Politechnika Łódzka

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej

Instytut Informatyki

Maciej Dzwonnik, 157831

System ekspertowy do prognozowania rynku finansowego przy użyciu analizy technicznej

Praca inżynierska
napisana pod kierunkiem
dr inż. Krzysztofa Lichego
oraz dr inż. Jana Stolarka

Łódź 2013

Spis treści

Rozdział 1

Wstęp

Niniejsza praca dotyczy inżynierii oprogramowania i systemów ekspertowych. W wyniku pracy powstał system ekspertowy do oceny spółek giełdowych za pomocą analizy technicznej. Temat ten został podjęty ze względu na zainteresowanie analizą rynków finansowych, głównie rynku akcji, a także chęć poszerzenia wiedzy z zakresu budowy systemów ekspertowych. Realizacja systemu została wykonana z wykorzystaniem podejścia funkcyjnego do tworzenia oprogramowania.

1.1 Cele pracy

W pracy stawiam następujące cele:

- Stworzenie systemu ekspertowego oceniającego spółki giełdowe przy pomocy analizy technicznej
- Wykazanie skuteczności powstałego systemu przez analizę porównawczą wyników generowanych przez aplikację i rzeczywistych zachowań cen spółek
- Opis zastosowania programowania funkcyjnego w procesie wytwarzania systemu ekspertowego

1.2 Przegląd literatury oraz uzasadnienie wyboru tematu

Temat analizy technicznej rynków finansowych rozwija się od przeszło 100 lat i wiedza z tej dziedziny cały czas jest rozszerzana. W obecnych czasach bardzo trudno jest jednej osobie, nawet jeśli jest ekspertem, przeprowadzić rzetelnie analizę techniczną wybranego rynku finansowego bądź aktywa na tym rynku w czasie, który pozwalałby na wykorzystanie wniosków z takiej analizy do wykonania pożądanych operacji na rynku. Stąd też potrzeba posiadania narzędzia, które będzie analizować przykładowo spółki giełdowe w czasie znacząco krótszym niż zrobiłby to człowiek,

a przy tym z taką samą bądź lepszą trafnością przewidywań zachowań cen. W mojej pracy podstawowym źródłem wiedzy na temat analizy technicznej jest książka "Analiza techniczna rynków finansowych" [25] autorstwa John'a J. Murphy'ego. Dodatkowo wspierałem się przewodnikiem po analizie technicznej napisanym przez Steven'a Achelis'a pt. "Technical Analysis from A to Z" [13]. Ta pozycja literaturowa zawiera przegląd i opisy sposobu interpretacji wielu wskaźników analizy technicznej w odniesieniu do aktualnej sytuacji rynkowej. Wcześniej wspomnianym narzędziem wykonującym analizę z powodzeniem może być system ekspertowy. Taki system może za użytkownika wykonać obliczenia dla wszystkich potrzebnych wskaźników, a także wykorzystując posiadaną bazę wiedzy i reguł przeanalizować zależności pomiędzy wyliczonymi wartościami. Fundamentalną pozycją z zakresu budowy systemów ekspertowych jest pozycja "Systemy ekspertowe" [24] polskiego autora Jana Mulawki. Posiłkowałem się również dwoma źródłami anglojęzycznymi. Pierwsze z nich to książka "Building Expert Systems in Prolog" [22] autorstwa Dennis'a Merritt'a. Dzięki tej książce mogłem lepiej przeanalizować budowę systemów ekspertowych, w których wnioskowanie opiera się w oparciu o reguły. Kolejnym źródłem jest książka "Artificial Intelligence and Expert Systems for Engineers" [15] posiadająca dwóch autorów - Krishnamoorthy i Rajeev. Z tej pozycji korzystałem praktycznie wyłącznie z rozdziału dotyczącego systemów opartych o bazę wiedzy. Swój system ekspertowy do analizy technicznej postanowiłem stworzyć z wykorzystaniem funkcyjnego podejścia do programowania. Wybór taki podyktowany był chęcią poszerzenia swojej wiedzy z zakresu metodologii programowania, a także możliwością porównania wybranego podejścia z podejściem klasycznym. Do pisania aplikacji wykorzystałem jeden z dialektów języka Lisp, język Clojure, który również poddam w swojej pracy podstawowej analizie. Więdzę na temat języka Clojure czerpałem przede wszystkim z książki "Programming Clojure" [19] napisanej przez Stuart'a Halloway'a oraz Aaron'a Bedra, ale także z "The Joy of Clojure" [18] autorstwa Michael'a Fogus'a. Ponadto korzystałem z książki "Learn You a Haskell for Great Good!" [21] Miran'a Lipovaca. Książka ta opisuje podstawy czysto funkcyjnego języka programowania jakim jest Haskell. Pozwoliła mi przestawić sposób myślenia na bardziej funkcyjny podczas pisania aplikacji. Dodatkowo w bardzo prosty sposób opisuje jak w funkcyjnych językach programowania przeprowadzane są obliczenia (ewaluacje funkcji), dzięki czemu lepiej mogłem wykorzystać język Clojure podczas tworzenia pracy.

1.3 Układ pracy

Struktura dalszej części pracy jest następująca: Rozdział 2 zawiera opis teorii podzielony na trzy części: Systemy ekspertowe, Programowanie funkcyjne oraz Analiza techniczna. Rozdział 3 opisuje technologie i narzędzia wykorzystane w pracy. Rozdział 4 przedstawia opis stworzonej aplikacji, a także wyniki przeprowadzonych przy jej pomocy badań. Rozdział 5 podsumowuje uzyskane wyniki oraz płynące z nich wnioski. Dodatek A zawiera płytę CD z aplikacją stworzoną w ramach pracy.

Rozdział 2

Teoria

2.1 Systemy ekspertowe

Według Jana Mulawki:
System ekspertowy jest programem komputerowym, który wykonuje złożone zadania o dużych wymaganiach intelektualnych i robi to tak dobrze jak człowiek, będący ekspertem w tej dziedzinie.[1]
Inne ujęcie definicji systemu ekspertowego mówi, że jest to aplikacja zawierająca niealgorytmiczną wiedzę pozwalającą na rozwiązywanie konkretnego typu problemów.[2]
Biorąc pod uwagę powyższe można powiedzieć, że system ekspertowy to aplikacja przeprowadzająca proces wnioskowania na podstawie zbioru specjalistycznej wiedzy. Wnioskowanie takie wymaga oprócz wiedzy, zbioru reguł, według których proces ten będzie przeprowadzany. Można wyróżnić następujące elementy składowe systemu ekspertowego:

- Baza wiedzy - Baza specjalistycznej wiedzy (np. zbiór reguł) z dziedziny, dla której tworzony jest system ekspertowy
- Baza danych stałych - Zbiór danych na temat analizowanego problemu
- Silnik wnioskujący - Mechanizm realizujący proces wnioskowania

Ponadto, program będący systemem ekspertowym może posiadać dodatkowo następujące elementy:

- Interfejs użytkownika - Graficzny bądź tekstowy interfejs dla użytkownika korzystającego z programu
- Edytor wiedzy - Edytor pozwalający rozszerzać bazę wiedzy
- Silnik objaśniający - Mechanizm wyjaśniający proces wnioskowania. Prezentuje "tok myślenia", którym system doszedł do uzyskanych wniosków

[1] [24], str. 20
[2] [22], str. 1

Już pod koniec lat siedemdziesiątych XX wieku zauważono, że największy wpływ na efektywność działania systemu ekspertowego ma baza wiedzy. Stwierdzono, że im pełniejsza wiedza, tym szybszy jest proces uzyskiwania wniosków przez program. Stworzenie dobrego systemu ekspertowego opiera się więc w dużej mierze na dostarczeniu dla niego odpowiednio dużej bazy dobrej jakościowo wiedzy. Jest to zadanie niełatwe, ponieważ wymaga pozyskania wiedzy od eksperta (bądź grupy ekspertów), który decyzje podejmuje na podstawie informacji o problemie, ale również w oparciu o swoje doświadczenie. Dodatkowo, zebraną wiedzę należy ustrukturalizować, tak aby była możliwa do przetwarzania przez system.[3]

Systemy ekspertowe możemy podzielić ze względu na kilka kryteriów, jednak ogólnie dzielą się one na:

- Doradcze - prezentują rozwiązania dla użytkownika, który potrafi ocenić ich jakość
- Podejmujące decyzję bez kontroli człowieka - system sam podejmuje decyzję, nie poddaje rozwiązania ocenie użytkownika
- Krytykujące - system ma przedstawiony problem i jego rozwiązanie, a następnie ocenia to rozwiązanie

Możemy również podzielić je na dwie grupy:

- Systemy dedykowane - kompletny system ekspertowy posiadający bazę wiedzy
- Systemy szkieletowe - system z pustą bazą wiedzy

W drugim przypadku proces tworzenia systemu ekspertowego polega na dostarczeniu bazy wiedzy. Innym kryterium podziału jest ze względu na logikę wykorzystywaną podczas prowadzenia procesu wnioskowania:

- Z logiką dwuwartościową (Boole'a)
- Z logiką wielowartościową
- Z logiką rozmytą

Można również dokonać podziału ze względu na rodzaj przetwarzanej informacji:

- Systemy z wiedzą pewną
- Systemy z wiedzą niepewną

Najbardziej szczegółowy podział to podział ze względu na zadania realizowane przez system ekspertowy:

- Interpretacyjne - dedukują opisy sytuacji z obserwacji lub stanu czujników

[3] [24], str. 21

- Predykcyjne - wnioskują o przyszłości na podstawie danej sytuacji
- Diagnostyczne - określają wady systemu na podstawie obserwacji
- Kompletowania - konfigurują obiekty w warunkach ograniczeń
- Planowania - podejmują działania, aby osiągnąć cel
- Monitorowania - porównują obserwacje z ograniczeniami
- Sterowania - kierują zachowaniem systemu
- Poprawiania - podają sposób postępowania w przypadku złego funkcjonowania obiektu
- Naprawy - harmonogramują czynności przy dokonywaniu napraw uszkodzonych obiektów
- Instruowania - systemy doskonalenia zawodowego (np. dla studentów)

Jak już wcześniej wspomniano, najistotniejszą częścią systemu ekspertowego jest baza wiedzy. Systemy ekspertowe których baza wiedzy składa się z faktów i reguł nazywa się systemami regułowymi. Zdecydowana większość systemów ekspertowych to właśnie systemy regułowe. Fakty w bazie wiedzy są to zdania oznajmujące, opisujące jakiś obiekt bądź jego stan. Reguły natomiast są zdaniami warunkowymi. Gdy spełnione są przesłanki występujące w części warunkowej takiego zdania, wtedy do bazy faktów dodawane jest nowe zdanie.

Sam proces wnioskowania, czyli przetwarzania reguł i faktów może odbywać się na kilka sposobów. Wyróżniamy następujące metody wnioskowania:

- Wnioskowanie w przód
- Wnioskowanie wstecz
- Wnioskowanie mieszane
- Wnioskowanie rozmyte

Metoda wnioskowania w przód polega na ciągłym przetwarzaniu faktów i reguł do momentu uzyskania wśród wygenerowanych faktów postawionego celu. Taki sposób wnioskowania wiąże się z powiększaniem się bazy wiedzy. Dzięki temu proces jest szybszy, jednak w szczególnych sytuacjach może dojść do całkowitego zapełnienia pamięci nowymi faktami.

Wnioskowanie wstecz, inaczej nazywane regresywnym, odbywa się w sposób odwrotny do wnioskowania w przód. Polega ono na potwierdzeniu głównej hipotezy sprawdzając czy przesłanki dla niej są prawdziwe. Jeśli nie wiadomo czy jakaś przesłanka jest prawdziwa, to traktowana jest ona jako nowa hipoteza i jest dla niej przeprowadzany analogiczny proces wnioskowania wstecz. Postępuje się w ten sposób do chwili znalezienia reguły, dla której wszystkie przesłanki są prawdziwe.

Mechanizm wnioskowania mieszanego polega na przełączaniu w zależności od sytuacji metody wnioskowania na wnioskowanie w przód bądź wstecz. Sam program zarządzający przełączaniem metod wnioskowania opiera się na zbiorze metareguł, które zawierają informacje o priorytecie rodzajów wnioskowania w zależności od sytuacji.

Koncepcja wnioskowania rozmytego opiera się o logikę rozmytą. Wnioskowanie takie można przeprowadzić dzięki reprezentacji wiedzy za pomocą zbiorów rozmytych i stosowaniu przekształceń danych wejściowych w postać końcową.

Systemy ekspertowe są to aplikacje mieszczące się w dziedzinie sztucznej inteligencji. Jest to nauka definiowana na wiele sposobów. Dwie definicje prezentujące odmienne aspekty badań prowadzonych w tej dziedzinie:

- Według Minsky'ego sztuczna inteligencja jest nauką o maszynach realizujących zadania, które wymagają inteligencji wówczas, gdy są wykonywane przez człowieka.[4]
- Według Feigenbauma sztuczna inteligencja stanowi dziedzinę informatyki dotyczącą metod i technik wnioskowania symbolicznego przez komputer oraz symbolicznej reprezentacji wiedzy stosowanej podczas takiego wnioskowania.[4]

2.2 Programowanie funkcyjne

Najbardziej powszechnym podejściem do programowania jest podejście impreratywne. Programowanie imperatywne opiera się na wykonywaniu kolejnych instrukcji zmieniających stan programu. W podejściu tym wykorzystywane są zmienne, które przechowuje informację o aktualnym stanie aplikacji bądź jej fragmentu. Istotą paradygmatu programowania imperatywnego jest możliwość modyfikowania danych przechowywanych w tych zmiennych, a co za tym idzie zmiana stanu aplikacji. W podejściu tym funkcje, pomimo przekazania takich samych parametrów, mogą zwracać różne wyniki. Spowodowane jest to tzw. efektami ubocznymi działania funkcji. Funkcja napisana z podejściem imperatywnym operuje nie tylko na parametrach do niej przekazanych, ale również na stanie aplikacji, czyli różnego rodzaju zmiennych. Zależnie od stanu w jakim podczas danego wywołania funkcji znajduje się program, można uzyskać różny wynik takiego wywołania.

Programowanie funkcyjne opiera się na ewaluacji funkcji, a nie przechowywaniu stanu aplikacji. W podejściu tym każda funkcja zwraca pewną wartość, a działania wykonuje wyłącznie na otrzymanych parametrach. Dzięki temu funkcje są pozbawione efektów ubocznych i mamy pewność, że wywołanie funkcji z takimi samymi parametrami zawsze zwróci taki sam wynik, niezależnie od chwili wywołania. Jako że każda funkcja jest ewaluowana do pewnej wartości, to mogą być one wykorzystywane jako argumenty innych funkcji. Stąd też jedna z głównych zalet podejścia funkcyjnego - modularność. Kod tworzony zgodnie z paradygmatem funkcyjnym jest łatwy do ponownego wykorzystania i dalszej rozbudowy.

[4] [24], str. 17

Kolejną istotną cechą jest leniwe wartościowanie. W przeciwieństwie do wartościowania zachłannego, które polega na wyliczaniu wartości wszystkich argumentów przed wywołaniem funkcji, nawet jeśli nie są one wykorzystywane, wartościowanie leniwe polega na ewaluacji wartości argumentów dopiero w chwili odwołania się do nich. Dzięki temu czas procesora jest wykorzystywany wyłącznie na obliczenia, które są konieczne w danym wywołaniu. Dzięki temu również można wyliczyć wartość funkcji nawet jeśli nie jest znana wartość któregoś z jej argumentów, pod warunkiem jednak, że nie jest on wykorzystywany w obliczeniach.

Programowanie funkcyjne w dużym stopniu oparte jest na rekurencji, czyli wywoływaniu funkcji przez samą siebie. Gdyby dla każdego takiego wywołania było zajmowane kolejne miejsce na stosie w pamięci operacyjnej systemu, bardzo szybko doszłoby do przepełnienia stosu. Stąd też powszechną techniką stosowaną przy podejściu funkcyjnym jest rekurencja ogonowa. Jest to rodzaj rekurencji, w którym wywołanie rekurencyjne jest ostatnią instrukcją funkcji. Dzięki temu nie ma potrzeby zajmowania nowego miejsca na stosie. Wykorzystywana jest dotychczasowa ramka stosu funkcji, jako że wszystkie operacje zostały już wykonane i zarezerowana pamięć nie jest już dłużej potrzebna. Taka operacja pozwala na optymalizację zarządzania pamięcią, a także ułatwia tworzenie kodu, jako że rekurencja zazwyczaj jest dużo bardziej naturalna niż iteracyjność. Istotne jest również to, że języki funkcyjne automatycznie przeprowadzają optymalizację rekurencji ogonowej, dzięki czemu programista nie musi się tym zajmować.

Jedną z najważniejszych cech funkcyjnych języków programowania jest posiadanie niemutowalnych struktur danych. Oznacza to, że wartość, bądź zbiór wartości raz przypisany do zmiennej bądź struktury danych nie może zostać zmieniony. Jest to cecha zaskakująca dla osób mających doświadczenie wyłącznie z programowaniem imperatywnym, które opiera się na modyfikowaniu wartości struktur danych i stanu aplikacji. Jednak programowanie funkcyjne opiera się na ewaluacji funkcji, a nie przechowywaniu danych w zmiennych. Dzięki niemutowalności danych unika się również wiele błędów związanych z efektami ubocznymi funkcji, które mogłyby te dane modyfikować.

Języki funkcyjne można podzielić na dwie grupy:

- Języki czysto funkcyjne
- Języki mieszane

Języki czysto funkcyjne są to języki, w których nie występują efekty uboczne, a operacje wejścia/wyjścia odbywają się np. z użyciem monad. W językach tych wartościowanie jest leniwe.

Jezyki mieszanie to takie, które dopuszczają stosowanie zmiennych, operacje wejścia/wyjścia i mieszanie stylu funkcyjnego z imperatywnym (a nawet obiektowym). Funkcje w tych językach mogą mieć również efekty uboczne.

2.3 Analiza techniczna

W tej pracy pojęcie analizy technicznej wykorzystywane jest w odniesieniu do rynku akcji. Analiza techniczna to badanie zachowań rynku, przede wszystkim przy użyciu wykresów, którego celem jest przewidywanie przyszłych trendów cenowych.[5] Zachowanie rynku oznacza w tym wypadku zmiany ceny konkretnego aktywa - akcji spółki giełdowej, a także wolumenu, czyli liczby akcji zmieniających właściciela podczas jednej sesji giełdowej (jednego dnia). Analiza techniczna oparta jest na 3 założeniach:

- Rynek dyskontuje wszystko
- Ceny podlegają trendom
- Historia się powtarza

Pierwszy punkt - rynek dyskontuje wszystko - zakłada, że rynkowa cena akcji w pełni odzwierciedla wszystkie czynniki mogące mieć wpływ na wartość aktywa, takie jak czynniki fundamentalne, polityczne czy psychologiczne. Analityk wykresów opiera się na prawie popytu i podaży. Jeśli popyt przewyższa podaż, to cena powinna rosnąć. Jeśli natomiast podaż przewyższa popyt - cena powinna spadać. Analityka nie interesują przyczyny zmiany popytu i podaży, a jedynie jej skutki.

To, że ceny podlegają trendom jest kolejną przesłanką na jakiej opiera się analiza techniczna. Jej uzupełnieniem jest stwierdzenie, że trend kontynuuje swój bieg w dotychczasowym kierunku tak długo, aż nie pojawią się przesłanki do tego, aby się zmienił.

Ostatnie założenie opiera się mocno na badaniu ludzkiej psychiki. W analizie technicznej zakłada się, że ludzka psychika, która ma przełożenie na trendy cenowe na rynku, raczej się nie zmienia. W związku z tym trendy występujące w przeszłości powinny powtórzyć się również w przyszłości.

Warto w tym miejscu wspomnieć również o drugim rodzaju analizy rynków finansowych - analizie fundamentalnej. Różni się ona tym, że koncentruje się na badaniu gospodarczych uwarunkowań popytu i podaży, które są przyczyną wzrostów, spadków lub stabilizacji cen. Przy podejściu fundamentalnym bada się wszystkie czynniki oddziałujące na cenę danego towaru, w celu określenia jego rzeczywistej wartości. Jeśli rzeczywista wartość danego towaru jest niższa od jego bieżącej ceny rynkowej, znaczy to, że towar ów jest "przewartościowany" i należy go sprzedać. Jeśli cena towaru jest niższa od jego rzeczywistej wartości, jest on "niedowartościowany" i należy go kupić.[6]

Analizę techniczną podzielić możemy na dwa obszary. Pierwszym z nich jest analiza wykresów i odnajdywanie na nich formacji cenowych, sugerujących kontynuację trendu, bądź jego odwrócenie. Drugi to analiza wskaźników analizy technicznej. Stworzono wiele wskaźników pomocnych w rozpoznawaniu określonych stanów w jakich znajduje się rynek, a także generujących sygnały takie jak sygnał kupna/sprzedaży, zmiany bądź kontynuacji trendu.

[5] [25], str. 1
[6] [25], str. 4

Rozdział 3

Technologie i narzędzia

3.1 Język programowania

Aplikacja została zrealizowana w języku Clojure [9] w wersji 1.5.1. Jest to język programowania działający na maszynie wirtualnej jawy (ang. JVM - Java Virtual Machine). Jest to język ogólnego przeznaczenia kompilowany do bajtkodu JVM. Clojure jest językiem funkcyjnym, jednym z dialektów języka Lisp. Tak jak Lisp jest językiem listowym (wszystkie elementy języka są listami), realizującym filozofię "kod jako dane".

Pierwszym powodem wyboru tego języka programowania jest to, że języki funkcyjne bardzo dobrze sprawdzają się przy tworzeniu systemów ekspertowych, a także pozwalają na generowanie nowego kodu aplikacji w trakcie jej działania. Samo programowanie funkcyjne jest natomiast dziedziną, która nie jest przybliżana podczas studiów. Stąd też chęć zapoznania się z tym paradygmatem programowania.

Kolejnym powodem użycia języka Clojure jest możliwość wykorzystania w nim elementów języka Java [10]. Dzięki temu w łatwy i szybki sposób mogłem stworzyć graficzny interfejs użytkownika dla aplikacji i więcej czasu poświęcić na te fragmenty aplikacji, które są nieodłącznymi elementami systemu ekspertowego.

Ostatnią przyczyną wybrania Clojure jest to, że programy pisane w tym języku kompilowane są do bajtkodu JVM, a więc można je uruchamiać na każdej maszynie i systemie operacyjnym posiadającym maszynę wirtualną jawy.

3.2 Oprogramowanie

Podczas tworzenia systemu ekspertowego dla tej pracy wykorzystywałem następujące oprogramowanie.

3.2.1 Środowisko programistyczne

Aplikacja była tworzone częściowo na komputerze z systemem operacyjnym Windows 7, a częściowo na komputerze z systemem Windows 8. Dzięki temu, że programy pisane w Clojure uruchamiane są na maszynie wirtualnej jawy, nie występowały

rzadne problemy podczas przenoszenia kodu pomiędzy tymi systemami i dalszym rozwijaniu go i uruchamianiu. Dzięki temu również praca byłaby możliwa chociażby na systemach Unixowych.

Do budowania aplikacji i zarządzania zależnościami wykorzystywałem środowisko Leiningen [11]. Jest to narzędzie do łatwego konfigurowania projektów tworzonych w języku Clojure.

Do budowania i uruchamiania aplikacji wykorzystywane jest środowisko JDK (ang. Java Development Kit) w wersji 1.7.0_25.

W trakcie pisania aplikacji korzystałem z edytora Clooj [7]. Jest to proste, zintegrowane środowisko programistyczne dla języka Clojure (jak i tworzone w tym języku). Edytor dostarcza podstawowe funkcjonalności, takie jak kolorowanie składni, dostęp do dokumentacji funkcji języka, czy możliwość łatwego tworzenia nowych plików źródłowych z automatycznym generowaniem nowych przestrzeni nazw.

3.2.2 Wykorzystane biblioteki

Podczas tworzenie aplikacji wykorzystywałem następujące biblioteki:

- Seesaw [6] - biblioteka do tworzenia interfejsów użytkownika w Clojure, jest zbudowana w oparciu o bibliotekę graficzną Swing
- Incanter [8] - biblioteka do obliczeń matematycznych, statystycznych i tworzenia wykresów w Clojure
- Instaparse [5] - biblioteka do tworzenia parserów gramatyk w Clojure
- clj-time [4] - biblioteka do operacji na datach i czasie w Clojure
- clj-http [3] - biblioteka do obsługi zapytań http w Clojure

3.3 Techniki i metodologie programistyczne

Projekt tworzony był z wykorzystaniem paradygmatu programowania funkcyjnego. Nie jest on jednak zrealizowany w czystym podejściu funkcyjnym. Przechowuje w postaci list wyliczone wartości wskaźników analizy technicznej, a także realizuje operację wczytywania danych o spółce giełdowej z pliku.

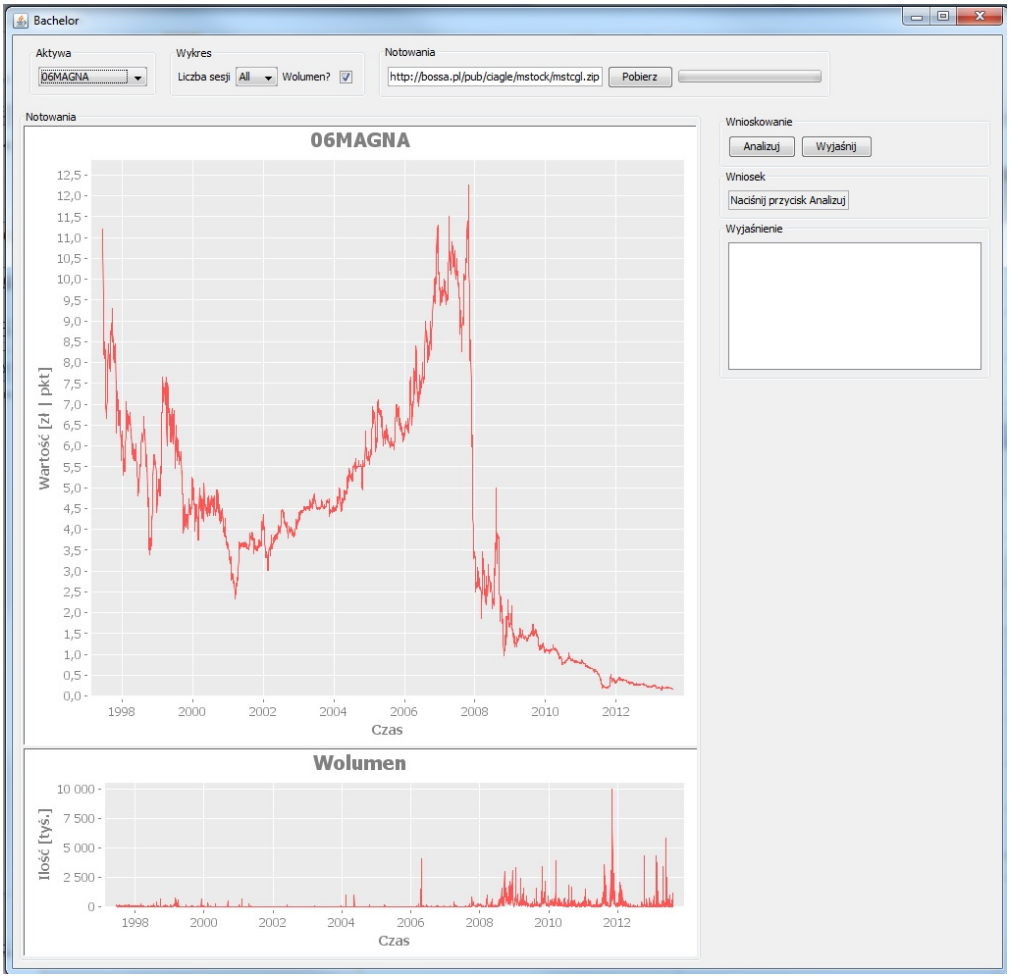
Rozdział 4

Wyniki badań eksperymentalnych

4.1 Opis stworzonej aplikacji

4.1.1 Podstawowy opis

Aplikacja powstająca podczas pisania tej pracy to system ekspertowy, którego zadaniem jest ocena spółek giełdowych i podejmowanie decyzji, czy w danej chwili konkretną spółkę warto kupić bądź sprzedać. Użytkownik aplikacji może pobrać notowania spółek giełdowych notowanych na Giełdzie Papierów Wartościowych w Warszawie z serwisu bossa.pl. Może również korzystać z dowolnego innego źródła notowań, pod warunkiem, że dane będą archiwum ZIP zawierającym pliki w formacie MST o nazwach odpowiadających nazwom spółek. Po tym na liście dostępne są spółki, których dane znajdowały się w archiwum. Aplikacja po wyborze konkretnej spółki tworzy wykres jej ceny i wolumenu w cały okresie w jakim była notowana. Użytkownik ma możliwość wyboru z listy ile sesji chce wyświetlić na wykresie, a także czy ma być tworzony wykres wartości wolumenu. Po wczytaniu danych o konkretnej spółce przycisk Analizuj pozwala na przeprowadzenie procesu wnioskowania dla danej spółki i określenie czy w danej chwili warto sprzedawać bądź kupować dane akcje.



Rysunek 4.1: Widok aplikacji po uruchomieniu

4.1.2 Struktura projektu

Projekt podzielony jest na 3 przestrzenie nazw:

- bachelor.core
- bachelor.wsk
- bachelor.zip

Przestrzeń nazw	Funkcjonalności
bachelor.core	Zawiera wszystkie funkcje parsera reguł, funkcje silnika wnioskującego, wczytywanie listy spółek, których dane znajdują się w katalogu z notowaniami. Tutaj również zdefiniowany jest interfejs użytkownika. Dodatkowo jest tu funkcja uruchamiająca wypakowywanie archiwum ZIP z notowaniami
bachelor.wsk	Tutaj zdefiniowana jest funkcja wczytująca notowania z pliku i tworząca z nich odpowiednią strukturę umieszczaną na liście notowań. W tej przestrzeni nazw zdefiniowane są również wszystkie wskaźniki wykorzystywane przez system wnioskujący, a także funkcje pomocnicze do wyliczania wskaźników i operacji na liście notowań. Jest tu również umieszczona funkcja tworząca listy wszystkich zdefiniowanych wskaźników, a także funkcje odwołujące się do tych list, które wykorzystywane są w regułach.
bachelor.zip	Zdefiniowana jest tu funkcja do pobierania archiwum ze wskazanego adresu internetowego i zapisu go na dysku. Znajdują się tu także wszystkie funkcje pomocnicze do rozpakowywania archiwum

4.1.3 Gramatyka reguł

Jedną z funkcjonalności systemu jest parser reguł, które są wczytywane z pliku tekstowego, a następnie na ich podstawie generowany jest kod nowych funkcji, które przekazywane są do mechanizmu wnioskującego. Aby można było stworzyć taki parser najpierw trzeba było zaprojektować gramatykę, zgodnie z którą można zapisywać reguły w pliku tekstowym. Gramatyka ta ma następującą postać:
RULE = EXPR >>FACT
EXPR = (EXPR OpL EXPR) | (WSK OpA NUM) | (WSK OpA WSK) | (FACT)
OpL = and | or
OpA = > | < | ==
WSK = [A-Z]+[A-Z0-9]*
FACT = [a-z]+[a-z0-9]*
NUM = [-+]?[0-9]+[.]?[0-9]*|[0-9]+

W pierwszej kolejności linia z regułą wczytana z pliku parsowana jest przez bibliotekę Instaparse [5] do listy tokenów, czyli par opisanych jako

[:TYP_TOKENA WARTOŚĆ]

Następnie element z listy o typie :EXPR przekazywany jest do funkcji, która sprawdza, którym wariantem wyrażenia jest przekazany token. Na tej podstawie tworzy listę symboli możliwych do zinterpretowania przez Clojure, które odpowiadają funkcjonalnie treści wyrażenia. Utworzona lista symboli jest wykorzystywana

jako „ciało” funkcji przez makro, które generuje nowe funkcje reprezentujące reguły. Każde tak wygenerowane wyrażenie musi zwracać wartość logiczną prawda bądź fałsz. Podczas procesu wnioskowania zależnie od tej wartości do listy faktów dodawany jest nowy fakt, bądź mechanizm przechodzi do ewaluacji kolejnej funkcji - reguły.

Dzięki bibliotece Instaparse [5] w łatwy sposób można powyższą gramatykę reguł przetworzyć na listę tokenów, a z takiej listy wygenerować funkcję, którą następnie można wywołać w celu sprawdzenia, czy wszystkie przesłanki danej reguły są prawdziwe i można dodać nowy fakt do bazy wiedzy.

```
(def grammar
  (insta/parser
    "RULE = EXPR' >> 'FACT
     EXPR = '('EXPR' 'OpL' 'EXPR')' | '('WSK' 'OpA' 'NUM')'
          | '('WSK' 'OpA' 'WSK')' | '('FACT')'
     OpL = 'and' | 'or'
     OpA = '>' | '<' | '=='
     WSK = #'[A-Z]+[A-Z0-9]*'
     FACT = #'[a-z]+[a-z0-9]*'
     NUM = #'[-+]?[0-9]+[.]?[0-9]*|[0-9]+'")
  )
```

Listing 4.1: Zdefiniowana gramatyka wykorzystywania do tworzenia listy tokenów z reguły

```
(defmacro generate-funcs
  "creates function from parse-tree"
  [parse-tree]
  (let [[_ expr _ [_ fact-value]] parse-tree
        arg (gensym "facts")
        expression (parse-expr expr arg)
        new-fact (str fact-value)]
    `(fn
       [~arg]
       (if ~expression
         (str ~new-fact)
         ())
       )
    )
  )
```

Listing 4.2: Makro generujące funkcję - wykorzystuje listę tokenów

```
1 (defn parse-expr
2   "parses expr struct into list of symbols"
3   [expr arg]
4   (cond
5     (empty? expr) ()
6     :else
7     (let [[_ _ [part1-type part1-val] _ [part2-type
          part2-val] _ [part3-type part3-val] _] expr]
8       (cond
9         (and (= :WSK part1-type) (= :OpA part2-type) (= :
            NUM part3-type))
10        (let
11         [wsk (symbol (str "bachelor.wsk/" part1-val))
12         opa (symbol part2-val)
13         num (Integer/valueOf part3-val)]
14         (list opa (list wsk) num))
15        (and (= :WSK part1-type) (= :OpA part2-type) (= :
            WSK part3-type))
16        (let
17         [wsk1 (symbol (str "bachelor.wsk/" part1-val))
18         opa (symbol part2-val)
19         wsk2 (symbol (str "bachelor.wsk/" part3-val))]
20         (list opa (list wsk1) (list wsk2)))
21        (= :OpL part2-type)
22        (let
23         [expr1 (parse-expr (get expr 2) arg)
24         opl (symbol part2-val)
25         expr2 (parse-expr (get expr 6) arg)]
26         (list opl expr1 expr2))
27        (= :FACT part1-type)
28        (let
29         [fact (str part1-val)
30         bool (symbol "boolean")
31         some (symbol "some")]
32         (list bool (list some #{fact} arg)))
33        :else
34        ())
35      )
36    )
37  )
```

Listing 4.3: Funkcja tworząca „ciało” funkcji z listy tokenów

4.1.4 Mechanizm wnioskujący

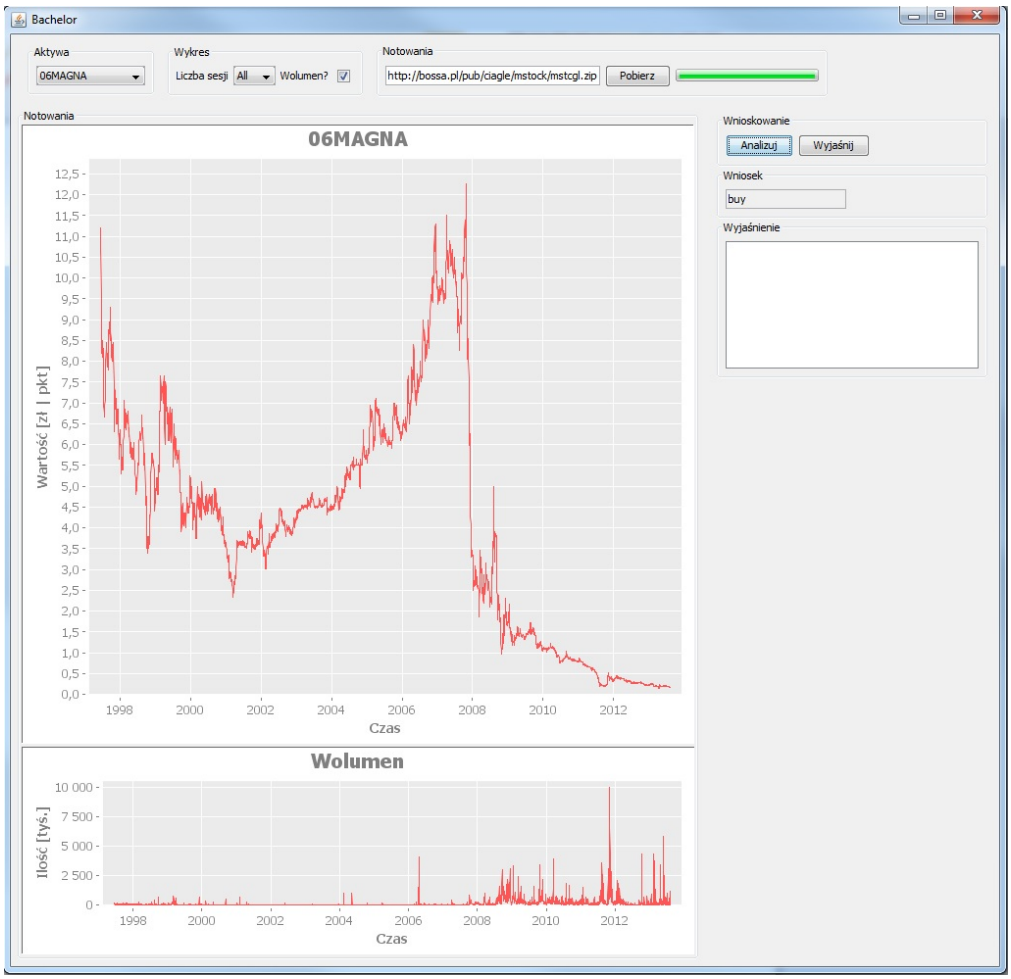
Silnik wnioskujący opiera się o wnioskowanie w przód i składa się z dwóch funkcji. Pierwsza z nich cyklicznie wywołuje drugą, przekazując do niej listę reguł i aktualną listę faktów. Druga funkcja uruchamia kolejno wszystkie reguły. Jeśli można uruchomić regułę i wartość logiczne jej wyrażenia to prawda, wtedy sprawdzane jest czy na liście faktów znajduje się już fakt generowany przez tą regułę. Jeśli nie, jest on dodawany i uruchamiana jest kolejna reguła. Po każdym wykonaniu funkcji sprawdzającej wszystkie reguły, główna funkcja wnioskująca sprawdza, czy na liście faktów pojawił się fakt mówiący o tym aby kupić bądź sprzedać dane akcje. Dodatkowo sprawdza też, czy zmieniła się liczba wygenerowanych faktów. Jeśli nie, oznacza to, że nie można uruchomić już żadnej nowej reguły i nie da się uzyskać wniosku kupuj bądź sprzedaj.

```
1 (defn inference
2   "evaluates all rules as long as don't get buy or sell
       conclusion"
3   [rules facts]
4   (cond
5     (done? facts) facts
6     (empty? rules) facts
7     (= (count facts) (count (vec (evaluateRules rules facts
         )))) facts
8     :else
9     (inference rules (vec (evaluateRules rules facts))))
10  )
```

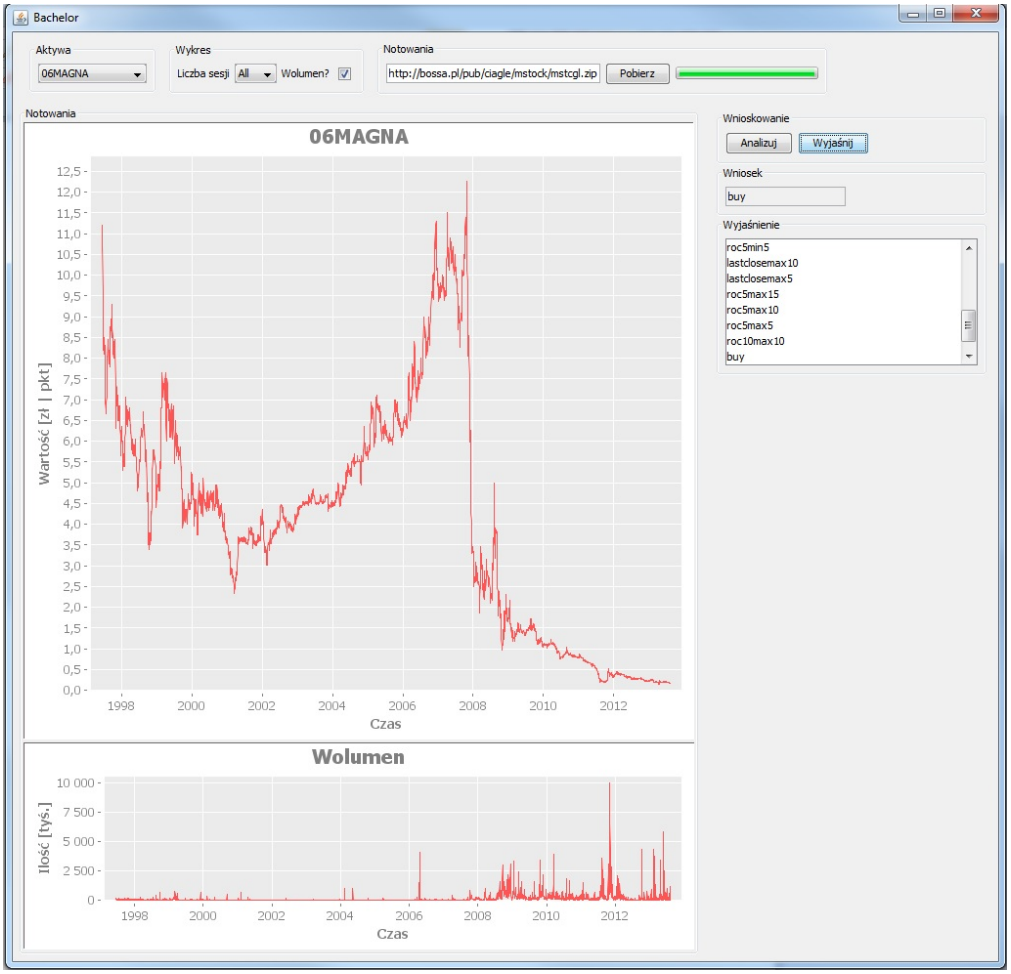
Listing 4.4: Główna funkcja wnioskująca

```
1 (defn evaluateRules
2   "evaluates rules"
3   [rules facts]
4   (cond
5     (empty? rules) facts
6     (empty? (eval (list (first rules) facts))) (
         evaluateRules (rest rules) facts)
7     (containsFact? (eval (list (first rules) facts)) facts)
          (evaluateRules (rest rules) facts)
8     :else
9     (cons (eval (list (first rules) facts)) (evaluateRules
         (rest rules) facts)))
10  )
```

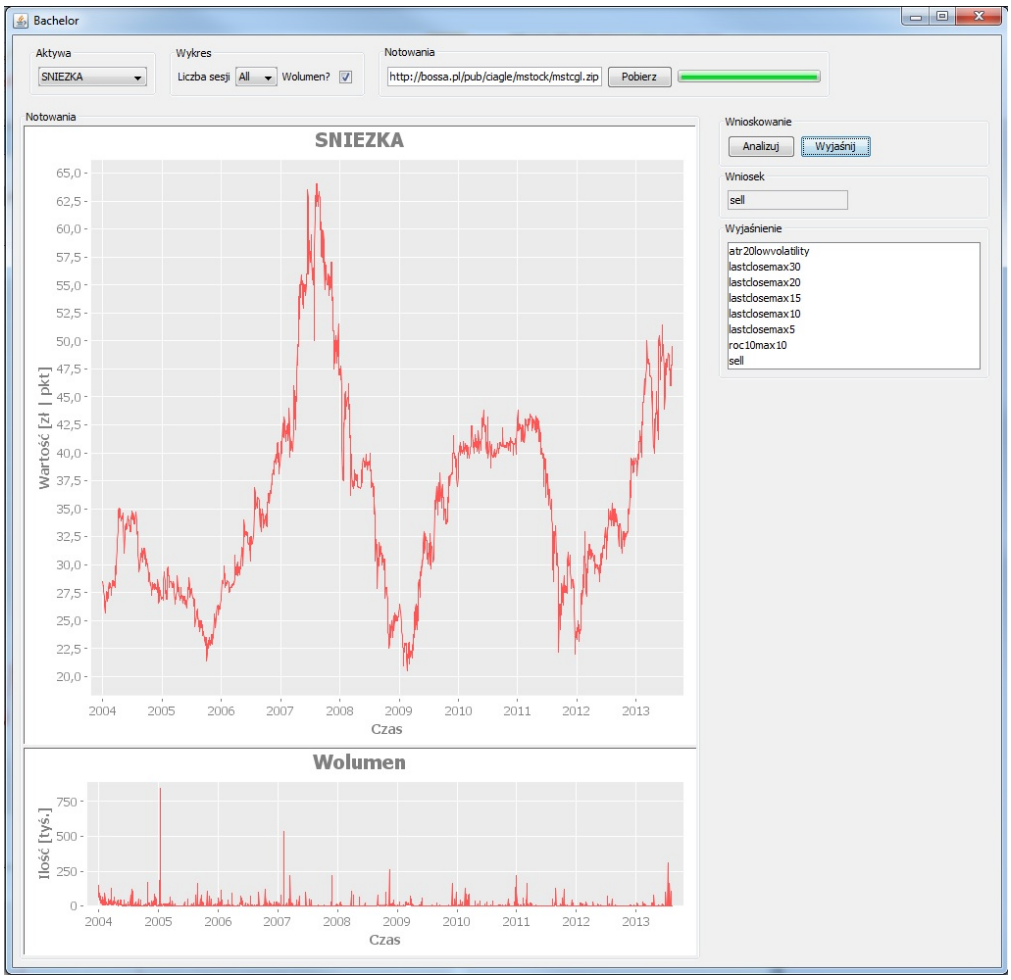
Listing 4.5: Funkcja sprawdzająca wszystkie reguły



Rysunek 4.2: Analiza notowań spółki



Rysunek 4.3: Przedstawienie listy wygenerowanych faktów podczas wnioskowania - wariant z wnioskiem kupuj



Rysunek 4.4: Przedstawienie listy wygenerowanych faktów podczas wnioskowania - wariant z wnioskiem sprzedaj

Rysunek 4.5: Przedstawienie listy wygenerowanych faktów podczas wnioskowania - wariant przy braku wniosków

4.1.5 Obsługa plików ZIP

Archiwum ZIP z notowaniami spółek z GPW w Warszawie obsługiwane jest w przestrzeni nazw bachelor.zip. Po naciśnięciu przycisku Pobierz wywoływana jest funkcja, która pobiera plik ZIP ze wskazanego w polu tekstowym adresu www. Następnie ścieżka do zapisanego pliku przekazywana jest do głównej funkcji rozpakowującej. Funkcja ta dla każdego elementu z archiwum tworzy nowy plik z nazwą tego elementu, a następnie zapisuje do niego tablicę bajtów z zawartością tego pliku. Ze względu na obsługę paska postępu wykonywanych operacji, główna funkcja wnioskująca znajduje się w głównej przestrzeni nazw, a wszystkie funkcje pomocnicze w przestrzeni bachelor.zip. Cały proces pobierania i rozpakowywania archiwum odbywa się w osobnym wątku, tak aby nie blokować działania reszty interfejsu użytkownika.

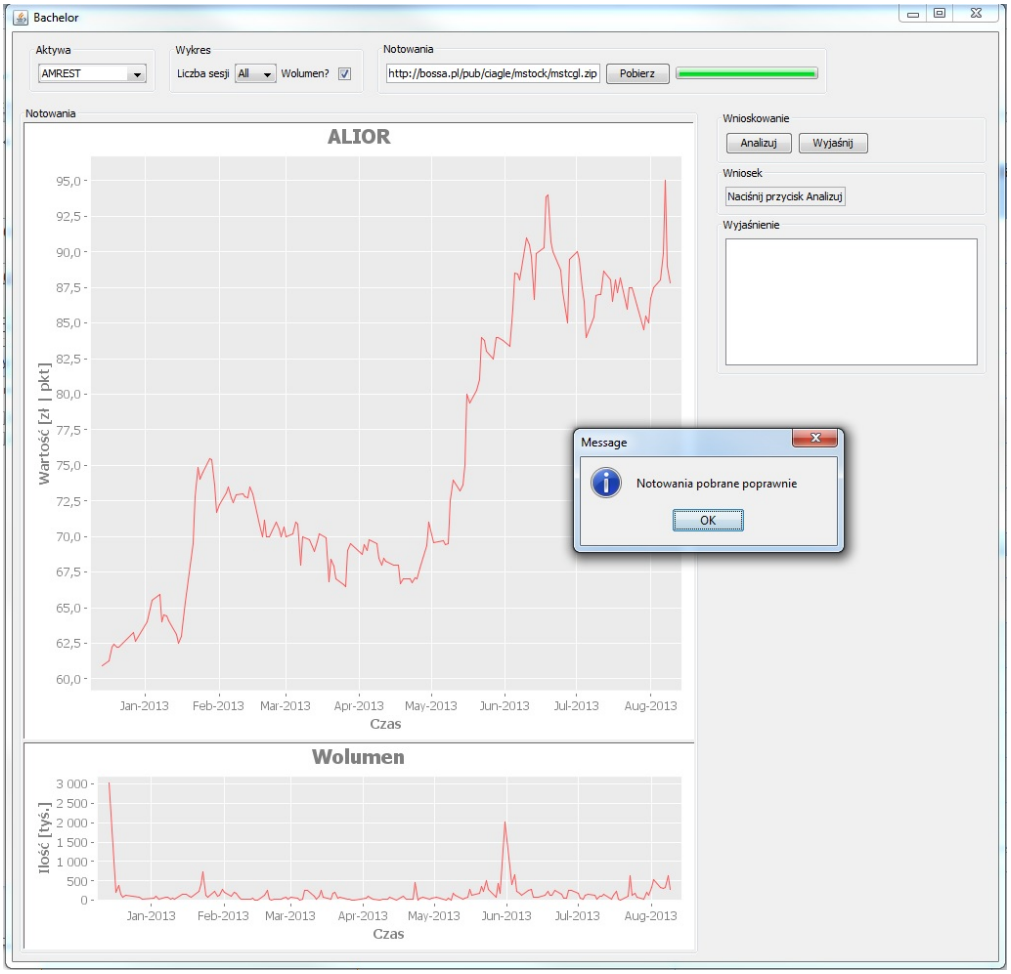
```
1 (defn writeZipEntry
2   "Writes zip entry to file"
3   [zip entry]
4   (with-open [w (io/writer (str "resources/notowania/" (.
      getName entry)))]
5     (.write w (getZipEntryContent zip entry)))
6   )
```

Listing 4.6: Funkcja zapisująca pojedynczy plik z archiwum

```
1 (defn getZipEntryContent
2   "Reads zip entry content"
3   [zip entry]
4   (slurp (.getInputStream zip entry))
5   )
```

Listing 4.7: Funkcja odczytująca zawartość pojedynczego pliku z archiwum

Rysunek 4.6: Pobieranie nowych notowań



Rysunek 4.7: Poprawne pobranie i rozpakowanie nowych notowań

4.1.6 Struktura danych o notowaniach

Dane o notowaniach spółek giełdowych, które wczytywane są z pliku, w aplikacji przechowywane są w następującej strukturze:

Company

- Name - nazwa spółki
- **Session**
  - Date - data sesji
  - Open - cena otwarcia sesji
  - High - cena maksymalna podczas sesji
  - Low - cena minimalna podczas sesji
  - Close - cena zamknięcia sesji
  - Vol - wolumen

```
1 (defstruct Company :name :session)
2
3 (defstruct Session :date :open :high :low :close :vol)
```

Listing 4.8: Struktura notowań zdefiniowana w języku Clojure

4.1.7 Wykresy

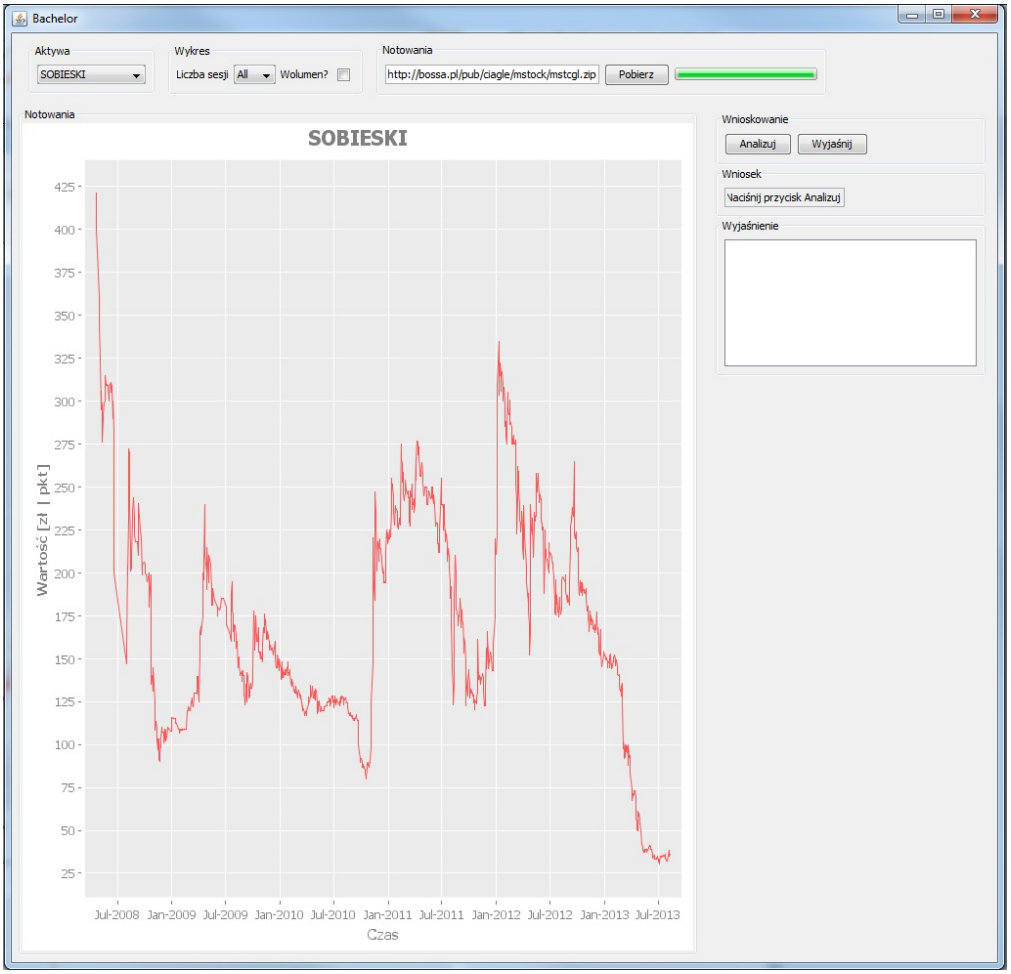
W aplikacji została wykorzystana biblioteka Incanter [8] pozwalająca generować wykresy z dostarczonych struktur danych. Wykorzystywany jest jedynie szablon wykresów czasowych, do którego przekazujemy listę dat jako wartości dla osi poziomej i drugą listę z wartościami dla osi pionowej. Domyślnie rysowane są dwa wykresy:

- Wartości akcji (indeksu) - główny wykres przedstawiający cenę akcji wybranej spółki, bądź wartość wybranego indeksu
- Wolumenu - dodatkowy wykres prezentujący wartość wolumenu

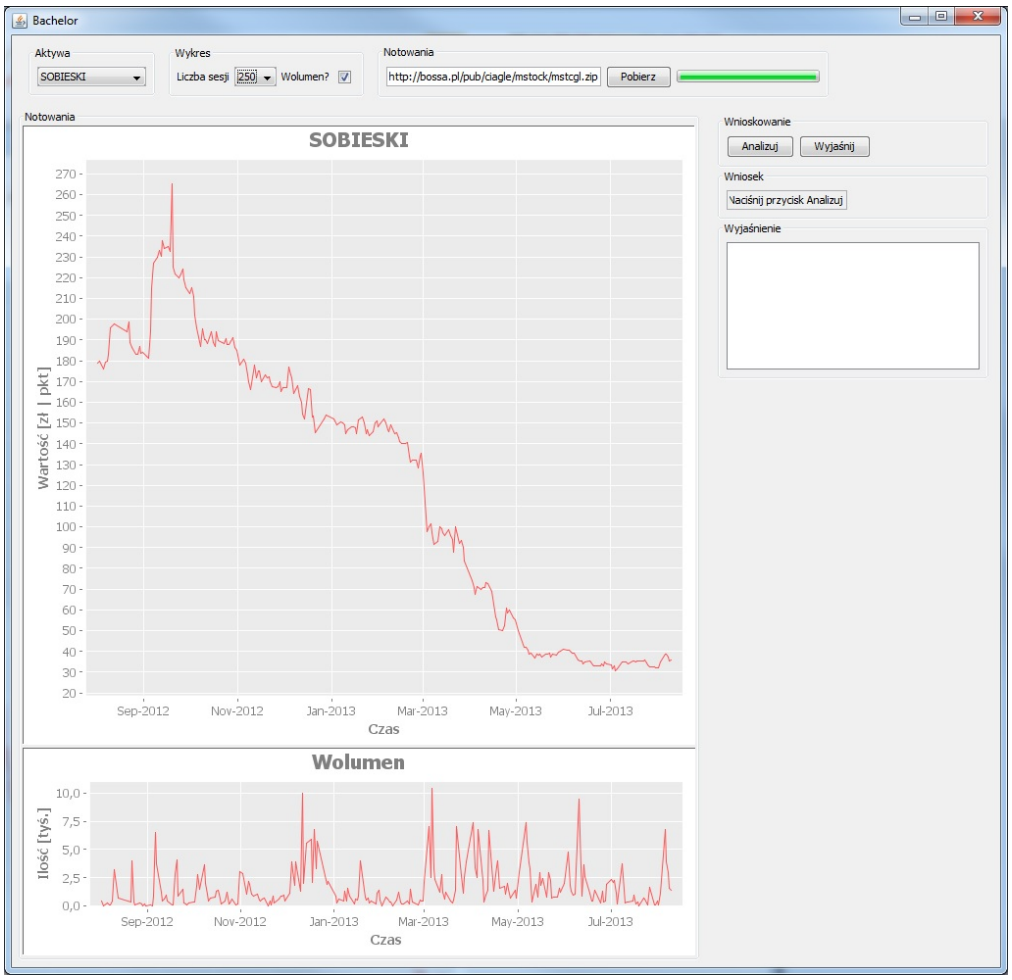
```
1 (ChartPanel.
2   (charts/time-series-plot
3     (x (read-string (config sessions :text)) (wsk/
          createCompany (config companiesList :text)))
4     (y (read-string (config sessions :text)) (wsk/
          createCompany (config companiesList :text)))
5     :title (config companiesList :text)
6     :x-label "Czas"
7     :y-label "Wartosc [zl | pkt]"))
```

Listing 4.9: Przykładowa funkcja rysująca wykres przy pomocy biblioteki Incanter [8]

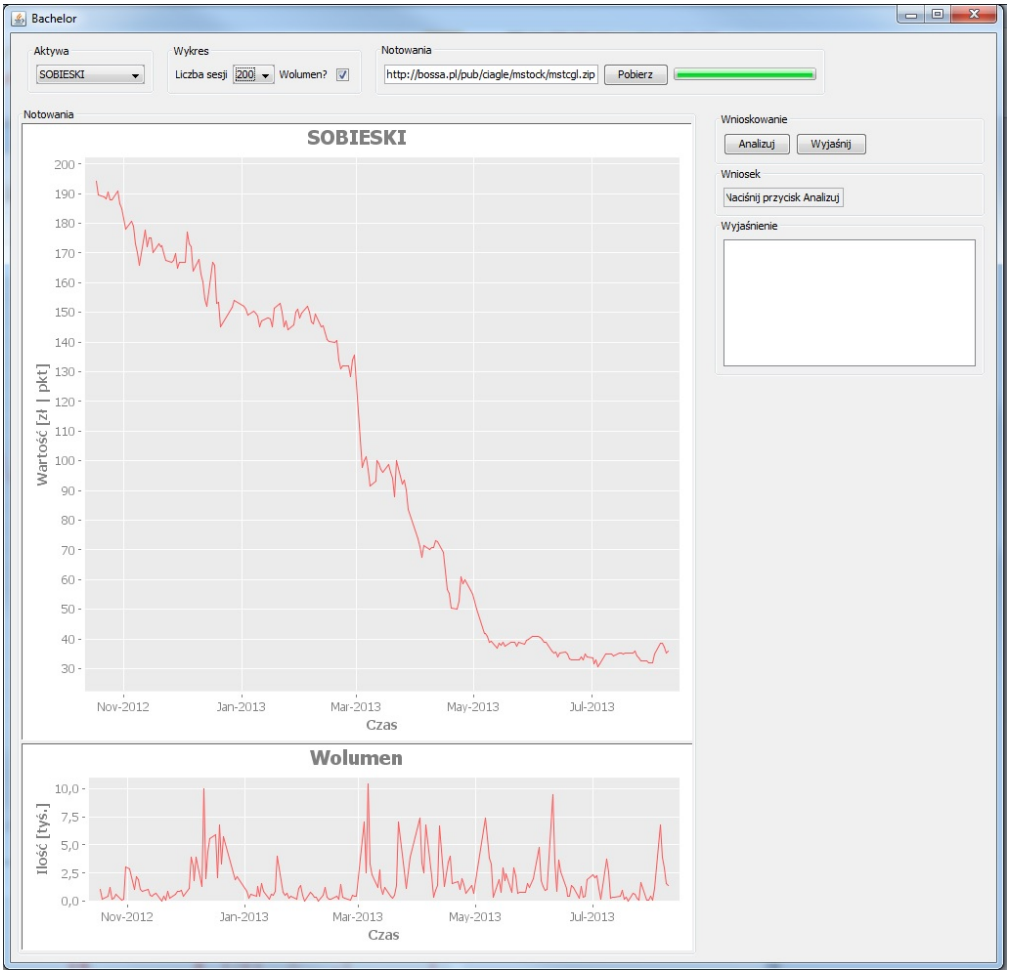
Istnieje możliwość ukrycia wykresu wolumenu, a także wyboru z rozwijanej listy dla ilu sesji mają być rysowane wykresy.



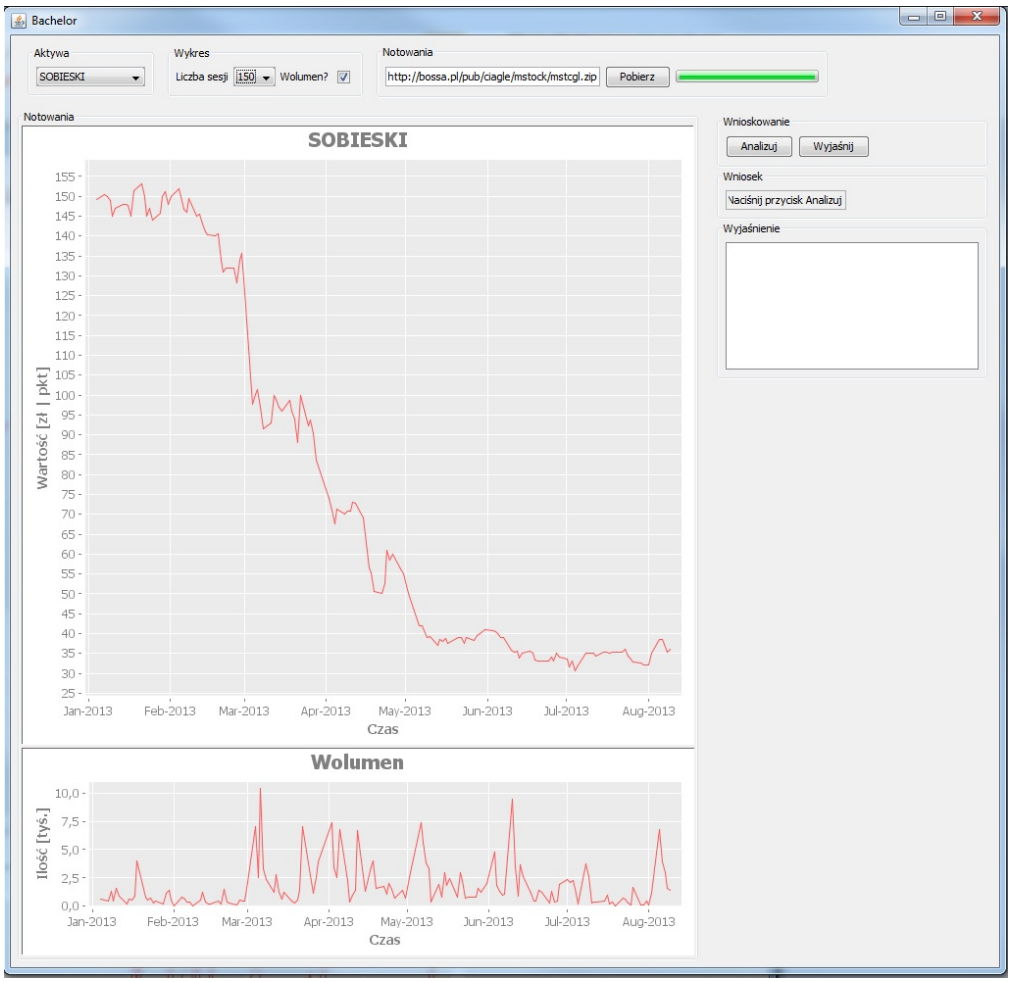
Rysunek 4.8: Widok z ukrytym wykresem wolumenu



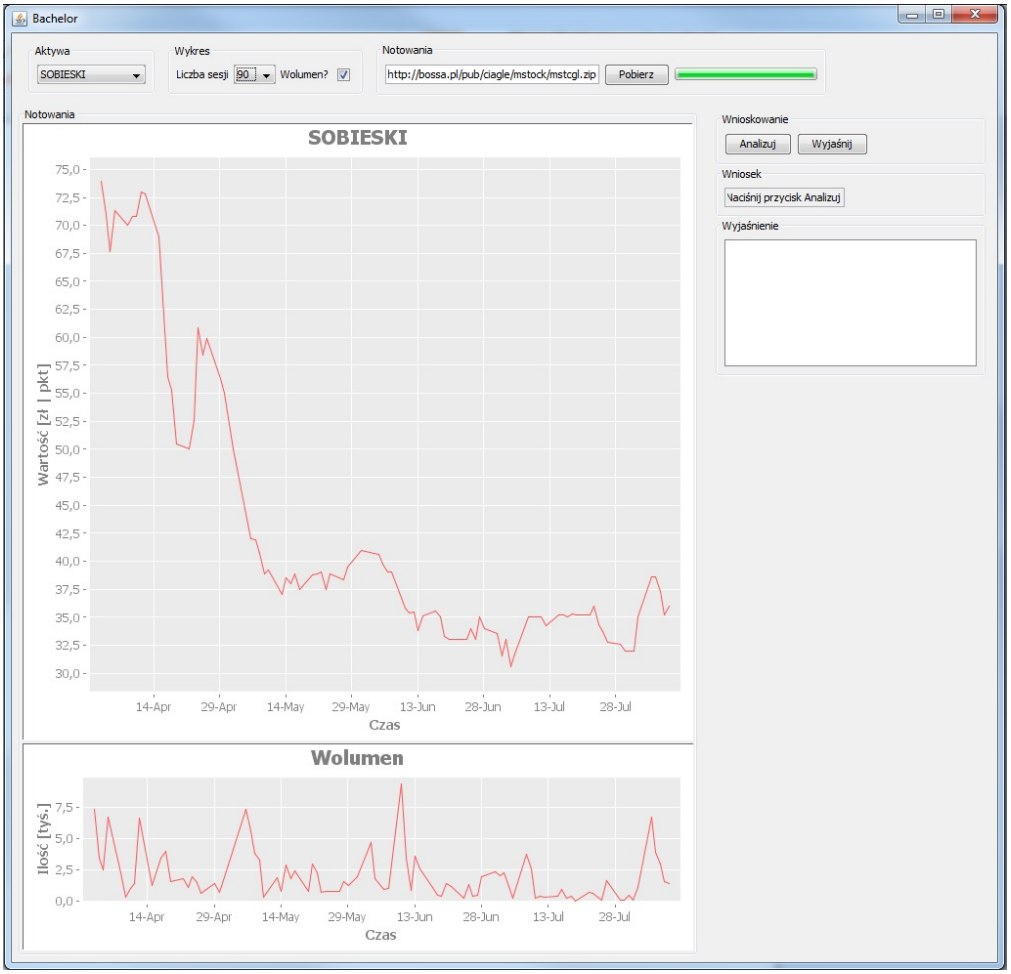
Rysunek 4.9: Widok wykresów dla 250 ostatnich sesji



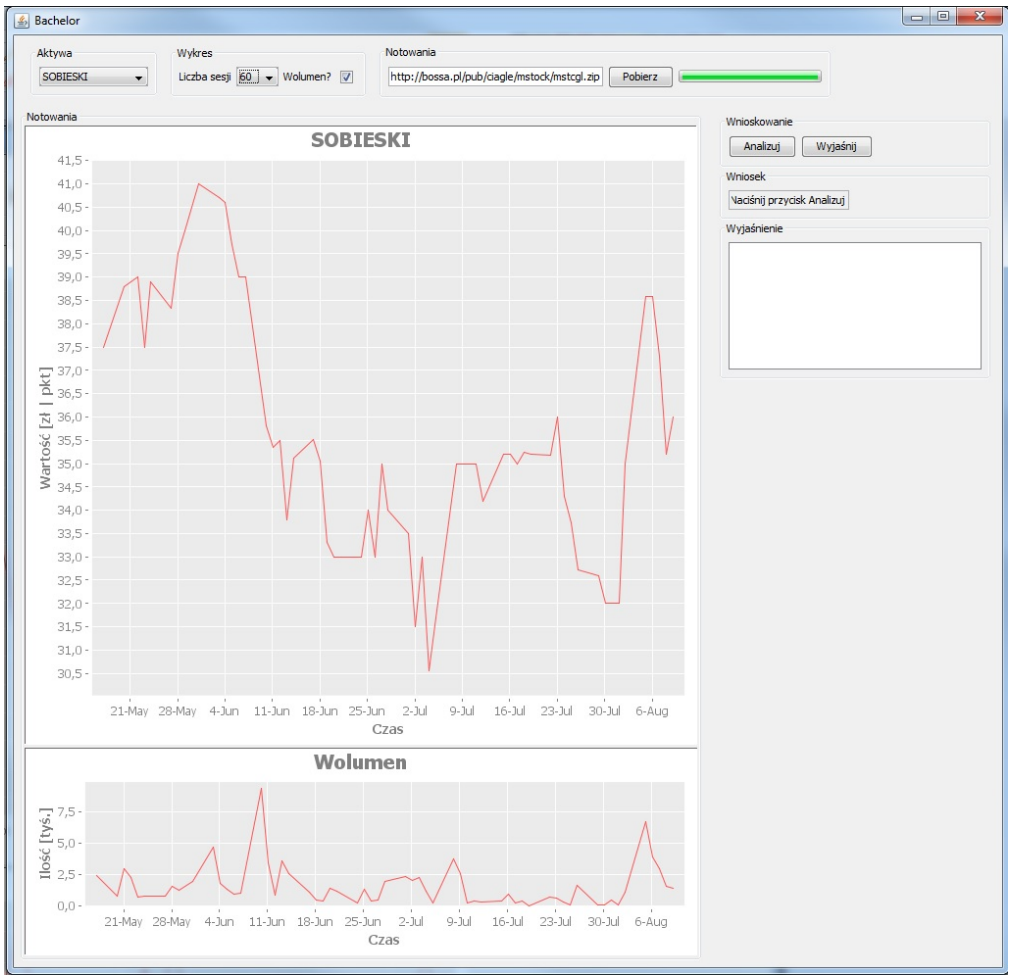
Rysunek 4.10: Widok wykresów dla 200 ostatnich sesji



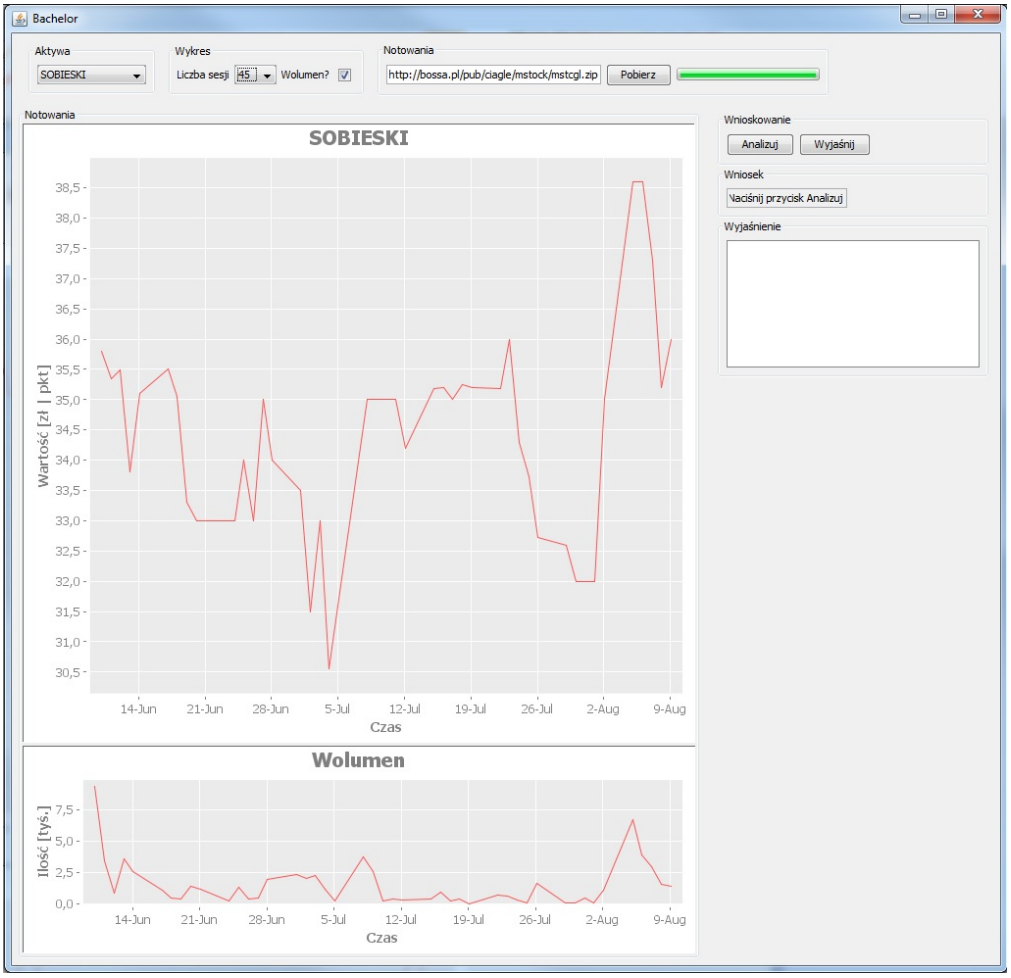
Rysunek 4.11: Widok wykresów dla 150 ostatnich sesji



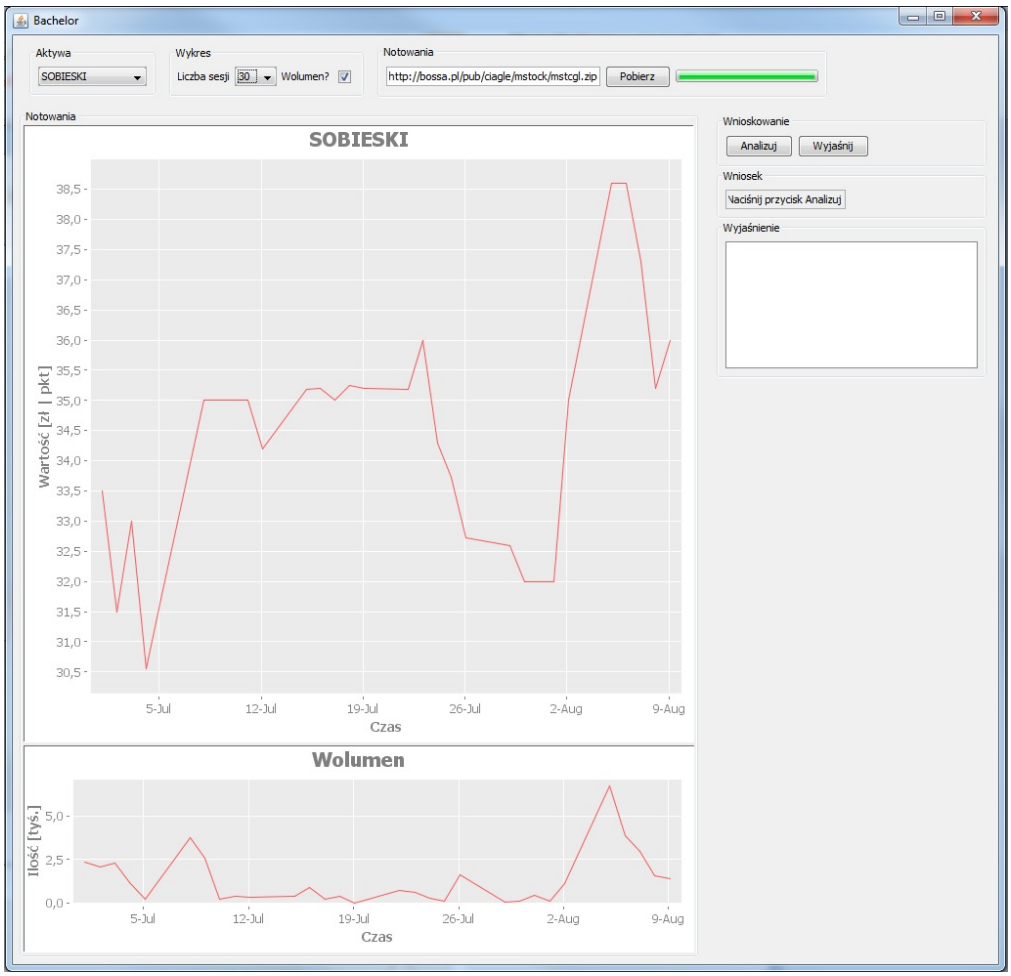
Rysunek 4.12: Widok wykresów dla 90 ostatnich sesji



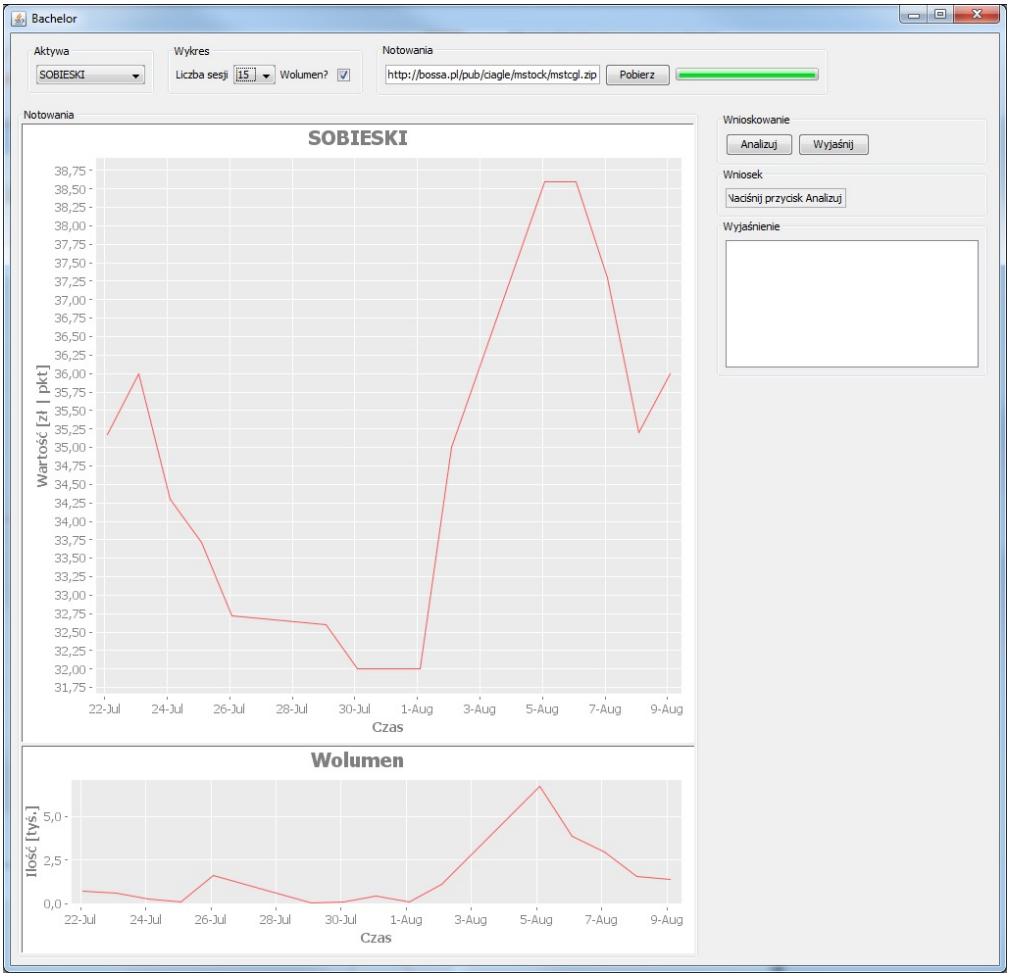
Rysunek 4.13: Widok wykresów dla 60 ostatnich sesji



Rysunek 4.14: Widok wykresów dla 45 ostatnich sesji



Rysunek 4.15: Widok wykresów dla 30 ostatnich sesji



Rysunek 4.16: Widok wykresów dla 15 ostatnich sesji

4.1.8 Obliczane wskaźniki analizy technicznej

Podczas procesu wnioskowania wykorzystywanych jest wiele wskaźników analizy technicznej. Poniżej znajduje się lista tych wskaźników wraz ze sposobem ich obliczania:

MFI - Money Flow Index

$$MFI = 100 \times \frac{pozytywny\ money\ flow}{pozytywny\ money\ flow + negatywny\ money\ flow}$$

, gdzie

- „pozytywny money flow" jest sumą *money flow* z dni, w których cena typowa przewyższała cenę typową z dnia poprzedniego
- „negatywny money flow" jest sumą *money flow* z dni, w których cena typowa była niższa niż w dniu poprzednim
- $money\ flow = cena\ typowa \times wolumen$
- $cena\ typowa = \frac{cena\ maksymalna + cena\ minimalna + cena\ zamkniecia}{3}$

ATR - Average True Range

Jest to średnia arytmetyczna z rzeczywistego zakresu zmian (ang. TR - True Range) dla zadanej liczby sesji. TR to największa co do modułu wartość z:

- różnicy między ceną najwyższą i najniższą podczas analizowanej sesji
- różnicy między ceną zamknięcia sesji poprzedzającej analizowaną sesję a ceną najwyższą podczas analizowanej sesji
- różnicy między ceną zamknięcia sesji poprzedzającej analizowaną sesję a ceną najniższą podczas analizowanej sesji

ROC - Wskaźnik zmiany ROC (ang. Rate of Change)

$$ROC(n,k) = \frac{C_n - C_{n-k}}{C_{n-k}} \times 100\%$$

, gdzie

- C - cena zamknięcia
- n - numer analizowanej sesji
- k - liczba sesji wstecz w odniesieniu do obecnie analizowanej

Momentum

$$Momentum(n,k) = C_n - C_{n-k}$$

, gdzie

- C - cena zamknięcia
- n - numer analizowanej sesji
- k - liczba sesji wstecz w odniesieniu do obecnie analizowanej

SMA - Prosta średnia krocząca (ang. Simple moving average)

$$SMA = \frac{C_0 + C_1 + \cdots + C_{n-1}}{n}$$

, gdzie

- C - cena zamknięcia
- n - liczba analizowanych sesji

WMA - Ważona średnia krocząca (ang. Weighted moving average)

$$WMA = \frac{nC_0 + (n-1)C_1 + \cdots + C_{n-1}}{n + (n-1) + \cdots + 2 + 1}$$

, gdzie

- C - cena zamknięcia
- n - liczba analizowanych sesji, a zarazem wagi kolejnych okresów

EMA - Wykładnicza średnia krocząca (ang. Expotencial moving average)

$$EMA = \frac{C_0 + (1-\alpha)C_1 + (1-\alpha)^2C_2 + (1-\alpha)^3C_3 + \cdots + (1-\alpha)^nC_n}{1 + (1-\alpha) + (1-\alpha)^2 + (1-\alpha)^3 + \cdots + (1-\alpha)^n}$$

, gdzie

- C - cena zamknięcia
- n - liczba analizowanych sesji
- $\alpha = \frac{2}{n+1}$

RSI - Wskaźnik siły względnej (ang. Relative Strength Index)

$RSI = 100 - \frac{100}{1+RS}$

, gdzie

- $RS = \left(\frac{a}{b}\right)$
- a - średnia wartość wzrostu cen zamknięcia z x sesji
- b - średnia wartość spadku cen zamknięcia z x sesji

Accumulation/Distribution

$Accum/Distr = Accum/Distr_{poprz} + wolumen \times CLV$

, gdzie

- $CLV = \frac{(C_{zamknicia} - C_{minimalna}) - (C_{maksymalna} - C_{zamknicia})}{C_{maksymalna} - C_{minimalna}}$

MACD - Moving Average Convergence/Divergence

Wskaźnik składa się z dwóch linii:

- Linia MACD - $MACD = EMA(26) - EMA(12)$
- Linia sygnału - powstaje ze średniej wykładniczej o okresie 9 z linii MACD

4.2 Badania

4.2.1 Opis przeprowadzonych analiz

Przy pomocy stworzonej aplikacji przeprowadziłem serię analiz różnych spółek notowanych na GPW w Warszawie. Analizy przeprowadziłem w perspektywie średnio i krótkoterminowej. Dla analizy średnioterminowej wykorzystywałem notowania sprzed 60 sesji, natomiast dla krótkoterminej sprzed 20. W obu przypadkach analizie poddanych zostało po 5 spółek z indeksów WIG20, mWIG40 i sWIG80. Dla WIG20:

- PKOBP
- KGHM
- PGE
- TPSA

- JSW

Dla mWIG40:

- ALIOR
- TVN
- NETIA
- INTERCARS
- GPW

Dla sWIG80:

- WAWEL
- AMICA
- ATM
- DEBICA
- BENEFIT

4.2.2 Wyniki badań

Załóżmy, że dla każdego zestawu danych w sytuacji wniosku „Kup" kupujemy 1000 akcji, a następnie sumujemy zyski/straty po upływie odpowiednio 20 bądź 60 sesji. Analogiczną operację przeprowadzamy dla faktu „Sprzedaj". Obliczamy różnicę w wartości 1000 akcji każdej spółki z zaleceniem sprzedaży odpowiednio 20 i 60 sesji wstecz. W sytuacji gdy nie ma wniosków nie jest podejmowane żadne działanie.

Analiza spółek z indeksu WIG20

Spółka	Cena 20 sesji wstecz	Cena aktualna	Wniosek
PKOBP	35.95	39.00	Kup
KGHM	119.50	125.95	Brak wniosków
PGE	14.50	15.12	Kup
TPSA	7.97	7.51	Sprzedaj
JSW	67.41	66.95	Kup

Tabela 4.1: Wyniki analizy spółek z WIG20 w krótkim terminie

Spółka	Cena 60 sesji wstecz	Cena aktualna	Wniosek
PKOBP	34.50	39.00	Kup
KGHM	140.50	125.95	Sprzedaj
PGE	17.48	15.12	Kup
TPSA	7.80	7.51	Sprzedaj
JSW	79.85	66.95	Brak wniosków

Tabela 4.2: Wyniki analizy spółek z WIG20 w średnim terminie

Poniżej zestawienie zysków/strat inwestycji prowadzonej zgodnie z zaleceniami systemu dla wybranych spółek z indeksu WIG20.

Spółka	Zakup 60 sesji wstecz	Zakup 20 sesji wstecz	Wartość zakupu	Wartość sprzedaży
PKOBP	TAK	TAK	70 450zł	78 000zł
KGHM	NIE	NIE	0 zł	0 zł
PGE	TAK	TAK	31 980zł	30 240zł
TPSA	NIE	NIE	0zł	0zł
JSW	NIE	TAK	67 410zł	66 950zł

Tabela 4.3: Wynik finansowy inwestycji w wybrane spółki z WIG20

Wartość zakupu wszystkich akcji:	169 840zł
Wartość sprzedaży wszystkich akcji:	175 190zł
Zysk netto:	5 350zł (3.15%)

Tabela 4.4: Podsumowanie inwestycji w wybrane spółki z WIG20

Spółka	Sprzedaż 60 sesji wstecz	Sprzedaż 20 sesji wstecz	Wartość sprzedaży	Wartość zakupu
PKOBP	NIE	NIE	0zł	0zł
KGHM	TAK	NIE	140 500zł	125 950zł
PGE	NIE	NIE	0zł	0zł
TPSA	TAK	TAK	15 770zł	15 020zł
JSW	NIE	NIE	0zł	0zł

Tabela 4.5: Wynik finansowy po sprzedaży wybranych spółek z WIG20

Wartość sprzedaży wszystkich akcji:	156 270zł
Wartość ponownego zakupu wszystkich akcji:	140 970zł
Zysk netto:	15 300zł (9.79%)

Tabela 4.6: Podsumowanie sprzedaży wybranych spółek z WIG20

Analiza spółek z indeksu mWIG40

Spółka	Cena 20 sesji wstecz	Cena aktualna	Wniosek
ALIOR	88.65	87.80	Kup
TVN	10.90	12.62	Brak wniosków
NETIA	4.33	4.99	Brak wniosków
INTERCARS	134.00	163.00	Sprzedaj
GPW	38.19	36.83	Brak wniosków

Tabela 4.7: Wyniki analizy spółek z mWIG40 w krótkim terminie

Spółka	Cena 60 sesji wstecz	Cena aktualna	Wniosek
ALIOR	80.00	87.80	Kup
TVN	9.60	12.62	Kup
NETIA	4.41	4.99	Brak wniosków
INTERCARS	109.00	163.00	Kup
GPW	40.75	36.83	Kup

Tabela 4.8: Wyniki analizy spółek z mWIG40 w średnim terminie

Poniżej zestawienie zysków/strat inwestycji prowadzonej zgodnie z zaleceniami systemu dla wybranych spółek z indeksu mWIG40.

Spółka	Zakup 60 sesji wstecz	Zakup 20 sesji wstecz	Wartość zakupu	Wartość sprzedaży
ALIOR	TAK	TAK	168 650zł	175 600zł
TVN	TAK	NIE	9 600 zł	12 620 zł
NETIA	NIE	NIE	0zł	0zł
INTERCARS	TAK	NIE	109 000zł	163 000zł
GPW	TAK	NIE	40 750zł	36 830zł

Tabela 4.9: Wynik finansowy inwestycji w wybrane spółki z mWIG40

Wartość zakupu wszystkich akcji:	328 000zł
Wartość sprzedaży wszystkich akcji:	388 050zł
Zysk netto:	60 050zł (18.31%)

Tabela 4.10: Podsumowanie inwestycji w wybrane spółki z mWIG40

Spółka	Sprzedaż 60 sesji wstecz	Sprzedaż 20 sesji wstecz	Wartość sprzedaży	Wartość zakupu
ALIOR	NIE	NIE	0zł	0zł
TVN	NIE	NIE	0 zł	0 zł
NETIA	NIE	NIE	0zł	0zł
INTERCARS	NIE	TAK	134 000zł	163 000zł
GPW	NIE	NIE	0zł	0zł

Tabela 4.11: Wynik finansowy po sprzedaży wybranych spółek z mWIG40

Wartość sprzedaży wszystkich akcji:	134 000zł
Wartość ponownego zakupu wszystkich akcji:	163 000zł
Zysk netto:	-29 000zł (-21.64%)

Tabela 4.12: Podsumowanie sprzedaży wybranych spółek z mWIG40

Analiza spółek z indeksu sWIG80

Spółka	Cena 20 sesji wstecz	Cena aktualna	Wniosek
WAWEL	861.00	975.00	Kup
AMICA	74.09	90.90	Kup
ATM	11.56	12.15	Brak wniosków
DEBICA	64.46	76.53	Brak wniosków
BENEFIT	275.00	279.45	Sprzedaj

Tabela 4.13: Wyniki analizy spółek z sWIG80 w krótkim terminie

Spółka	Cena 60 sesji wstecz	Cena aktualna	Wniosek
WAWEL	808.95	975.00	Brak wniosków
AMICA	72.67	90.90	Sprzedaj
ATM	11.75	12.15	Brak wniosków
DEBICA	65.41	76.53	Brak wniosków
BENEFIT	259.00	279.45	Brak wniosków

Tabela 4.14: Wyniki analizy spółek z sWIG80 w średnim terminie

Poniżej zestawienie zysków/strat inwestycji prowadzonej zgodnie z zaleceniami systemu dla wybranych spółek z indeksu sWIG80.

Spółka	Zakup 60 sesji wstecz	Zakup 20 sesji wstecz	Wartość zakupu	Wartość sprzedaży
WAWEL	NIE	TAK	861 000zł	975 000zł
AMICA	NIE	TAK	74 090 zł	90 900 zł
ATM	NIE	NIE	0zł	0zł
DEBICA	NIE	NIE	0zł	0zł
BENEFIT	NIE	NIE	0zł	0zł

Tabela 4.15: Wynik finansowy inwestycji w wybrane spółki z sWIG80

Wartość zakupu wszystkich akcji:	935 090zł
Wartość sprzedaży wszystkich akcji:	1 065 900zł
Zysk netto:	130 810zł (13.99%)

Tabela 4.16: Podsumowanie inwestycji w wybrane spółki z sWIG80

Spółka	Sprzedaż 60 sesji wstecz	Sprzedaż 20 sesji wstecz	Wartość sprzedaży	Wartość zakupu
WAWEL	NIE	NIE	0zł	0zł
AMICA	TAK	NIE	72 670 zł	90 900 zł
ATM	NIE	NIE	0zł	0zł
DEBICA	NIE	NIE	0zł	0zł
BENEFIT	NIE	TAK	275 000zł	279 450zł

Tabela 4.17: Wynik finansowy po sprzedaży wybranych spółek z sWIG80

Wartość sprzedaży wszystkich akcji:	347 670zł
Wartość ponownego zakupu wszystkich akcji:	370 350zł
Zysk netto:	-22 680zł (-6.52%)

Tabela 4.18: Podsumowanie sprzedaży wybranych spółek z sWIG80

Podsumowanie inwestycji we wszystkie spółki

Wartość zakupu wszystkich akcji:	1 432 930zł
Wartość sprzedaży wszystkich akcji:	1 629 140zł
Zysk netto:	196 210zł (13.69%)

Tabela 4.19: Podsumowanie inwestycji we wszystkie spółki

Wartość sprzedaży wszystkich akcji:	637 940zł
Wartość ponownego zakupu wszystkich akcji:	674 320zł
Zysk netto:	-36 380zł (-5.70%)

Tabela 4.20: Podsumowanie sprzedaży wszystkich spółek

Jak widać z powyższych podsumowań system pozwala zarobić dzięki inwestycji w akcje giełdowe spółek. W większości przypadków prawidłowo daje sygnały zakupu. Niestety zdecydowanie gorzej wypada w przypadku wyłapywania sygnałów do sprzedaży. Jednak zyski z zakupu akcji bilansują częściowe straty poniesione podczas sprzedaży.

Rozdział 5

Podsumowanie i wnioski

Podczas pisania pracy udało zrealizować wszystkie cele z rozdziału 1.1. Powstał system ekspertowy, którego zadaniem jest prognozowanie przyszłego zachowania rynku finansowego, a konkretniej spółek giełdowych. System ten udało się zrealizować z wykorzystaniem paradygmatu programowania funkcyjnego, co zaowocowało poszerzeniem wiedzy na temat sposobów wytwarzania oprogramowania i tworzenia kodu źródłowego. Użycie języka Clojure [9] było bardzo dobrą decyzją. Jest to język mieszany, tzn. jest dialektem najpopularniejszego języka funkcyjnego - Lisp, jednocześnie daje możliwość korzystania z bibliotek języka Java [10]. Takie połączenie pozwoliło w łatwy sposób stowrzyć graficzny interfejs użytkownika dla aplikacji. Jednocześnie cała logika systemu mogła zostać zaimplementowana zgodnie z podejściem funkcyjnym.

W rozdziale 4.2 przeprowadzono analizę 15 wybranych spółek giełdowych notowanych na GPW. Spółki wybierane były po 5 spośród największych, średnich i małych spółek. Wyniki przedstawione w rozdziale 4.2.2 pokazują, że system w zdecydowanej większości przypadków poprawnie daje sygnału do zakupu akcji konkretnej spółki. Gorzej wypada w przypadku wychwytywania sygnałów sprzedaży. Podsumowując jednak całą inwestycję, zarówno zyski z zakupu akcji jak i straty podczas sprzedaży, mimo wszystko bilans jest dodatni, co świadczy o skuteczności działania systemu w szerokim ujęciu. Z pewnością niezwykle trudnym zadaniem jest skonstruowanie bazy reguł w ten sposób, aby wychwytywać wszystkie sygnały poprawnie. Można nawet pokusić się o stwierdzenie, że jest to zadanie niemożliwe do zrealizowania. Spowodowane jest to tym, że analityk oceniając jakąś spółkę i analizując dla niej wskaźniki często ufa swoim subiektywnym odczuciom. Nie interpretuje każdego wskaźnika zawsze w dokładnie ten sam sposób. Ponadto analityk opiera się na swoim doświadczeniu. Przez to opisanie jego pracy w sposób deklaratywny jest niemożliwe.

Praca pokazuje, że programowanie funkcyjne z powodzeniem może być zastosowane do tworzenia systemów ekspertowych. Dzięki wykorzystaniu list jako podstawowych struktur do przechowywania analizowanych danych w bardzo łatwy sposób można te dane przetwarzać. Pozwalają na to zaawansowane, a zarazem proste

w użyciu, mechanizmy filtrowania list, a także mapowania funkcji na kolejne elementy w liście. W kodzie źródłowym napisanym zgodnie z podejściem funkcyjnym występuje bardzo dużo wywołań rekurencyjnych, jest to cecha charakterystyczna języków funkcyjnych. Dzięki temu kod pisze się łatwiej, ponieważ rekurencja jest dużo bardziej naturalna dla człowieka. Powoduje to, że system może powstawać szybciej, a tworzony kod jest krótszy, ponieważ programista nie zastanawia się jakie kolejne operacje aplikacja ma wykonać, a prawie że naturalnie opisuje czym poszczególne funkcje/struktury są. Języki funkcyjne pozwalają również na generowanie nowego kodu w trakcie wykonywania aplikacji, dzięki czemu dobrze napisany system może się rozwijać nawet bez ingerencji programisty w kod źródłowy. Ponadto kod - funkcje - napisany w sposób funkcyjny, jeśli jest wystarczająco ogólny, daje się bez większego problemu wykorzystać ponownie w przyszłości.

Z pewnością istnieje możliwość dalszych badań w zakresie pracy i rozwoju aplikacji. Pierwszym krokiem byłoby skalibrowanie reguł systemu tak, aby wyłapywał precyzyjniej i w większej ilości sygnały zarówno do zakupu jak i sprzedaży akcji. Kolejny etap to zintegrowanie systemu z notowaniami w czasie rzeczywistym i dostosowanie reguł do prognozowania trendów w ciągu pojedynczej sesji giełdowej. Możliwe jest również rozbudowanie mechanizmu wnioskującego o heurystyki, które przyspieszyłyby proces wnioskowania, co byłoby bardzo istotne w przypadku zleceń zakupu/sprzedaży w ciągu dnia. Dodatkowo istnieje możliwość rozwoju mechanizmu wyjaśniającego tak, aby oprócz listy wygenerowanych faktów przedstawiał dokładniejsze wyjaśnienie procesu wnioskowania, np. w postaci drzewa ze ścieżką uruchamiania kolejnych reguł i dodawania nowych faktów do bazy wiedzy.

Bibliografia

[1] KBibTeX. http://home.gna.org/kbibtex/.

[2] Kile - an Integrated LaTeX Environment. http://kile.sourceforge.net/.

[3] Repozytorium biblioteki clj-http. https://github.com/dakrone/clj-http.

[4] Repozytorium biblioteki clj-time. https://github.com/clj-time/clj-time.

[5] Repozytorium biblioteki instaparse. https://github.com/Engelberg/instaparse.

[6] Repozytorium biblioteki seesaw. https://github.com/daveray/seesaw.

[7] Repozytorium projektu clooj. https://github.com/arthuredelstein/clooj.

[8] Strona domowa biblioteki incanter. http://incanter.org/.

[9] Strona domowa języka clojure. http://clojure.org/.

[10] Strona domowa języka java. http://www.java.com/pl/.

[11] Strona domowa projektu leiningen. http://leiningen.org/.

[12] TeX Live. http://www.tug.org/texlive/.

[13] S. Achelis. *Technical Analysis from A to Z*. 2000.

[14] R. Akerkar. *Knowledge-Based Systems*. 2009.

[15] S. Rajeev C.S. Krishnamoorthy. *Artificial Intelligence and Expert Systems for Engineers*. 1996.

[16] J. Durkin. *Expert Systems: Design and Development*. 1994.

[17] R. Edwards. *Technical Analysis of Stock Trends, Tenth Edition*. 2012.

[18] M. Fogus. *The Joy of Clojure*. 2011.

[19] S. Halloway. *Programming Clojure*. 2012.

[20] P. Jackson. *Introduction To Expert Systems (3rd Edition)*. 1998.

[21] M. Lipovaca. *Learn You a Haskell for Great Good!: A Beginner's Guide*. 2011.

[22] D. Merritt. *Building Expert Systems in Prolog*. 1989.

[23] G. Michaelson. *An Introduction to Functional Programming Through Lambda Calculus*. 2011.

[24] J. Mulawka. *Systemy ekspertowe*. 1997.

[25] J. Murphy. *Analiza techniczna rynków finansowych*. 2008.

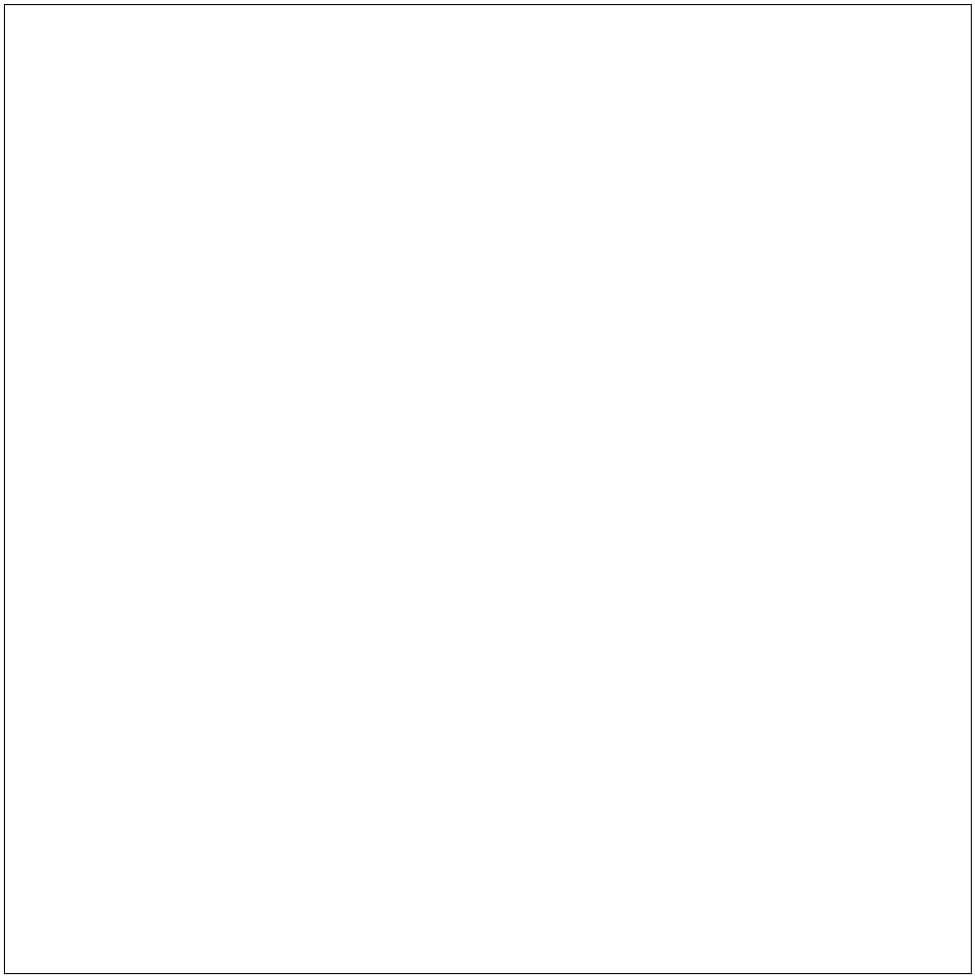
[26] A. Niewiadomski. Szablon pracy dyplomowej. `http://ics.p.lodz.pl/~aniewiadomski/mgr/szablon-konspekt.pdf`, 2010.

[27] T. Oetiker. The not so short introduction to LaTeX2e. `http://tobi.oetiker.ch/lshort/lshort.pdf`, 2011.

[28] C. Okasaki. *Purely Functional Data Structures*. 1999.

Dodatek A

Płyta CD



Zawartość katalogów na płycie:

dat : pliki z danymi wykorzystane w trakcie badań

dist : dystrybucyjna wersja aplikacji przeznaczona do uruchamiania

doc : elektroniczna wersja pracy magisterskiej oraz dwie prezentacje wygłoszone podczas seminarium dyplomowego

ext : aplikacje dodatkowe potrzebne do uruchomienia stworzonej aplikacji (środowisko uruchomieniowe Java)

src : kod źródłowy aplikacji