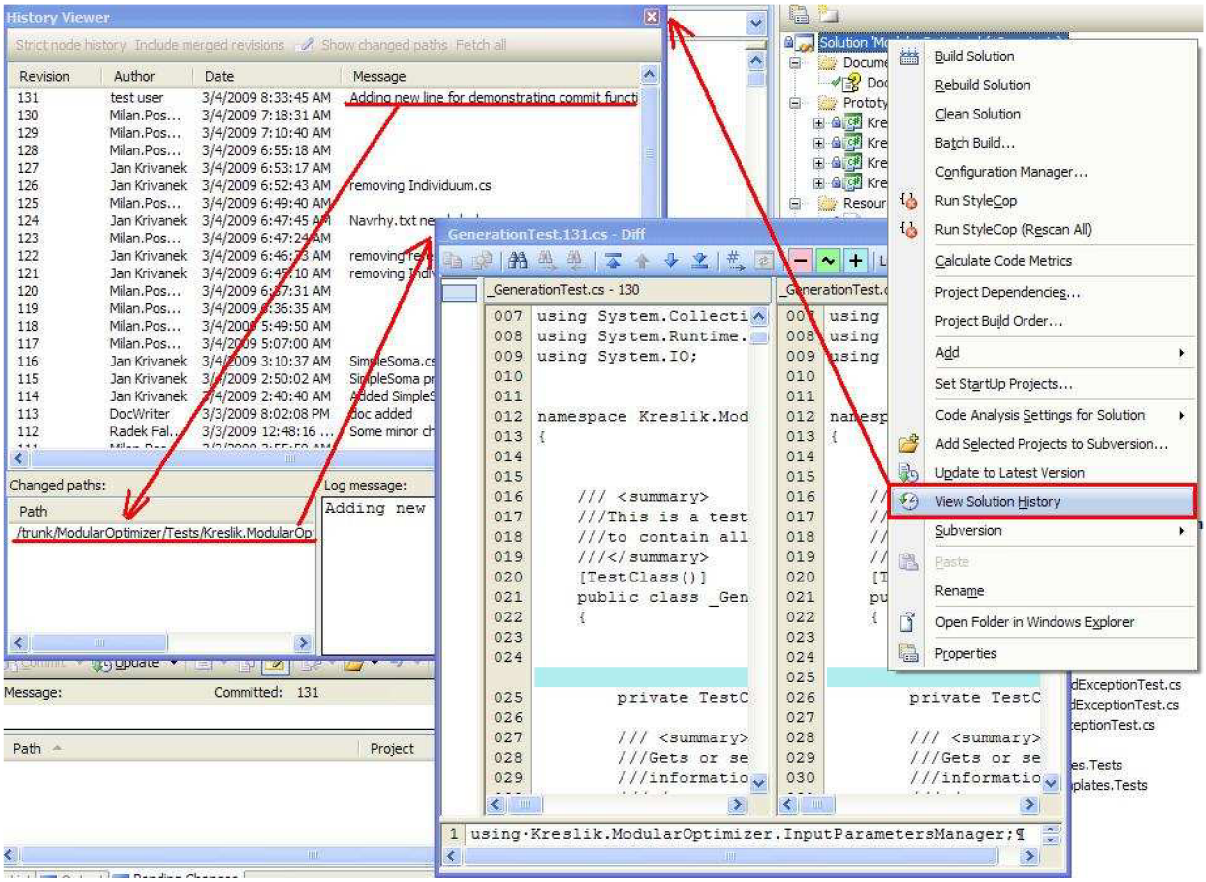
Jako vývojové prostředí bylo použito *Visual Studio 2008* (vydání *Team System* pro možnost provádění analýzy pokrytí kódu pomocí testů jednotek). Pro správu zdrojových kódů bylo zvoleno verzované úložiště zdrojových kódů *VisualSVN* a *AnkhSVN* doplněk do *Visual Studia* pro jednoduché připojení k úložišti přímo z vývojového prostředí. Pro automatické sestavování a spouštění testovacích a analytických úloh byl zvolen sestavovací server *JetBrains TeamCity* (této problematice se podrobněji věnuje kapitola 5.3.1 Autonomní kontinuální integrace).

5.2 Minimalizace technických rizik v projektu

Součástí vývoje projektu (ať už individuálního či o to více týmového) by mělo být centrální verzované úložiště kódů. To nám jednak umožní pracovat na projektu (synchronně) z více míst, dále pak prohlížet historii změn v projektu, případně navracení k těmto změnám – pro ukázku této funkčnosti viz Obr. 5.1.

V našem projektu jsme pro tyto účely zvolili úložiště s možností bezplatné instalace a používání – *VisualSVN*, instalované na serveru dostupném přes *WAN*. Připojení k tomuto serveru je možné jakýmkoliv *svn* klientem – pro účely snadného a rychlého ovládání jsme zvolili volně

dostupný nástroj integrovatelný přímo do vývojového prostředí – *AnkhSVN*. Práce s úložištěm tak byla umožněna pomocí kontextových menu bez nutnosti přerušit práci s kódem.



Obr. 5.1.: Ukázka prohlížení historie změn v projektu a jejich porovnání s aktuálním stavem.

Centrální úložiště samo o sobě ovšem neřeší snadné zotavení se ze ztráty zdrojových kódů. Projekt sice existuje ve více kopiích (kromě centrální verze má každý svoji lokální pracovní kopii), obnovení kódů z těchto kopií by ovšem muselo proběhnout ručně. Funkce source control jsme proto ještě doplnili každonočním zálohováním obsahu úložiště na záložní disk na jiném fyzickém serveru.

5.3 Měření kvality projektu

Samotná konstrukce softwarového díla (tedy fáze mezi návrhem a testováním a předáváním díla) tvoří dle zdroje [17] 30 – 80 % úsilí a času práce na projektu. Je v ní tedy vysoká pravděpodobnost zanesení chyb a zmaření cílů naplánovaných ve fázi designu. Kvalitu kódu a celého projektu je tedy třeba kontrolovat již v průběhu této fáze a ne až po jejím skončení.

Pro bližší přehled toho, co rozumíme pod pojmem kvality projektu a kódu a jakými technikami jí můžeme dosahovat a měřit, viz zdroj [17], pro naše účely budeme kvalitu kódu především posuzovat pomocí následujících technik:

- ***Testování jednotek*** – technika častěji označovaná jako „Unit Testing“, technika při níž spolu s každou jednotkou kódu (modulem, třídou, metodou) napíšeme i test této jednotky prověřující její správnou funkčnost.
- ***Statická analýza kódu*** – analýza kódu schopná odhalit mnohé nešvary kódu, jež mohou vést k jeho špatné udržovatelnosti, k zanášení chyb a zesložiťování oprav (například nevhodné strukturování kódu, jmenné konvence apod.)
- ***Kontinuální integrace*** – technika označovaná jako „Continuous Integration“, anebo častěji při jejím zautomatizování jako „ACI“ – „Autonomous Continuous Integration“. Při každé změně kódu se přeloží a sestaví celý projekt – tím se minimalizuje zanášení chyb projevujících se až po integraci projektu. Tuto techniku je možné spojit s výše uvedenými technikami pokud jsme schopni je plně zautomatizovat.
- *Revize kódu* – pokud máme možnost práce v týmu je vhodné kromě automatizovaných nástrojů využít vzájemnou kontrolu a připomínkování kódu mezi členy týmu.

První tři uvedené techniky byly implementovány jakou automatizovaná součást měření kvality tohoto projektu, rozebereme si je tedy podrobněji. Pro jejich efektivní a nenáročné použití byly naimplementovány jako součást *autonomní kontinuální integrace*.

5.3.1 Autonomní kontinuální integrace

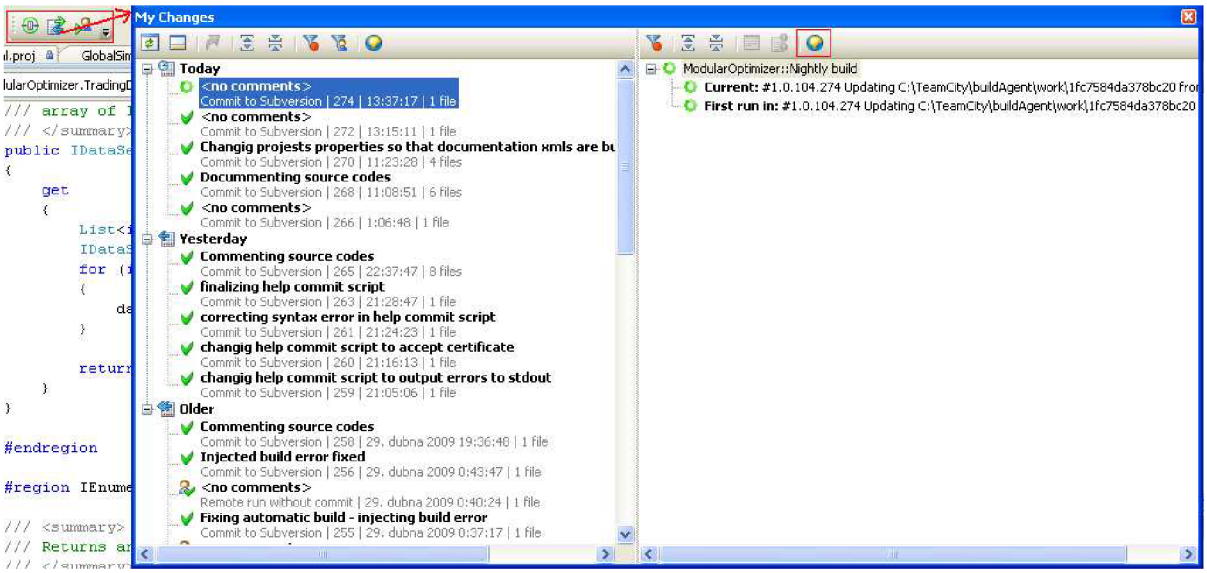
Tato technika umožňuje jednak neustálou kontrolu interakce všech modulů projektu, tak způsob, jak mít neustále po ruce jednotlivé verze výstupních sestavení projektu. Při této technice je veškerý překlad kódu, jeho sestavování a případně další dodatečné procedury automatizovány a obvykle probíhají na centrálním serveru vždy po přidání další revize kódů do verzovaného úložiště.

K implementaci takovéto technologie se v současné době používá speciální software označovaný jako „*build server*“ (sestavovací server). Z mnoha dostupných nástrojů této kategorie, schopných pracovat s kódy platformy .NET, jsme zvolily nástroj *TeamCity* od firmy *JetBrains* (alternativami mohly být například *CrusieControl.NET*, *Draco.NET*, *TeamFoundation server* a další). Zvolen byl především z důvodů licenčních (volně dostupný) a poté díky unikátní funkčním přednostem (vzdálené práce pomocí bohatého webového rozhranní, nebo integrovatelného rozšíření do vývojového prostředí, možnost spouštění osobních sestavení bez nutnosti vkládat kódy do úložiště kódů a dalších).

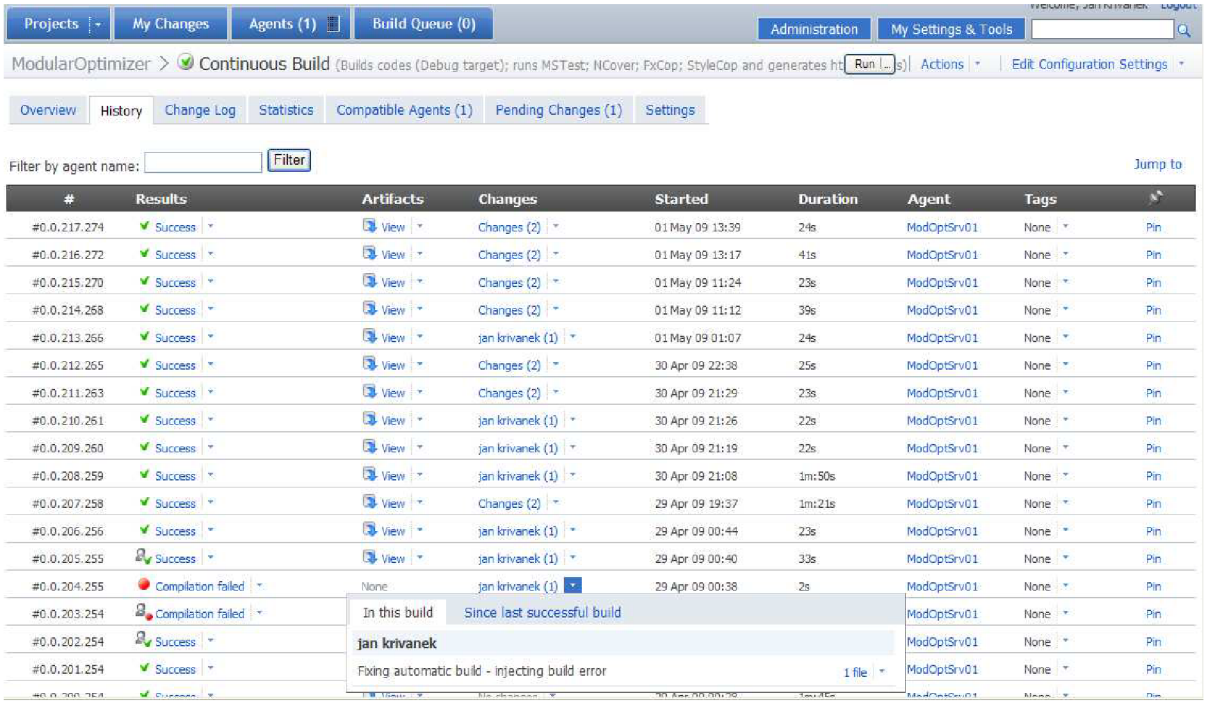
Podrobnější popis instalace a konfigurace tohoto nástroje přesahuje rozsah této práce a je součástí elektronických příloh. Zde pouze zmíníme, že nástroj byl nakonfigurován tak, aby prováděl automatické překlady a sestavení kódu a následné spouštění testů a analýz a generování reportů z těchto akcí po vložení kódů do *VisualSVN* úložiště. Dále bylo nastaveno provádění automatických nočních sestavení výstupního kódu spolu s vygenerováním programové dokumentace a jejího

zpětného uložení do úložiště kódů (takže další den je aktuální dokumentace dostupná přímo z vývojového prostředí po aktualizování revize projektu).

Vývojář tak může přímo z vývojového prostředí prohlížet průběh automatických sestavení, tak jak je tomu znázorněno na obrázku Obr. 5.2., popřípadě může výsledky sestavení a jejich podrobnosti a reporty sledovat z webového prohlížeče jak je tomu uvedeno na obrázku Obr. 5.3.



Obr. 5.2.: Prohlížení automatických sestavení z vývojového prostředí

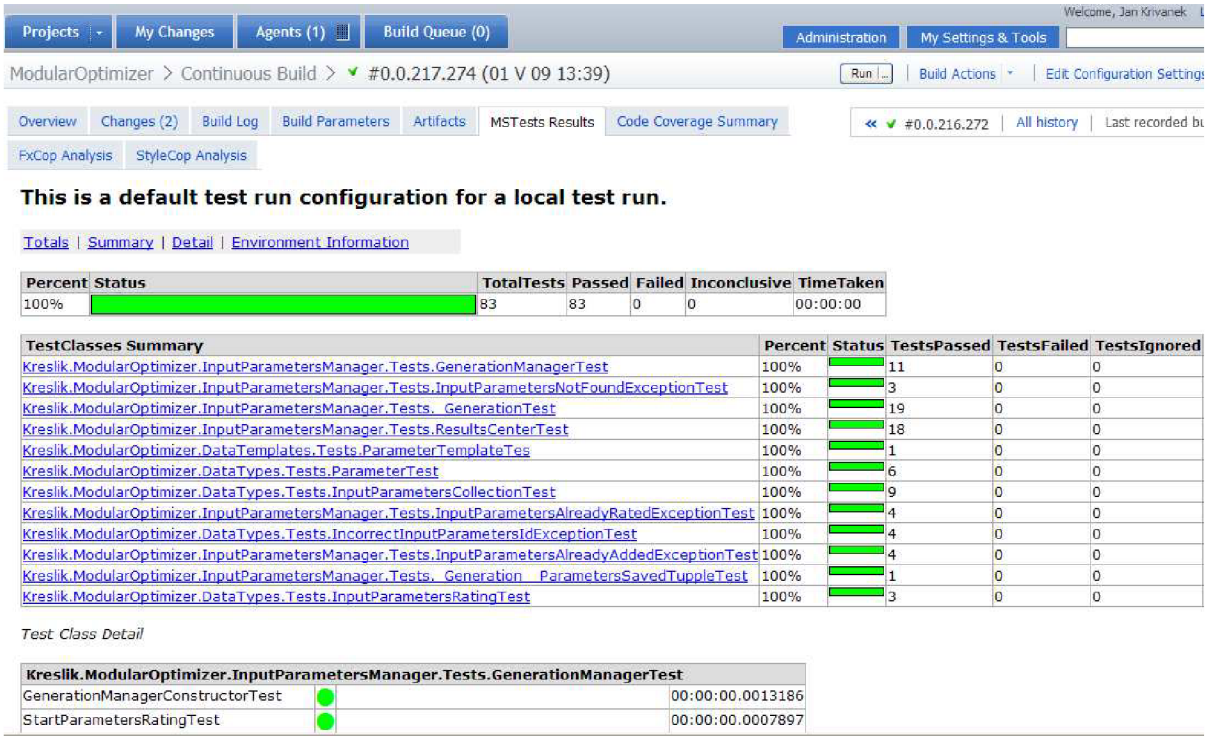


Obr. 5.3.: Prohlížení přehledu sestavení pomocí webového prohlížeče

5.3.2 Testování jednotek

Jak bylo uvedeno výše – tato technika znamená psaní testů spolu s každou jednotkou kódu (třídou, metodou apod.), tak aby mohla být ověřena základní funkčnost jednotky. Testy je poté vhodné spouštět při každé změně kódu, aby se minimalizovalo riziko zanesení chyb při úpravách existujícího kódu. Testy jednotek jsou tedy ideální součástí výše popsané autonomní kontinuální integrace.

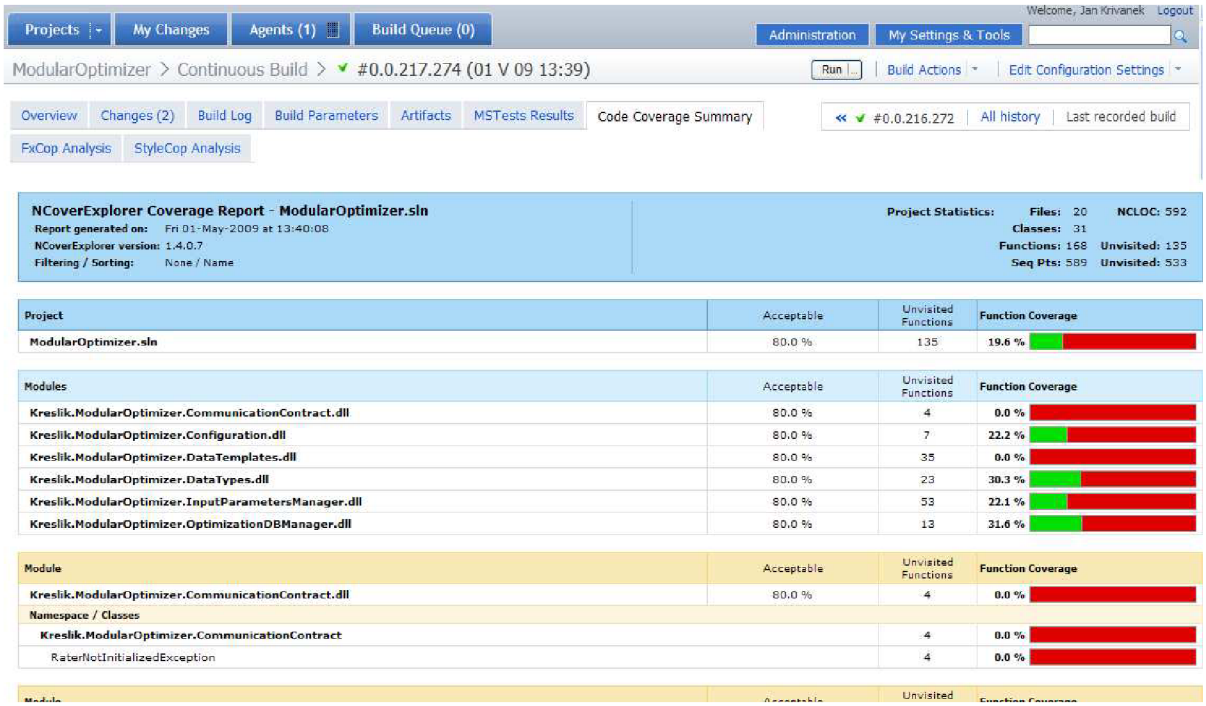
Framework pro spouštění testů byl zvolen *MStest*, jež je součástí vývojářských nástrojů instalovaných s vývojovým prostředím. Tento framework exportuje výsledky testů ve formě *.trx* souborů, pro jejich transformaci v přehledný *html* report byl využit volně dostupný nástroj *trx2html* (je součástí balíčku použitých nástrojů na přiloženém CD). Příklad takto transformovaného výstupu zpřístupněného přes webové rozhraní integračního serveru je na snímku Obr. 5.4.



Obr. 5.4.: Přehledné výstupy testů jednotek proběhlých na integračním serveru.

Při psaní testů jednotek si vývojář sám určuje, kdy je test úspěšný a kdy selhal. Toto může vést ke psaní testů, které prochází ale ve skutečnosti jen málo testují to co mají. Pro určení kvality kódu je mimo jiné směrodatné jaká část kódu byla ve skutečnosti otestována – tato metrika se často označuje jako takzvaná „*Code coverage*“, tedy *pokrytí kódu*.

Do našeho integračního serveru jsme tedy přidali použití dalšího volně dostupného nástroje – *NCover*. Tento nástroj umožňuje měřit pokrytí kódu testy jednotek. Pomocí dalšího volně dostupného nástroje - *NCoverExplorer* – je možné výstupy analýzy pokrytí kódu transformovat do přehledného *html* reportu. Získáme pak výstup obdobný výstupu Obr. 5.5., ze kterého jsme schopni posoudit kvalitu testování a určit, které části kódu je vhodné protestovat důkladněji.



Obr. 5.5.: Výstup analýzy pokrytí kódu testy jednotek

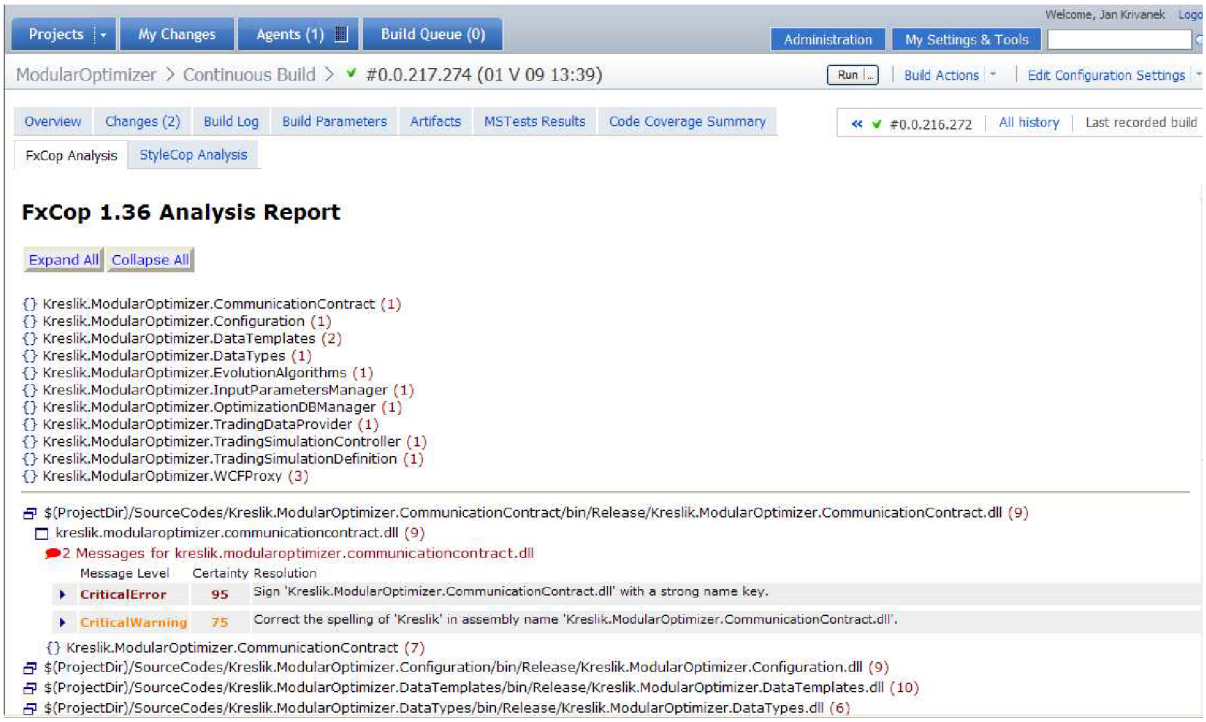
5.3.3 Statická analýza kódu

Jedna z vlastností kvalitního kódu je jeho dobrá udržovatelnost (viz [17]). Udržovatelnost kódu je jedna z jeho statických vlastností (dána strukturováním kódu, dodržováním jmenných konvencí, stupněm provázanosti komponent a podobně). řada z těchto vlastností je možné sledovat automaticky – v prostředí .NET k tomuto účelu vznikl volně dostupný nástroj *FxCop* (později odkoupený firmou Microsoft a integrovaný jako tzv. *Code Analyzis* do nové verze *Visual Studia* v edicích *Team System*).

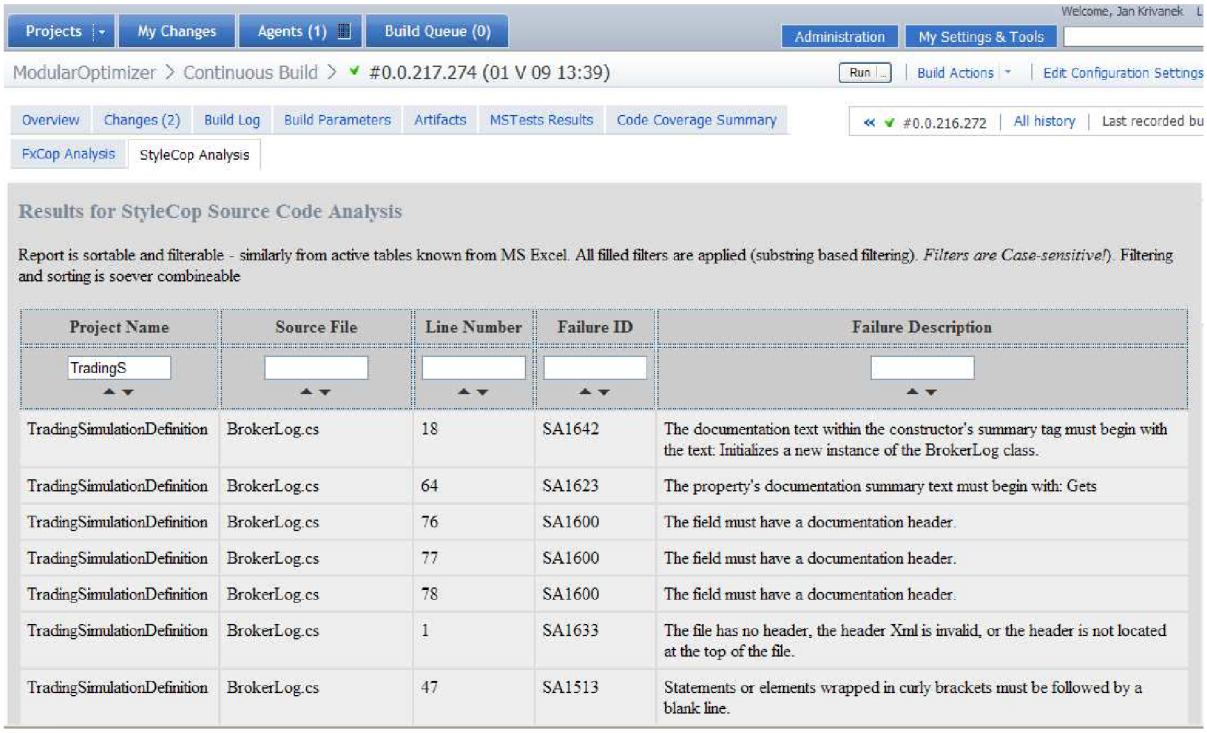
Tento nástroj byl tedy přidán do integračního serveru a příklad jeho výstupu můžeme vidět na schématu Obr. 5.6. Hlášení obsahují umístění, možné příčiny a možná řešení nalezených chyb nebo nesrovnalostí. Automatická analýza nemůže být dokonalá (například pro dodržování jmenných konvencí používá slovník, který obsahuje pouze konečnou podmnožinu anglických slov – některá hlášení tedy mohou být způsobena neznalostí použitého slova nástrojem *FxCop*), nástroj je tedy možné konfigurovat pomocí souboru pravidel (a pravidla tak odebírat nebo přidávat vlastní). Součástí projektu je tedy i soubor s pravidly pro použití nástrojem, tak aby se zamezilo zbytečným hlášením a vývojáři se mohli zaměřit na relevantní chyby.

V současnosti uvolnila firma *Microsoft* pro volné použití svůj interně používaný nástroj pro kontrolu kvality kódu v jazyce *C#* (nástroj zatím dokáže zpracovat kód psaný pouze v tomto jazyce) – *StyleCop*. *StyleCop* je obdobou nástroje *FxCop*, ovšem s daleko podrobnějšími a pedantičtějšími pravidly. Nástroj je poměrně nový a zatím neexistuje volně dostupné rozšíření pro generování přehledných *html* výstupů (pro účely integračních serverů). Takovéto rozšíření jsme si tedy

naimplementovali sami přímo jako součást sestavovacího skriptu (*MSBuild.xml.proj* dostupného jako součást projektu na přiloženém CD), příklad generovaného výstupu je na schématu Obr. 5.7.



Obr. 5.6.: Příklad výstupu analýzy kódu nástrojem FxCop

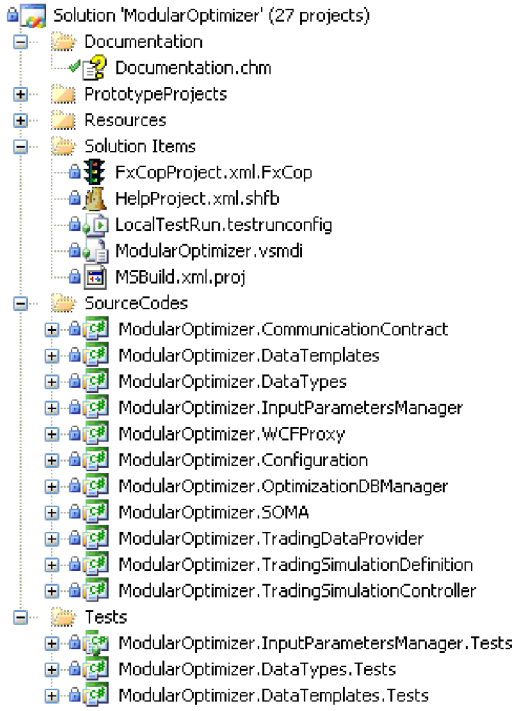


Obr. 5.7.: Příklad výstupu analýzy kódu nástrojem StyleCop

5.4 Uspořádání zdrojových kódů

Se značnou rozsáhlostí kódů v projektu, využíváním techniky prototypování a testování jednotek a používání sestavovacích skriptů pro autonomní integraci je v tomto projektu vhodně zvolené uspořádání zdrojových kódu obzvláště důležité. Zdrojové kódy tvoří šest následujících logických jednotek:

- *Produkční zdrojové kódy* – tedy zdrojové kódy projektů, jež jsou pevnou součástí systému v konečné fázi budou dohromady tvořit kompletní funkční systém.
- *Zdrojové kódy prototypů* – jsou to zdrojové kódy projektů testujících funkčnost podčástí systému (například prototyp zkoušející funkčnost komunikace, prototyp demonstrující funkčnost optimalizace apod.)
- *Zdrojové kódy testů jednotek* – části produkčních projektů (v ideálním případě všem) odpovídají projekty s testy jednotek všech prvků produkčního projektu.
- *Projektové konfigurační soubory* – soubory definující překlad, sestavení, spouštění testů, spouštění analýzy kódů a vytvoření nápovědy (většina souborů je ve formátu *xml*, popřípadě podmnožině *xml* pro popis překladu a sestavování – v jazyce *MSBuild*).
- *Dokumentace* – soubor dokumentace ve formátu *.chm.*
- *Další podpůrné zdroje* – např. *sql* skripty, databáze v souboru apod.



Obr. 5.8.: Uspořádání zdrojových kódů a projektů v systému optimalizace predikcí

Podle tohoto logického rozčlenění bylo ve struktuře zdrojových kódu navrženo i členění do odpovídajících podsložek řešení, jak naznačuje snímek Obr. 5.8.

5.5 Dokumentování

Programová dokumentace zdrojových kódů projektu vznikala iterativně jako součást nočních sestavení sestavovacího serveru. Překladač jazyka *C#* (*csc.exe*) je schopný generovat *xml* dokumentace ze speciálních komentářů přímo ve zdrojových kódech (takzvaných *xml komentářů* – komentářů začínajících trojicí znaků / a zapisovaných v podobě *xml*), dokumentování kódu tedy provádí programátor během psaní zdrojového kódu.

Standardní nástroje platformy *.NET* ovšem nenabízí žádný nástroj, který by uměl vygenerované *xml* dokumentace dále zpracovat. Vznikla tak řada volně dostupných nástrojů třetích stran. Mezi přední patřil nástroj *NDoc*, jež ovšem přestal být svým autorem udržován a vyvíjen a tak se do popředí dostal nástroj *SandCastle*. Tento velice užitečný nástroj pro generování dokumentace ve formě *html* stránek nebo Windows help souborů (*.chm*) nemá grafické uživatelské rozhraní a je poměrně složité jej konfigurovat – bývá tedy často používán spolu s nástrojem SHFB – *SandCastle Help File Builder*, který poskytuje rozhraní k nástroji *SandCastle* (ne náhodou je toto rozhranní velice podobné dříve zavedenému nástroji *NDoc*).

Oba popisované nástroje jsou součástí sestavovacího serveru (pro shlédnutí přesného použití odkazujeme čtenáře na sestavovací skript *MSBuilt.xml.proj*, který je součástí přiložených zdrojových kódů). Výstupem je pak programová dokumentace ve formě Windows help souboru – *Documentation.chm*. Tento soubor je pak automaticky vložen do úložiště kódů a programátor tak má k dispozici vždy aktuální verzi dokumentace bez nutnosti ji jakkoliv sestavovat.

5.6 Podíl autora na projektu

Vzhledem k faktu, že popisovaný projekt vznikal jako týmový projekt řešený na Fakultě informačních technologií pod vedením autora, je vhodné uvést detailně konkrétní podíl autora na tomto projektu. Projekt řešený týmem je vlastně programovou částí této diplomové práce – uvedení do teoretické problematiky tohoto projektu, které je předvedeno v prvních třech kapitolách práce a vyhodnocení projektu v posledních dvou kapitolách práce není součástí projektu a vzniklo pouze pro účely ucelené diplomové práce.

Z programové části projektu je autor práce autorem celého návrhu architektury projektu a jeho a jeho zautomatizované technické podpory (centrální sestavovací server s automatickými sestavováními a zálohováním kódů a sestavení, databázový server apod.) a dále pak je autorem zdrojových kódů následujících balíčků (dynamicky propojovaných knihoven):

- balíček s datovým kontraktem,
- balíček s komunikačním kontraktem,
- balíček s definicí prohledávaného stavového prostoru,
- balíček pro správu vektorů,

- balíček se správou simulací pro ohodnocení vektorů,
- balíček s definicí obchodního prostředí,
- balíčky s implementací obchodního prostředí,
- balíček pro optimalizační výpočty se základním genetickým algoritmem a
- podpůrné balíčky (pro obsluhování konfiguračních souborů, logování běhu, ukládání výsledků do databáze).

U následujících balíčků je pouze autorem návrhu rozhraní (datového i komunikačního – to je součástí prvních dvou výše uvedených balíčků), zadavatelem jejich implementace a osobou zodpovědnou za správnou implementaci řešiteli:

- balíček pro optimalizační výpočty s algoritmem SOMA a
- balíčky s komunikační funkcionalitou a správou běhu systému

Součástí odevzdávané verze projektu není následující balíček, který patří až do třetí iterace vývoje preojektu – tedy fáze, která bude pokračovat po úspěšném uzavření prvních dvou fází projektu, jež jsou popisovány v této práci (pro podrobnější popis fází vývoje tohoto projektu viz kapitolu 4.3 Iterativní vývoj systému):

- Balíček pro sdílení pomocných výpočetních dat.

6 Vyhodnocení

V průběhu celé implementační fáze byly vytvářeny prototypy projektů, jež testovaly funkčnost samostatných modulů (balíčků), případně tyto prototypy propojovaly více modulů a testovaly jejich správnou spolupráci. Toto testování ovšem potvrzovalo pouze technickou správnost vyvíjeného systému. Jeho logickou správnost (respektive schopnost správně vykonávat požadovanou činnost – tedy validita systému) mohla být potvrzena až po vytvoření prototypu spojujícího podstatnou část plánovaných modulů (tedy po vytvoření první alfa verze finálního systému).

Tato kapitola se bude zabývat především konkrétní instancí problému ze třídy všech možných problémů optimalizovatelných tímto systémem (tedy problému hledání nejvhodnější podoby modulu schopného předikovat časové finanční řady a tyto predikce využít k získání největšího a zároveň nejstabilnějšího zisku). Vzhledem k rozsáhlosti této třídy problémů a vyvíjeného systému jsou i jednoduché komparativní studie (porovnávající jak schopnost jednoho optimalizačního modulu řešit různé instance problémů, tak schopnosti různých optimalizačních modulů řešit jednu instanci problémů) možným netriviálním a komplexním směrem budoucího navázání na tuto práci. Spíše než objektivní studií je tak tato kapitola případovou studí – ovšem zvolenou dostatečně náhodně k tomu, aby měla pro celou problematiku dostatečně reprezentativní charakter.

6.1 Data pro simulace

Pro simulace obchodního prostředí jsme zvolili časovou řadu vývoje kurzu EUR/USD ve dnech 29. 3. 2009 až 3.4. 2009 s nejvyšší dostupnou granularitou (tedy nová data každých přibližně 10 až 100 milisekund). K dispozici tak máme přes 600 tisíc datových bodů. Ve skutečnosti však tyto datové body nejsou zcela atomické a mohou reprezentovat více než jeden časový okamžik (obsahují jak cenu nabídky tak poptávky a časy platností těchto cen – ty však mohou být odlišné). Data je tedy nutné předzpracovat čímž získáme přes 1,2 miliónů datových bodů uskupených ve zhruba 700 tisících různých časových okamžicích. Existují tedy časové okamžiky, kdy je dostupná pouze jedna strana ceny (nabídka versus poptávka) a okamžiky, kdy může být pro jednu stranu trhu dostupných více cen (s odlišnou velikostí dostupné částky).

Tato řada pak slouží během simulace k poskytování dat obchodnímu systému a uspokojování objednávek modulem brokera. Simulátor k datům přistupuje systémem popsaným v kapitole 4.2.7 Balíček se správou simulací pro ohodnocení vektorů. Pro vytváření statistik obchodování je ovšem potřeba také znát nejlepší ceny nabídky a poptávky v každém časovém okamžiku, tím se dostáváme k dalšímu předzpracování dat – příklad takto předzpracované řady lze shlédnout v materiálu Příloha 2.

Obchodní systém si může pro své potřeby od simulátoru vyžádat libovolné jiné předzpracované řady (musí být ovšem implementovány v dostupném modulu), simulátor pak umožňuje efektivní sdílení takovýchto dat. Další příklad takové řady lze vidět na grafu v Příloha 3, bližší popis významu této řady je součástí následující kapitoly popisující implementovaný obchodní systém.

6.2 Predikční obchodní modul

Při vytváření predikčních modelů se v zásadě postupuje podle obecných principů popsaných v kapitole 2.2.2 Postup modelování časových řad, obchodní aplikace takového modelu pak jen generuje příkazy na základě takovýchto predikcí. Podobně jsme postupovali i my při implementaci našeho obchodního modulu. Pro jednoduchost a názornost modelu jsme se ale zaměřili pouze na cyklickou složku (a ignorovali trendovou a sezónní složku). Pro časovou řadu s cyklickou složkou bez sezónní a trendové složky je typické, že se neustálými výkyvy posouvá přes svoji určitou nulovou hladinu – toto je také typické krátkodobé chování (během hodin až dní) cen aktiv na trhu (pokud se nevyskytují významné události silně ovlivňující cenu), tedy možné očekávané chování pro naše data..

Indikátorů popisujících cyklické složky řad a generujících hodnoty na základě nichž je možné obchodovat je velké množství. My jsme zvolili implementaci vlastního, jež by byl parametrizovatelný (a tím optimalizovatelný) a zároveň založený na nějakém účinném reálném indikátoru. Tím reálným základem se pro nás stal indikátor *stochastic* (viz [20]).

Náš upravený indikátor pracoval jen na základě předchozích minim a maxim. K tomuto účelu jsme vytvořili dva typy předzpracovaných řad – jeden udávající, pro kolik bezprostředně předcházejících bodů je tento bod extrémem (minimem nebo maximem), na základě níž vznikla řada udávající, kolik bezprostředně předcházejících extrémů aktuální bod překonal (pro kolik předchozích vrcholů má aktuální bod vyšší hodnotu, respektive pro kolik sedel má aktuální bod nižší hodnotu). Příklad dvou takto vypočtených instancí poslední uvedené řady (počet bezprostředně překonaných maxim ceny poptávky a počet bezprostředně překonaných minim ceny nabídky) můžeme vidět na grafu v Příloha 3.

Implementovaný systém byl pak následně parametrizovatelný dvěma přirozenými čísly udávajícími počet bezprostředně překonaných maxim ceny poptávky po kterých má systém vytvořit objednávku pro prodej fixního obnosu měny, respektive počet bezprostředně překonaných minim ceny nabídky, po kterých má systém vytvořit objednávku pro nákup fixního obnosu měny. Bylo tedy třeba optimalizovat uspořádané dvojice $(a,b) \in \{1,...,100\} \times \{1,...,100\}$. Postup optimalizování (generování a ohodnocování) těchto dvojic je obsahem následující kapitoly.

6.3 Implementovaný evoluční algoritmus

Pro povahu úlohy – tedy optimalizování vektorů přirozených čísel – byla zvolena implementace upraveného tradičního genetického algoritmu (viz kapitolu 3.1.1 Genetické algoritmy). Jedinci byli kódováni jako n-tice přirozených čísel a algoritmus používá následující operátory (s možností v konfiguračním souboru nastavit jejich četnost):

- ***Mutace*** – na náhodné pozici vektoru je generováno náhodné číslo z oboru hodnot této dimenze vektoru
- ***Křížení (jednobodové)*** – od náhodně vybrané pozice dvou rodičovských vektorů dojde k prohození hodnot těchto vektorů. Rodičovské vektory byly vybírány *ruletovou proporcionální* metodou na základě jejich fitness.
- ***Elitismus*** – Nejlepší jedinec generace je automaticky beze změny vložen do následující generace.

Pro povahu stavového prostoru s velikou náhodností *fitness landscape* byly nastavené vysoké četnosti operací *mutace* a *křížení* a byla zcela zakázaná operace *elitismu* – tím jsme dosáhli rozprostřenějšího prohledáván i v zdánlivě nevhodných oblastech. Tím se ale algoritmus blížil slepému prohledávání – proto jsme přidali jeden operátor vlastní:

- ***Prohledávání okolí elitních jedinců*** – pro nastavitelný počet nejlepších jedinců se vygenerují potomci v náhodné vzdálenosti (s nastavitelným maximem vzdálenosti) od těchto jedinců – tím pokračujeme v prohledávání lokálních maxim a zkoumání, zda se nejedná o globální maximum. V zásadě se jedná o mírně upravený elitismus.

Optimalizace byla provedena tímto algoritmem s následujícím způsobem nastavenými hodnotami (v konfiguračním souboru *ModularOptimizer.GeneticAlgorithm.dll.xml*):

- řád mutací – 20%
- řád rekombinací - 50%
- elitismus – ne
- počet elitních jedinců k prohledávání okolé – 3
- maximální krok při prohledávání okolí elitních jedinců – 5 (tj 5 krát granularita dimenze)
- počet jedinců v generaci (velikost populace) – 20
- zkončit pokud počet generacé překročí maximum – ano
- maximum počtu generací – 100
- zkončit pokud nedochází k vylepšování řešení – ano
- přípustný počet generací s nestoupající fitness – 40
- zkončit pokud fitness dosáhne určité hodnoty – ne

6.4 Hodnotící modul

Modul, který počítá pro běhy obchodovaní hodnotu fitness udávájící jejich úspěšnost je poměrně jednoduchý z hlediska implementačního, o to složitější z hlediska navržení správného měřítka hodnocení obchhodování.

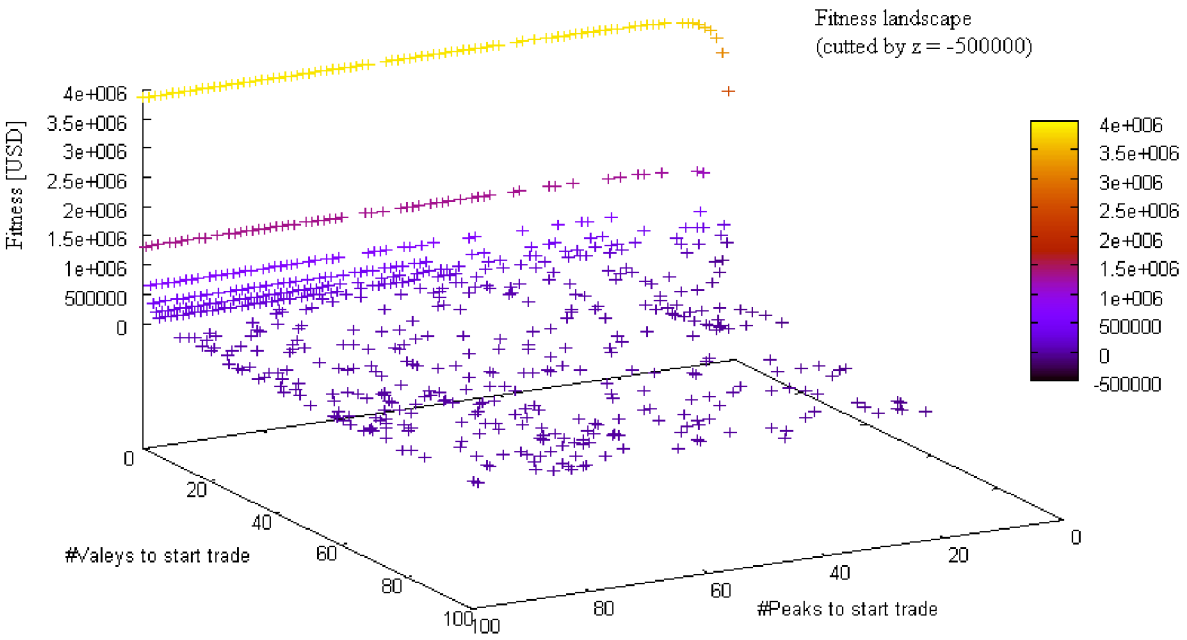
Hodnotící modul má ve chvíli hodnocení k dispozici kompletní statistiky obchodování (vývoj průběžného zisku – tzv *equity*, pozice na trhu, ceny pozice na trhu, aktuálního maximálního propadu equity, seznam všech proběhlých *obchodů* – tj. posloupností pozic majících na začátku a konci společné body s nulovou hladinou – a jejich statistky jako nejvyšší zisk a ztráta v průběhu obchodu, závěrečný zisk atd.). Jistě nejdůležitější statistikou je hodnota závěrečného zisku (*equity*), který udává kolik systém vydělal. Tato hodnota se ovšem může při mírné změně dat značně změnit. Proto je také důležitým ukazatelem *maximálné propad equity* v průběhu obchodování (*maximal equity drawdown*), který naznačuje jak by se mohla závěrečná equita změnit. Hodnotu fitness jsme tedy počítali na základě následně zvoleného vzorce:

$$fitness(\text{TradingLog}) = ClosingEquity - \frac{MaximalEquityDrawDown}{2} \qquad (6.1.)$$

6.5 Výstupy optimalizace

Celá optimalizace byla provedena pouze v lokální verzi s dvěmi paralelními simulačním vlákny. Celková doba optimalizace činila přibližně tři a půl hodiny a během této doby bylo vygenerováno sto generací o dvaceti jedincích a pro každého unikátního jedince byla provedena kompletní simulace obchodování. Vygenerovaní jedinci a jejich fitness počítaná na základě vzorce (6.1.) byly v průběhu optimalizace ukládáni do databáze. Po skončení tak bylo možné získané výsledky vizualizovat – k tomu jsme použili volně dostupný nástroj *GNUPlot*.

Podoba prozkoumané části fitness funkce lze vidět na Obr. 6.1 (Nebarvený graf a jeho kolmé průměty do bočních rovin je pak součástí příloh jako Příloha 4, grafy interpolovaných hodnot jsou součástí příloh jako Příloha 5, Příloha 6, Příloha 7). Pro lepší přehlednost je graf obarven podle hodnoty fitness a zbaven jedinců s hodnotou fitness horší než -5000000 (v průběhu celé optimalizace byly vygenerovaní tři takovýto jedinci - všichni s hodnotou fitness v rozsahu -10000000 až -9000000).



Obr. 6.1.: Troj dimenzionální graf prozkoumaných hodnot fitness funkce.

6.5.1 Nejúspěšnější jedinec

V průběhu optimalizace (animovaný obrázek průběhu prozkoumávání stavového prostoru je součástí elektronických příloh) byl nalezen nejlepší jedinec s genotypem (98,0) s hodnotou fitness zhruba 3,9 miliónů. Vizualizace průběhu průběžného zisku a aktuálního maximálního propadu zisku je možné shlédnout na grafu Příloha 8.

Byl tedy úspěšně prohledán definovaný stavový prostor a nalezen nejúspěšnější jedinec, jehož průběh obchodování vykazuje velmi dobré výsledky (většinu času je jeho zisk i po odečtení celkové hodnoty maximálního propadu zisku je v kladných hodnotách).

Jakékoliv kladné výsledky je ovšem nutné podrobit důkladné kritické analýze. Tou může být v našem případě současné zobrazení ceny obchodovaného aktiva a průběžného zisku do jediného grafu (po úpravě měřítek) – tak jako to můžeme vidět na grafu Příloha 9. Na takovém grafu můžeme vidět nápadně vysokou korelaci zobrazovaných průběhů. Také genotyp nejúspěšnějšího jedince – (98,0) – může být nápadný. Z grafu průběhu cen (viz Příloha 2) je zřetelný růstový trend cen. Pokud na predikci a indikaci vhodných okamžiků k nákupu použijeme indikátor založený na cyklické povaze datové řady, bude takovýto indikátor generovat převážně signály v jednom směru (pro náš případ nákup). Optimalizace navíc tento indikátor nastavila tak, že má prakticky nulový práh generování signálů k nákupu (stačí, aby se změnil směr vývoje ceny na záporný) a velmi vysoký práh prodeje (je nutno překonat 98 bezprostředně předchozích lokálních maxim ceny) – systém tak téměř

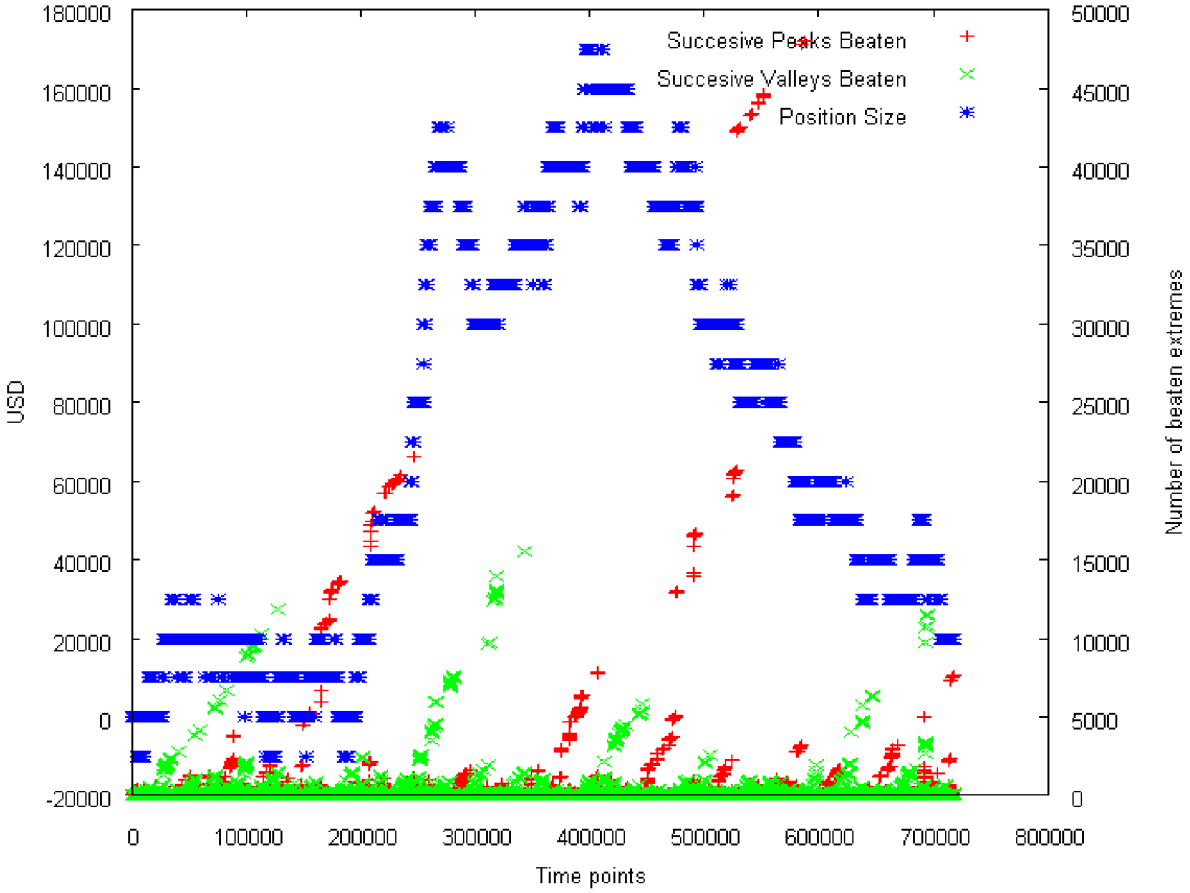
neustále nakupuje, čímž zvyšuje svoji pozici, při růstu ceny pozice je to ovšem cesta k největšímu zisku.

Z technického hlediska je všechno naprosto správně. Otázkou je ovšem použitelnost takovéhoto řešení – lépe použitelných řešení bychom mohli dosáhnou použitím ještě většího vzorku dat nebo úpravou funkce fitness, která by penalizovala obchodní systémy držící (nebo navyšující) příliš dlouho otevřenou pozici a tím se vystavující riziku náhlého propadu ceny (pokud by ovšem tento propad nebyl zcela monotónní, pak by indikátor začal brzy generovat signály pro prodej – propad zisku systému by tak nemusel nutně nastat – v Příloha 9 je možné vidět, že propad průběžného zisku má pozvolnější trend než propad ceny aktiva).

6.5.2 Průměrný jedinec

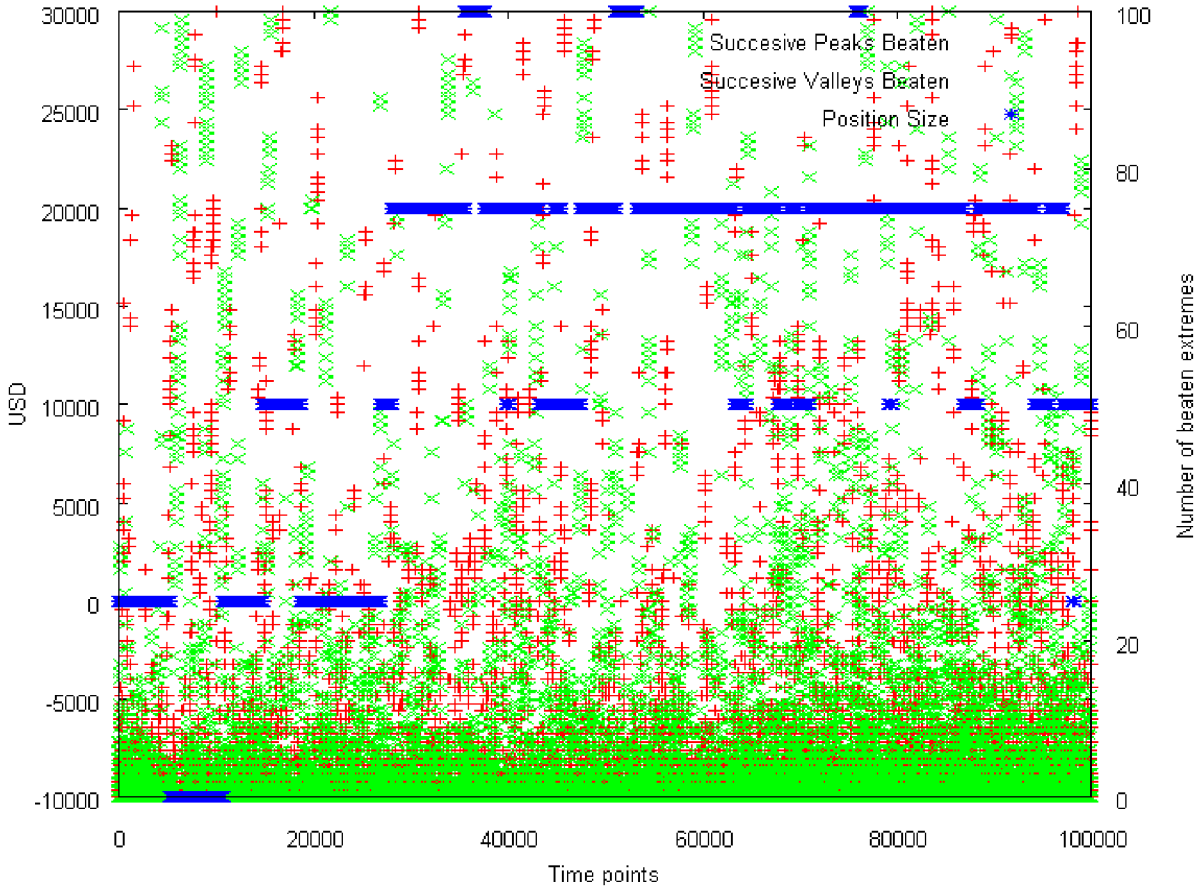
Pro ilustraci toho, jak by se mohl chovat teoreticky více adaptabilní jedinec (tj. jedinec nepreferující směr vývoje ceny a obchodující na základě delšího vývoje) jsme vybrali jedince s genotypem (42, 40) (Tedy čekající na překonání 42, respektive 40, bezprostředně předcházejících minim, respektive maxim pro vygenerování příkazu pro nákup, respektive prodej). Tento jedinec získal v simulovaném obchodním prostředí s výše představeným vývojem cen fitness zhruba 800 (tj., stále je schopen generovat zisk i po odečtu poloviny maximální možné ztráty, který není v poměru ke zvolené fixní investované částce 10000 na daný časový úsek zanedbatelný).

Na Obr. 6.2 je možné vidět průběh pozice tohoto systému v průběhu celého obchodování. Je vidět, že tento průběh není rozhodně monotónní, ovšem v oblastech výrazného růstu/klesání ceny (jež vyvolává generování řady signálů pro nákup a prodej) svoji pozici patřičně mění.



Obr. 6.2.: Vývoj velikosti pozice na trhu a generovaných signálů pro změnu pozice pro celé obchodování

Detail tohoto grafu je na obrázku Obr. 6.3 – zde je možné vidět množství informací na základě nichž se obchodní systém rozhoduje o změně své pozice na trhu. Také je vidět jistá „hystereze“ systému – i přes velké množství signálů mění svoji pozici zřídka (je možné rozeznat kolem 30 změn pozice na 100000 časových bodů).



Obr. 6.3.: Vývoj velikosti pozice na trhu a generovaných signálů pro změnu pozice pro detail obchodování

6.5.3 Závěr případové studie

Lze tedy říci, že jedinci protěžovaní touto optimalizací jsou pravděpodobně schopni se prosadit i ve změněných podmínkách – k podložení této domněnky by bylo ovšem nutné další rozsáhlé testování (především velké množství dat). V daných podmínkách však bezesporu nalezení jedinci vykazovali dobré až nadprůměrné výsledky, z grafů byla možná vyčíst zřejmá závislost parametrů predikčních modelů na jejich výsledcích – tedy byla potvrzen úvodní myšlenka možnosti evolučně optimalizovat predikční modely časových řad aplikovaných na obchodování na finančních trzích.

7 Závěr

Výsledkem předkládané práce je jednak sumarizace teoretických znalostí v oblasti teorie časových řad, jejich analýzy a aplikace na finančních trzích a v oblasti evolučních algoritmů, jejich klasifikaci a použití. Následně pak praktická aplikace těchto znalostí v podobě optimalizačního systému predikčních modelů, jež je vysoce škálovatelný (možnost paralelizace výpočtů na úrovni vláken i procesů na více fyzických výpočetních zdrojích za současného sdílení předpřipravených simulačních dat) a modulární (možnost měnit jakoukoliv podčást řešení. Změna instance optimalizovaného problému popřípadě modulů hodnotících a optimalizujících tento problém bez nutnosti sestavování projektu a ručního rozmisťování aktuálních balíčků na výpočetní stroje).

Největším potenciálním úskalím této práce byl fakt, že se jednalo o praktickou realizaci myšlenky, jež nelze dohledat v dostupných odborných pramenech. Z uvedeného sumarizovaného teoretického pozadí úlohy vyplývá jistá pravděpodobnost úspěchu takovéto myšlenky (evolučně optimalizovat predikční modely časových finančních řad a získat dostatečně prediktivní modely), definitivní potvrzení ovšem vyžadovalo samotnou implementaci a tedy bylo možné vyhodnotit až po kompletní realizaci díla.

Všechny ostatní popsané vyskytnuvší se problémy byly vyřešeny a zkoumaná myšlenka byla realizována v podobě implementace optimalizačního systému podle uvedeného návrhu. Implementovaný systém je do jisté míry možno použít pro optimalizaci libovolného problému k němuž umíme najít funkci fitness přiřazující každému řešení reálnou hodnotu udávající úspěšnost tohoto řešení. V práci byly do systému implementovány moduly umožňující simulovat obchodování pomocí predikčního modelu a hodnotit predikční model podle úspěšnosti tohoto obchodování. Možným praktickým použitím této práce je ovšem možnost optimalizovat jakýkoliv jiný problém po implementaci vhodného hodnotícího modulu.

Úspěšnost systému byla vyhodnocena případovou studií optimalizace genetickým algoritmem jednoho prediktivního modelu pro datovou řadu o více než 700000 rozdílných časových bodech. V této případové studii byla prokázána korelace mezi hodnotami generovaných vektorů parametrů a úspěšností predikčních modelů upravených těmito vektory – tedy nutný základ pro úspěšné provádění optimalizací. K dalšímu podložení této myšlenky je ovšem nutné provést rozsáhlé komparativní studie (různých predikčních modelů, simulačních dat a optimalizačních modulů), které mohou být netriviálním komplexním směrem navázání na tuto práci.

Hlavním přínosem tohoto díla by tedy měla být jeho praktická část demonstrující použití evolučních algoritmů na netriviálním reálném problému s možností systém snadno modifikovat na řešení optimalizující odlišné problémy. Hlavní možnosti vylepšení a navázání na stávající podobu práce však autor vidí v pokračování v optimalizacích predikčních modelů a to především v kromě zmíněných komparativních studiích také v důkladném profilování existujícího řešení a jeho

následného optimalizování za účelem zvýšení výpočetního výkonu, dále v zautomatizování vyhodnocování úspěšnosti jednotlivých optimalizačních modulů, schopnosti reagovat na změny dat a definic stavového prostoru za běhu optimalizací a další.

Literatura

[1] Brockwell, P. J., Davis, R. A.: Introduction to Time Series and Forecasting, Second edition, Springer, 2002. ISBN 0-387-95351-5

[2] Barbazon, A.: Biologically Inspired Algortithms for Financial Modelling. Springer 2006. ISBN-10 3-540-26252-0

[3] Sekanina, L.: Biologií inspirované počítače, Studijní opora, část první. FIT VUT 2006.

[4] Řezanková, H., Marek, L., Vrabec, M.: IASTAT – Interaktivní učebnice statistiky, projekt FRVŠ F5 0009/2000 a FRVŠ F5 1450/2001.

[4] Wikipedia: Time series, článek dostupný elektronicky na adrese http://en.wikipedia.org/wiki/Time_series (listopad 2008)

[5] Chang, Y., Park, Y. J.: Index models with integrated time series, Journal of econometrics, 2003. Dokument dostupný elektronicky na adrese http://www.ruf.rice.edu/~econ/papers/1998papers/03Chang.pdf (listopad 2008)

[6] Slaný, K.: Evoluční adaptace prediktorů v reálném čase, Teze disertační práce. FIT VUT v Brně 2008.

[7] Plummer, E. A.: Time series forecasting with feed-forward neural networks: guidelines and limitations. University of Wyoming 2000.

[8] Fárková, L.: Generující funkce a náhodná procházka, Bakalářská práce. Masarykova Universita 2006. Dokument dostupný elektronicky na adrese http://is.muni.cz/th/106394/prif_b/GFaNP.pdf (listopad 2008)

[9] Gladiš, D.: Naučte se investovat, druhé, rozšířené vydání. Grada 2005. ISBN 8024712059

[10] Schwarz, J., Sekanina, L.: Aplikované evoluční algoritmy, studijní opora. FIT VUT v Brně 2006.

[11] Wikipedia: Evolutionary Algorithm. článek dostupný na elektronické adrese http://en.wikipedia.org/wiki/Evolutionary_algorithm (prosinec 2008)

[12] Eiben, A. E., Smith, E.: Introduction to Evolutionary Computing. Springer Verlag, November, 2003, ISBN 3540401849. Kniha je omezeně dostupná na elektronické adrese http://www.cs.vu.nl/~gusz/ecbook/ (prosinec 2008)

[13] Sipper, M. a kol.: A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems. IEEE Transactions on Evolutionary Computation, 1997.

[14] Hruška, T.: Informační systémy: Pojem informačního systému, data, procesy, transakce, FIT VUT v Brně 2008

[15] Gamma, E, Helm E., Vlissides, J.: Design Patterns: elements of reusable object-oriented software, Addison-Wesley, Massachusetts 1997, ISBN 0-201-63361-2

[16] Zeigler, P. B., Praehofer, H., Kim, T. G.: Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems, Academic Press, 2000, ISBN 0127784551

[17] McConnell, S. ,: Code Complete: A Practical Handbook of Software Construction, Microsoft Press, Redmond - Washington 1993, ISBN 1-55615-484-4

[18] Wikipedia: Mono. článek dostupný na elektronické adrese http://cs.wikipedia.org/wiki/Mono_(platforma) (březen 2009)

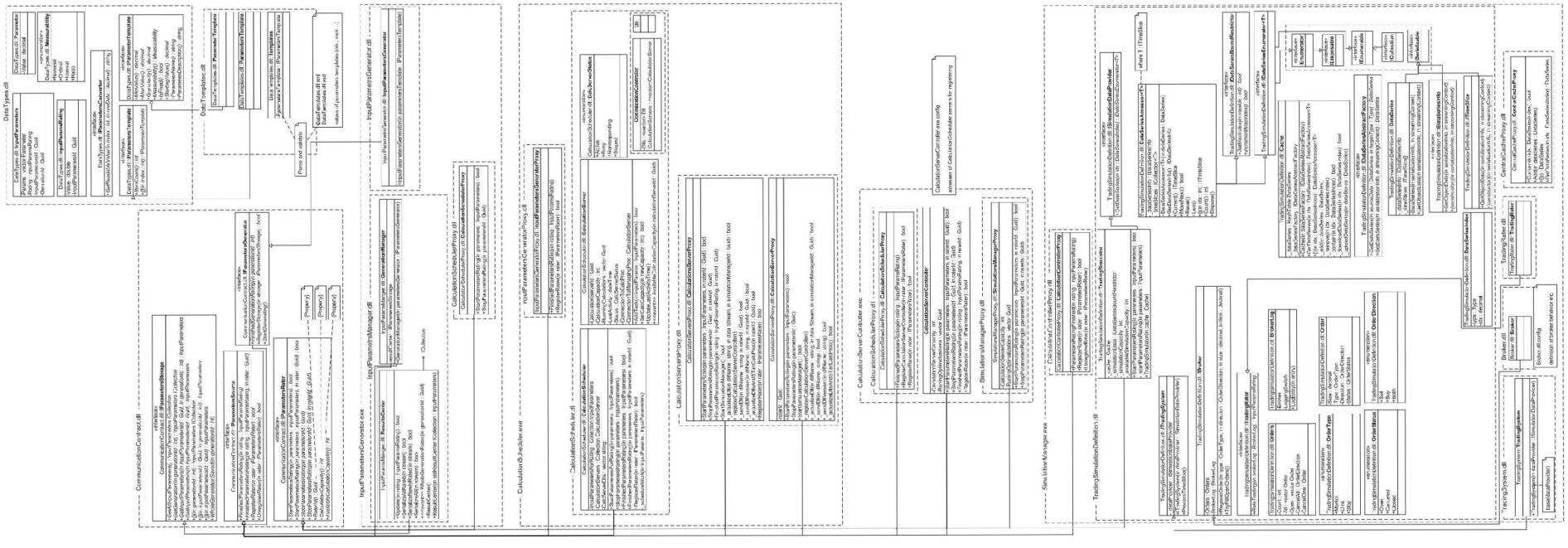
[19] Wikipedia: dotGNU. článek dostupný na elektronické adrese http://cs.wikipedia.org/wiki/DotGNU (březen 2009)

[20] Wikipedia: Stochastic oscillator. článek dostupný na elektronické adrese http://en.wikipedia.org/wiki/Stochastic_oscillator (duben 2009)

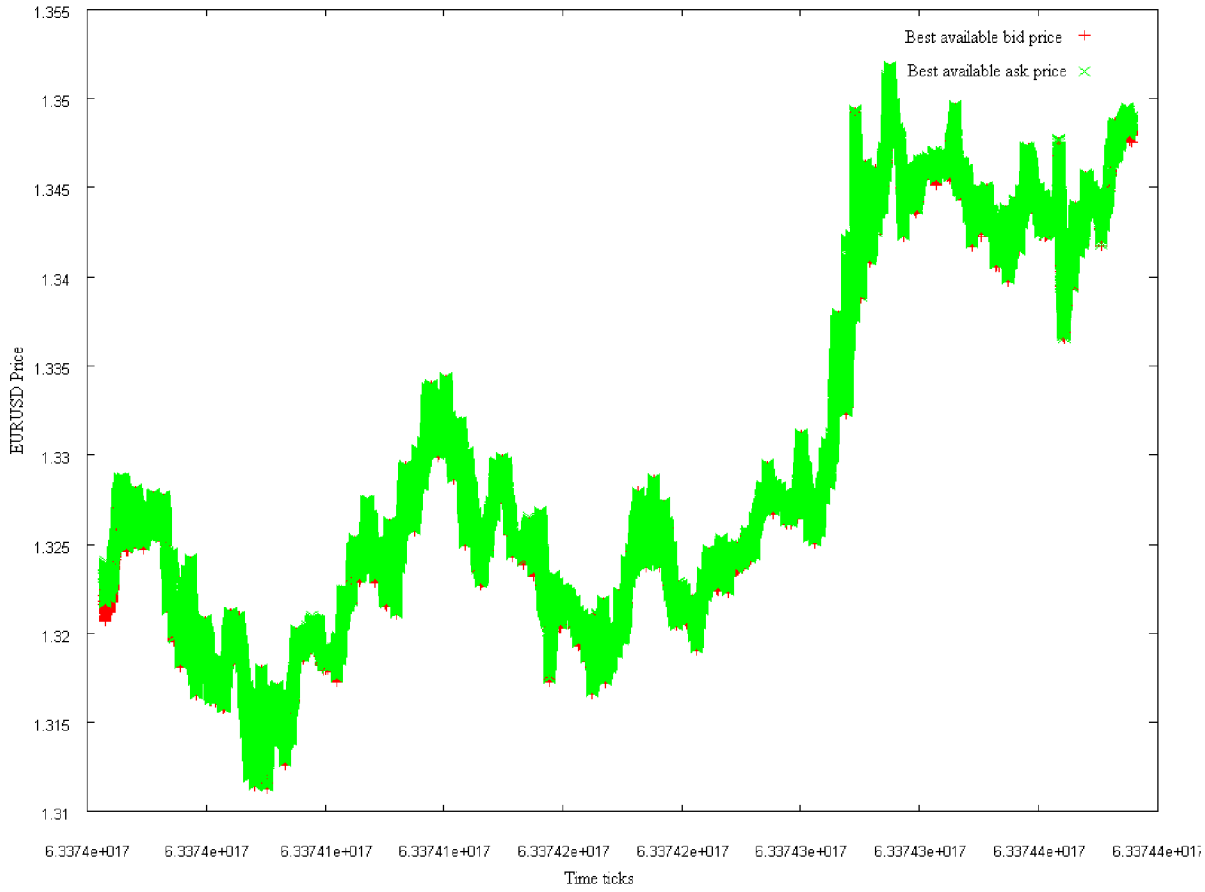
Seznam příloh

Příloha 10: CD se složkami
- doc: elektronická forma textu diplomové práce
- src: všechny vytvořené zdrojové kódy, spolu s programovou dokumentací
- bin: příklad přeloženého optimalizačního systému

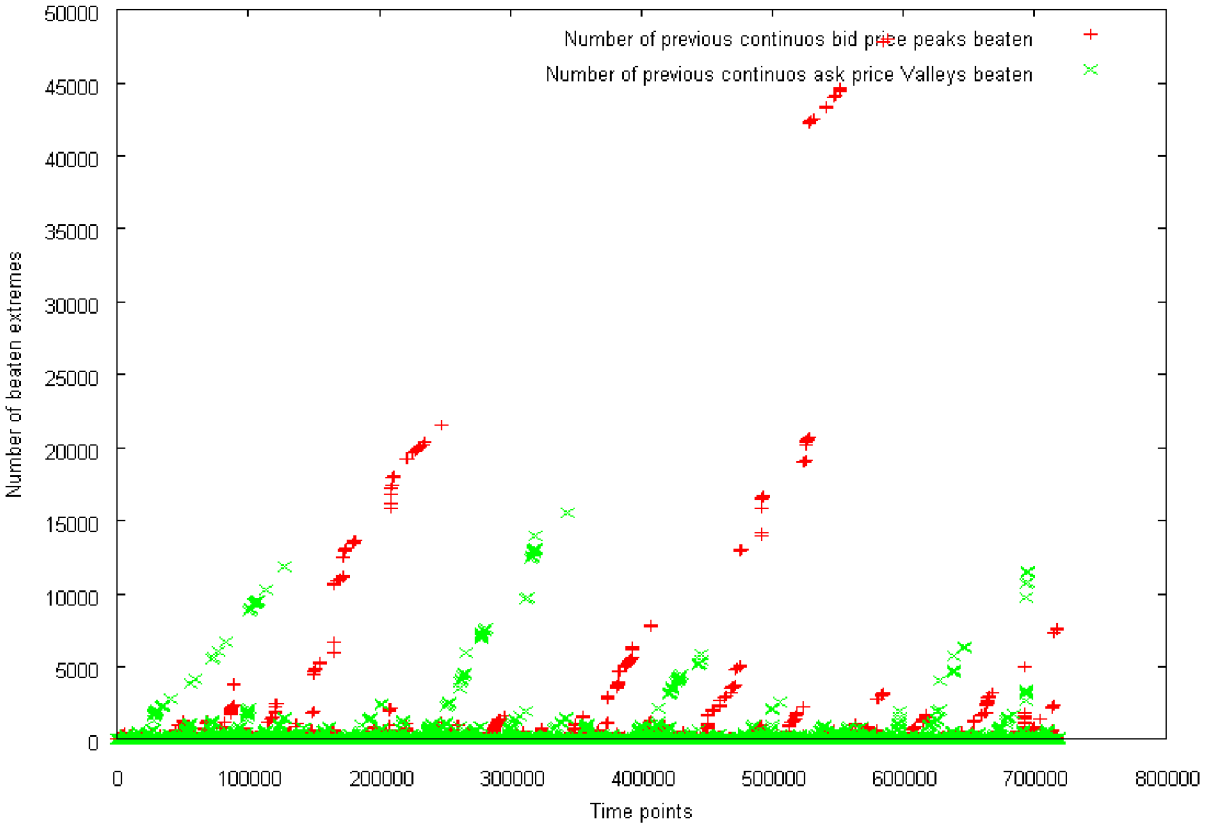
Příloha 1: Diagram návrhových tříd modelovaného systému



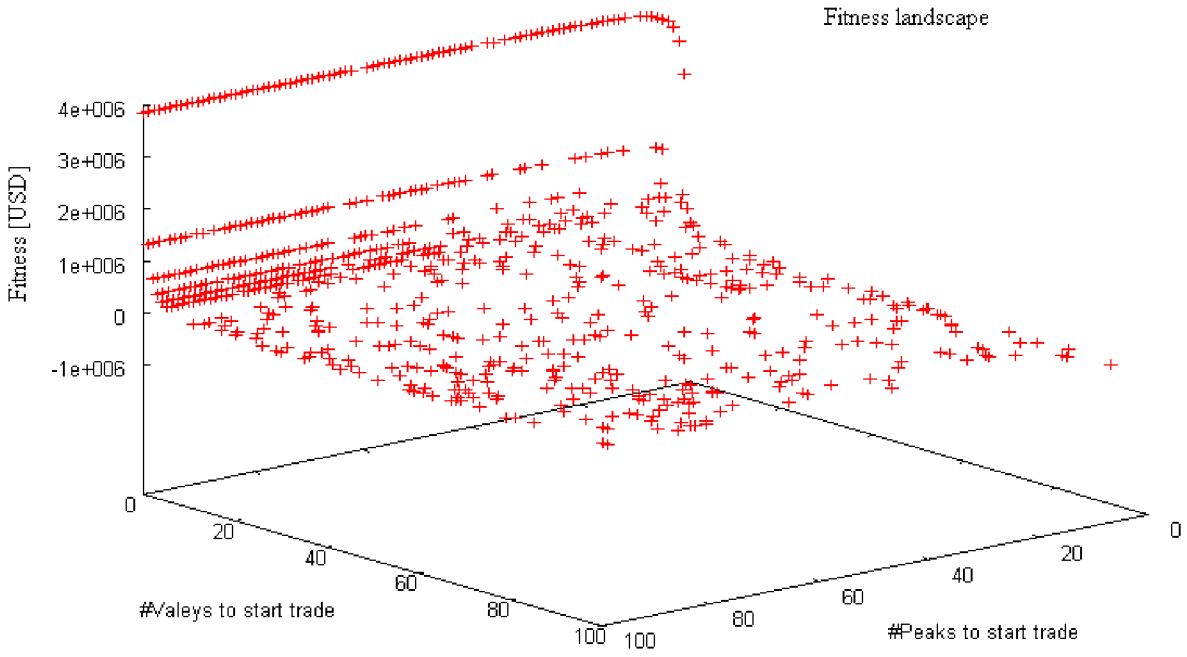
Příloha 2: Vývoj cen nabídky a poptávky měnového páru EURUSD 29.3. – 3.4. 2009

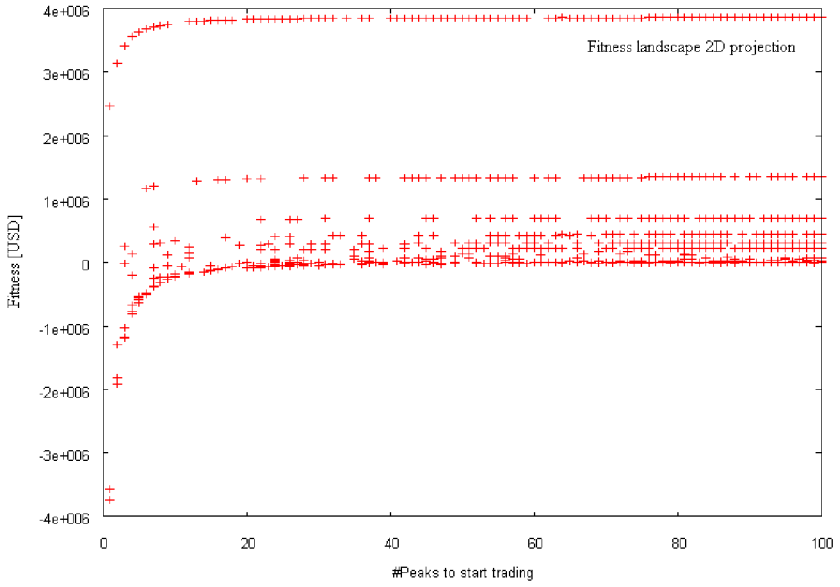


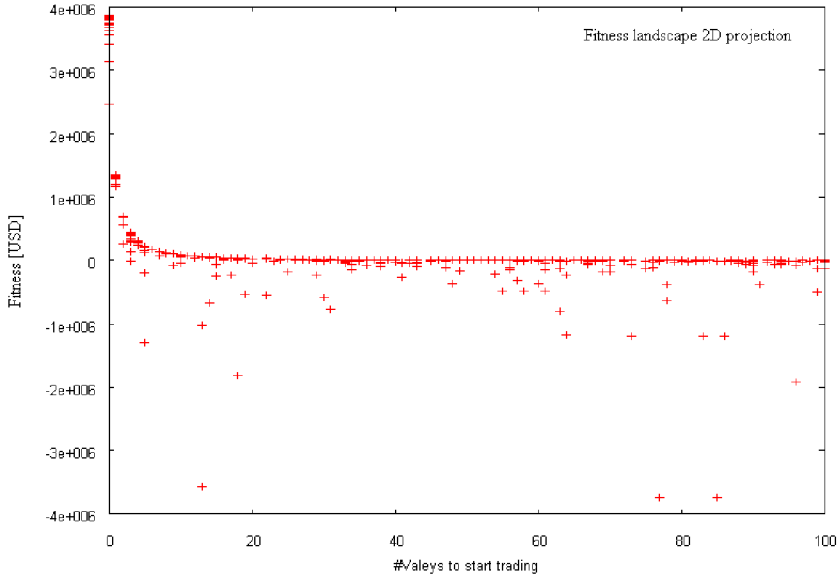
Příloha 3: Indikátor udávající počty překonaných bezprostředně předcházejících extrémů



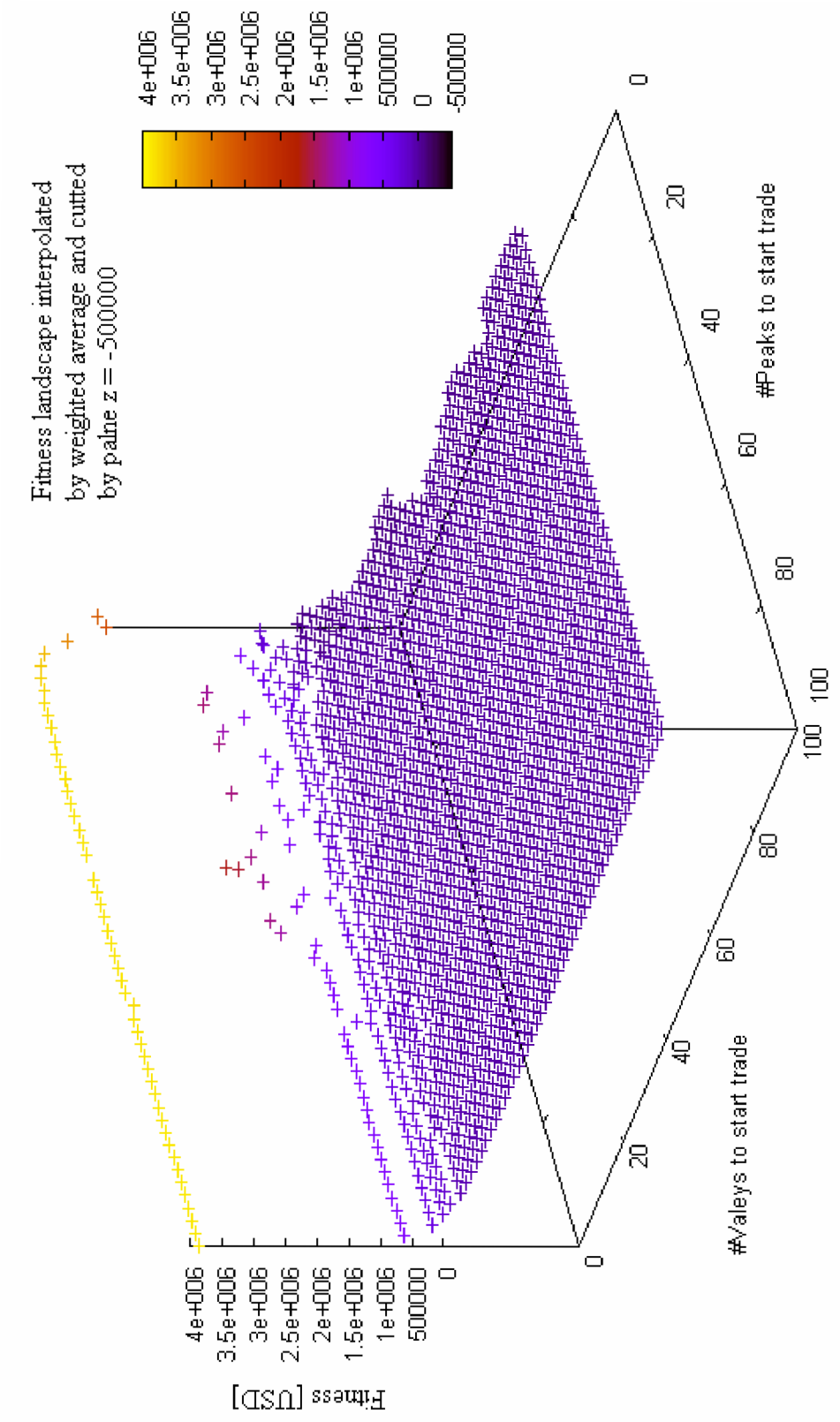
Příloha 4: Získané body fitness funkce zobrazené v 3D grafu a v kolmých průmětech



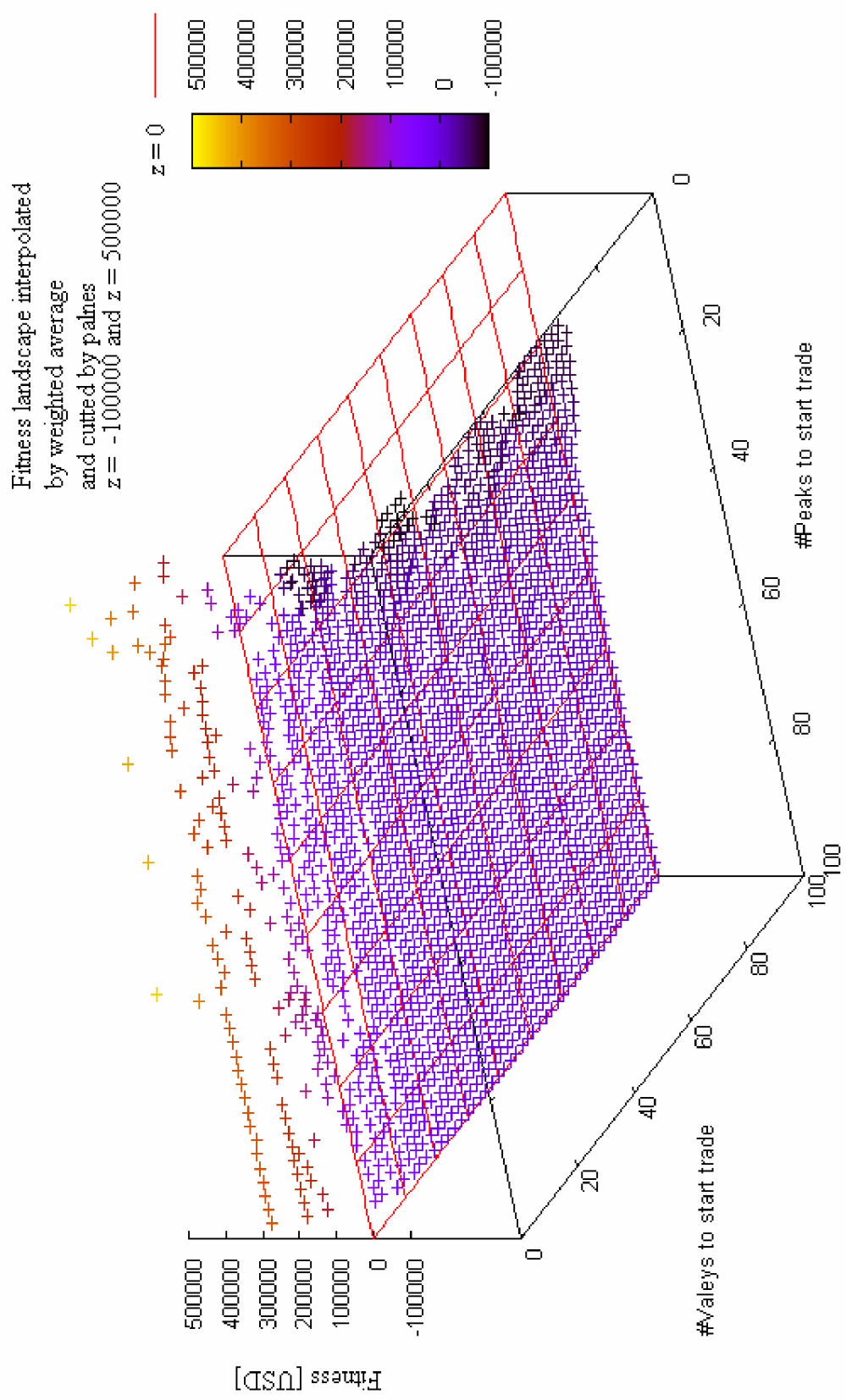




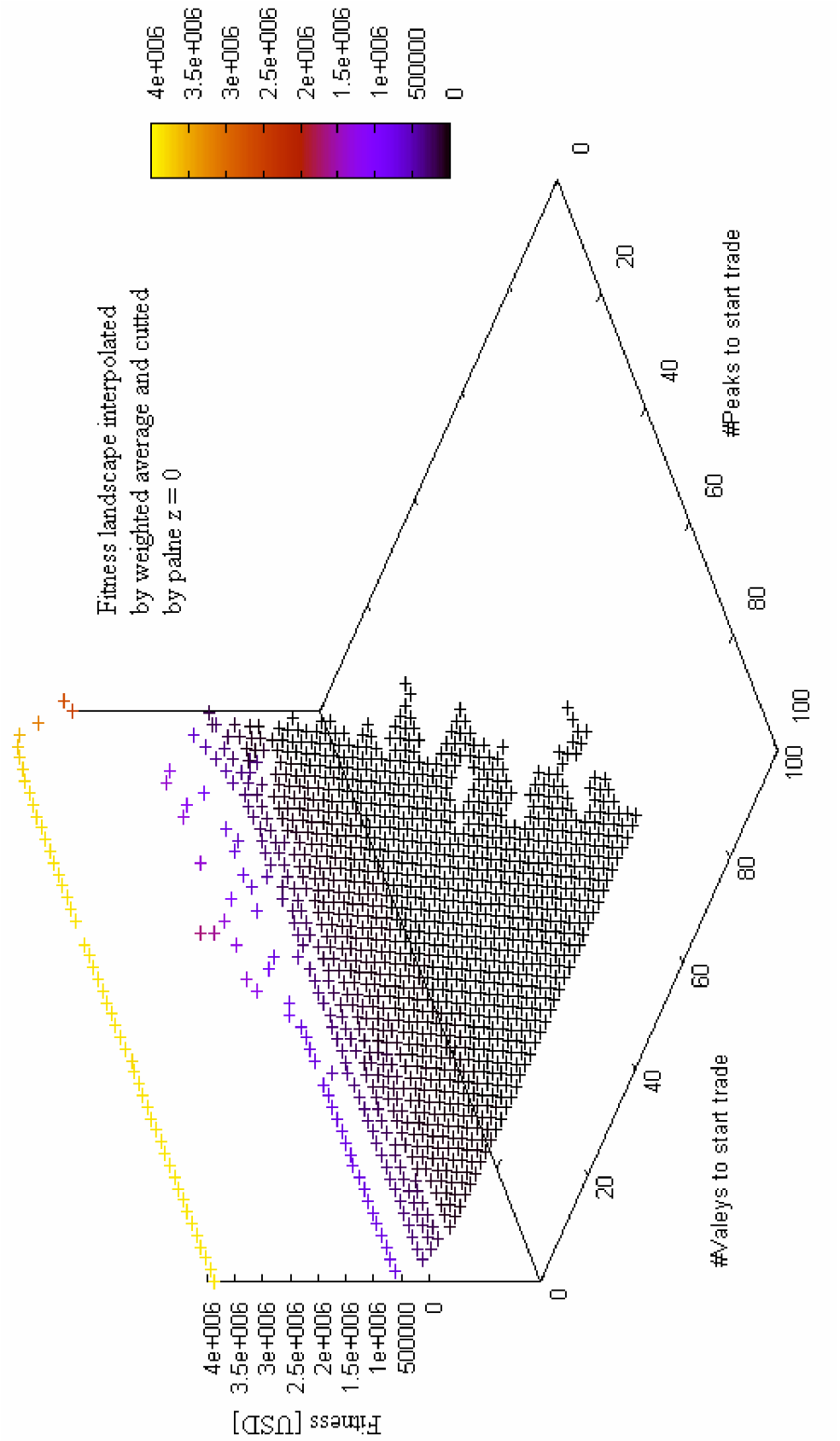
Příloha 5: Řez získaných bodů fitness funkce interpolované váženým průměrem rovinou z = –500000



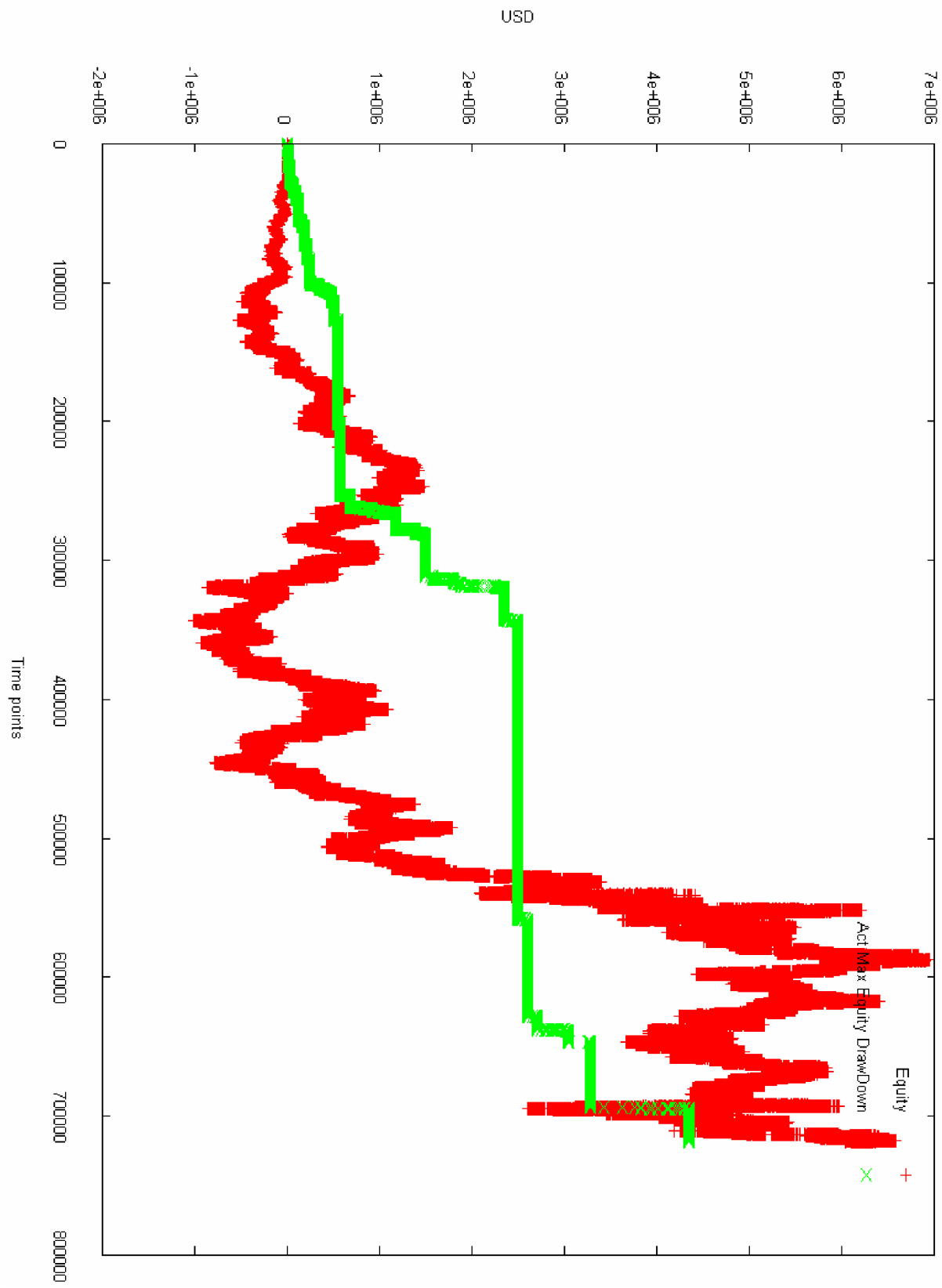
Příloha 6: Řez získaných bodů fitness funkce interpolované váženým průměrem rovinami z = –500000 a z = 100000



Příloha 7: Řez získaných bodů fitness funkce interpolované váženým průměrem rovinou z = 0



Příloha 8: Průběh průběžného zisku a průběžného maximálního propadu zisku pro nejúspěšnějšího vygenerovaného jedince



Příloha 9: Korelace průběžného zisku nejúspěšnějšího vygenerovaného jedince a vývoje ceny obchodovaného měnového páru