VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AOS: SPRÁVA VIRTUÁLNÍCH OBJEDNÁVEK

AOS: VIRTUAL ORDER MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LEOŠ NAVRÁTIL**
AUTHOR

VEDOUCÍ PRÁCE — **Mgr. ROMAN TRCHALÍK, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/18749/2016/xnavra50

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Navrátil Leoš**
Obor: Informační technologie
Téma: **AOS: Správa virtuálních objednávek**
AOS: Virtual Order Management
Kategorie: Informační systémy

Pokyny:

1. Seznamte s obchodní platformou Ninja Trader nebo Meta Trader.
2. Analyzujte požadavky na zpracování dat objednávek (obchodních pokynů) a vytvořte návrh služby/démona pro jejich evidenci.
3. Dle pokynů vedoucího implementujte službu (server) pro zpracování objednávek a klienta pro Ninja Trader nebo Meta Trader, který vyvíjené službě bude předávat informace o zadané objednávce. Implementovaná služba by měla umožňovat síťovou komunikaci se vzdáleným klientem běžícím i na jiném stroji.
4. Výsledek Vaší práce prezentujte ve vhodné formě.

Literatura:

- Ninja Script Reference [cit. 2016-10-13] <https://ninjatrader.com/support/helpGuides/nt7/?alphabetical_reference.htm>
- ESPOSITO, Antonio. *Learning .NET High-performance Programming*. Packt Publishing, 2015. ISBN 9781785288463.
- R.M.C. Pinto. Strategic methods for automated trading in Fore. JCM Silva, 2012. ISBN: 978-1-4673-5117-1
- J. J. Murphy. Study Guide to Technical Analysis of the Financial Markets, 1999. ISBN: 978-0735200654
- R. D. Edwards. Technical Analysis of Stock Trends, Tenth Edition. 2012. ISBN: 978-1439898185

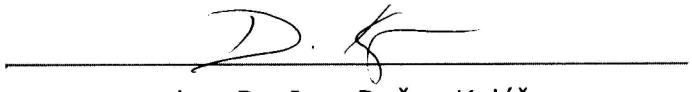
Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Trchalík Roman, Mgr., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2



doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cílem této bakalářské práce je navrhnout a implementovat centrální systém pro zadávaní, uzavírání a evidenci objednávek pro větší počet vzdálených klientů. Tohoto cíle bude dosaženo pomocí abstrakce obchodního pokynu v obchodní platformě do virtuální objednávky v centrálním systému. Navržený systém by měl podporovat nákup či prodej instrumentu současně u více klientů s předem definovaným objemem tak, aby docházelo k co nejmenšímu ovlivnění trhu ze strany makléře. Jinými slovy, systém rozloží nákup do více pokynů u různých makléřů.

Abstract

The aim of this bachelor's thesis is to design and implement a central system which will allow to enter, close and register orders from multiple remote clients. This goal will be achieved by abstracting from a trade order in trading platform using virtual order within central system. The suggested system should support buying or selling of desired instrument from multiple clients with defined volume at once in order to diminish broker's influence over the market. In other words, the system will spread out the order to multiple brokers.

Klíčová slova

NinjaTrader, MetaTrader, .NET Framework, WPF, MVVM, správa virtuálních objednávek, WebSocket

Keywords

NinjaTrader, MetaTrader, .NET Framework, WPF, MVVM, virtual order management, WebSocket

Citace

NAVRÁTIL, Leoš. *AOS: Správa virtuálních objednávek*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Mgr. Roman Trchalík, Ph.D.

AOS: Správa virtuálních objednávek

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Mgr. Romana Trchalíka, Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Leoš Navrátil
15. května 2017

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce za konzultace, odbornou pomoc a v neposlední řadě za jeho pozitivní přístup.

Obsah

1 Úvod **3**

2 Použité nástroje a technologie **4**
2.1 Automatický obchodní systém 4
2.2 Makléři 5
2.3 NinjaTrader 5
2.3.1 Obchodní pokyny 7
2.3.2 Market Data 8
2.3.3 NinjaScript 8
2.3.4 Indikátory 10
2.3.5 Strategie 10
2.4 MetaTrader 11
2.4.1 Jazyk MQL 11
2.5 Síťová komunikace 12
2.5.1 Protokol WebSocket 12
2.5.2 JSON 13
2.6 Vývojové prostředí a databáze 13

3 Návrh systému **15**
3.1 Analýza zadání 15
3.2 Možné přístupy k návrhu 16
3.3 Databázové schéma 17
3.3.1 Normální formy 19
3.4 Prvky systému 20

4 Implementace **23**
4.1 Serverová aplikace 23
4.1.1 Zobrazení připojených klientů 24
4.1.2 Zpracování objednávek a obchodních dat 24
4.1.3 GUI a jeho funkce 25
4.2 Strategie pro NinjaTrader 27
4.3 Strategie pro MetaTrader 28

5 Testování **30**

6 Závěr **32**

Literatura **33**

Přílohy **34**

A Snímky aplikace **35**

B Obsah DVD **37**

C Užitečné pojmy a zkratky **38**

Kapitola 1

Úvod

Cílem této bakalářské práce je navrhnout a implementovat systém, který bude umožňovat efektivní uložení dat reprezentujících obchodní pokyny vydané klientem, připojeným do finanční sítě. Systém byl rozšířen o evidenci obchodních dat. Architektura systému je klient-server.

Klienta představuje program s grafickým rozhraním pro stolní počítače. Takovýto software je běžně používán pro uskutečňování finančních obchodů s makléřem, což je zprostředkovatel obchodů na burze. Příkladem jsou programy NinjaTrader a MetaTrader, které mimo uskutečňování obchodů dovolují trhy analyzovat a testovat obchodní strategie na historických datech (backtesting).

Právě data popisující vývoj trhu hrají stěžejní roli při jeho analýze. Při procházení historických dat v NinjaTraderu se zobrazují již pouze zhuštěné informace, což pro vývoj kvalitní obchodní strategie nemusí stačit, protože takto se trh v reálném světě nechová. Data typu Market Replay by měla být přesnější, avšak je možné je získávat pouze zpětně za dané období.

Naproti tomu implementovaný systém, coby výsledek této práce, eviduje data a objednávky v reálném čase, přesně tak, jak je klient obdrží, a tak pokládá základ pro budoucí zpracování dat na serveru.

Evidence objednávek na serveru dá obchodníkovi přehled o stavu jeho investic, pokud se rozhodne své záměry rozprostřít mezi více makléřů, například z důvodu zachování obchodního tajemství strategie. Systém umí objednávky a obchodní data přehledně zobrazit, řadit a filtrovat dle zadaných kritérií. Dále umožňuje export dat do souboru pro Microsoft Excel a zaslat konkrétním připojeným klientům příkaz k provedení objednávky či k vykreslení symbolu do grafu.

Výsledný systém se skládá ze zásuvných modulů (strategií) pro NinjaTrader, MetaTrader, z knihovny zapouzdřující komunikaci pro MetaTrader a ze serverové aplikace přistupující do databáze Microsoft SQL Express LocalDB. Z důvodu podpory platformy oběma klienty byl zvolen vývoj na Windows, což vyústilo ve WPF (Windows Presentation Foundation) aplikaci, napsanou v jazyce C# v Metro stylu. Práce by mohla být dále využita jako základ pro tvorbu serverové obchodní strategie s využitím evidovaných dat.

V kapitole 2 se čtenář seznámí se základy nutnými pro pochopení problematiky a s technologiemi využitými při realizaci práce. V kapitole 3 je rozebrán architektonický návrh systému. Dále pak v kapitole 4 autor rozebírá implementační detaily. V poslední kapitole 5 jsou vyhodnoceny implementované celky.

Kapitola 2

Použité nástroje a technologie

V této kapitole jsou popsány pojmy týkající se světa financí, software pro stolní počítače pro uskutečňování online obchodů na burze a také protokol *WebSocket* a formát *JSON*, které byly využity při realizaci systému.

2.1 Automatický obchodní systém

AOS je počítačový program, který vytváří objednávky a odesílá je na burzu. Jedná se o elektronické obchodování. Nejzákladnějšími parametry objednávky jsou: jméno obchodovaného instrumentu, typ pokynu (prodej, či nákup) a množství. V minulosti obchodování prováděli obchodníci bez využití informačních technologií, avšak v současné době dosahuje podíl AOSů ve FOREXu až 80% objemu transakcí [8].

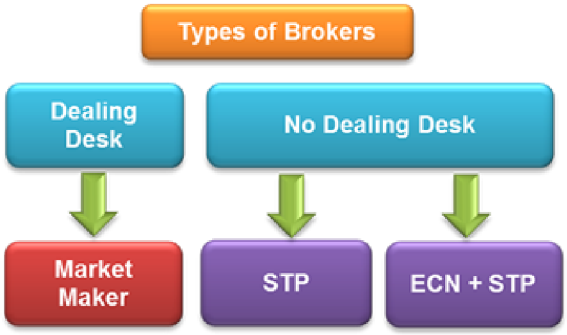
Automatické obchodní systémy jsou řádově rychlejší než člověk, navíc nemají emoce, které by mohly negativně ovlivnit jejich rozhodování. Jejich obchodování je založeno na nějaké obchodní strategii. Tato strategie může fungovat na principu fundamentální nebo technické analýzy trhu [9]. Technická analýza spočívá v předpovědi budoucího vývoje trhu na základě předchozích dat, zejména ceny a objemu (kolik cenných papírů se obchodovalo v daném časovém období). Příkladem může být strategie založená na indikátoru RSI (Relative Strength Index) nebo Stochastic. Naproti tomu fundamentální analýza je zaměřena na správném odhadu vnitřní ceny akcie pomocí informací, které jsou přístupné veřejnosti. Fundamentální analýza samotná je známá dlouhou dobu (prvotní publikace na toto téma od Benjamina Grahama již od roku 1934), ale jako součást strategií je relativně nová, protože je většinou těžké kvantifikovat informace z ní plynoucí, nezbytné k učinění rozhodnutí.

HFT

V dnešní době také stále častěji setkáváme s vysokofrekvenčním obchodováním (angl. high-frequency trading). HFT není lehké definovat, avšak míní se jím obchodování charakterizované uskutečňováním velkého množství objednávek za krátký časový úsek (v řádu mikrosekund). Využívá se složitých algoritmů k analýze více trhů současně a objednávky jsou zadávány po splnění daných podmínek.

2.2 Makléři

Makléřem (angl. broker) je míněna společnost registrovaná v obchodním rejstříku, umožňující provádět obchodování na kapitálovém trhu.



Obrázek 2.1: Typy FOREXových brokerů Zdroj: [1]

Brokery můžeme rozdělit na dva hlavní typy: *Dealing Desk* a *No Dealing Desk*. *Dealing Desk* je synonemem pro *Market Makera* a *No Dealing Desk* můžeme dále dělit na *ECN* a *STP*. *No Dealing Desk* brokeři sami nevytváří protistranu obchodu, pouze spojují dvě strany, které spolu chtějí obchodovat.

- **Market Maker (MM)** - Market Maker vytváří trh pro své klienty. Určuje ceny nákupu a prodeje, poskytuje klientům likviditu a peníze vydělává pomocí *spreadu*, což je rozdíl mezi nákupní a prodejní cenou. Takovou ztrátu utrpí investor, pokud daný instrument nakoupí a ihned prodá. MM sám vytváří protistranu, tedy pokud chce obchodník prodat, MM od něj nakoupí a naopak. Co dále udělá s aktivem je na něm. Většinou nabízí fixní spread, což pro něj znamená velmi malé riziko. Nejsou zde žádné poplatky (komise) za uskutečnění obchodu.

- **Straight Through Processing (STP)** - Někteří brokeři o sobě uvádějí, že jsou typu ECN, ale ve skutečnosti se jedná pouze o STP brokery. STP broker přeposílá objednávky klientů přímo svým poskytovatelům likvidity. Od poskytovatelů likvidity získá broker nejlepší ceny pro Ask (cena pro nákup) a Bid (cena pro prodej), těmto cenám rozšíří spread o svůj výdělek. Tyto ceny poskytuje klientům. Nejčastěji poskytují variabilní spread, protože spread od poskytovatelů likvidity se rozšiřuje a totéž musí udělat STP, aby si zachoval svůj zisk.

- **Electronic Communications Network (ECN)** - Zde se setkává nabídka a poptávka různých subjektů (Market Makeři, banky, soukromníci, nebankovní instituce). Za provedení obchodu se platí brokerovi poplatek - komise. Transakce se realizují mezi dvěma obchodníky uvnitř sítě. ECN umožňuje klientovi vidět hloubku trhu (angl. Depth of Market). Hloubka trhu ukazuje objednávky, které jsou právě zpracovávány na různých tržních hladinách. Dává tak informaci o likviditě daného aktiva.

2.3 NinjaTrader

NinjaTrader je obchodní platformou umožňující provádět online obchody. Obchodovat lze akcie, futures kontrakty a cizí měny (angl. FOREX). Pro analýzy a backtestování strate-

gií je možno jej používat zdarma. NinjaTrader umožňuje vytvoření demo účtu u jedné ze spřátelených firem. V současné době se jedná o firmy *Dorman Trading*, *Phillip Capital*, *Forex.com* a *FXCM*. Za provedení obchodu ostrého (live) účtu se platí poplatek, neboli *Commission*, který se u jednotlivých makléřů liší. V případě verze *Direct*, která je zdarma je poplatek vyšší než pokud si obchodník NT koupí, či předplatí.

Funkce	**Doživotní licence**	**Pronájem**	**Direct**	**Free (Simulátor)**
Pokročilé grafy a analýza	✓	✓	✓	✓
Ostré obchodování (live trading)	✓	✓	✓	
Simulace obchodování	✓	✓	✓	✓
Automated Trading	✓	✓		✓
ATM Strategie	✓	✓		✓
Chart Trader	✓	✓		✓
OCO pokyny	✓	✓		✓
Vývoj vlastní strategie/indikátoru	✓	✓		✓
Klávesové zkratky	✓	✓		✓
Poplatek - Futures kontrakty	0.53$ / kontrakt	0.73$ / kontrakt	0.95$ / kontrakt	
Poplatek - FOREX	0.04$ za 1K FX lot	0.05$ za 1K FX lot	0.06$ za 1K FX lot	
Cena	999$, nebo 4x splátka po 299$	od 50$ měsíčně	ZDARMA	ZDARMA

Tabulka 2.1: Možnosti licencování NinjaTraderu a poplatky

NinjaTrader nabízí na rozdíl od konkurenčního MetaTraderu možnost napojit se na simulovaný zdroj dat, kdy aplikace náhodně generuje pohyby trhu. Je možné určovat jen směr - růst, či pád. Tohoto bylo hojně využíváno ve vývoji, například o víkendech, kdy se neobchoduje. Dalším typem připojení je *Playback Connection*, které přehrává záznam pohybů na trhu z minulosti. Přehrávat lze buď *Historical Data*, nebo *Market Replay*.

Veškerý vývoj byl prováděn na NinjaTraderu ve verzi 8.0.4.0 s účtem u FXCM. Verze 8 značně rozšiřuje volně poskytnuté API [7], díky tomu je možné objednávky registrovat, protože ve verzi 7 mohla strategie reagovat jen na změny stavu objednávek, které sama vytvořila.

Při zadávání objednávek se program jevil stabilně, nicméně při vývoji strategie často docházelo k pádům aplikace, případně k chybám. Jednou došlo k poškození interní databáze programu, která se dá obnovit v nastavení.

Obrázek 2.2: Graf měnového páru Forex USDJPY

2.3.1 Obchodní pokyny

Obchodní pokyny neboli objednávky jsou příkazy brokerovi k nákupu či prodeji aktiva. Objednávky mohou být v NT zadány přes více oken. *Basic Entry* je určeno k obchodování s vlastním kapitálem (angl. Equities), *SuperDOM* je určen pro futures kontrakty a okna *FX Pro* a *FX Board* slouží k obchodování FOREXu.

Následující tabulka obsahuje výčet všech typů objednávek v NT.

Typ pokynu	**Limit Price**	**Stop Price**
Market	-	-
Limit	*	-
Stop Limit	*	*
Stop Market	-	*
MIT (Market if touched)	-	*

Tabulka 2.2: Typy obchodních pokynů v NT, * označuje nutnost zadat cenu

Každá objednávka daného typu může být určena ke koupi (Buy - za cenu Ask Price), nebo k prodeji (Sell - za cenu Bid Price).

Nejjednodušším typem objednávky je typ Market, který je zároveň nejčastěji využíván. Objednávka typu Market je uskutečněna za nejlepší cenu, dosažitelnou v době zpracování. Objednávky typu Limit dává brokerovi pokyn k prodeji, či k nákupu za stanovenou limitní cenu. Pro nákup musí být limitní cena stanovená pod aktuální hladinu trhu (pod Ask), zatímco pro prodej musí být limitní cena nad trhem (nad Bidem). Objednávka s limitem se nemusí uskutečnit, ani když se trh dotkne její limitní ceny, protože takových objednávek může být víc od jiných obchodníků. Objednávky typu Stop Market slouží následujícím účelům:

- k minimalizaci ztrát na nákupní (long), nebo prodejní (short) pozici
- k ochraně výdělku na existující nákupní, nebo prodejní pozici
- k vytvoření nové nákupní, nebo prodejní pozice

Cena stop při nákupu Stop Market objednávky se umísťuje nad trh, při prodeji pod trh. Jakmile se cena limitu dotkne, objednávka se přemění na typ Market a uskuteční se za nejlepší možnou cenu. Objednávka typu Stop Limit bývá zadána se dvěma cenami. Cena stop funguje stejně jako u objednávky Stop Market. Druhou limitní cenou obchodník specifikuje, že nechce, aby byla objednávka uskutečněna přes tento limit. Pokyny typu Stop Limit by neměly být používány k uzavření pozice.

U měnových párů (FOREX) vždy jednu měnu kupujeme a druhou zároveň prodáváme. Instrument EURUSD ukazuje, kolik dolarů je potřeba ke koupi jednoho eura. Pokyn ke koupi EURUSD znamená, že kupujeme euro, a zároveň prodáváme dolar.

Zadaná objednávka postupně prochází zpracováním právě připojeným brokerem, a tak mění svůj stav (angl. state), například u typu Market objednávka (úspěšná) běžně prochází stavy: *Submitted*, *Accepted*, *Working* a *Filled*.

Arbitráž
Pokud obchodník provede nákup nástroje (angl. instrumentu) a vzápětí jej prodá na jiném trhu se ziskem, jedná se o arbitráž. Riziko je při tomto obchodování malé, ale musí se k tomu naskytnout příležitost a náklady za provedení obchodu musí být menší než případný zisk. Takto často obchodují velké instituce, například banky a fondy.

2.3.2 Market Data

Informace o pohybech trhu určitého aktiva se nazývají *Market Data*. Aktualizace přicházejí od poskytovatele připojení vždy, když dle brokera dojde ke změně. Část těchto informací vykreslují klienti do grafu (Last, případně Ask či Bid) a proto je potřeba se k jejich odběru přihlásit otevřením nového grafu pro konkrétní instrument. NinjaTrader při přijímání dat garantuje správné pořadí operací. V celé práci je pod pojmem *Market Data* míněno data první úrovně (Level 1 data). Jedná se o tyto informace:

- jméno instrumentu (např. USDJPY, GC 04-17)
- ask - poptávková cena
- bid - nabídková cena
- last - cena, za kterou proběhla poslední objednávka
- ask volume - objem nákupů zrealizovaný během časové periody
- bid volume - objem prodejů zrealizovaný během časové periody
- datum změny

2.3.3 NinjaScript

NinjaTrader lze rozšířit o indikátory, nebo strategie s pomocí NinjaScriptu. NinjaScript je jazyk odvozený od *C#*, je založen na událostech (angl. event-driven) a umožňuje přístup k pokročilým součástem celé platformy. Rozšíření lze vyvíjet buď ve vestavěném editoru, nebo ve *Visual Studiu*. Autorovi plně postačila první možnost. Zdrojový kód je po zkompilování do DLL knihovny k dispozici ke spuštění a obvykle není nutné restartovat NinjaTrader. Výsledná knihovna se jmenuje *NinjaTrader.Custom.dll* a obsahuje všechny vyvíjené strategie a indikátory. Je umístěna v adresáři `C:\Users\[User]\Documents\NinjaTrader8\bin\Custom`.

Ve verzi 8.0.4.0 občas docházelo k problémům spojeným s linkováním, a tak na platformě běžely i strategie, které byly vypnuté. Problém s dá řešit smazáním výše zmíněné knihovny a rekompilací.

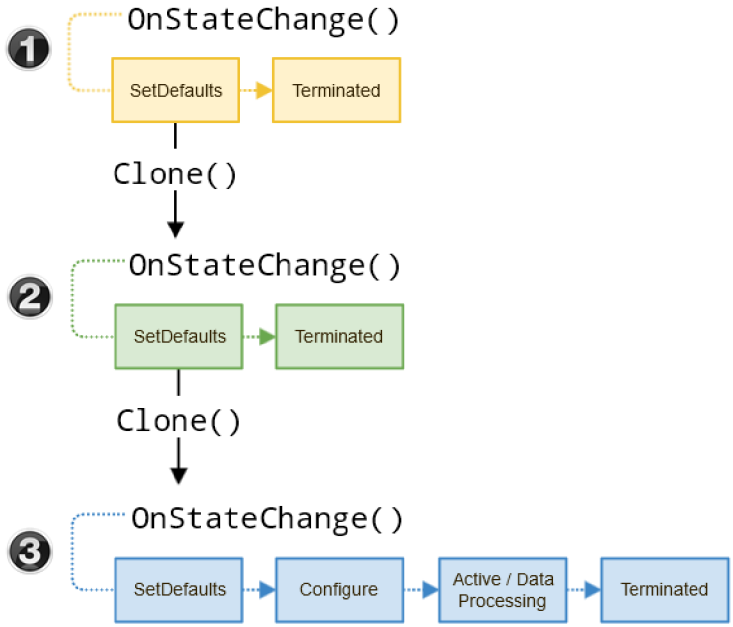
NinjaTrader je moderní aplikací podporující vykonávání kódu ve více vláknech, proto je potřeba u některých funkcí na toto myslet. Synchronizace vláken se standardně řeší pomocí dispečera (angl. dispatcher) a příkladem, kde je potřeba synchronizovat je vykreslování ikon do grafu, jinak dochází k pádu aplikace.

Při vývoji je možné využít jakoukoliv externí knihovnu napsanou v C# (.NET Framework 4.5). Stačí ji přidat jako referenci a našeptávač (technologie *IntelliSense*) již zobrazuje jmenný prostor knihovny.

Zajímavý je životní cyklus při spouštění rozšíření. Při spouštění indikátoru či strategie se nevytváří jen jedna instance. Sekvence přidání rozšíření do grafu spočívá v následujících krocích:

1. Uživatel otevře graf a pravým tlačítkem myši si otevře nabídku *Indicators*, nebo *Strategies*.
2. Ze seznamu *Available* vybere rozšíření, které chce přidat.
3. Vyplní jeho vstupní parametry a potvrdí jej pomocí tlačítka *Ok* nebo *Apply*.

Během těchto kroků se klonováním vytváří tři instance daného rozšíření. První instance vznikne ve výčtu *Available*, protože NT musí zjistit pojmenování rozšíření z kódu (z inicializačního bloku SetDefaults, viz níže). Druhá se vytváří při přidání rozšíření do výčtu *Configured* a umožňuje zadání vstupních parametrů. Nakonec po posledním kroku je třetí instance nakonfigurovaná a běžící nad grafem. V té době první dvě instance už neexistují.



Obrázek 2.3: Životní cyklus indikátoru/strategie Zdroj: [7]

Je důležité uvědomit si, že veškerý kód, který má být proveden jednorázově až po zadání vstupních parametrů a potvrzení, nestačí dát do konstruktoru, ale místo toho musí být v metodě *OnStateChange*, a to ještě pokud je aktuálním stavem stav *DataLoaded*. Jinak dochází k trojitému volání a duplikaci chování uvnitř konstruktoru (například otevírání síťových socketů).

```
protected override void OnStateChange()
{
  if (State == State.SetDefaults)
  {
    // inicializace
    Description = @"Client-side strategy.";
    Name        = @"VirtualOrderRegisterClient";
  }
  else if (State == State.DataLoaded)
  {
    // sem patri kod, ktery se provede pouze jednou pri potvrzeni
    // strategie
  }
}
```

2.3.4 Indikátory

Technický indikátor je matematická funkce, která obchodníkovi na základě cenového průběhu trhu poskytne informaci k predikci budoucího vývoje. V obchodních platformách jsou k dispozici desítky již předprogramovaných indikátorů.

2.3.5 Strategie

Strategie je podobně jako indikátor rozšířením NT, ale má větší možnosti. Je stěžejním prvkem automatického obchodního systému. Tvorba strategie začíná skrz průvodce, kde je možné definovat vlastní vstupní parametry strategie a přidat metody reagující na konkrétní události (jako *OnMarketData*). Následně je možné manuálně editovat zdrojový kód a využít při tom i volání metod, které v úvodním průvodci nejsou zobrazeny, nicméně jsou dostupné skrz API [7].

Typicky strategie vyhodnocuje aktuální pohyb trhu na základě pravidel a ve vhodných chvílích vstoupí, či vystoupí z obchodu.

V této práci je strategie využita k navázání spojení se serverem, k odesílání informací o uskutečněných obchodních pokynech, o přijatých pohybech trhu a také k zadávání objednávek na příkaz serveru.

Backtesting

K ověření úspěšnosti strategie je potřeba ji otestovat, přičemž testovat ji v ostrém provozu je nepřípustné, každá chyba by obchodníka stála peníze. Z toho důvodu se strategie testují na historických datech, ověřujeme, zda by idea stojící za konáním strategie byla zisková. Aby měl backtesting smysl, testovací vzorek dat musí obsahovat periody měnících se tržních podmínek s rostoucími i klesajícími trendy a celkově se snažit co nejvíc přiblížit reálnému trhu.

Papertrading
Jedná se o fázi vývoje, která následuje po backtestingu. Obchodník testuje strategii na ostrých datech, ale ještě ne se skutečnými penězi. Při modifikaci strategie je potřeba znovu provést nejprve backtesting a pak papertrading.

2.4 MetaTrader

MetaTrader je stejně jako NinjaTrader elektronickou obchodní platformou. Vyvíjí jej firma MetaQuotes Software, klient je dostupný pro OS Windows, Linux, macOS, iOS a Android. V současnosti je program k dispozici ve verzích 4 a 5, pro implementaci jsem zvolil verzi 5, jelikož je novější a další vývoj výrobce by se měl ubírat tímto směrem. Je nutné podotknout, že verze 4 je stále široce využívána a její uživatelé často nechtějí přejít na verzi 5 (obsahuje MQL5) z důvodu nekompatibility již vytvořených skriptů, indikátorů a strategií. MetaTrader je v praxi o něco pomalejší než NinjaTrader.

Obrázek 2.4: Platforma MetaTrader

2.4.1 Jazyk MQL

MetaQuotes Language 5 je integrovaný programovací jazyk určený pro rozšíření funkčnosti *MetaTraderu*. Syntax jazyka je odvozená od jazyka *C++*, nicméně jeho možnosti jsou značně omezené. Toto je z důvodu snahy výrobce o zajištění bezpečnosti platformy a také lze spekulovat, že rozšíření obchodní logiky mimo adresáře programu není v jeho záměrech, neboť je to poměrně složité. API jazyka MQL5 je zdokumentováno na stránkách výrobce [2].

Vývojové prostředí nezvládne přeložit obvyklé direktivy preprocesoru, jako jsou *pragma* a *ifdef* a ukazatelé fungují jinak, než v C++, což činí kompilaci rozsáhlejšího kódu, například externí knihovny do jednoho souboru s příponou *.ex5* nemyslitelné.

MetaTrader obsahuje *MetaEditor*, což je vývojové prostředí sloužící k zjednodušení vývoje a testování knihoven, indikátorů a *Strategií*.

Expert Advisor

Je ekvivalentem strategie NinjaTraderu - umožňuje automatickou analýzu a realizaci obchodních procesů. Důležité metody:

- void OnInit() - inicializační metoda, je zavolána po startu.
- void OnDeinit(const int reason) - metoda je zavolána při ukončení strategie.
- void OnTick() - metoda je zavolána, jakmile je generována událost *NewTick*. K té dochází, pokud se cena v grafu, nad kterým strategie běží změní.
- void OnTradeTransaction(const MqlTradeTransaction& trans, const MqlTradeRequest& request, const MqlTradeResult& result) - je zavolána při změně stavu účtu. Takovou změnou může být objednávka brokerovi, nebo změna stavu dosud nevyřízené objednávky.

2.5 Síťová komunikace

Strategie běžící na klientech potřebují předat data serveru, zatímco server potřebuje zaslat klientům příkazy k provedení objednávek.

2.5.1 Protokol WebSocket

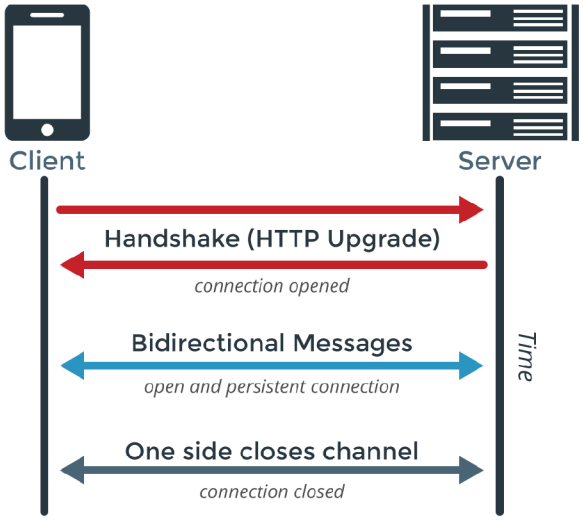
Jedná se o protokol založen na TCP, umožňující obousměrnou komunikaci mezi klientem a vzdáleným serverem. Protokol využívá implicitně porty 80 (HTTP) a 443 (HTTPS). Důvodem pro jeho vznik byla skutečnost, že webové aplikace často potřebují skutečně obousměrnou komunikaci, ale s využitím dřívějších technologií byly problémy (viz *HTTP Long Polling* [4]). WebSocket byl navržen tak, aby dobře fungoval s již existující webovou infrastrukturou. Každé spojení tohoto protokolu začíná z důvodu kompatibility jako spojení HTTP, následně proběhne *Handshake*, kterým klient signalizuje, že chce změnit (požadavkem Upgrade) protokol z HTTP na WebSocket. Spojení může být buď nezabezpečené (s URI prefixem ws://), nebo zabezpečené (s URI prefixem wss://).

Protokol je popsán standardem RFC 6455 [5] a je v současné době podporován ve všech známých prohlížečích (Chrome, Edge, Explorer, Opera, Firefox).

V této bakalářské práci bylo zapotřebí využít tři rozdílné implementace protokolu WebSocket, jednu pro server v C# (Fleck), druhou pro NinjaTrader klienta (WebSocket4Net) také v C# a třetí pro MetaTrader klienta (knihovna cpprestsdk) v jazyce C++.

Ukázka opening handshake - Upgrade požadavek (zaslaný od klienta k serveru) [5]:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```



Obrázek 2.5: Protokol WebSocket Zdroj: [3]

2.5.2 JSON

JSON je zkratka zastupující *JavaScript Object Notation* a jedná o zápis dat takovým způsobem, aby mohla být kompaktně a asynchronně přenášena sítí, například mezi webovým prohlížečem a serverem. JSON je odvozen od *JavaScriptu* a je definován RFC 7159 [6]. Obdobou JSONu je značkovací jazyk *XML*, avšak *XML* je obvykle spojeno s větší režií na přenos kvůli nutnosti uzavírání párových značek (tagů). JSON má široké uplatnění, především ve výměně krátkých strukturovaných dat. JSON může obsahovat kolekce a struktury.

JSON umožňuje přenos těchto datových typů: number, string, boolean, array, object, null.

2.6 Vývojové prostředí a databáze

Vývoj byl cílen na platformu Windows z důvodu kompatibility NinjaTraderu i MetaTraderu. Serverová aplikace byla napsána v jazyce C#, GUI definováno v XAMLu. Použitý software:

- .NET Framework 4.6.1 - běhové prostředí, implicitně nainstalované skrz aktualizace Windows.
- Visual Studio 2015 Update 3 - IDE využité při vývoji.
- Microsoft SQL Server 2014 Express LocalDB - minimalistická verze Microsoft SQL Serveru pro ukládání perzistentních dat. Instalační soubor *SqlLocalDB.msi* je dostupný na adrese:

  https://www.microsoft.com/en-us/download/details.aspx?id=42299
- NinjaScript Editor - pro vývoj strategie v NinjaTraderu.
- MetaEditor - vývoj strategie (Expert Advisora) v MetaTraderu.

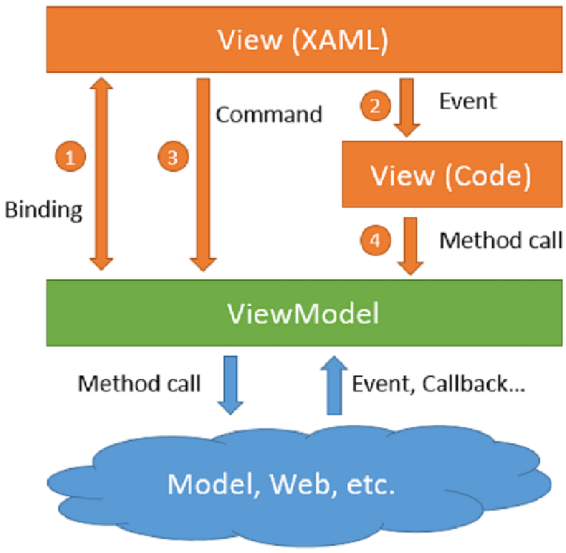
XAML

Deklarativní značkovací jazyk XAML se využívá ve WPF aplikacích, technologii Silverlight, při vývoji aplikací Windows Store a dalších. Je založen na jazyku XML. XAML definuje prvky uživatelského rozhraní a jejich provázání s daty. Vývoj a zobrazení výsledné grafické podoby je možné buď ve Visual Studiu, nebo v aplikaci Microsoft Expression Blend, která je k tomu přímo určená a umožňuje vývoj animací. Možnost využívat XAML přišla s uvedením .NET Frameworku 3.0.

MVVM

Návrhový vzor MVVM (Model-View-ViewModel) slouží k oddělení vývoje GUI od aplikační logiky a jedná se o variaci vzoru Presentation Model, jejímž autorem je Martin Fowler. Jednou z výhod je možnost souběžné práce návrháře na GUI i programátora na implementaci logiky aplikace a celkově tento přístup vede k lepší udržovatelnosti aplikace. Cílem MVVM je minimalizovat programový kód zodpovědný za vykreslování GUI, tzv. *code-behind* a místo toho deklarovat prvky GUI a jejich datové vazby.

- **Model** je implementace datového modelu s validačními pravidly, v případě této práce jsou to třídy, které se mapují do databáze s využitím objektově relačního mapování.
- **View** je uživatelským rozhraním deklarovaným v XAMLu.
- **ViewModel** je prostředník mezi vrstvami Model a View. Prvky UI ve View jsou provázány s daty ve ViewModelu - zde získávají svůj obsah a mohou do ViewModelu i zapisovat (v případě *obousměrného provázání*). Aby se zajistilo překreslení vrstvy View při změně dat ve ViewModelu, musí tato data implementovat rozhraní *INotifyPropertyChanged*. Obdobně je kolekcí u potřeba implementovat *INotifyCollectionChanged*, o což se v základu stará kolekce *ObservableCollection*.



Obrázek 2.6: Návrhový vzor MVVM

Kapitola 3

Návrh systému

Po uvedení do obchodních platforem a nezbytných technologií bude vyložen cíl práce a následně bude vypracován návrh řešení.

3.1 Analýza zadání

Cílem této práce je vytvořit systém, který bude evidovat a spravovat objednávky zadané obchodními platformami do perzistentního úložiště. Autor se rozhodl zadání rozšířit a evidovat i informace o pohybu trhů v reálném čase. Předpokládá se zobrazení objednávek a pohybů trhů v grafickém rozhraní na serveru. Implementovaná služba musí umožňovat síťovou komunikaci se vzdáleným klientem běžícím i na jiném stroji. Klienta bude potřeba navrhnout a implementovat jako rozšíření některé ze stávajících obchodních platforem. Autor se po e-mailové korespondenci s Tomášem Rozehnalem, zástupcem profesionálního vývojářského týmu společnosti FXstreet.cz a obchodníka zaměřeného na HFT, rozhodl soustředit svou pozornost na tvorbu strategie pro NinjaTrader.

Právě NinjaTrader je dle jeho doporučení v reálném obchodování výrazně rychlejší než MetaTrader, a tak se hodí pro uskutečnění klientské části správy objednávek. Server by měl být schopen iniciovat komunikaci se strategií běžící nad grafem instrumentu v NinjaTraderu a zaslat jí příkaz k provedení objednávky u stávajícího makléře. Toto by bylo upotřebitelné, pokud by vývoj této práce pokračoval a logika serveru by byla rozšířena o strategii (ve smyslu programu, který zvažuje aktuální obchodní situaci podle dat z trhů a rozhoduje o vstupu na trh, či o výstupu z něj).

Obchodník používající systém by získal tyto výhody:

- Kompletní skrytí logiky obchodní strategie před makléřem, jelikož by se nenacházela na stejném počítači, na kterém běží obchodní platforma (NinjaTrader). Vývoj fungující obchodní strategie stojí obrovské úsilí vzhledem k času a je právem střeženým tajemstvím každého úspěšného obchodníka. V této konfiguraci by server po učinění rozhodnutí obchodní strategie pouze zaslal klientovi příkaz, například nákup za aktuální cenu do daného grafu.

- Spojení dat o pohybech trhů od různých makléřů (FXCM, Interactive Brokers, ...) a z různých platforem (NinjaTrader, MetaTrader). To samé platí pro evidenci obchodních pokynů.

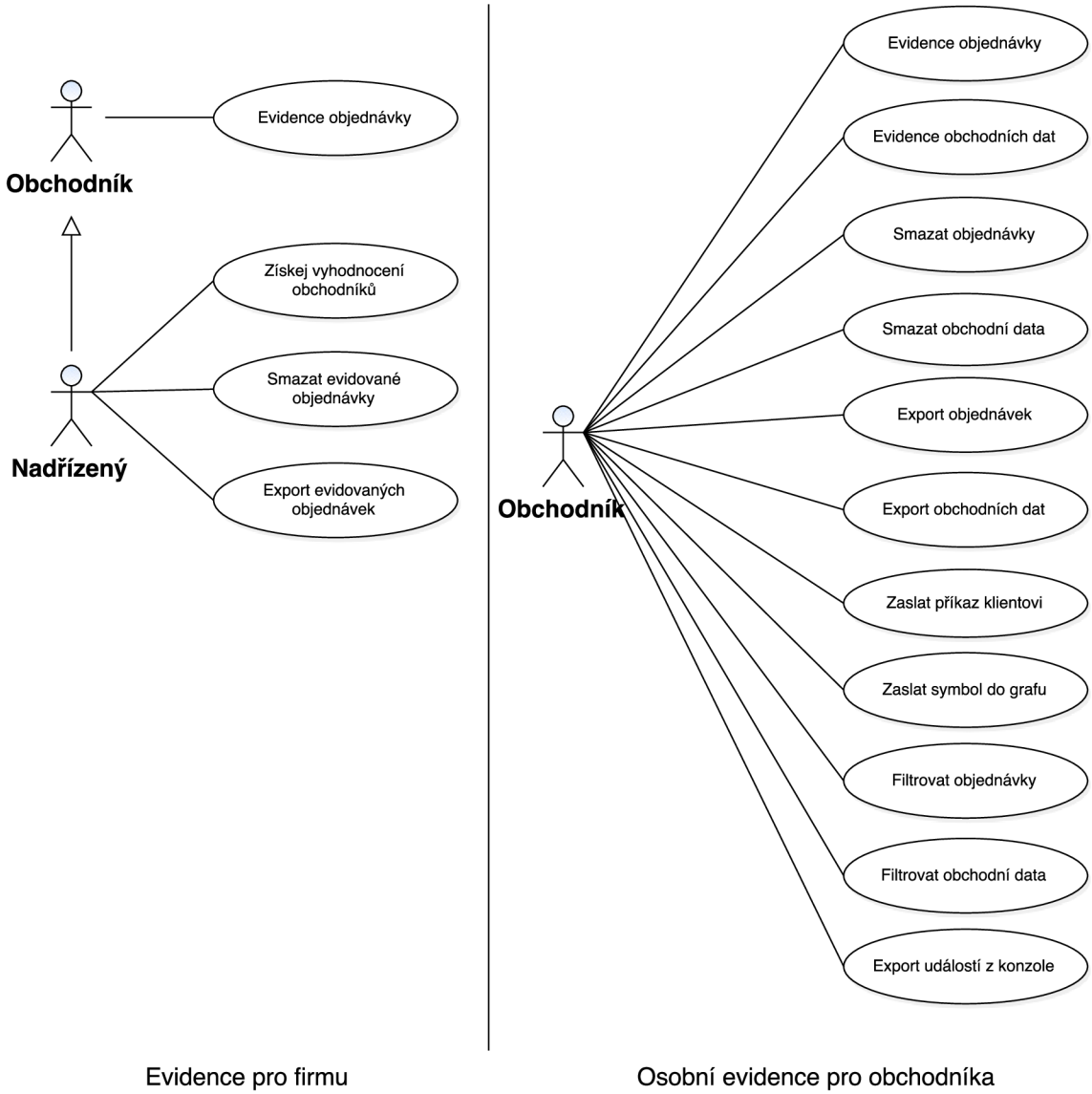
- Existence historické databáze evidovaných dat (objednávek a pohybů trhu). Úspěšní obchodníci přikládají možnosti zpětně analyzovat situaci na trhu velký význam.

3.2 Možné přístupy k návrhu

Systém pro evidenci objednávek lze pojmout nejméně dvojím způsobem. Za prvé je možné uvažovat o evidenci pro firmu zaměstnávající obchodníky používající klienty jako je NinjaTrader. Pokud by manuální obchody, které provádějí uskutečňovali na grafu s běžící klientskou strategií, server by evidoval informace o provedení každé objednávky. Nadřízený těchto obchodníků by měl díky tomu celistvý přehled o tom, jak se kterému obchodníkovi vede a mohl by serverová data dále vyhodnocovat.

Druhým možným přístupem je osobní evidence pro obchodníka, který má zájem na evidenci svých obchodů z důvodu zpětné analýzy svých rozhodnutí.

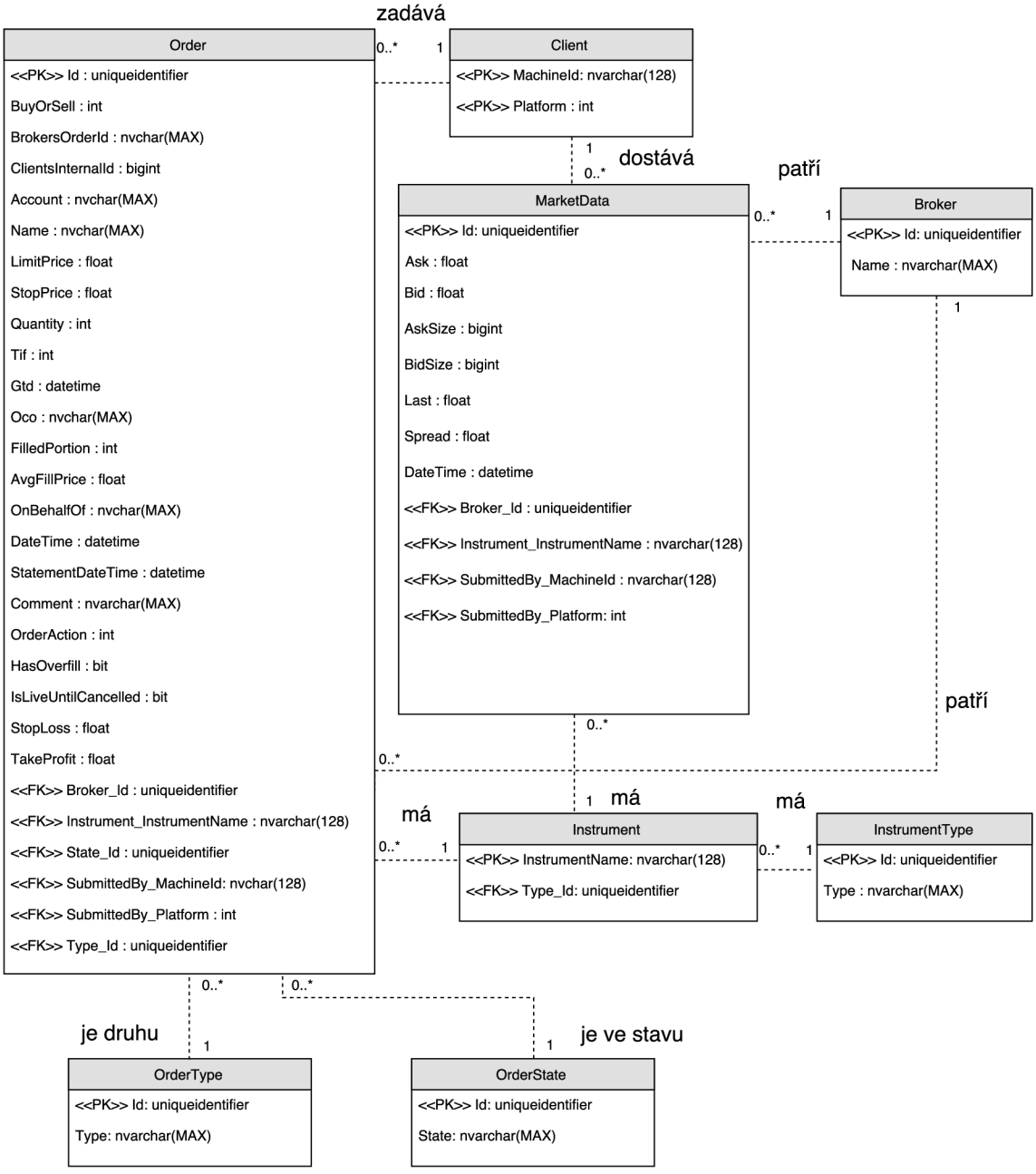
Architektonický návrh obou systémů by byl odlišný. Po konzultaci s vedoucím se autor rozhodl navrhnout a implementovat druhou možnost.



Obrázek 3.1: Diagram případů užití

3.3 Databázové schéma

Jelikož potřebuje aplikace uložit perzistentní data, je potřeba využít databázi.



Obrázek 3.2: Databázové schéma

Výše uvedené schéma je ER diagramem, rozšířeným o datové typy a informace o klíčových atributech. ERD modeluje entity aplikační domény a statické vztahy mezi nimi. Vztahové množiny mezi entitami obsahují kromě svého jména i kardinalitu, což je maximální počet vztahů daného typu, ve kterých se může jedna entita participovat.

Důležitou entitou je `Client`. Představují jej dva atributy souhrnně tvořící složený primární klíč. Atribut `Platform` specifikuje typ klientské platformy. Může se jednat o NinjaTrader, MetaTrader, případně o další v budoucnu přidanou platformu. Tato informace také umožňuje odlišit zdroj obchodních dat v případě stejného instrumentu a brokera (makléře). To se může hodit v případě, že na počítači běží NT i MT souběžně (obchodní data i objednávky by se jinak zamíchaly). `MachineId` zastupuje jméno počítače, které obchodník zadá do své klientské strategie. Nechává na uživateli, zda se evidované objednávky zařadí do stejné kategorie, pod stejné `MachineId`, což se může hodit v případě výpadku klientského počítače a potřeby rychle jej nahradit. Jediné, co by bylo potřeba zajistit je programatický posun (offset) čítače provedených objednávek, protože objednávka (entita `Order`) je jednoznačně identifikována těmito atributy:

- MachineId
- Platform
- ClientsInternalId (v entitě Order)

přičemž `ClientsInternalId` je interní čítač, který je při každé nové objednávce inkrementován (v NT začíná na od čísla jedna).

Tyto tři atributy slouží jako vodítko, když objednávka mění svůj stav (na který odkazuje cizí klíč do entity `OrderState`. Jinými slovy, serverová aplikace musí vědět, které konkrétní objednávce má změnit stav (jedná se o aktualizační sémantiku). K jednoznačné identifikaci by teoreticky mohl být použit i atribut `BrokersOrderId`, avšak ten je v režii makléře a často se mění bez toho, aby to mohla běžící strategie poznat (viz [7] - Order). Pro identifikaci či jako primární klíč je tedy nevyhovující. Pokud by došlo k výpadku klienta, musel by uživatel ručně zeditovat strategii a na místě, kde se odesílá `ClientsInternalId` na server, přičíst nejvyšší číslo z evidovaných objednávek na serveru, čímž by se vyloučily konflikty.

Entita objednávka (`Order`) obsahuje všechny informace dostupné NinjaTraderem a je rozšířena o atributy `StopLoss` a `TakeProfit`, které využívá pouze MetaTrader. Systém zatím nedokáže evidovat objednávky z MT, protože ten používá zcela jiný životní cyklus při zpracování, ale databázové schéma je na to připraveno. Atribut `Tif` (Time in Force) specifikuje režim při zpracování objednávky - určuje jak dlouho se bude objednávka snažit o zpracování. V serverové aplikaci je definována výčtovým typem (enumerací) v kódu a nejčastějšími hodnotami jsou *Day*, *Gtc* a *Gtd*

- Day - objednávka se bude snažit o provedení do konce obchodního dne.
- Gtc (Good-til-Cancelled) - objednávka funguje, dokud není provedena, nebo zrušena uživatelem, či dle podmínek makléře.
- Gtd (Good-til-Date) - objednávka se snaží o provedení do konce zadaného obchodního dne.

`OrderState` obsahuje výčet všech různých stavů, které mohou objednávky z NT a MT nabývat, a při prvním startu programu je jako tabulka předvyplněna. Je dobré držet `OrderState` jako samostatnou tabulku, protože kdyby bylo potřeba nějaký stav přejmenovat, například z důvodu vydání nové verze klientské platformy, tak to lze provést na jednom místě.

Entita `OrderType` definuje šest typů objednávek, je vyčleněna podobně jako stav objednávky z důvodu rozšiřitelnosti. Tyto typy ovlivňují životní cyklus objednávky, typ *Market* je uskutečněn ihned, zatímco *Limit* čeká na dosažení ceny.

Pro evidenci obchodních dat slouží entita `MarketData`. Protože neexistuje kombinace atributů, která by zajistila jednoznačnou identifikaci entity, je zvolen dodatečný primární klíč `Id`. `MarketData` udržuje informace, které klienti dostávají od svého makléře (ceny `Ask` a `Bid`, velikosti nabídky a poptávky, cenu poslední transakce `Last` a `Spread`).

Objednávka i obchodní data jsou propojena s entitami `Instrument`, `InstrumentType` a `Broker`. `Instrument` definuje instrument, tedy aktivum a je jednoznačně určený řetězcem `InstrumentName`, který server obdrží od strategie. Entita `InstrumentType` klasifikuje Instrument do kategorie (Stock, Forex, Future, Option, Index, Unknown). Podobně entita `Broker` slouží k evidenci makléře specifikovaného jeho jménem.

3.3.1 Normální formy

Kvalitní databázové schéma v relačním modelu dat splňuje normální formy, které definují požadavky na vlastnosti schématu relace z pohledu závislostí mezi atributy. Transformace do vyšších forem probíhá skrz postupnou dekompozici. Normalizace zachovává závislosti a zabraňuje redundanci dat. Normální formy mají hierarchický charakter, takže n-tá normální forma splňuje podmínky pro všechny předchozí (nižší) normální formy a ještě něco navíc. V praxi se obvykle schéma normalizuje do třetí normální formy, nebo do BCNF.

- **Nultá normální forma (0NF)** - Relace se nachází v 0NF, když alespoň jeden atribut obsahuje více než jednu hodnotu [10].
- **První normální forma (1NF)** - Relace se nachází v 1NF, právě když všechny její jednoduché domény obsahují pouze atomické hodnoty[10].
- **Druhá normální forma (2NF)** - Relace se nachází v 2NF, právě když je v 1NF a každý její neklíčový atribut je plně funkčně závislý na každém kandidátním klíči[10].
- **Třetí normální forma (3NF)** - Relace se nachází v 3NF, právě když je ve 2NF a neexistuje žádný neklíčový atribut, který je tranzitivně závislý na některém kandidátním klíči[10].
- **Boyce-Coddova forma (BCNF)** - Relace se nachází v BCNF, jestliže pro každou netriviální funkční závislost $X \rightarrow Y$ je X superklíčem[10].

Po transformaci ERD 3.2 na schéma relační databáze je potřeba provést normalizaci. Normalizované schéma je téměř totožné s ERD (neobsahuje jména vztahových množin). Jediným kandidátním klíčem v tabulce `Order` je umělý (angl. surrogate) atribut `Id`, protože dvojice `(BrokersOrderId, Broker_Id)` není kandidátním klíčem (`BrokersOrderId` může být změněn, tedy není jednoznačným identifikátorem) a trojice (`ClientsInternalId, Submitted_By_MachineId, Submitted_By_Platform`) také nelze použít, protože více počítačů se stejným jménem a platformou mohou mít stejný čítač `ClientsInternalId`. Výsledné databázové schéma se nalézá v BCNF.

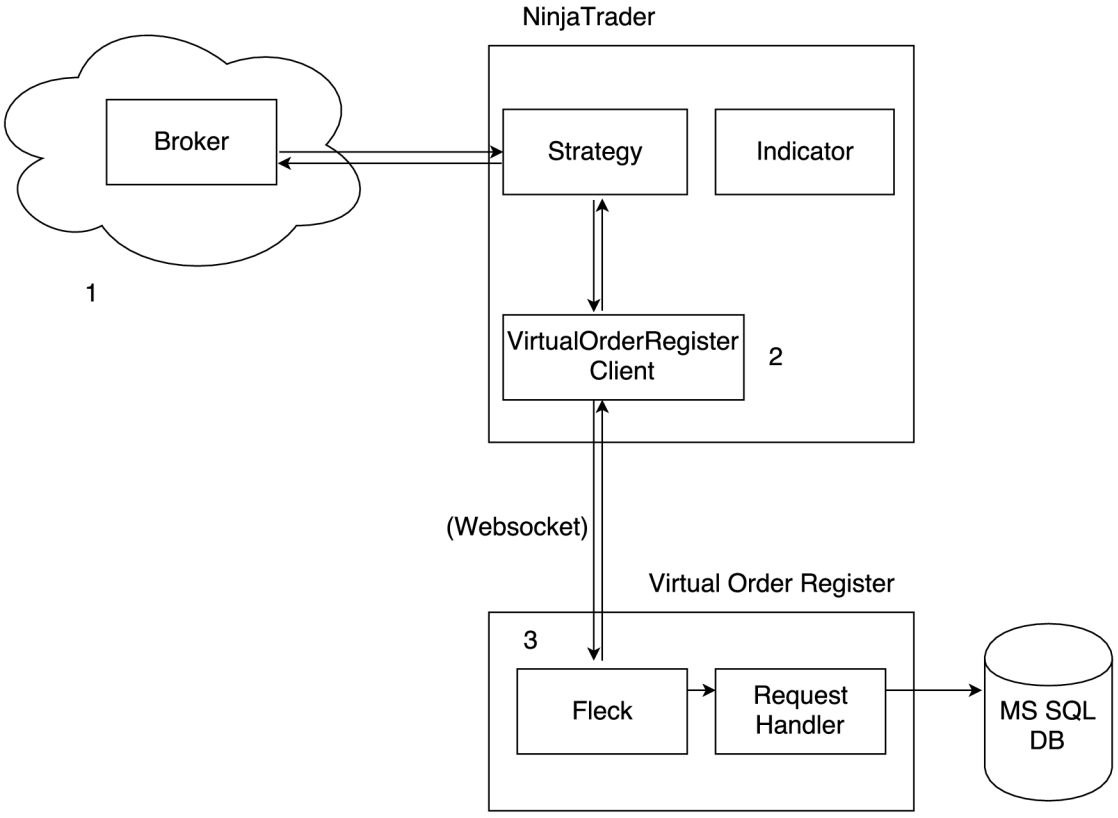
3.4 Prvky systému

Autor se rozhodl dekomponovat systém do následujících částí:

- Rozšíření klientů
  (a) Strategie pro NinjaTrader
  (b) Strategie pro MetaTrader
- Serverová aplikace

Nejsložitějším prvkem bude *serverová aplikace*, která musí data přijímat, zpracovávat i odesílat a přitom zvládat vykreslovat uživatelské rozhraní a umožňovat filtrování a exporty dat.

Strategie pro NinjaTrader
Strategie běží nad grafem určitého instrumentu. Před spuštěním by měla přijmout vstupní parametry. Jedná se o *adresu serveru* a *port*, na kterém běží aplikace s evidencí a také řetězec se *jménem klienta*, který poslouží jako MachineId. Pro adresu a port můžeme použít jednu proměnnou typu řetězec, kterou nazveme `ServerAdressPort`. Uživatel zadá IP adresu a za ní dvojtečku a potom cílový port. Server bude implicitně naslouchat na portu 80. V případě, že uživatel zadá IPv6 adresu, je potřeba zabalit ji do hranatých závorek. Po zadání vstupů a spuštění vyhodnocuje strategie události skrz API NinjaTraderu.



Obrázek 3.3: Diagram datových toků

Obrázek 3.3 popisuje tok dat v systému. Veškerá obchodní data pochází z trhu (1). Makléř (broker) má k dispozici podmnožinu tohoto trhu, který monitoruje a posílá informace o pohybech ceny instrumentů NinjaTraderu. V rámci NinjaTraderu běží uživatelské rozšíření (strategie) pojmenované *VirtualOrderRegisterClient* (2).

Jakmile strategie detekuje příchozí obchodní data, serializuje je a přes protokol WebSocket odešle na server k evidenci. V případě detekování manuální objednávky zadané obchodníkem musí strategie zkontrolovat, zda se shoduje instrument grafu, ve kterém běží s instrumentem podané objednávky. Až pokud toto platí, může odeslat informace o objednávce na server. Toto ověření je třeba k zamezení duplicitní evidence objednávky, kterou detekují dvě strategie běžící nezávisle na sobě, například pro instrument `EURUSD` a `AUDUSD`.

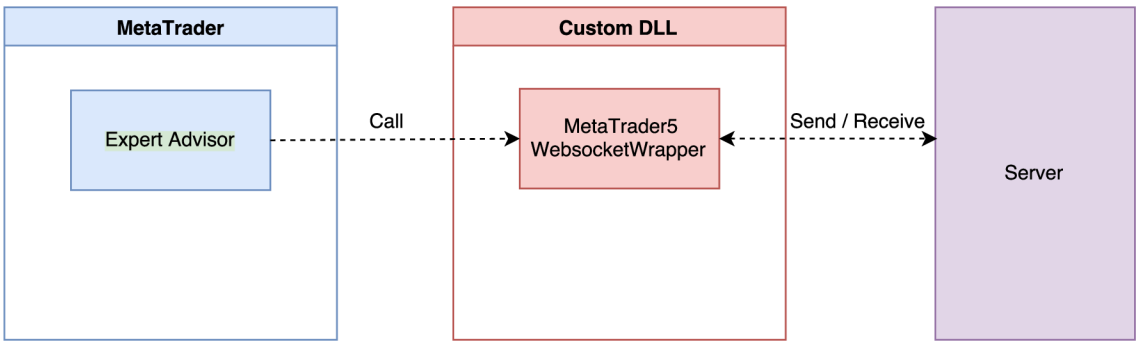
Na serveru běží v rámci .NET Frameworku běhové prostředí *Common Language Runtime*, ve kterém je spuštěna serverová aplikace. Aplikace dále naslouchá na portu 80 na všech rozhraních stroje a podporuje protokol WebSocket (3). Příchozí požadavky jsou zpracovány interní komponentou nazvanou `Request Handler`, která provede deserializaci, validaci a data uloží, nebo je aktualizuje v databázi.

V NinjaTraderu je možné pomocí funkce `Print` vypisovat ladící informace do okna nazvaného *NinjaScript Output*. Strategie do něj bude zaznamenávat potencionální chybové stavy, výjimky a také informovat, když přijme příkaz od serveru k provedení objednávky.

Strategie pro MetaTrader

Rozšíření pro MetaTrader se v mnohém podobá strategii v NinjaScriptu. Také běží nad grafem a má přístup k aplikačně programovému rozhraní pro zpracování událostí, avšak nedokáže z důvodu omezení jazyka MQL5 samostatně komunikovat se serverem. Proto autor musel vyvinout vlastní DLL knihovnu, která funguje jako prostředník mezi MQL5 a komunikačními knihovnami napsanými v jazyce C++.

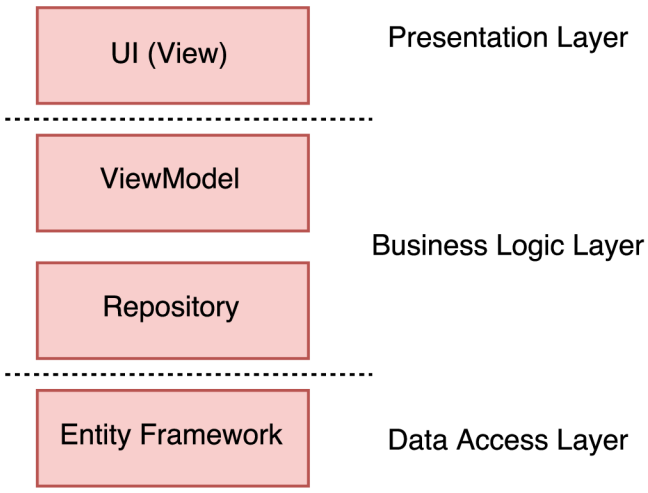
Tato autorova knihovna se importuje do strategie (Expert Advisora), přičemž je nutno definovat signatury použitých metod. Podstatné je, že strategie může volat metody knihovny, ale naopak to není možné. Problém nastává v případě, když by systém měl být rozšířen o posílání příkazů k provedení objednávky ze serveru na MetaTrader klienta. Autorova knihovna je schopná požadavek zachytit, avšak nemá žádný prostředek k notifikaci strategie k provedení pokynu skrz MetaTrader API. Řešením by mohlo být uložení požadavku do fronty v knihovně a periodické testování strategií, zda existují nějaké požadavky k provedení.



Obrázek 3.4: Rozšíření architektury pro MT

Serverová aplikace
Jak už bylo napsáno výše, cílem serverové aplikace je spolehlivě a efektivně evidovat objednávky a obchodní data, perzistentně je uložit do databáze a zobrazovat a spravovat je z GUI. V případě NinjaTraderu by měla umět zaslat pokyn k provedení objednávky, či kreslit grafiku do grafu.

Aplikace bude postavena na návrhových vzorech MVVM a Repository (pro přístup k datům). Repozitář přidává separační vrstvu mezi data a aplikační vrstvu (angl. Business Logic Layer). Nejnižší datovou vrstvu představuje Entity Framework, který umožňuje objektově-relační mapování. Tento přístup vede k lepší návrhu a udržovatelnosti kódu.



Obrázek 3.5: Jednotlivé vrstvy aplikace

Síťová komunikace
Prvky systému vyžadují obousměrnou komunikaci, kterou zajistí protokol WebSocket. V případě přenosu citlivých dat přes internet autor předpokládá využití šifrovaného VPN tunelu mezi klienty a serverem, avšak typický případ užití by měly tvořit stroje v lokální síti, kde na jednom počítači provádí obchodník obchody a druhý počítač eviduje objednávky a je schopen zpětně dokreslit stav trhu pomocí zaznamenaných obchodních dat.

Kapitola 4

Implementace

V této kapitole je popsaná implementace systému. Z fáze návrhu je zřejmé, že v implementační části je potřeba realizovat čtyři komponenty. Jedná se o strategie pro NinjaTrader a pro MetaTrader, komunikační knihovnu pro MetaTrader a nakonec o serverovou aplikaci. Při vývoji serverové aplikace byl hojně využíván verzovací systém git, dostupný v rámci Visual Studia.

4.1 Serverová aplikace

Pro implementaci serveru byl zvolen jazyk C#, který nabízí vyšší výkon oproti interpretovaným jazykům, jako je například Python, ale finální volba jazyka je spíše osobní preferencí autora. Aplikace byla dekomponována do čtyř projektů v rámci jednoho řešení (solution) Visual Studia. Těmito projekty jsou:

- **Desktop** - zde jsou umístěny deklarace všech oken (Windows) a znovupoužitelné komponenty (UserControls) v XAMLu, zdroje (ikona a obrázky) a převodníky hodnot (Converters).
- **Model** - obsahuje definice všech entit (většina jich dědí od třídy `BaseModel.cs`) a šablon pro komunikaci s klienty (ve složce `Templates`). Šablony jsou obyčejné třídy obsahující patřičné proměnné.
- **Services** - tento projekt sestává z migrace obsahující počáteční stav databáze a repozitářů spojujících kompozitní entity z více tabulek. Zajímavá je třída `DbInitializer .cs`, jejíž metoda `Seed` je volána při startu aplikace, kdy se zajistí vytvoření počáteční databáze, pokud ještě neexistuje. Tato třída předvyplňuje do databáze předem známé stavy, jako jsou typy objednávek.
- **ViewModel** - třídy, které udržují informace o stavu uživatelského rozhraní se nacházejí ve složce `ViewModels`. ViewModely dědí od bázových tříd `ViewModelCollection` a `ViewModelBase` umístěných ve složce `Framework`. Složka `Commands` obsahuje třídy, které implementují rozhraní `ICommand` skrz přepsání (override) metod bázové třídy `CommandBase`. Tyto příkazy slouží k realizaci dílčích funkcí různých ViewModelů (například filtraci, export do Excelu, odpojení připojeného klienta a další). Konečně, ve složce `Messages` třídy tvořící šablony pro posílání zpráv mezi jednotlivými ViewModely dle návrhového vzoru *Messenger*.

Dále bylo potřeba vyřešit objektově relační mapování entit (v projektu `Model`) do tabulek v databázi. K tomuto posloužil Entity Framework, který umožňuje přidat k vlastnostem entit anotace, které ovlivní vzniklou tabulku. Jedná se o přístup *Code-First*, programátor tedy vůbec nemanipuluje s databázovým schématem a ani nemusí psát SQL kód.

4.1.1 Zobrazení připojených klientů

V úvodní záložce programu, nazvané `Connected Clients` je zobrazen seznam všech připojených klientů s možností spojení uzavřít. Klient je na tento seznam přidán, pokud jsou splněny dvě podmínky - musí být otevřené spojení na server a musí serveru poslat první obchodní data. Poté je veden jako *online*, dokud není spojení uzavřeno z iniciativy serveru, nebo samotného klienta. Stavové informace o klientovi nejsou udržovány v databázi, ale ve struktuře v paměti RAM, protože udržování IP adresy v databázi neodpovídá realitě - klient může například přepnout z LAN síťové karty na Wi-fi adaptér a DHCP mu přidělí jinou IP v rámci bezdrátové sítě. Stavové informace jsou vedeny jako statická vlastnost `ConnectedClients` v souboru `MainViewModel.cs`, která je pomocí jednosměrného provázání spojena s grafickým prvkem `ItemsControl` v XAMLu v souboru `MainWindow.xaml`. Jedná se o kolekci třídy `WebsocketMapping`, která obsahuje MachineId, Platformu, Instrument, Brokera a také instanci implementující `IWebSocketConnection`, což je rozhraní websocketové knihovny *Fleck*. Tato instance zpřístupňuje zaslání zprávy klientovi, prověřování funkčnosti (ping) a pokyn k uzavření spojení. Autor chtěl původně využít oficiální implementaci Microsoftu v oboru názvů `System.Net.WebSockets`, ale to nebylo možné, protože Windows 7 neobsahují kompletní implementaci této funkce (je dostupná ve Windows 8 a Windows Serveru 2012). Proto byla zvolena open-source knihovna *Fleck*.

Online klienty z RAM lze přiřadit k objednávkám, nebo obchodním datům v databázi na základě dvojice MachineId a Platform.

Přidávání a odebírání online klientů je implementováno v souboru `RequestHandler.cs`, jedná se o metody `CheckConnection` a `RemoveWebsocket`. Metoda `CheckConnection` zkontroluje příchozí data a pokud klient, který data poslal není veden v seznamu online, tak jej tam a priori přidá. Přitom kontroluje, zda je stále otevřený socket (schránka), ze kterého byla data přijata.

Metoda `RemoveWebsocket` je zavolána po kliknutí na tlačítko *Disconnect* a pouze uzavře konkrétní schránku a odstraní záznam z mapování *Connected Clients*.

4.1.2 Zpracování objednávek a obchodních dat

Při návrhu na obrázku 3.3 jsme se seznámili se strukturou aplikace. Nyní je zapotřebí objasnit evidenci dat, která probíhá na pozadí. Při startu aplikace je instanciována třída `MainViewModel`, která obsahuje všechny ostatní ViewModely. V konstruktoru se mimo jiné vytvoří i instance `WebSocketServer`, která pochází z výše zmíněné knihovny *Fleck*. `WebSocketServer` přijímá při vytvoření řetězec ve tvaru IP adresy a portu jako parametr a rozhraní s danou IP adresou naslouchá na daném portu. Bylo použito adresy, která se interně ve *Flecku* přeloží na *IPAddress.IPv6Any*, tedy server startuje na všech dostupných IPv4 i IPv6 rozhraních.

Příchozí požadavky jsou poslány instanci třídy *RequestHandler*, která je zpracovává. Obslužný kód je deklarován jako kritická sekce v zámku `lock`. Jedná se tedy o vzájemné vyloučení. Je to tak z toho důvodu, že požadavky od klientů mohou přicházet skutečně kdykoliv, a tak by docházelo k souběhu (angl. race condition), například při vkládací (insert-if-not-exists) sémantice. Mějme dvě souběžné transakce, přičemž obě se pokusí zjistit, zda

v databázi již existuje instrument s daným jménem. Obě transakce provedou čtení a jelikož záznam neexistuje, rozhodnou se obě pro zápis, což skončí výjimkou Entity Frameworku, protože instrument je jednoznačně identifikován právě jménem (je to primární klíč).

Samotný příkaz lock nemá na výkon příliš vliv, nicméně ostatní požadavky od klientů jsou uspány a zpracovány, až na ně přijde řada. Alternativním přístupem by bylo zpracování výjimky (zahození duplicitního požadavku) a zpracování jeho zbytku; pak by nebylo potřeba zamykat vlákna.

V metodě `Handle` v souboru `RequestHandler.cs` je požadavek deserializován knihovnou *Newtonsoft.JSON*, která je rychlá a velmi populární. Je k tomu využita šablonová třída `WrapperTemplate`, která definuje všechny proměnné, které klienti zasílají. Klient pro NinjaTrader obsahuje ve svém kódu identickou šablonu, klient pro MetaTrader šablonu neobsahuje, ale dodržuje ji. Po zkontrolování spojení se vytváří nová instance `DbContextManager`, která je odvozená od DbContextu, což je třída určená pro interakci s datovou vrstvou. `DbContextManager` je rozšířen o deklaraci tabulek podle přístupu *Code-first* a obsahuje *Connection String*, který definuje jméno databáze (MSSQLLocalDB) a jméno katalogu (VirtualServerRegister), kde budou uložena veškerá data aplikace.

Po deserializaci se zpracování větví v závislosti na tom, zda se jedná o příchozí obchodní data, nebo objednávku. Pro všechny dílčí entity (Client, Broker, Instrument, InstrumentType) platí, že jsou vyhledány v databázi a pokud neexistují, jsou přidány. Zároveň se zasílá informace o přidání do ViewModelů, k čemuž je potřeba využít návrhový vzor *Event aggregator*, jehož autorem je Martin Fowler (více v odstavci 4.1.3). Nakonec jsou nová obchodní data a objednávky přidány do databáze. Pokud objednávka s trojicí vlastností (MachineId, Platform a ClientsInternalId) již existuje, došlo k její změně a je potřeba ji aktualizovat.

4.1.3 GUI a jeho funkce

Tato část pojednává o implementaci grafického uživatelského rozhraní a s ním souvisejících funkčních prvků.

MahApps

Již při návrhu se chtěl autor odchýlit od tradičního vzhledu Windows WPF aplikací a využít NuGet balík *MahApps* k vytvoření dojmu *Modern UI* rozhraní. Po přidání do projektu `Desktop` je potřeba přidat XAMLové zdroje *MahApps* do souboru `App.xaml` a všechna okna odvodit v deklaraci vzhledu (XAML) od `controls:MetroWindow` místo od `Window`. Podobně je zapotřebí v code-behindu dědit od `MetroWindow` místo od `Window`.

Stavová ikona

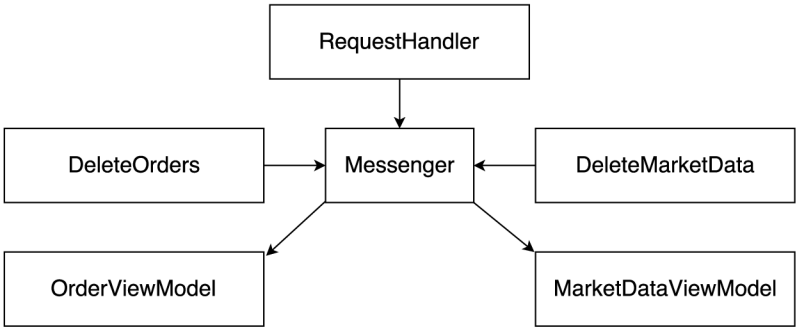
V pravém horním rohu aplikace se nalézá stavová ikona, která informuje uživatele o tom, zda právě probíhá sběr obchodních dat, či nikoliv. O tom, jestli vykreslit online, nebo offline ikonu rozhoduje `StatusConverter` a ikona samotná je tvořena XAMLovým elementem `Rectangle` s neprůhlednou maskou `OpacityMask`.

Obrázek 4.1: Indikace připojeného klienta

Obrázek 4.2: Sběr dat neprobíhá

Návrhový vzor Event aggregator/Messenger
Jelikož do aplikace neustále proudí nová data, je potřeba informovat jednotlivé ViewModely o změně (přidání, aktualizace stavu objednávky). Data jsou sice `RequestHandlerem` v pozadí uložena do databáze, avšak to nezpůsobí aktualizaci zobrazovacích komponent, která je potřeba. Přitom není příliš vhodné, aby se tyto komponenty dotazovaly databáze periodicky na všechna data, protože netrvá dlouho (v závislosti na připojených klientech a jejich tickům) a jen obchodní data samotná velice rychle překročí hranici deset tisíc záznamů v databázi.

K řešení tohoto problému slouží návrhový vzor *Event aggregator*, jež je implementován v sadě nástrojů *MvvmLight*. Aktualizace je řízena událostmi (zprávami), takže změny se projeví ihned. ViewModely ve svých konstruktorech zavolají metodu `void RegisterMessages`, která zaregistruje odběr zpráv daného typu pomocí metody `Messenger.Default.Register`. Zasláním zprávy je možné kdekoliv v aplikaci efektivně donutit ViewModely změnit svůj obsah, bez toho, aby prováděly hledání v databázi.



Obrázek 4.3: Komunikace jednotlivých ViewModelů skrz Messenger

Třídy `DeleteOrders` a `DeleteMarketData` slouží k smazání záznamů, buďto objednávek, nebo obchodních dat. Informaci o smazání je také potřeba propagovat patřičnému ViewModelu. Tyto dvě třídy využívají návrhový vzor *Command*, který byl využit všude, kde stisk tlačítka vede na provedení určité akce.

Filtrace
Při editaci filtrovacího dialogu se hodnoty jednotlivých prvků propisují do vlastností s prefixem "Filter" konkrétního ViewModelu. Až po stisknutí tlačítka *Apply!* dochází ke spuštění Commandu, který do pohledu `ICollectionView` na kolekci `Items` přidá filtrovací lambda výraz s kritérii.

Mazání záznamů
Mazání všech objednávek či obchodních dat nemůže probíhat, pokud jsou připojeni aktivní klienti, protože by se mohlo stát, že by databáze byla smazána během pokynu k aktualizaci

objednávky. Při mazání se využívá přímý SQL příkaz, protože ostatní přístupy (RemoveRange) vedou na procházení *foreach* cyklem pro všechny entity v tabulce a to je velice pomalé.

Export do Excelu
Při exportování dat do souboru čitelného pro Microsoft Excel byla využita open-source knihovna `EPPlus`. Autor předpokládá, že na serveru, kde aplikace poběží nemusí být dostupný MS Excel, a proto není možné použít vestavěnou knihovnu `Microsoft.Office.Interop.Excel`. Pro výběr umístění souboru se využívá dialog `Win32` API. Knihovna `EPPlus` si jako zdroj dokáže vzít přímo zobrazovací komponentu DataGrid, nicméně díky určování typu každé buňky trvá export velice dlouho (i více než 30 minut). Autor se rozhodl manuálně specifikovat všechny sloupce a jejich záhlaví. Tento přístup exportuje i desetitisíce buněk v řádu několika sekund.

Výsuvná nabídka a pokyny klientům
V záložce Connected Clients je možné po kliknutí na tlačítko se symbolem "plus" otevřít překrývací nabídku pro odeslání příkazů k objednávce více klientům naráz. K získání tohoto efektu bylo využito komponenty *Flyout*, do které bylo vnořena okno UserControl `SendCommandToClientView.xaml`.

Odeslání příkazu ke klientovi je triviální, v cyklu foreach se projdou všichni připojení klienti a pokud je u daného klienta zaškrtnut vybírací CheckBox, tak je sestavena šablonová třída `CommandToClient`, která je serializována a poslána prostřednictvím WebSocketové instance klientovi.

4.2 Strategie pro NinjaTrader

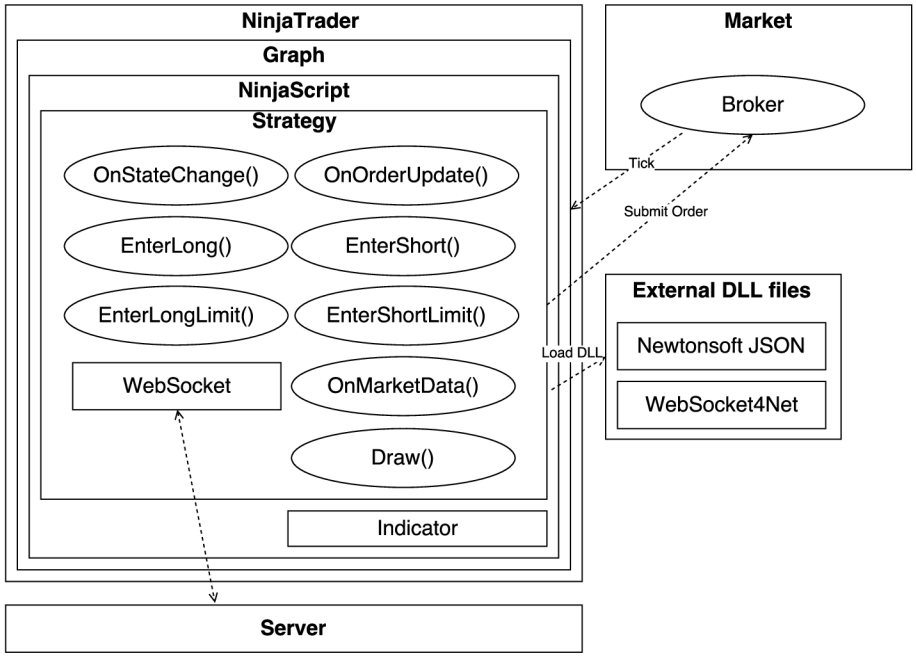
Strategie se řídí životním cyklem popsaným v sekci 2.3, kde je popsána i inicializace. Je potřeba nastavit parametr `Calculate` na hodnotu `OnEachTick`, což způsobí, že pokaždé, když přijde tick od makléře, je zavolána funkce `OnBarUpdate`. Pokud by graf nebyl přepočítáván při každém ticku, nebylo by možné zadávat příkazy od serveru dokud graf nevykreslí alespoň jeden sloupec (doba trvání je ovlivněna periodou grafu). V inicializační fázi je také potřeba zaregistrovat se k odběru změny stavu objednávky na současném účtu (metoda `OnOrderUpdate`). Dále je potřeba vytvořit instanci klienta pro protokol WebSocket. Autor zvolil knihovnu *WebSocket4Net*. Knihovna otevře nezašifrované spojení na server dle zadané adresy a portu, pokud na cílovém stroji neběží serverová služba, tak je strategie ukončena.

Obchodní data
Pro odběr obchodních dat je využita metoda `OnMarketData`, kde se čeká na změnu ceny Last, protože obchodní data je potřeba vzorkovat konzistentně a cena Last se změní vždy, když někdo nakoupí, nebo prodá daný instrument. Šablona `MarketDataTemplate` je naplněna daty, serializována a odeslána.

Objednávky
Evidence objednávek se nalézá v metodě `OnOrderUpdate` a je analogická k odběru obchodních dat, jen je potřeba poznamenat, že `OnOrderUpdate` je napojena na celý uživatelský účet, tudíž se musí ověřit, že nedostáváme data od objednávek provedených na jiném grafu, což by způsobovalo duplikování dat.

Příkazy od serveru
Veškerá logika zpracování příkazů se nalézá v metodě `websocketMessageReceived`, kde se příkaz deserializuje z formátu JSON a následně je využita instance výčtového typu `OrderCommand`, která určí, zda se použije jedna z metod `EnterLong`, `EnterShort`, `EnterLongLimit`, nebo `EnterShortLimit`. První dvě jmenované metody slouží pro zadání objednávek typu *Market*, zatímco poslední dvě metody pro specifikaci typu *Limit*.

Druhým možným typem příkazu je vykreslení symbolu do grafu. Protože každý symbol je jednoznačně identifikován řetězcem, musí se pro každý v rámci grafu generovat nová hodnota. Tohoto je docíleno vytvořením nového globálního identifikátoru `guid`.



Obrázek 4.4: Architektura klienta

4.3 Strategie pro MetaTrader

Každá strategie nebo jakékoliv jiné rozšíření běží ve vlastním vlákně. Po přidání strategie do grafu je načtena do paměťového prostoru terminálu. Jako první dochází k inicializaci globálních proměnných, poté je zavolán konstruktor. Poté strategie čeká na události od klientského terminálu (MetaTradera).

Rozšíření MetaTraderu obsahuje import komunikační knihovny `MetaTrader5Websocket Wrapper.dll`, o které se autor zmiňuje v poslední sekci. Import je deklarace preprocesoru a musí obsahovat výčet metod. Skrz tyto metody strategie deleguje funkce, které není možné implementovat v MQL5 z důvodu omezení zmíněných v návrhu 3.4. Seznam importovaných funkcí:

- void **connect**(string addressPort)
- void **close**()
- void **sendMarketData**(string, string, double, double, double, double, string)

Jako první je zavolána metoda `OnInit`, ve které strategie pouze předá řetězec obsahující IP adresu serveru a port do komunikační knihovny. Za předpokladu, že se DLL úspěšně spojí se serverem vrátí tato funkce návratový kód `INIT_SUCCEEDED`.

Stěžejní metodou je `OnTick`, která je provedena při každém zpracování pohybu ceny grafu. Prostřednictvím importované metody `sendMarketData` jsou do DLL knihovny zaslány tyto informace: *Uživatelem specifikované jméno klienta, název instrumentu, cena Ask, Bid a Last, Spread a jméno makléře.*

Metoda `OnDeinit` zajišťuje uzavření spojení v případě ukončení strategie. Funkce `OnTradeTransaction` je zaregistrována pro odběr událostí spojených se změnou stavu objednávky, pro budoucí rozšíření.

Komunikační knihovna

Tvorba knihovny pro MetaTrader má svá specifika - musí se jednat o DLL projekt v C++ a musí splňovat konvence volání *__stdcall*, nebo *__cdecl*. Signatury metod je nutné obohatit o exportní makro `_DLLAPI`:

```
#define _DLLAPI extern "C" __declspec(dllexport)
```

Pro serializaci a připojení k serveru skrz protokol WebSocket byla vybrána knihovna *C++ REST SDK*, která podporuje komunikaci prostřednictvím moderního asynchronního C++ kódu.

Kapitola 5

Testování

Aplikace byla testována průběžně, vždy po přidání nových funkcí. Celé testování probíhalo na lokální síti s pěti stolními počítači a několika zařízeními připojenými bezdrátově. Komunikující PC měly vypnutý Firewall. Systém byl testován na operačních systémech Windows 7 a 10 v 64-bitové verzi na skutečných i virtualizovaných počítačích (Oracle VirtualBox). Aplikace bezproblémově fungovala jak v protokolu IPv4, tak i v IPv6. Evidence objednávek naslouchající na portu 80 pracovala i za síťovým přepínačem *MikroTik RouterBOARD RB260GS*. Bohužel, aplikace nebyla zpřístupněna a otestována z internetu, nicméně autor předpokládá, že po správné konfiguraci sítě včetně přesměrování portu 80 na směrovači by aplikace měla fungovat stejně, jako na síti lokální. Testování na lokální síti bylo konzultováno s vedoucím a bylo považováno za dostačující.

Serverová aplikace se nespustí, pokud na systému nejsou splněny softwarové předpoklady 2.6, například není nainstalován MS SQL Server 2014 (LocalDB).

Výkon

První start aplikace trvá déle, protože je potřeba inicializovat obvykle vypnutou službu *sqlserv.exe*. Čas narůstá s počtem záznamů v databázi, před ostrým nasazením by mohlo být vhodné uvažovat o opatření hlavních zobrazovacích komponent (DataGridy) v záložkách *Market Data* a *Orders* o mechanismus postupného načítání záznamů (lazy loading). S více než dvaceti tisíci položkami v databázi (tabulka Market Data) aplikace startovala za několik sekund, takže optimalizace by byla na místě až při případném dalším prodlužování startu. Vyšší výkon by dále mohla poskytnout plnohodnotná verze SQL Serveru, jelikož použitá verze LocalDB je primárně určena k vývoji.

Přes výše zmíněné nápady ke zlepšení se serverová aplikace jevila dostatečně stabilní a výkonná, při 6 připojených klientech různého typu zvládala obsloužit všechny příchozí i odchozí požadavky.

IPv6

Dle nástroje `netstat` lze snadno ověřit, že serverová aplikace naslouchá jak na adrese `0.0.0.0`, tak na `[::]`, na portu 80 a využívá protokol TCP v transportní vrstvě. Obě adresy jsou nesměrovatelné (angl. non-routable) a zastupují všechna rozhraní stroje v daném protokolu IPv4, či IPv6. Připojení na server pomocí IPv6 lze realizovat ze strategií NinjaTraderu, stačí připojovací IPv6 adresu umístit do hranatých závorek a přidat dvojtečku s portem. Strategie pro MetaTrader protokol IPv6 nepodporuje, pravděpodobně kvůli použité komunikační knihovně.

Strategie pro MetaTrader
Při vývoji klienta v MQL5 docházelo často k potížím, které nebylo triviální diagnostikovat, jelikož s pádem *strategie* byl vždy ukončen i MetaTrader, a tak byly jakékoliv indície vedoucí k příčině ztraceny. Jediným vodítkem byla intuice a sporadické výpisy ze zásobníku, které ovšem často nebyly zalogovány.

Potíže se však podařilo překonat a strategie spolehlivě zasílá serveru obchodní data. Lze si povšimnout, že data chodí subjektivně v jiných intervalech než od NinjaTraderu, toto je však zapříčiněno implementací MetaTraderu. Při ukončení spojení ze strany serveru může občas dojít k pádu strategie, ale nikdy nespadne MetaTrader a jelikož je spojení již ukončené, nemělo by to mít výrazný vliv na funkčnost. Omezení plynoucí z jazyka MQL5 (nemožnost použít "plnohodnotné" ukazatele a obtížné řízení toku programu při přidání externí knihovny) značně ztěžují vývoj a odladění rozšíření.

Strategie pro NinjaTrader
Spolupráce systému s NinjaTraderem probíhala během testování velmi dobře. Autor nenarazil na žádný případ selhání evidence objednávky ani obchodních dat. NinjaTrader zvládá obsloužit více souběžně probíhajících požadavku k nákupu, či kreslení do grafu od serveru.

Systém přirozeně nefungoval mimo obchodní dny, tedy například o víkendu, protože makléř neposílá klientovi žádná data a simulace také žádná negeneruje. Toto lze během testování obejít změnou *Trading Hours* grafu v menu *Data Series* na *Default 24 x 7* v kombinaci s používáním simulovaného zdroje dat, nicméně v reálném provozu není co evidovat.

Strategie vypisuje případná chybové hlášení do okna *NinjaScript Output*. V případě kritické chyby využívá navíc hlasovou notifikaci.

Při testování také nejednou došlo k vyčerpání prostředků na demo účtu v NinjaTraderu. Platforma odmítne zadat další objednávku s hláškou *Not enough excess margin*. Toto lze řešit restartováním účtu, případně úpravou finančního stropu.

Platforma	**Evidence obchodních dat**	**Evidence objednávek**	**Pokyn k objednávce od serveru**	**Kreslení do grafu**
NinjaTrader	✓	✓	✓	✓
MetaTrader	✓	-	-	-

Tabulka 5.1: Implementovaní klienti a jejich možnosti

Kapitola 6

Závěr

Cílem práce bylo navrhnout a implementovat systém pro evidenci a správu virtuálních objednávek. Pro realizaci bylo nutné seznámit se s obchodními platformami NinjaTrader a MetaTrader a možnostmi jejich rozšíření, dále pak vhodně zvolit prostředek obousměrné síťové komunikace, serializace a nakonec vyvinout grafickou aplikaci pro uložení a zobrazení výsledků. Autor shledává protokol WebSocket elegantním a spolehlivým řešením komunikace, byť vychází z webových technologií a jeho využití u aplikací pro stolní počítače zatím není příliš časté.

Kompletní řešení se skládá ze strategie pro NinjaTrader, strategie pro MetaTrader, komunikační knihovny pro MetaTrader a ze serverové WPF aplikace komunikující s databází. Aplikace vyžaduje běhové prostředí .NET Framework a je určena pro Windows. Výsledný systém dokáže objednávky spravovat, evidovat, zobrazit v reálném čase, umožňuje je řadit, filtrovat, mazat a exportovat do formátu čitelného pro tabulkový procesor Excel k dalšímu vyhodnocení. Správa objednávek využívá princip abstrakce od fyzické objednávky u klienta a spočívá v možnosti serveru zaslat klientovi příkaz k nákupu, či prodeji, a to buď s provedením ihned (typ *Market*), nebo s provedením při vhodných cenových podmínkách (typ *Limit*). Za úspěch považuji rozšíření systému oproti původnímu zadání o platformu *MetaTrader*, pro kterou systém eviduje pohyby trhu. Databázové schéma bylo navrženo s ohledem na možnost dalšího rozšíření o evidenci objednávek z MetaTraderu a mělo by je pojmout, i přesto, že se od objednávek NinjaTraderu v mnohém liší.

Z hlediska dalšího pokračování práce na tomto systému by bylo vhodné vyvinout serverovou obchodní strategii, která by ke svému rozhodování využívala evidovaná data z trhů, zadané objednávky a vyvinuté komunikační prostředky. V kapitole 2 byly zmíněny arbitrážní obchody. Realizace arbitrážních obchodů na základě dat od NinjaTraderu není příliš reálná, protože by to vyžadovalo, aby server s běžící aplikací i oba servery vybraných brokerů byly velmi blízko vedle sebe a zisk z obchodu by musel být vyšší než komise. V praxi se tohoto dosahuje metodou high-frequency trading a obchodníci si k tomu běžně platí optickou linku do bankovních institucí, důležitá je rychlost zpracování a odezva. Z toho důvodu by strategie byla pravděpodobně založená na metodách technické analýzy trhu, které nemají požadavky na extrémní rychlost provedení objednávky.

Během testování byla úspěšně ověřena komunikace v protokolech IPv4 i IPv6 (NinjaTrader) a byly objeveny slabé stránky jazyka MQL5 v MetaTraderu. Evidence obchodních dat i objednávek a jejich správa na lokální síti fungovala tak, jak byla navržena.

Literatura

[1] Forex Broker Types: Dealing Desk and No Dealing Desk. [Online; navštíveno 11. 4. 2017].
URL http://www.babypips.com/school/kindergarten/choosing-a-broker/different-types-of-brokers.html

[2] MQL5 Reference - How to use algorithmic/automated trading language for MetaTrader 5. [Online; navštíveno 11. 4. 2017].
URL https://www.mql5.com/en/docs

[3] WebSockets vs REST: Understanding the Difference. [Online; navštíveno 29. 1. 2017].
URL https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/

[4] What is HTTP Long Polling? [Online; navštíveno 29. 1. 2017].
URL https://www.pubnub.com/blog/2014-12-01-http-long-polling/

[5] RFC 6455 - The WebSocket Protocol. 2011, [Online; navštíveno 4. 3. 2017].
URL https://tools.ietf.org/html/rfc6455

[6] RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format. 2014, [Online; navštíveno 4. 3. 2017].
URL https://tools.ietf.org/html/rfc7159

[7] API Reference. [2017], [Online; navštíveno 4. 3. 2017].
URL http://ninjatrader.com/support/helpGuides/nt8/en-us

[8] Haynes, R.; Roberts, J.: Automated Trading in Futures Markets, 2015, [Online; navštíveno 4. 3. 2017].
URL http://www.cftc.gov/idc/groups/public/@economicanalysis/documents/file/oce_automatedtrading.pdf

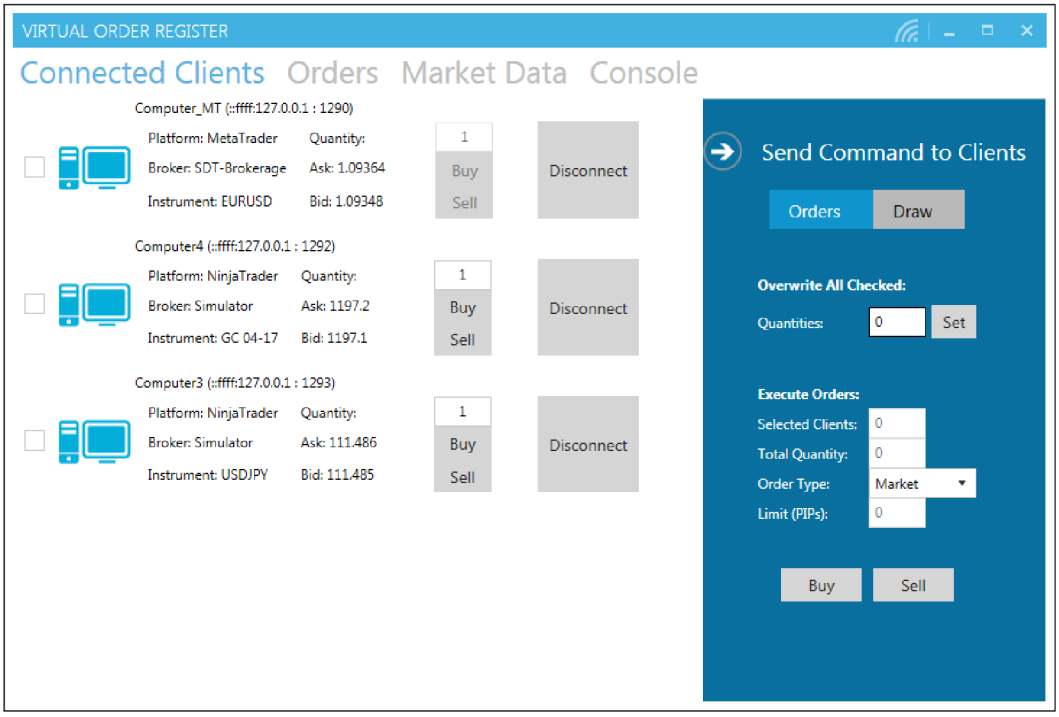
[9] Pinto, R. M. C.; Silva, J. C. M.: Strategic methods for automated trading in Forex. In *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*, Nov 2012, ISSN 2164-7143, s. 34–39, doi:10.1109/ISDA.2012.6416509.

[10] Zendulka, J.; Rudolfová, I.: Databázové systémy: Studijní opora. 2006, [Online; navštíveno 24. 4. 2017].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIDS-IT%2Ftexts%2FIDS_predn.pdf&cid=9969

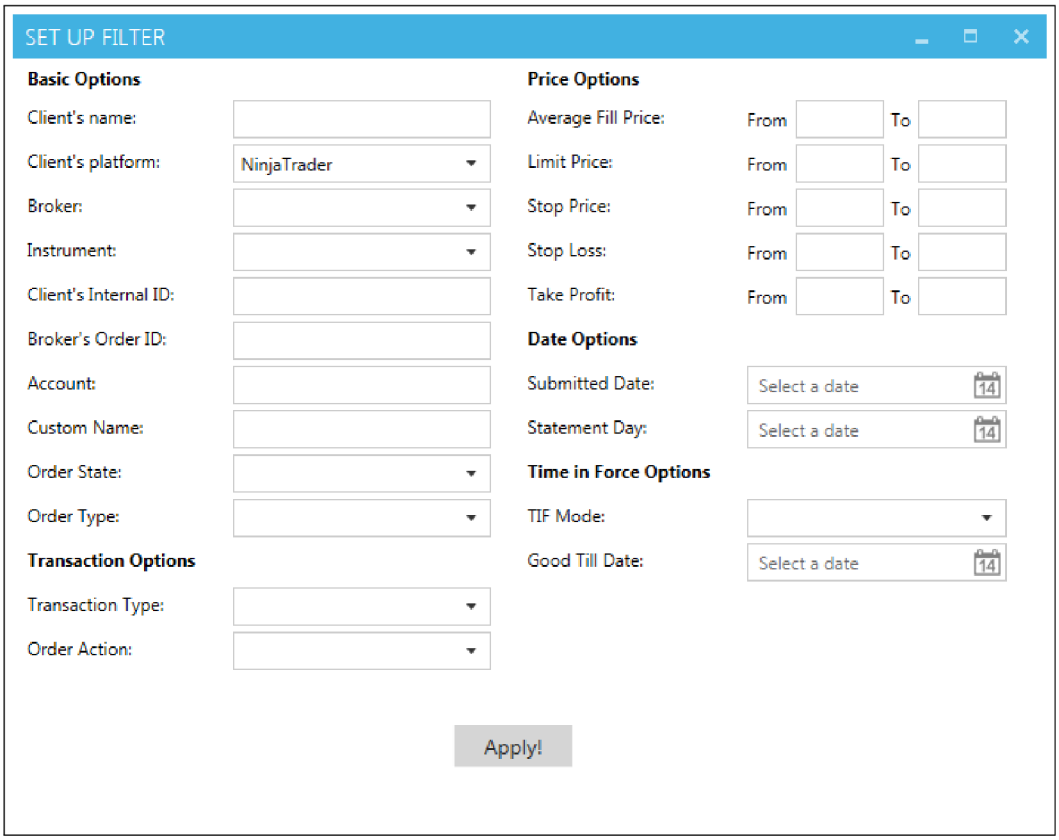
Přílohy

Příloha A

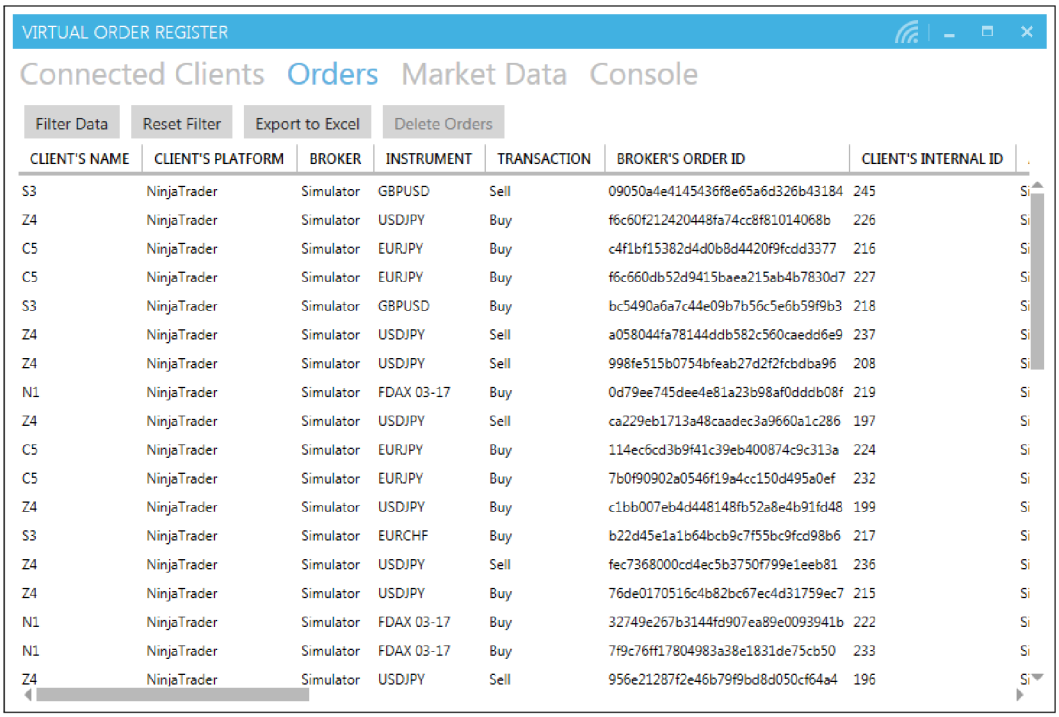
Snímky aplikace



Obrázek A.1: Přehled připojených klientů



Obrázek A.2: Filtrování objednávek



CLIENT'S NAME	CLIENT'S PLATFORM	BROKER	INSTRUMENT	TRANSACTION	BROKER'S ORDER ID	CLIENT'S INTERNAL ID
S3	NinjaTrader	Simulator	GBPUSD	Sell	09050a4e4145436f8e65a6d326b43184	245
Z4	NinjaTrader	Simulator	USDJPY	Buy	f6c60f212420448fa74cc8f81014068b	226
C5	NinjaTrader	Simulator	EURJPY	Buy	c4f1bf15382d4d0b8d4420f9fcdd3377	216
C5	NinjaTrader	Simulator	EURJPY	Buy	f6c660db52d9415baea215ab4b7830d7	227
S3	NinjaTrader	Simulator	GBPUSD	Buy	bc5490a6a7c44e09b7b56c5e6b59f9b3	218
Z4	NinjaTrader	Simulator	USDJPY	Sell	a058044fa78144ddb582c560caedd6e9	237
Z4	NinjaTrader	Simulator	USDJPY	Sell	998fe515b0754bfeab27d2f2fcbdba96	208
N1	NinjaTrader	Simulator	FDAX 03-17	Buy	0d79ee745dee4e81a23b98af0dddb08f	219
Z4	NinjaTrader	Simulator	USDJPY	Sell	ca229eb1713a48caadec3a9660a1c286	197
C5	NinjaTrader	Simulator	EURJPY	Buy	114ec6cd3b9f41c39eb400874c9c313a	224
C5	NinjaTrader	Simulator	EURJPY	Buy	7b0f90902a0546f19a4cc150d495a0ef	232
Z4	NinjaTrader	Simulator	USDJPY	Buy	c1bb007eb4d448148fb52a8e4b91fd48	199
S3	NinjaTrader	Simulator	EURCHF	Buy	b22d45e1a1b64bcb9c7f55bc9fcd98b6	217
Z4	NinjaTrader	Simulator	USDJPY	Sell	fec7368000cd4ec5b3750f799e1eeb81	236
Z4	NinjaTrader	Simulator	USDJPY	Buy	76de0170516c4b82bc67ec4d31759ec7	215
N1	NinjaTrader	Simulator	FDAX 03-17	Buy	32749e267b3144fd907ea89e0093941b	222
N1	NinjaTrader	Simulator	FDAX 03-17	Buy	7f9c76ff17804983a38e1831de75cb50	233
Z4	NinjaTrader	Simulator	USDJPY	Sell	956e21287f2e46b79f9bd8d050cf64a4	196

Obrázek A.3: Evidované objednávky

Příloha B

Obsah DVD

Na přiloženém DVD se nalézá:

- soubor `readme.txt` - pokyny k instalaci
- složka `doc` - obsahuje obrázky k popisu instalace
- soubor `setup.ps1` - skript pro zajištění závislostí klientského systému
- složka `src`
  - `NT-strategy` - zdrojové soubory strategie pro Ninjatrader
  - `MT-strategy` - zdrojové soubory strategie pro MetaTrader
  - `MetaTrader5WebsocketWrapper` - zdrojové soubory knihovny pro MetaTrader
  - `VirtualOrderRegister` - zdrojové soubory serverové aplikace
- složka `thesis`
  - `src` - zdrojové soubory LaTeX bakalářské práce
  - `pdf` - bakalářská práce

Příloha C

Užitečné pojmy a zkratky

- `AOS` - automatický obchodní systém
- `FOREX` - Foreign-exchange, mezinárodní obchodní systém pro směnu měnových párů
- `Objednávka typu MIT` - Market-if-touched, pokud cena dosáhne limitu, dojde k přeměně na objednávku typu Market
- `MQL5` - MetaQuotes Language 5, jazyk odvozený od C++, používá se pro rozšíření MetaTraderu
- `Expert Advisor` - programovatelné rozšíření MetaTraderu, ekvivalent strategie
- `HTTP` - Hypertext Transfer Protocol, internetový protokol
- `IDE` - Integrated Development Environment, vývojové prostředí
- `GUI` - grafické uživatelské rozhraní
- `WPF` - Windows Presentation Foundation, knihovna, součást .Net Frameworku pro tvorbu uživatelského rozhraní (GUI)
- `MVVM` - Model–view–viewmodel, návrhový vzor pro tvorbu GUI
- `JSON` - JavaScript Object Notation, zápis dat, vhodný pro serializaci
- `XAML` - Extensible Application Markup Language, deklarativní jazyk pro specifikaci GUI
- `ERD` - Entity-relationship diagram, entitně vztahový diagram v softwarovém inženýrství
- `NT` - NinjaTrader
- `MT` - MetaTrader
- `API` - Application Programming Interface, rozhraní pro programování aplikací
- `VPN` - virtuální privátní síť
- `UI` - uživatelské rozhraní
- `SQL` - Structured Query Language, strukturovaný dotazovací jazyk, používá se ve vztahu k relačním databázím