POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Metod Programowania

Programowanie aplikacji biznesowych

Mateusz Pawłowski

Nr albumu 4427

Wirtualna gra giełdowa na platformie J2ME

Praca inżynierska napisana pod kierunkiem mgr Wojciecha Drabika

Warszawa, luty, 2011

Spis treści

Wstęp

Dziedzina biznesowa.

Pierwsza giełda papierów wartościowych w Warszawie funkcjonowała w latach 1817-1939. Po II wojnie światowej próby utworzenia giełdy nie powiodły się w narzuconym systemie gospodarczym.

Obecna GPW w Warszawie funkcjonuję od 16 kwietnia 1991, jest to data pierwszej sesji giełdowej. GPW jest instytucją która organizuje obrót papierów wartościowych. Umożliwia nabywanie oraz sprzedawanie papierów wartościowych po ustalonych kursach. Kursy papierów wartościowych ustalane są na podstawie zleceń kupna oraz sprzedaży, dlatego rynek giełdowy nazywany jest rynkiem kierowanym zleceniami.[10]

Potrzeba stworzenia systemu.

W obecnej chwili nie ma aplikacji mobilnej która umożliwiałaby wirtualną grę na warszawskiej GPW. Na polskim rynku jest podobna aplikacja mobilna, stworzona przez firmę „statica" która, udostępnia aktualne notowania giełdowe w czasie rzeczywistym, ale nie umożliwia gry. Charakter tej aplikacji jest jedynie informacyjny.

Wirtualna gra giełdowa na urządzenia mobilne ma za zadanie umożliwić użytkownikowi symulacje gry na rzeczywistych danych giełdowych dotyczących akcji oraz wgląd w te dane. Taka symulowana gra, ułatwia zapoznanie się z funkcjonowaniem giełdy i może być pomocna tym którzy planują w przyszłości rozpocząć prawdziwą grę giełdową. Gracz może inwestować w akcje bez obawy o utratę pieniędzy, ponadto w każdej chwili symulacje może rozpocząć od nowa.

Cel pracy.

Celem niniejszej pracy inżynierskiej jest stworzenie i opisanie prototypu systemu umożliwiającego wirtualną grę giełdową na platformie J2ME (Java Platform Micro Edition).

Sposoby pozyskiwania danych giełdowych.

Aby symulacja gry była możliwa konieczny jest dostęp do aktualnych danych giełdowych. Dane warszawskiej GPW, w formie elektronicznej można pozyskiwać na dwa sposoby [11]:

1. Bezpośrednio z GPW
   - Wymaga to podpisania umowy z GPW o dystrybucję serwisów giełdowych. Jest to rozwiązanie płatne. Dane z giełdy przesyłane są za pomocą sygnału satelitarnego. Wymagana jest instalacja anteny oraz odbiornika wraz z komputerem posiadającym oprogramowanie umożliwiające odbiór danych.
2. Od innego niż GPW dystrybutora.
   - W tym przypadku należy skontaktować się z dystrybutorem i uzgodnić sposób i warunki komercyjne przesyłania informacji. GPW nie pośredniczy i nie ingeruje w takim przypadku o ile warunki nie naruszają wcześniejszej umowy GPW z dystrybutorem.

Zarówno dane bezpośrednio z GPW jak i od innego dystrybutora mogą być przesyłane w dwóch trybach aktualizacji o ile dystrybutor je udostępnia.

- Przesyłanie danych w czasie rzeczywistym.
- Przesyłanie danych z 15-minutowym opóźnieniem.

Do dystrybutorów innych niż GPW należą między innymi:

- agencje informacyjne
- domy maklerskie
- portale internetowe np. „www.onet.pl www.bossa.pl www.interia.pl" najczęściej udostępniają dane opóźnione 15 minut.

Informacja ze strony „www.gpw.pl„ dotycząca autoryzacji i udostępniania danych opóźnionych. „Dane opóźnione pozyskane ze źródeł innych, niż GPW lub jeden z autoryzowanych przez nią dystrybutorów, mogą być udostępniane klientom bez podpisywania umowy z GPW (a co za tym idzie bez autoryzacji GPW). Rozwiązanie takie nie pociąga za sobą żadnych opłat na rzecz GPW, Giełda jednak nie może być wskazana jako źródło informacji."

Ograniczenia.

Ze względu na bezpłatny dostęp do danych giełdowych, zdecydowano że prototypu będzie korzystać z danych opóźnionych 15 minut, dostępnych na portalu „www.onet.pl„ w postaci pliku o formacie „csv".

Dane historyczne zostaną pobrane z serwisu „www.bossa.pl„ w postaci plików o formacie „mst" a następnie skonwertowane na potrzeby systemu.

Określenie wymagań kanałów komunikacji dla aplikacji.

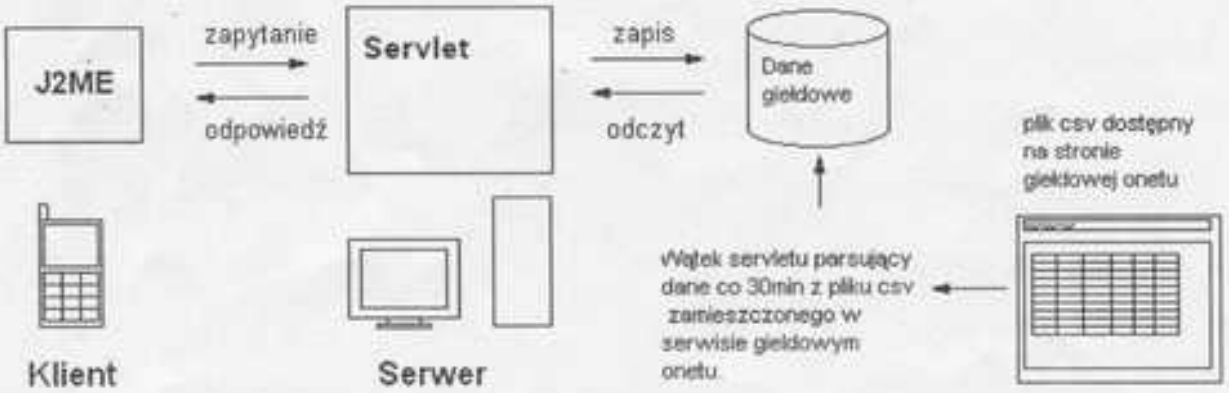
Architektura klient-serwer.

Jest to architektura rozdzielenia funkcjonalności aplikacji działającej w sieci na dwie lub więcej warstw. [13] Klienci komunikują się z serwerem wysyłając żądania które następnie są realizowane przez ten serwer.

- Architektura klient-serwer dwu-warstwowa..
- Klient pełni funkcję warstwy prezentacji. Znajduje się w niej interfejs-użytkownika
- Funkcjonalność serwera znajduje się w jednej warstwie.
- Architektura klient-serwer trój-warstwowa. [14]
- Klient pełni funkcję warstwy prezentacji. Znajduje się w niej interfejs-użytkownika.
- Funkcjonalność serwera rozdzielona jest na dwie warstwy.
- Warstwę logiki biznesowej – odpowiedzialną za przetwarzanie danych
- Warstwę danych – odpowiedzialna za składowanie i udostępnianie danych.

Podział systemu.

Z powodu małej ilości pamięci trwałej jaką dysponują urządzenia mobilne zdecydowano aby system działał w architekturze klient-serwer (Rys.1). Urządzenie mobilne, które jest urządzeniem słabszym o mniejszych możliwościach (klient) będzie komunikować się z urządzeniem mocniejszym udostępniającym dane giełdowe (serwer). Ponadto serwer będzie aktualizował dane i składował je. W pamięci trwałej urządzenia mobilnego będą przechowywane jedynie informacje o stanie konta gracza oraz posiadanych przez niego akcjach. Zarówno aktualne dane giełdowe jak i historyczne będą udostępniane przez serwer na żądanie klienta.

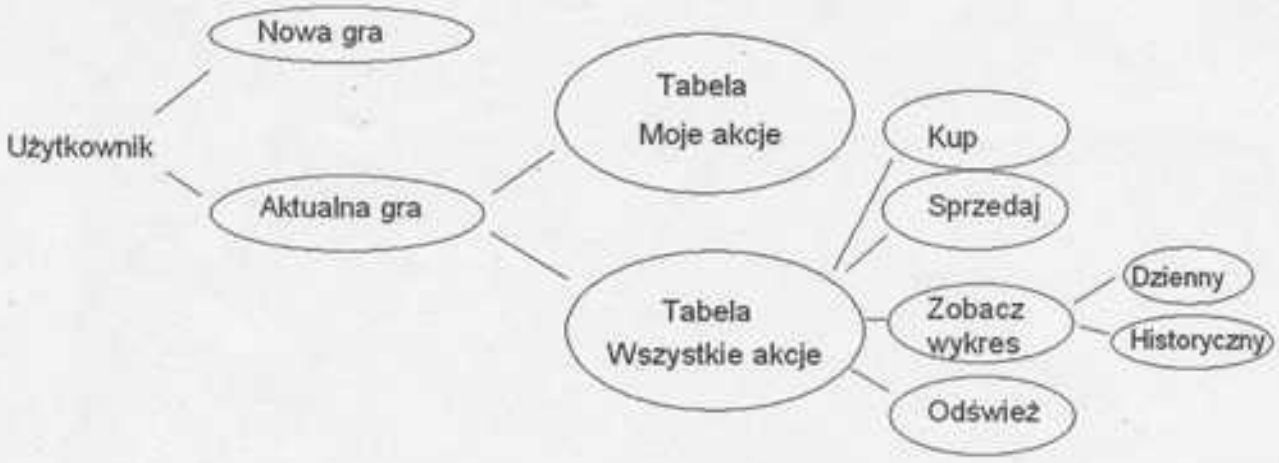


Rys. 1. Uproszczony schemat przedstawiający podział systemu.

Aplikacja po stronie klienta - (midlet na platformie J2ME).

Java Platform Micro Edition – J2ME – jest najczęściej spotykaną platformą programistyczną dla urządzeń mobilnych. Została zaprojektowana specjalnie dla urządzeń mobilnych takich jak telefony komórkowe i palmtopy [9]. Większość telefonów komórkowych dostępnych na rynku posiada maszynę wirtualną umożliwiającą uruchamianie programów na platformie J2ME, dlatego zdecydowano aby aplikacja po stronie klienta działała w oparciu o tę platformę.

- Funkcjonalność aplikacji klienta (Rys. 2). Aplikacja klienta powinna umożliwiać:
- Utworzenie nowej gry (tworzy nowy profil gracza wraz ze startową sumą pieniędzy, w przypadku gdy istniała wcześniej aktualna gra zostaje zresetowana nowa gra staje się aktualną)
- Kontynuacje bieżącej gry. (umożliwia dalszą grę na bieżącym profilu gracza)
- Podczas aktualnej gry
- Przeglądanie tabeli moje akcje. (tabela przechowująca informacje o ilości akcji posiadanych przez gracza zapisywana w profilu gracza)
- Przeglądanie tabeli wszystkie akcje (zawierającej aktualne kursy akcji)
- Możliwość odświeżenia tabeli.
- Kupno wybranej akcji
- Sprzedaż wybranej akcji
- Dla wybranej akcji możliwość oglądania wykresów
- Dziennego (aktualizowany co 30min)
- Historycznego (stworzony na podstawie kursów zamknięcia kolejnych dni)
- Informacja o aktualnym stanie konta gracza.



Rys. 2. Przypadki użycia dla aplikacji klienta.

Aby aplikacja realizowała powyższe funkcje, w komunikacji z serwerem będzie mogła wysyłać żądania o następujące dane.

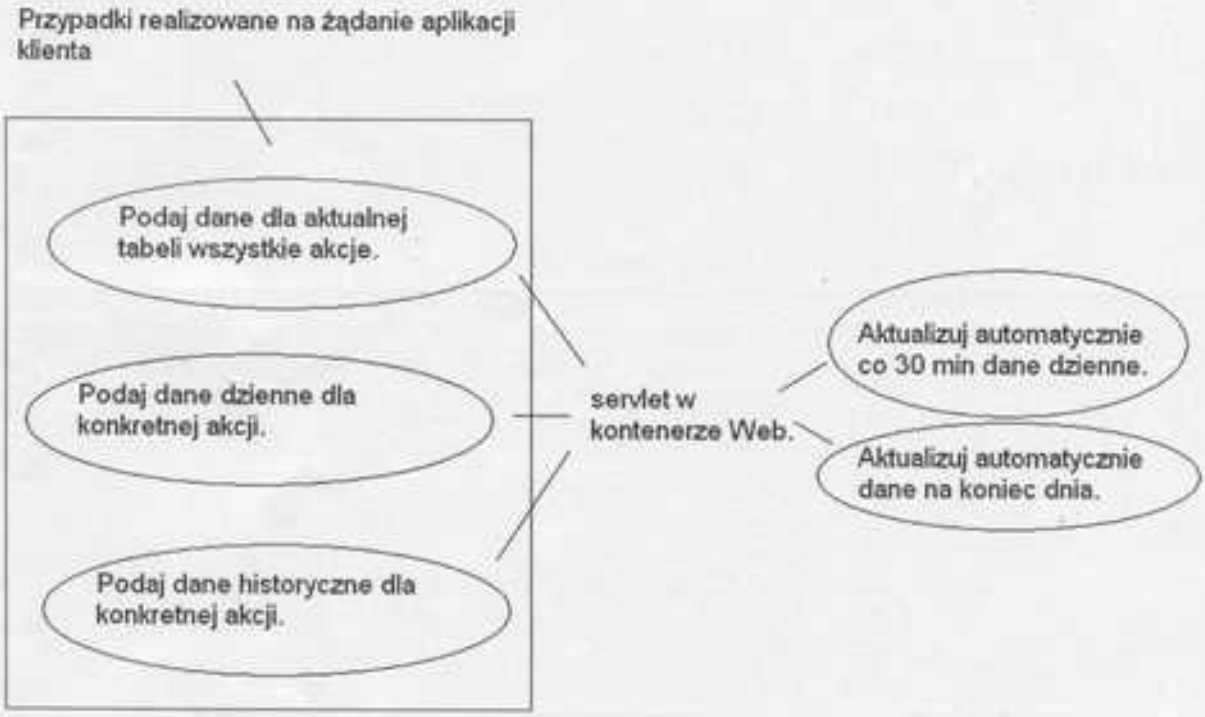
- Pobierz dane potrzebne do stworzenia aktualnej tabeli wszystkie akcji.
- Pobierz dane potrzebne do stworzenia wykresu dziennego dla konkretnej akcji.
- Pobierz dane potrzebne do stworzenia wykresu historycznego dla konkretnej akcji.

Aplikacja po stronie serwera - (serwlet w kontenerze Web).

Serwlet: jest klasą napisaną w języku Java która działa w modelu (żądanie, odpowiedź) przyjmuje żądania klientów, wykonuje swoje działanie na serwerze a następnie przesyła odpowiedź do klienta. Zarówno J2ME jak i „Java Servlet Technology" umożliwiają komunikacje za pomocą protokołu „HTTP". Klient wysyła żądania za pomocą poleceń „get" lub „post". Takie rozwiązanie sprawia że klientem może być dowolny program korzystający z protokołu „HTTP" np.:

- przeglądarka internetowa
- program klient na komputerze klasy PC
- program klient na urządzeniu mobilnym

- Funkcjonalność aplikacji serwera (Rys. 3). Servlet powinna umożliwiać:
- Automatyczną cykliczną np.(co 30 minut) aktualizacje danych giełdowych dziennych z internetu i ich składowanie.
- Automatyczną aktualizacje danych giełdowych z końca każdego dnia i ich składowanie.
- Udostępnienie aktualnie dostępnych na serwerze kursów dla wszystkich akcji.
- Udostępnienie dla wybranej akcji danych
- dziennych
- historycznych



Rys. 3. Przypadki użycia dla aplikacji serwera.

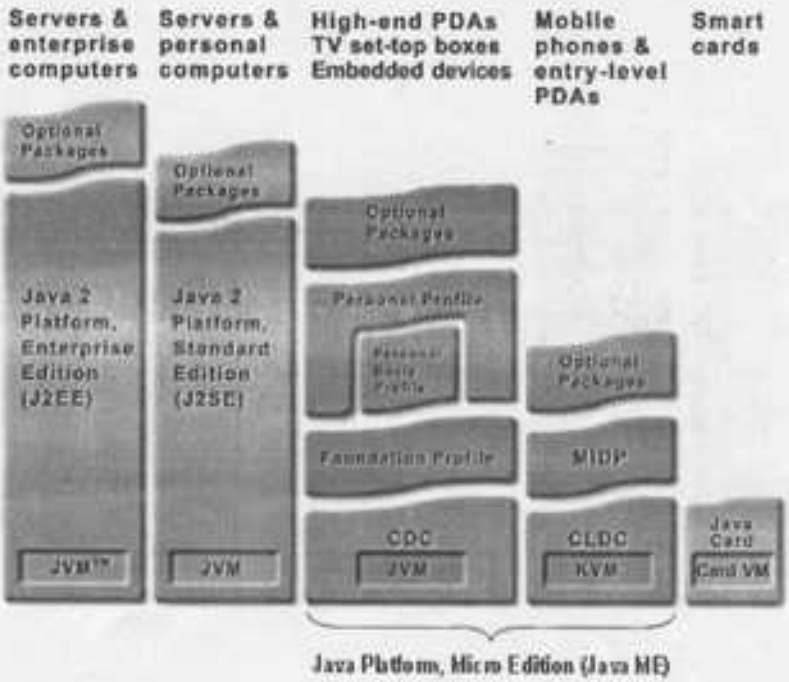
Wybrane technologie.

J2ME (Java Platform, Micro Edition)

Obecna skrócona nazwa technologi to Java ME (wcześniejsza J2ME) - Java Platform Micro Edition to platforma (zbiór technologi i specyfikacji) języka programowania java zaprojektowana specjalnie dla urządzeń mobilnych przez firmę Sun Microsystems [8]. Technologię J2ME tworzą trzy podstawowe elementy (konfiguracje, profile, pakiety opcjonalne) (Rys. 4).

- Konfiguracje to specyfikacje które opisują podstawowe biblioteki oraz możliwości maszyny wirtualnej dla różnych urządzeń.
- Profile uzupełniają konkretną konfigurację o wyspecjalizowane API. Profil uzupełniając konfiguracje tworzy środowisko w którym działa aplikacja. np. Konfiguracja CLDC i profil MIDP to połączenie które występuje najczęściej.
- Pakiety opcjonalne to dodatkowe API

API definicja - „(ang. Application Programming Interface) - interfejs programowania aplikacji, sposób, w jaki komunikuje się ona z systemem operacyjnym i innymi programami. " (źródło wikipedia - „http://pl.wikipedia.org/wiki/API_%28informatyka%29")



Rys. 4. Porównanie platform (J2EE, J2SE, Java ME) i ich komponentów. (źródło „http://java.sun.com/javame/technology/index.jsp")

Konfiguracje i profile.[9],[5].

Platforma Java ME została podzielona na dwie podstawowe konfiguracje CLDC oraz CDC. (Rys.4)

- CLDC - „Connected Limited Device Configuration" konfiguracja stworzona dla urządzeń o małej mocy obliczeniowej i mniejszej pamięci takich jak (telefony komórkowe, pagery, PDA). Urządzenia te:
  - Korzystają z maszyny wirtualnej KVM (Kilobyte Virtual Machine)
  - Dysponują pamięcią 32kb - 512kb
  - Posiadają procesory 16 – 32 bitowe które pracują z częstotliwością od 16-60MHz

- CDC - „Connected Device Configuration" konfiguracja stworzona dla urządzeń mocniejszych takich jak (palmtopy, video telefony, komunikatory, set-top-box) które:
  - Korzystają ze standardowej maszyny wirtualnej JVM.
  - Dysponują pamięcią 1MB-10MB

Profile wchodzące w skład konfiguracji CLDC

- MIDP-(Mobile Information Device Profile) (JSR118)– najczęściej stosowany profil, ze względu na najszersze pole urządzeń na których działa. (telefony komórkowe, PDA). Charakterystyka dla tego profilu w wersji (MIDP 2.0) to:
  - Wsparcie dla poprzednich wersji MIDP 1.0
  - Wyświetlacz o minimalnym rozmiarze (96x54 pixeli) i minimum 2 kolorach.
  - Minimum 128kb pamięci ulotnej na uruchomienie aplikacji.
  - Bezprzewodowy dwukierunkowy dostęp do sieci. Możliwość nawiązywania połączenia z wykorzystaniem protokołu HTTP.
  - Możliwość zapisu i odczytu danych do pamięci trwałej urządzenia z wykorzystaniem RMS - (Record Management System).
  - Zdolność odtwarzania i generowania dźwięków.
  - Urządzenie musi umożliwiać sterowanie za pomocą klawiatury lub ekranu dotykowego.

- IMP - (Information Module Profile) (JSR195) – profil stworzony dla urządzeń o ograniczonych możliwościach wyświetlaczy lub nawet ich braku. Wykorzystują go maszyny sprzedające, systemy ochrony, telefony awaryjne itp.

Profile wchodzące w skład konfiguracji CDC

- Foundation Profile (JSR219) – profil stworzony dla aplikacji które wykorzystują pełne możliwości JVM oraz API platformy J2SE.

- Personal Basis Profile (JSR217) – jest rozszerzeniem Foundation Profile, które wspiera tworzenie lekkiego GUI (graficzny interfejs-użytkownika) wykorzystuje niepełne AWT-(Abstract Window Toolkit – zbiór bibliotek graficznych Java).

- Personal Profile (JSR216) – jest rozszerzeniem Personal Basis Profile wykorzystującym pełne AWT.

(JSR) – Java Specification Request – dokumenty towrzone w ramach Java Community Process opisują różne specyfikacje platformy Java oraz ich rozwój. .

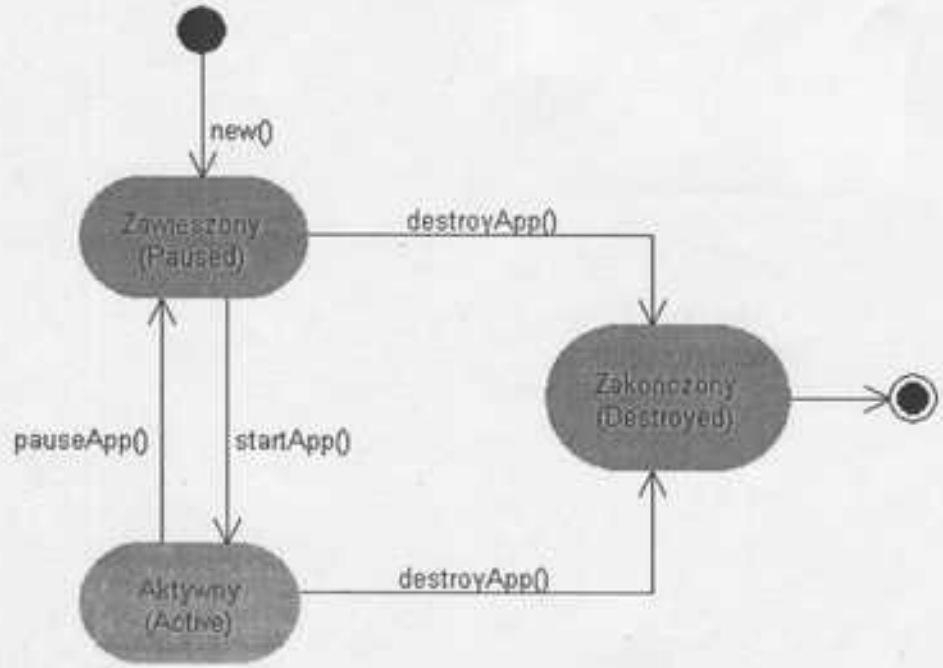
Cykl życia Midletu.[3]

MIDlet to aplikacja mobilna, która działa w oparciu o profil MIDP pod kontrolą AMS [3].
AMS („Application management software") Oprogramowanie znajdujące się na urządzeniu mobilnym, dostarcza środowisko JRE w którym działa MIDlet, zarządza cyklem życia aplikacji, obsługuje błędy oraz zdarzenia zewnętrzne jak np. przychodzącą rozmowę telefoniczną.

Główna klasa MIDlet'u musi dziedziczyć po klasie Midlet z pakietu javax.microedition.midlet i definiować następujące trzy metody startApp(), pauseApp(), destroyApp() odpowiedzialne za stan aplikacji (Rys. 5). Możliwe stany w jakich może znaleźć się aplikacja:

- paused – gdy instancja midletu istnieje i jest nieaktywna.
- active - gdy aplikacja jest aktywna.
- destroyed – gdy aplikacja została zakończona.

Początkowy stan MIDlet'u to paused dopiero AMS w pewnym momencie po utworzeniu wywołuje metodę startApp(). Może się zdarzyć że AMS w odpowiedzi na przychodzącą rozmowę wprowadzi aplikacje w stan paused wywołując metodę pauseApp(). Stany active oraz paused mogą być wywołane wiele razy podczas cyklu życia MIDlet'u stan destroyed tylko raz.



Rys 5. Schemat przedstawiający stany MIDlet'u oraz przejścia między nimi (źródło „http://www.midlety.net/site/teoria/MIDP10.html")

GUI (niskopoziomowe, wysokopoziomowe) zalety + wady. [7]

J2ME posiada specjalne API wspomagające tworzenie GUI(Graficzny interfejs-użytkownika). Z poziomu logicznego można wyróżnić:

- API wysokopoziomowe - posiadające gotowe komponenty takie jak np: TextBox, Alert, List, Form. Zaletą tego API jest szybkość tworzenia oprogramowania, oraz jego przenośność na inne urządzenia. Dostajemy gotowe do użycia obiekty. Wadą jest brak możliwości ingerowania w sposób wyświetlania takich gotowych komponentów. Zdarza się też że ten sam komponent wygląda inaczej na różnych urządzeniach.
- API niskopoziomowe - dające możliwość pełnej kontroli nad grafiką. Programista otrzymuje obiekt Canvas - płótno po którym może rysować za pomocą różnych funkcji graficznych, może także wyświetlać grafikę bitmapową. Mamy dostęp do każdego pixel'a. API niskopoziomowe najczęściej jest stosowane w grach oraz w aplikacjach w których ważna jest prezentacja. Wadą jest dłuższy czas tworzenia oprogramowania związany z utworzeniem własnych komponentów.

RMS – zapis danych w pamięci urządzenia mobilnego. [6]

RMS (Record Management System) to specjalne API (pakiet javax.microedition.rms) dla platformy J2ME umożliwiający zapis oraz odczyt danych z pamięci trwałej urządzenia mobilnego. Oznacza to trwałość danych po zakończeniu działania MIDlet'u i możliwość późniejszego odczytu przy kolejnym uruchomieniu aplikacji. RMS to obecnie jedyny sposób dostępu do pamięci trwałej z poziomu MIDlet'u. Dane przechowywane są za pomocą rekordów w bardzo prostej bazie danych- „record store" którą obrazuje tabelka nr 1.

ID rekordu	Dane
1	Tablica bajtów
2	Tablica bajtów
3	Tablica bajtów
...	...

Tabelka nr. 1. źródło „http://www.ibm.com/developerworks/library/j-j2me3/"

ID rekordu - liczba typu integer. Rekordy w ramach danego record store numerowane są od 1 rosnąc o 1 (1,2,3 ... itd.). W przypadku usunięcia np. rekordu nr 2 powstaje luka numery pozostałych rekordów pozostają bez zmian. Dodany do takiego zestawu 1,3 nowy rekord będzie oznaczony jako nr 4. Dlatego przy odczycie rekordów zamiast klasycznego inkrementowania od 1 do n należy zastosować enumerator.

Dane - dane przechowywane są w tablicy bajtów, dlatego też muszą być wcześniej zamienione na taki typ danych. Konwersje może ułatwić zastosowanie klasy-strumieniowej ByteArrayOutputStream wraz z klasą DataOutputStream.

Baza danych-"record store" rozpoznawana jest po nazwie, która może mieć do 32 znaków. W ramach jednego MIDlet Suite nie może być dwóch baz o tej samej nazwie. MIDlet Suite czyli zestaw (jest to kolekcja jednego lub więcej MIDlet'ow spakowanych razem w archiwum jar). W przypadku usunięcia MIDlet Suite wszystkie rekordy stworzone w ramach tego zestawu także zostaną usunięte. Rekordy w bazie danych mogą być współdzielone zarówno w ramach tego samego zestawu MIDlet'ów jak i z innego MIDlet' Suite. Rodzaj współdzielenia można zdefiniować podczas tworzenia danego record store.

Tworzenie „record store"

„przykład 1.

RecordStore openRecordStore(String recordStoreNazwa, boolean createIfNecessary)

przykład 2.

RecordStore openRecordStore(String recordStoreNazwa, boolean createIfNecessary, int authmode, boolean writable)

przykład 3.

RecordStore openRecordStore(String recordStoreNazwa, String vendorName, String suiteName)"

Listingi metod tworzenia record store źródło „http://www.ibm.com/developerworks/library/j-j2me3/"

Pierwszy parametr typu String recordStoreNazwa określa nazwę zbioru rekordów. Drugi parametr typu boolean createIfNecessary, określa czy należy utworzyć zbiór w przypadku gdyby nie istniał. W przykładzie drugim mamy dwa dodatkowe parametry authmode oraz writable. Parametr authmode określa dostęp do zbioru i może przyjąć wartość RecordStore.AUTHMODE_ANY – dostęp ma każdy zestaw MIDlet'ów lub RecordStore.AUTHMODE_PRIVATE – dostęp ma tylko ten zestaw który sam utworzył zbiór. Parametr writable definiuje czy zapis może być dokonany przez inne zestawy MIDlet'ów. Przykład trzeci pokazuje możliwość otworzenia record store stworzonego przez inny zestaw. Parametry vendorName-nazwa sprzedawcy oraz suiteName-nazwa zestawu muszą być zgodne z manifestem MIDlet Suite.

Zapis, odczyt, usuwanie danych z record store.

Zapis do istniejącego zbioru umożliwia metoda addRecord w klasie RecordStore

np. *nowy_record_store.addRecord(rekord,0,rekord.length)*

Pierwszy parametr rekord to tablica bajtów z danymi, drugi i trzeci określają zakres w tablicy bajtów, który chcemy zapisać. Poprawnie wykonana metoda powinna zwrócić ID nowego rekordu.

Metoda setRecord umożliwia zmiane istniejącego rekordu o danym ID.

np. *nowy_record_store.setRecord(ID,rekord,0,rekord.lenght)*

Odczyt rekordu o podanym ID do tablicy bajtów umożliwia metoda getRecord

np. *byte[]tablica_bajtow=nowy_record_store.getRecord(ID)*

Usuwanie pojedynczego rekordu

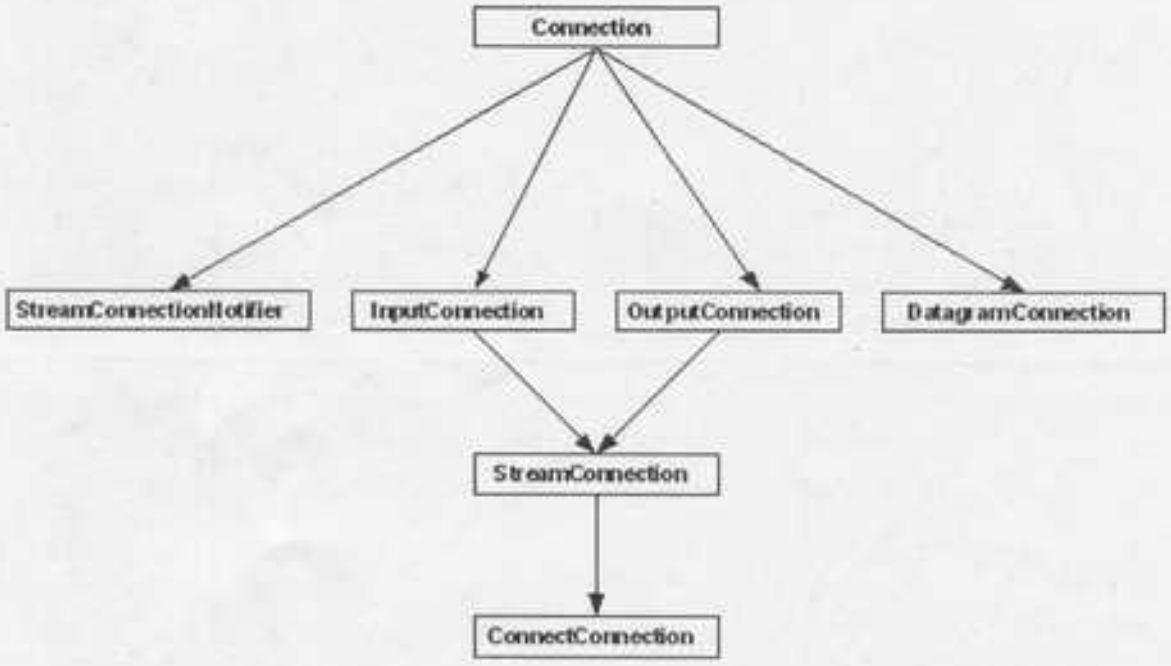
np. *nowy_record_store.deleteRecord(ID)*

Usuwanie całego zbioru

np. *RecordStore.deleteRecordStore(nazwa_zbioru)*

Komunikacja z web aplikacją (komunikacja z serwletem). [2]

Aby J2ME mogło wspierać możliwie dużo różnych urządzeń mobilnych do konfiguracji CLDC oraz CDC wprowadzono GCF - (Generic Connection framework) (Rys. 6). GCF definiuje w sposób możliwie najprostszy (na wysokim poziomie) hierarchie interfejsów i klas potrzebnych do wykonywania operacji sieciowych i plikowych na urządzeniu mobilnym. Implementacja interfejsów oraz decyzja o tym które będą zaimplementowane zależy od producenta konkretnego urządzenia.



Rys. 6. Interfejsy wchodzące w skład „Generic Connection framework" źródło („http://www.wirelessdevnet.com/channels/java/features/j2me_http.phtml")

Jest jedna klasa Connector która fabrykuje obiekty typu Connection, których jest siedem

- Connection - główny interfejs z którego dziedziczą pozostałe.
- ContentConnection
- DatagramConnection
- InputConnection
- OutputConnection
- StreamConnection
- StreamConnectionNotifier

Interfejsy wchodzące w skład GFC znajdują się w pakiecie „javax.microedition.io".
Rodzaj połączenia zależy od parametrów wprowadzonych w metodzie open obiektu Connector.
np. Connector.open(String polaczenie)
parametr polaczenie jest zmienna typu String odczytywaną zgodnie z formatem URL
(protokół:adres:paramtery).

Przykładowe utworzenie połączenia do pliku:

```
Connection polaczenie = Connector.open(„file://pik.txt");
```

Przykładowe połączenie http:

```
Connection polaczenie_http = Connector.open(„http://www.adres_strony.pl");
```

Aby MIDlet mógł komunikować się z serwletem musi być wstanie posyłać polecenia metodą GET wykorzystując protokół HTTP a następnie odczytać odpowiedź.

Klasa HttpConnection umożliwia obsługe połączeń HTTP. HttpConnection nie jest częścią GFC ale dziedziczy z ContentConnection i jest zdefiniowana w profilu MIDP.

Java Servlet Technology.

Technologia umożliwiająca tworzenie aplikacji WEB w języku Java to Java Servlet Technology.

Web aplikacja servlet.

Serwlet: jest klasą napisaną w języku Java która działa w modelu (żądanie, odpowiedź) przyjmuje żądania klientów, wykonuje swoje działanie na serwerze a następnie przesyła odpowiedź do klienta [12]. Servlet stanowi warstwę, która pośredniczy pomiędzy żądaniami wysyłanymi przez przeglądarkę WWW lub program kliencki korzystający z protokołu HTTP a np. bazą danych lub aplikacją wykonywaną na serwerze HTTP [4]. Servlety działają w środowisku serwera aplikacji, który zarządza ich działaniem np.(ładowanie i wywoływanie, obsługa żądań, bezpieczeństwo). Aby servlet działał musi zostać wpierw wdrożony w serwerze aplikacji oraz sam serwer aplikacji musi być uruchomiony. Ponieważ cały proces jest trochę skomplikowany, można skorzystać z odpowiedniego IDE np. NetBeans które automatyzuje wdrażanie i uruchamianie.
Klasa Servlet'u musi dziedziczyć po klasie HttpServlet i przesłaniać metody doGet oraz doPost.

Przykładowy fragment kodu serwletu, który po wywołaniu za pomocą Get zwróci napis Witaj.

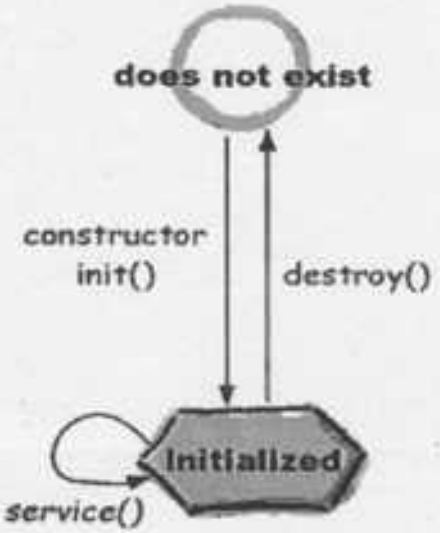
```
„import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PrzykladowyServlet extends HttpServlet {
        public void doGet(HttpServletRequest zapytanie, HttpServletResponse odpowiedz) throws
        ServletException, IOException {
                PrintWriter out = odpowiedz.getWriter();
                out.println(„Witaj");
        }
}"
```

Cykl życia serwletu. [1]

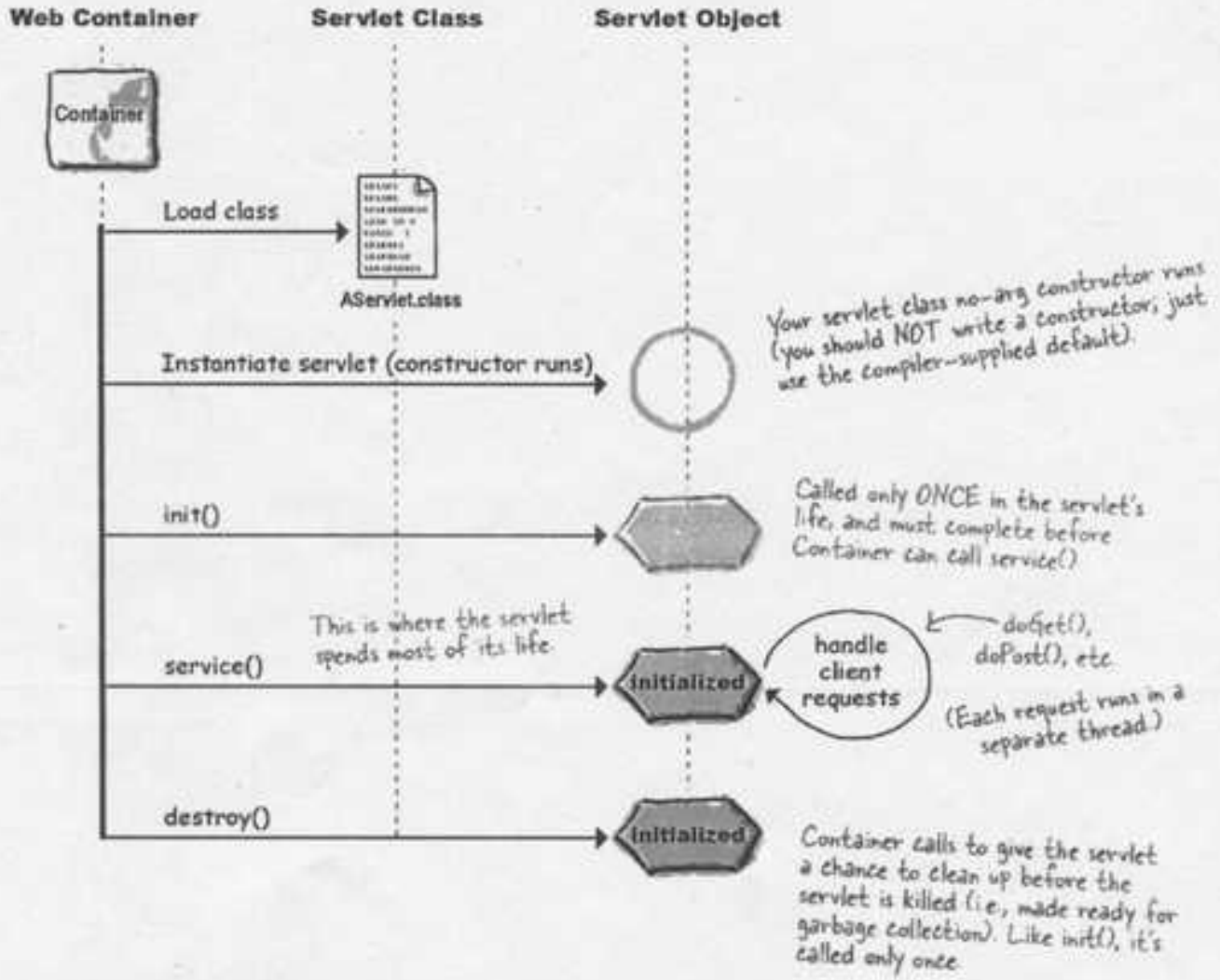
Serwlet może się znajdować w jednym głównym stanie zainicjalizowany (Rys. 7).

Jeśli serwlet nie znajduje się w stanie zainicjalizowany, oznacza to że znajduje się w stanie inicjalizowany (gdy wykonywana jest metoda init) albo w stanie niszczony (gdy wykonywana jest metoda destroy) albo po prostu nie istnieje.



Rys 7. Rysunek przedstawia stany w jakich może znajdować się serwlet oraz metody wywołujące określony stan. Źródło - „Bryan Basham, „Head First Servlets & JSP", O'Reilly, 2004".

W procesie tworzenia serwletu (Rys. 8), pierwszą czynnością którą wykonuje kontener webowy jest wczytanie klasy serwletu, następnie tworzony jest obiekt serwletu po czym wywołana zostaje metoda init(). Metoda init() jest wywołana tylko raz w cyklu życia serwletu i służy czynnościom które powinny być wykonane jeszcze przed wykonaniem metody service() np. łączenie z bazą danych. Po zakończeniu procesu inicjalizacji serwlet znajduje się w stanie zainicjalizowany i oczekuje na żądania klienta. W momencie wysłania przez klienta żądania GET lub POST tworzony jest osobny wątek i wywołana metoda service() która wywołuje metodę doGet lub doPost w zależności od nagłówka żądania. Metoda destroy może być wywołana tylko raz i służy do usunięcia z pamięci kopi serwletu. Destroy może być wywołane celowo na żądanie lub z powodu nieaktywności serwletu przez dłuższy czas.



Rys. 8. Proces tworzenia serwletu zarządzany przez kontener webowy. Źródło - „Bryan Basham, „Head First Servlets & JSP", O'Reilly, 2004".

Komunikacja za pomocą protokołu http (GET i POST). [1],[4].

Komunikacja do serwletów opiera się na odbieraniu żądań posyłanych (najczęściej metodą GET lub POST) za pomocą protokołu http. W najprostszym przypadku aby wysłać polecenie GET wystarczy w pasku adresowym przeglądarki internetowej wpisać adres serwletu wraz z parametrami które chcemy przesłać do serwletu.

Np."http://serwer/moj_serwlet?parametr1=a&parametr2=3&imie=Adam"

Parametry znajdują się po znaku zapytania i oddzielone są znakiem ampersand. W powyższym przykładzie widać adres serwletu http://serwer/moj_serwlet oraz 3 parametry po znaku zapytania parametr 1 o wartości a, parametr 2 o wartości 3 i parametr imię o wartości Adam. Jeżeli serwlet był zainicjalizowany kontener wywoła metodę service która wywoła metodę doGet wraz z obiektem HttpServletRequest i HttpServletResponse. Wynik zależy od kodu który zaimplementujemy w metodzie doGet. Korzystając z obiektu HttpServletRequest możemy pobrać parametry zapytania i

na ich podstawie wygenerować odpowiednią odpowiedź korzystając z HttpServletRequest.

Przykładowy fragment kodu serwletu reagującego na metodę GET i witającego użytkownika po imieniu wprowadzonym w parametrze imie.

```
„import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PowitanieServlet extends HttpServlet {
        public void doGet(HttpServletRequest zapytanie, HttpServletResponse odpowiedz) throws
        ServletException, IOException {
                PrintWriter out = odpowiedz.getWriter();

                String wartosc_parametru_imie = request.getParameter(„imie");
                out.println(„Witaj "+wartosc_parametru_imie);
        }
}"
```

W przypadku żądania GET parametry dołączane były do części adresu. Żądanie POST zawiera dodatkowe ciało w którym są umieszczane parametry, dzięki temu można przesłać większą liczbę informacji. Aby wysłać żądanie POST z przeglądarki użytkownik powinien wysłać np. formularz z ustawioną metodą METHOD="POST".
Protokół HTTP oferuje także inne metody które można wykorzystać w programowaniu serwletów.

Metody zleceń HTTP:

- GET – Żądanie o zasoby znajdujące się pod podanym adresem URL.
- POST – Umożliwia wysłanie danych o nielimitowanej długości pod podany adres URL.
- HEAD – Żąda zwrócenia nagłówków.
- TRACE – Żąda zwrócenia komunikatu żądania.
- PUT – Żąda umieszczenia zasobu pod wskazany adres.
- DELETE – Żąda usunięcia zasobu.
- OPTIONS – Zwraca listę metod na które zasób pod podanym adresem może odpowiedzieć.

Implementacja.

Aplikacja klienta.

Struktura aplikacji.

Główną klasą aplikacji jest Gielda_klient_v2 która rozszerza klase Midlet. Wewnątrz tej klasy tworzony jest obiekt klasy Plotno i obiekt klasy Display oraz zostaje przekazane wyświetlanie grafiki na Plotno.

```
„public class Gielda_klient_v2 extends MIDlet implements CommandListener{
public Gielda_klient_v2() throws IOException {
                    plotno=new Plotno(true,this);

                    ekran = Display.getDisplay(this);
                    ekran.setCurrent(plotno);
```

}„ Fragment głównej klasy Gielda_klient_v2 Midletu.

Wewnątrz obiektu Plotno tworzone są pozostałe obiekty potrzebne do działania aplikacji.

Sterowanie aplikacją.

Sterowanie aplikacją odbywa się za pomocą kursorów kierunkowych joysticka telefonu. (Góra, dół, lewo, prawo), przycisku fire służącemu do akceptacji aktualnego wyboru oraz cyferblatu w przypadku wprowadzania wartości liczbowych. Klasa Plotno dziedziczy po klasie GameCanvas i umożliwia rysowanie oraz sterowanie aplikacją za pomocą przycisków telefonu ponadto implementuje interfejs Runnable tworząc główny wątek aplikacji. Wątek klasy Plotno jest jakby główną pętlą aplikacji w której cyklicznie sprawdzany jest stan klawiatury i odświeżana grafika.

```
„ public void run() {
        while(true){
                    this.klawiatura();
                    this.paint(g);
            try
            {
            Thread.sleep(1);
            }catch(InterruptedException e){}
        }
```

}" Fragment metody run klasy Plotno.

Wewnątrz obiektu klasy Plotno znajdują się referencje do pozostałych obiektów tworzonych na potrzeby aplikacji. Te pozostałe obiekty implementują interfejs o nazwie Interfejs, który służy do sterowania aplikacją.

```
„public interface Interfejs {
        public void gora();
        public void dol();
        public void lewo();
        public void prawo();
        public Interfejs fire();
```

```
public void cyfra_0();
public void cyfra_1();
public void cyfra_2();
public void cyfra_3();
public void cyfra_4();
public void cyfra_5();
public void cyfra_6();
public void cyfra_7();
public void cyfra_8();
public void cyfra_9();
public void cyfra_cofnij();
public String zwroc_nazwe();
public void rysuj(Graphics g);
```

}" Kod interfejsu Interfejs.

Dzięki takiemu rozwiązaniu z metody klawiatura obiektu klasy Plotno można sterować aplikacją odwołując się tylko do interfejsu „inter" a nie do konkretnego obiektu.

np. „

```
if ((klawisze & LEFT_PRESSED)!=0)
    {
        if(anykey==false) {inter.lewo();}
    }
```

" Fragment metody klawiatura klasy Plotno czytającej przyciski.

Aby zmienić obiekt którym chcemy sterować należy przypisać interfejs do innego obiektu. Dzieje się to najczęściej po wciśnieciu przycisku fire na jakiejś opcji. Pod Interfejs inter zostaje przypisany jakiś inny obiekt implementujący Interfejs.

„

```
if ((klawisze & FIRE_PRESSED)!=0)
    {if(anykey==false)
        {
            if (inter.zwroc_nazwe()=="Wyjdź"){this.gra.notifyDestroyed();}
            if (inter.fire()!=null) inter=inter.fire();
        }}
```

" Fragment metody klawiatura klasy Plotno czytającej przyciski.

Własny interfejs-użytkownika GUI.

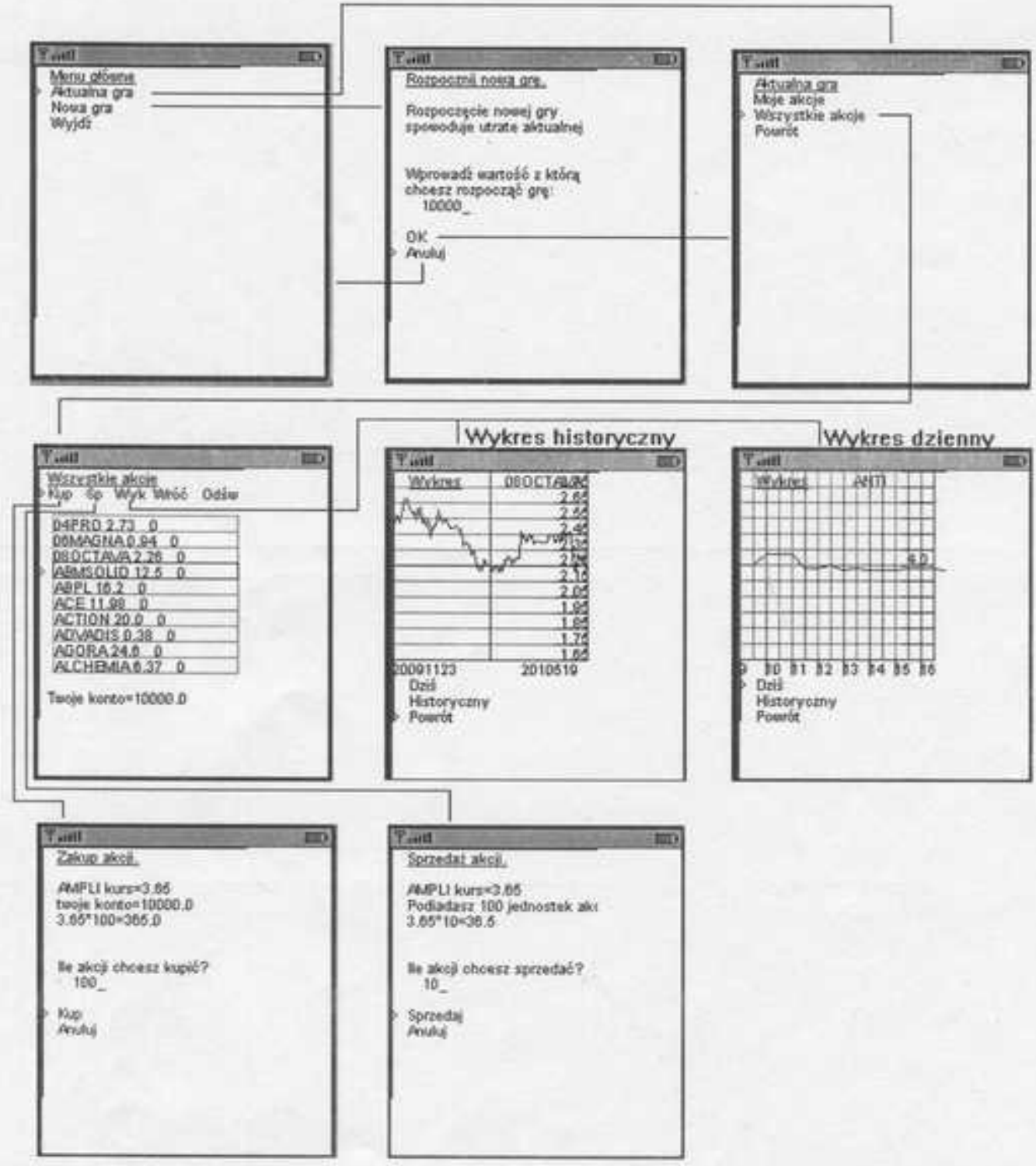
Na potrzeby aplikacji stworzono specjalne klasy (Rys. 10) które implementują interfejs sterujący „Interfejs". Klasy te to Menu, Menu_wszystkie_akcje, Cyfry_box, Cyfry_box_kup, Cyfry_box_sprz, Wykres.

Rys. 9. Klasa Menu.

Klasa Menu (Rys. 9) implementuje interfejs (Interfejs), posiada opis (String title), współrzędne położenia (int pol_x, int pol_y), dwa wektory (Vector menu_lista) (Vector menu_lista_2), zmienną int kursor wskazującą na aktualnie zaznaczony element menu listy.
Vector menu_lista przechowuje nazwy opcji wyboru które można wskazać kursorem.
Vector menu_lista_2 przechowuje referencje do obiektu na który przerzucić sterowanie gdyby element z menu_lista został wybrany może to być np. inne menu. Metoda dodaj(String element, Object przelacz_na) umożliwia dodanie do menu elementu wraz z referencja do obiektu na który przelaczyc sterowanie. Metody (kursor_up, kursor_down) przemieszczają kursor po liście. Metoda rysuj(Graphics g) jest odpowiedzialna za narysowanie obiektu na płótnie Plotno. Metoda fire() zwraca Interfejs dzięki temu sterowanie można przełączać między obiekty implementujące interfejs.

Menu_wszystkie_akcje to obiekt który dziedziczy po Menu. Wewnątrz obiektu znajdują się referencje do obiektów Ekstensja_akcji, Cyfry_box_kup, Cyfry_box_sprz, Wykres.
Menu_wszystkie_akcje umożliwia przeglądanie listy dostępnych akcji za pomocą przycisków (góra, dół), oraz dokonywanie operacji na zaznaczonej kursorem-przeglądania akcji (Kupno, Sprzedaż, Wykres, Odśwież, Wróć). Wybór operacji ustalany jest za pomocą przycisków (lewo, prawo) które sterują kursorem-operacji. Menu_wszystkie_akcje posiada metodę wczytaj_akcje() która wysyła do serwletu żądanie o listę aktualnych akcji wraz z kursami.

Cyfry_box to obiekt zbudowany na bazie obiektu Menu w którym są dwie opcje (OK, Anuluj) oraz pole wprowadzenia wartości liczbowej za pomocą cyferblatu. Obiekt jest wykorzystany przy rozpoczęciu nowej gry, umożliwia wprowadzenie kwoty startowej.
Na bazie obiektu Cyfry_box stworzono obiekty Cyfry_box_kup oraz Cyfry_box_sprz, które umożliwiają zakup oraz sprzedaż wcześniej wybranej akcji uwzględniając dostępne środki finansowe gracza. Gracz wprowadza ilość akcji które chciałby zakupić lub sprzedać. Aplikacja automatycznie informuje czy jest to możliwe biorąc pod uwagę (ilość posiadanych akcji oraz pieniędzy).
Wykres jest klasą która została rozbudowana na bazie klasy menu - umożliwia oglądanie wykresów (dziennego oraz historycznego) wcześniej wybranej akcji. W przypadku opcji historyczny istnieje możliwość przewijania wykresu w czasie za pomocą przycisków (lewo, prawo). Klasa Wykres posiada metode wczytaj_dane_do_wykresu(boolean hist). Metoda ta wysyła żądanie do serwletu o dane do wykresu dzienego jeżeli zmienna hist=false, w przeciwnym przypadku gdy zmienna hist=true metoda żąda danych do wykresu historycznego.



Rys. 10. Przejścia w sterowaniu między obiektami aplikacji przedstawione za pomocą linii.

Wysyłanie zapytań do web aplikacji i odczytywanie odpowiedzi.

Midlet wysyła żądanie GET do serwletu w następujących przypadkach:

- Przy pierwszym utworzeniu obiektu menu_wszystkie_akcje prosi o przesłanie listy akcji wraz z kursami. Wysyłając żądanie GET *"http://localhost:8084/Gra_gieldowa_server/Gielda_servlet?pobierz_akcje=all";*
- Po wywołaniu operacji odśwież dla menu_wszystkie_akcje.
- Przy przełączeniu sterowania na obiekt Wykres żąda przesłania danych dla wcześniej wybranej akcji o danej nazwie. W zależności od zmiennej boolean hist żąda danych historycznych albo dziennych.*"http://localhost:8084/Gra_gieldowa_server/Gielda_servlet?pobierz_akcje="+this.akcja.nazwa_spolki+"&historyczne="+hist;*

```
„public void wczytaj_dane_do_wykresu(boolean hist) throws IOException
    {
        this.wyczysc_vectory_wykresu();
        //zmienna boolean hist true lub false decyduje o wykresie ktory odebrac
        this.historyczny=hist;

String url= "http://localhost:8084/Gra_gieldowa_server/Gielda_servlet?
pobierz_akcje="+this.akcja.nazwa_spolki+"&historyczne="+hist;

        HttpConnection c = null;
          InputStream is = null;
          StringBuffer b = new StringBuffer();

          try {
              System.out.println("1. wywolano invoke servlet");
              c = (HttpConnection)Connector.open(url);

              System.out.println("2. utworzono connector");
              c.setRequestMethod(HttpConnection.GET);
                  System.out.println("3. ustawiono request method na get");

                  is = c.openDataInputStream();
                  System.out.println("5. otworzono inputstream");

                      int ch;
                      while ((ch = is.read()) != -1) {
                          b.append((char) ch);
                      }
                      String text=b.toString();

                      System.out.println(text);
```

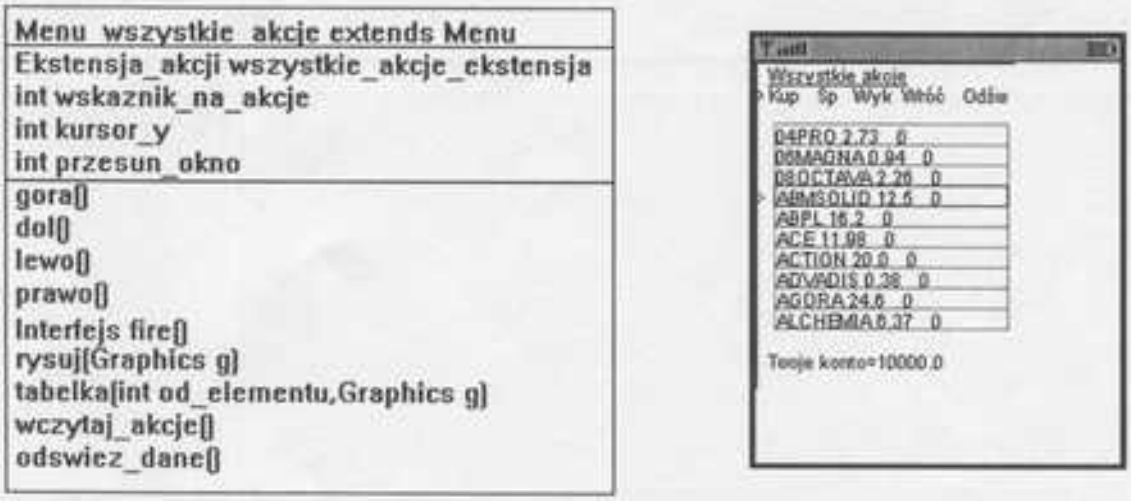
" Fragment metody wczytaj_dane_do_wykresu(boolean hist) należącej do obiektu Wykres.

W metodzie tworzony jest obiekt Connector z żądaniem zapisanym w Stringu url. Następnie ustawiona zostaje metoda żądania na GET po czym otwierany zostaje strumień danych InputStream. Dane są zapisywane do StringBuffera b i przypisane do pomocniczego Stringa text. W dalszej części metody następuje parsowanie Stringa text i zapełnienie wektorów danych i czasu dla wykresu.

Przechowywanie i prezentacja danych (tabel, wykresów akcji.)

Wirtualne obiekty akcji przechowywane są w wektorze ekstensja obiektu Ekstensa_akcji (Rys. 11). Dane akcji odczytywane są w metodzie tabelka przy wyświetlaniu obiektu menu_wszystkie_akcje.

Rys. 11. Klasa Ekstensja_akcji która przechowuje w wektorze instancje Akcji.



Rys. 12. Klasa Menu_wszystkie_akcje w której znajduje się ekstensja akcji oraz widok tej klasy podczas działania aplikacji.

Wewnątrz klasy Menu_wszystkie_akcje (Rys. 12) metoda tabelka(int od_elementu,Graphics g) jest odpowiedzialna za wyświetlenie 10 kolejnych akcji z ekstensji które tworzą tabelkę. Przedział 10 kolejnych wyświetlanych elementów w tabelce jest ruchomy. W przypadku gdy kursor ustawiony jest na ostatnim 10 elemencie okna i został wciśnięty przycisk dół przedział przesuwa się o jeden element w górę.

```
// wypisanie okna (od elementu do elementu) z listy ekstensji
for(int k=od_elementu;k<od_elementu+10;k++){

g.drawString(""+this.wszystkie_akcje_ekstensja.wypisz_ekstensje_nr(k), 10,40+(k-
od_elementu)*10,Graphics.BASELINE|Graphics.LEFT);
}
```

Fragment metody tabelka klasy menu_wszystkie_akcje. Odpowiedzialny za wyświetlenie 10 kolejnych elementów.

W przypadku prezentacji wykresów dane dla konkretnej akcji są pobierane od serwletu za każdym razem gdy użytkownik wybierze operacje Wyk z menu_wszystkie_akcje. Wówczas do obiektu Wykres zostaje przekazana referencja akcji która jest właśnie zaznaczona w menu_wszystkie_akcje i wywołana zostaje metoda wykres.wczytaj_dane_do_wykresu(true). Następnie sterowanie zostaje przekazane do obiektu Wykres.

```
„if(this.menu_lista.elementAt(this.kursor).toString()=="Wyk")
        {
                int wsk=this.wskaznik_na_akcje;
                Akcja ak=(Akcja)
    (this.wszytskie_akcje_ekstensja.ekstensja.elementAt(wsk));

                this.wykres.akcja=ak;

                try{
                        this.wykres.wczytaj_dane_do_wykresu(true);
                }catch(Exception exc){}
        }"
```

Klasa Wykres (Rys. 13) posiada pomocnicze wektory dla danych dziennych oraz historycznych które są zapełniane tymczasowo na potrzeby narysowania wykresu.

Rys. 13. Klasa Wykres.

Vector dane_wykresu zawiera wartości kursów zamknięcia poszczególnych historycznych dni. Vector czas_wykresu zawiera date odpowiadającą danemu kursowi.

Vector dane_wykresu_dzien zawiera wartośći kursów dla obecnego dnia.

Vector czas_wykresu_dzien zawiera godzine zarejestrowania kursu.

Metoda rysuj(Graphics g) w klasie Wykres rysuje aktualnie wybrany wykres w zależności od zmiennej boolean historyczny.

```
„for(int wyk_x=0;wyk_x<this.dane_wykresu_dzien.size()-1;wyk_x++)
        //for(int wyk_x=0;wyk_x<130;wyk_x++)
        {
            Float c=(Float)this.dane_wykresu_dzien.elementAt(wyk_x);
            Float c2=(Float)this.dane_wykresu_dzien.elementAt(wyk_x+1);

            float d=c.floatValue()*100;
            float d2=c2.floatValue()*100;

            g.setColor(0,0,255);
            g.drawLine(wyk_x*8, 60-(int)d+srodek_intdz, wyk_x*8+8, 60-(int)d2+srodek_intdz);
            g.setColor(0,0,0);

            //g.drawString(""+this.czas_wykresu_dzien.elementAt(wyk_x),wyk_x*22,
100,Graphics.BASELINE|Graphics.LEFT);
            //System.out.println(""+this.czas_wykresu.elementAt(wyk_x));
        }" Fragment metody rysuj odpowiedzialny za rysowanie wykresu dziennego.
```

Pętla for iteruje po elementach w wektorze z danymi. W każdym kroku pętli rysowana jest linia od punktu x do x+1 o odpowiadającej mu wartości z wektora danych.

Zapis oraz odczyt stanu gry –operacje na bazie danych RMS.

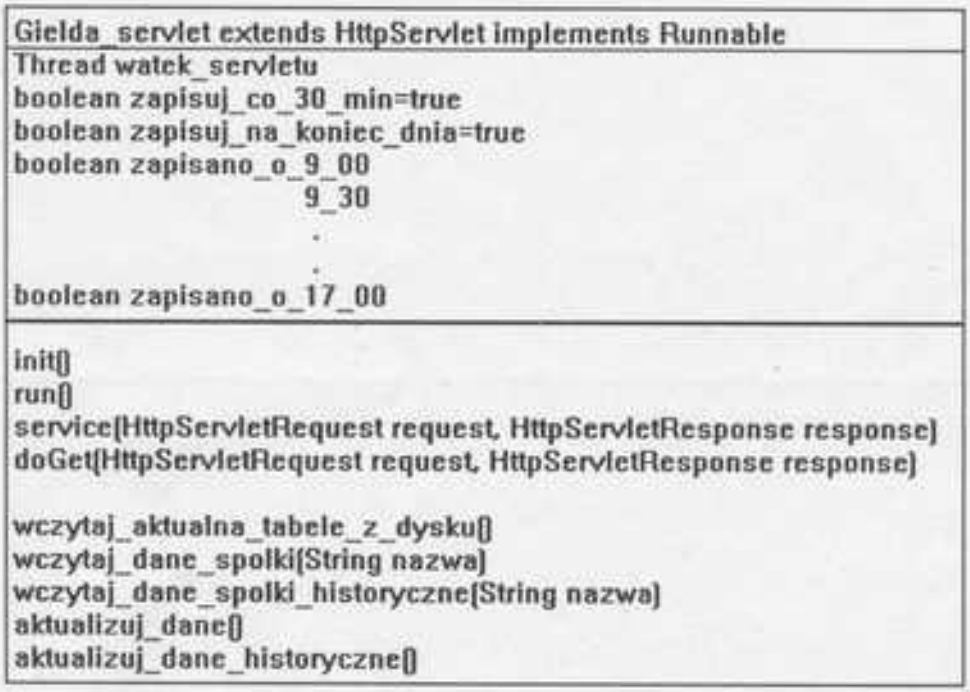
Operacje zapisu oraz odczytu stanu gry nie zostały jeszcze zaimplementowane.

Według założeń projektu mają być przechowywane najbardziej istotne informacje czyli stan konta gracza oraz ilość akcji posiadanych przez gracza. Informacje te byłyby przy każdej operacji zakupu i sprzedaży akcji aktualizowane i zapisywane do RMS. Każde nowe rozpoczęcie aplikacji w trybie aktualna gra powodowałoby wczytanie tych wartości oraz możliwość dalszej gry.

Aplikacja serwera.

Web aplikacja jako serwlet.

Główną klasą po stronie serwera jest serwlet Gielda_servlet (Rys. 14) który pracuje w kontenerze Apache Tomcat'a. Aplikacja została utworzona jako WebApplication przy pomocy kreatora projektów NetBeans IDE 6.8. co zautomatyzowało proces wdrażania i uruchamiania. Klasa „Gielda_servlet.java" dziedziczy po klasie HttpServlet oraz implementuje interfejs Runnable.



Rys. 14. Klasa Gielda_servlet.

Zmienna *boolean zapisuj_co_30_min* decyduje o tym czy dane giełdowe mają być pobierane dla danego dnia co pół godziny.

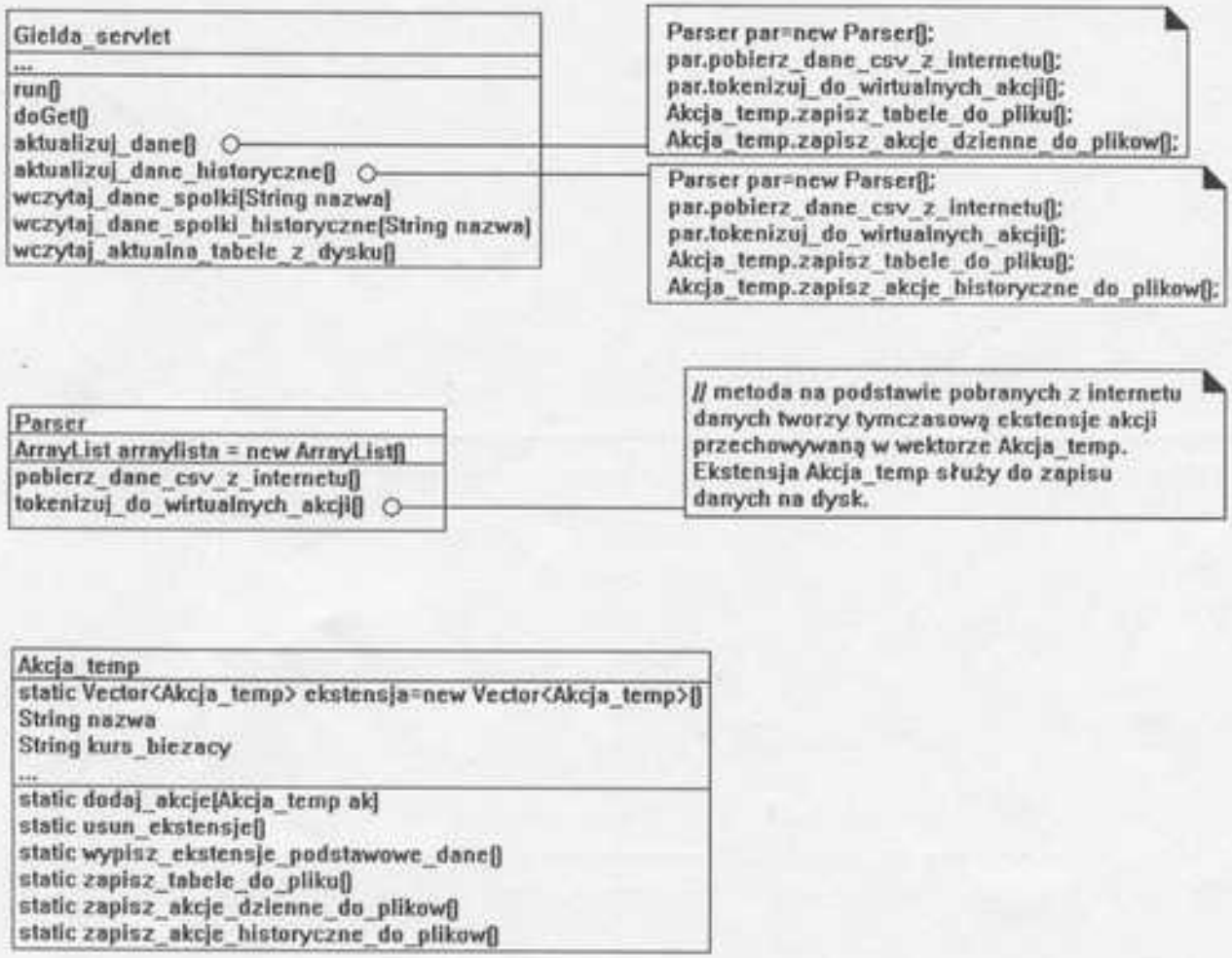
Zmienna *boolean zapisuj_na_koniec_dnia* decyduje czy zapisywać dane giełdowe z zamkniecia sesji giełdowej danego dnia.

Zmienne *boolean zapisano_o_9:00* ... 9:30 ... 10:00 ..... 17:00 to zmienne pomocnicze mówiące o tym czy konkretny zapis został wykonany.

init() - metoda inicjalizacyjna serwletu tworzy i uruchamia wątek serwletu.

run() - metoda interfejsu Runnable stanowi treść wątku. W serwlecie odpowiedzialna jest za sprawdzanie bieżącej godziny i wywołanie metod aktualizujących dane (aktualizuj_dane(), aktualizuj_dane_historyczne()) o określonych godzinach. Metody te służą do pobrania danych z internetu i zapisania ich na dysku. Proces zapisu przedstawia (Rys. 15).

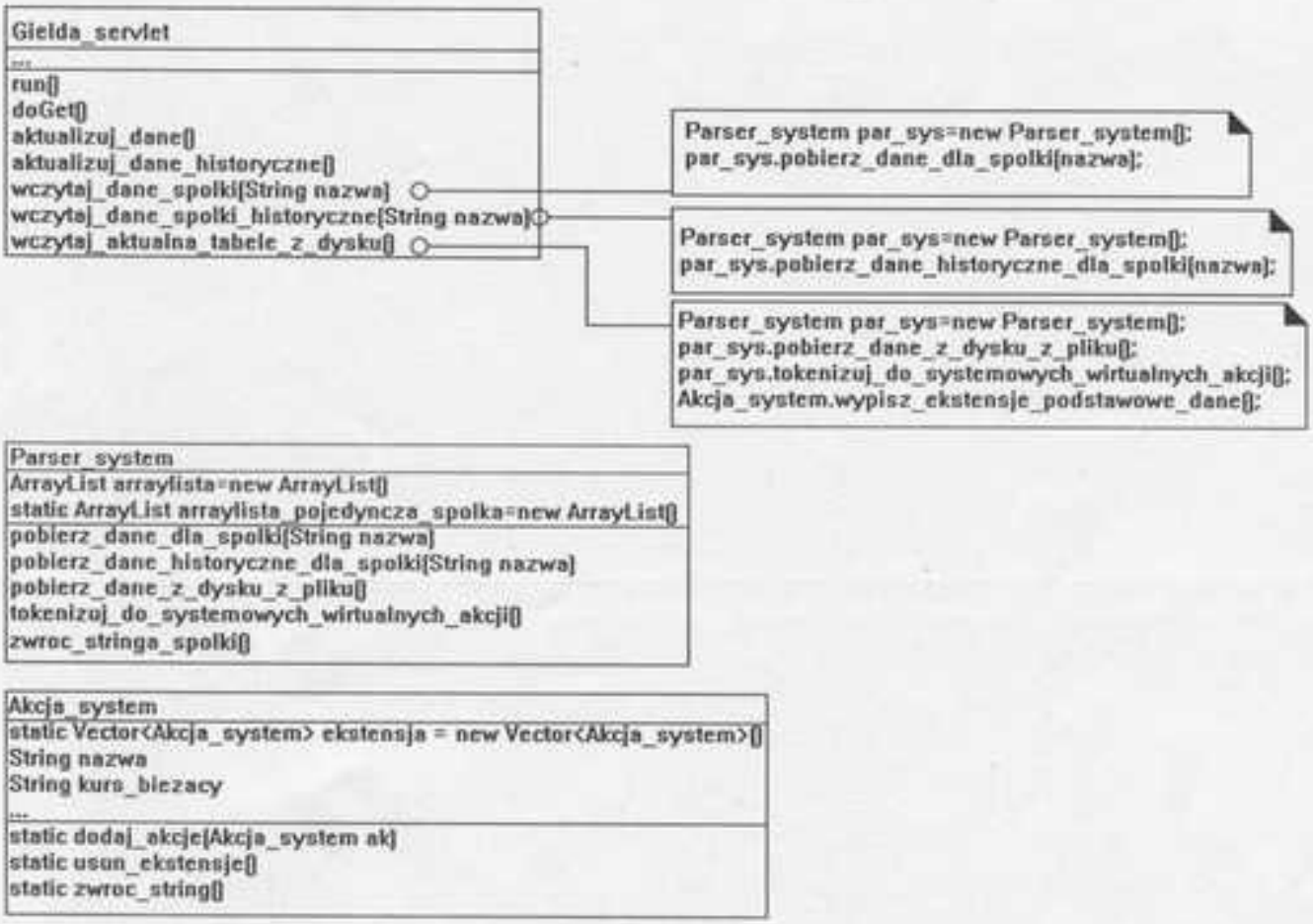
doGet – metoda wewnątrz której znajduje się kod serwletu odpowiedzialny za realizowanie żądań. Wywołuje metody wczytujące dane z dysku (wczytaj_aktualna_tabele_z_dysku(), wczytaj_dane_spolki(), wczytaj_dane_spolki_historyczne()) i zwraca odpowiedź do klienta. Proces odczytu danych z dysku przedstawia (Rys. 16).



Rys. 15. Klasy obiektów które biorą udział podczas pobierania danych z internetu i zapisie ich na dysk.

aktualizuj_dane() - metoda tworzy obiekt Parser który pobiera listę wszystkich akcji wraz z ich aktualnymi kursami z internetu(plik csv zamieszczony w serwisie giełdowym onet'u) a następnie wywołuje metode Parser.tokenizuj_do_wirtualnych_akcji() która tworzy ekstensje akcji o nazwie Akcja_temp. Po utworzeniu ekstensji wywoływane zostają z obiektu Akcja_temp metody zapisujące dane aktualną tabele wszystkich akcji oraz dane dzienne na dysk. Dane dzienne są pamiętane tylko dla bieżącego dnia.

aktualizuj_dane_historyczne() - metoda która działa w ten sam sposób co metoda aktualizuj_dane(). Zapisuje do plików historycznych kolejny kurs zamknięcia.



Rys. 16. Klasy obiektów które biorą udział w procesie odczytu danych z dysku.

wczytaj_dane_spolki(String nazwa) – metoda wczytuje z pliku na dysku aktualne dane dzienne spółki o nazwie podanej jako argument nazwa. Dane dzienne to lista notowań rejestrowanych danego dnia co pół godziny.
Dane umieszczane są w pomocniczej ArrayLiscie arraylista_pojedyncza_spolka obiektu Parser_system.

wczytaj_dane_spolki_historyczne(String nazwa) – metoda wczytuje dane historyczne z pliku dla spółki o podanej nazwie. Dane notowane dzień po dniu zawierają datę oraz kurs zamknięcia danego dnia. Dane umieszczane są w pomocniczej ArrayLiscie arraylista_pojedyncza_spolka obiektu Parser_system.

wczytaj_aktualna_tabele_z_dysku() - metoda wczytuje z dysku plik zawierający nazwy wszystkich spółek wraz z ich bieżącymi kursami. Dane zostają wczytane do zwykłej ArrayListy arraylista obiektu Parser_system a następnie na ich podstawie tworzona jest ekstensja akcji przechowywana w wektorze ekstensja klasy Akcja_system..

Wątek w serwlecie. Pobieranie danych giełdowych z internetu.

Serwlet implementuje interfejs Runnable. Wewnątrz metody init() tworzony i uruchomiony zostaje wątek serwletu.

```
„public void init(ServletConfig config) throws ServletException
 {
   super.init(config);
   watek_servletu=new Thread(this);
   watek_servletu.setPriority(Thread.MIN_PRIORITY);
   watek_servletu.start();
 }"
```

Kod metody init() serwletu.

Wewnątrz metody run() znajduje się kod wątku który jest cyklicznie wykonywany co 1000 milisekund. Sprawdzany jest bieżący czas i na jego podstawie dokonywana jest aktualizacja danych giełdowych co pół godziny zaczynając od godziny 9:30 a kończąc na godzinie 17:00. Jeden raz w ciągu dnia o godzinie 18 aktualizowane są dane historyczne.

```
„if(godzina==15 && minuta==0)
 {
   aktualizuj_dane();
   System.out.println("zapisano o Godzina "+godzina+":"+minuta+":"+sekunda);
   this.zapisano_o_14_30=true;
 }"
```

Fragment kodu metody run() serwletu. Przykładowy fragment kodu o godzinie 15:00 wywołuje metodę aktualizuj_dane();

Pobieranie danych z internetu realizowane jest przez metodę pobierz_dane_csv_z_internetu() obiektu klasy Parser. Metoda wykorzystując obiekty klas URLConnection, InputStreamReader, BufferedReader odczytuje linia po linii plik csv pod wskazanym adresem url. Odczytane linie zostają umieszczone w ArrayLiscie arraylista obiektu Parser w postaci typu String.

Parsowanie danych i rzutowanie do wirtualnych obiektów Akcja_temp.

Dane pobrane z internetu znajdują się w ArrayList arraylista obiektu klasy Parser. Zanim zostaną zapisane są rzutowane na obiekty Akcja_temp znajdujące się w ekstensji ekstensja klasy Akcja_temp. Zadanie wydobycia poszczególnych danych i ich wprowadzenia do ekstensji realizuje

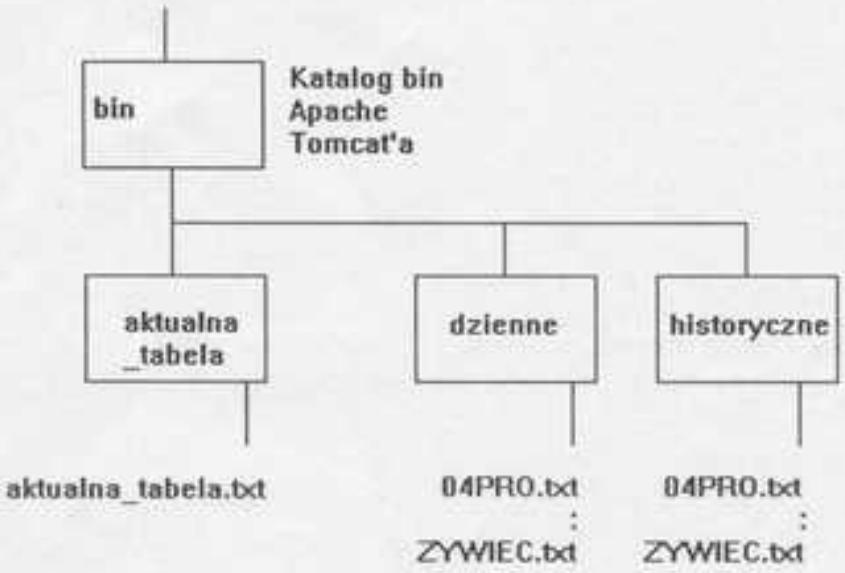
metoda tokenizuj_do_wirtualnych_akcji() obiektu klasy Parser. Ponieważ dane w arraylista przechowywane są jako linie danych typu String należy je wydobyć stosując obiekt StringTokenizer. Każda linia danych typu String zawiera informacje o jednej akcji. Informacje o atrubutach konkretnej akcji oddzielone są znakiem średnika „;". Następnie każda odczytana akcja zostaje utworzona w ekstensji.

„new Akcja_temp(tydz_min,tydz_max,czas,nazwa,kurs_biezacy,zmiana_proc,zmiana_pln,kurs_odniesienia,kurs_otwarcia,kurs_min,kurs_max,obroty_wolumen,obroty_wartosc);"

Fragment tworzący instancję obiektu Akcja_temp, która automatycznie (konstruktor wywołuje metodę dodaj_akcje(this)) dodawana jest do wektora ekstensja.

Zapis danych na dysku.

Gdy bieżące notowania o wszystkich akcjach znajdują się wewnątrz ekstensji ekstensja klasy Akcja_temp mogą zostać zapisane w strukturze katalogów przedstawionej na (Rys. 17).



Rys. 17. Struktura katalogów przechowujących pliki z danymi giełdowymi.

Katalog bin to katalog Apache Tomcat'a wewnątrz którego należy wgrać strukturę katalogów wraz z danymi historycznymi (o ile jesteśmy w ich posiadaniu). W przypadku braku takiej struktury zostanie ona automatycznie utworzona przy pierwszych zapisach danych na dysk (dane historyczne będą rejestrowane od nowa).

Katalog aktualna_tabela – przechowuje plik aktualna_tabela.txt zawierający ostatnio zaktualizowaną listę nazw akcji wraz z ich kursami.

Katalog dzienne – przechowuje pliki wszystkich spółek w formacie nazwa_spolki.txt. Każdy plik w katalogu dzienne zawiera notowania tylko dla bieżącego dnia rejestrowane co 30min.

Katalog historyczne - przechowuje pliki wszystkich spółek w formacie nazwa_spolki.txt. Każdy plik w katalogu historyczne zawiera notowania na koniec dnia zapisywane dzień po dniu wraz z datą.

Zapis danych z ekstensji realizują metody (zapisz_tabele_do_pliku(), zapisz_akcje_dzienne_do_plikow(), zapisz_akcje_historyczne_do_plikow()) klasy Akcja_temp. Metody wykorzystują do zapisu obiekt klasy FileWriter.

„*public static void zapisz_tabele_do_pliku()*

```
{
  String plik="aktualna_tabela.txt";
  try{
    FileWriter fw = new FileWriter("aktualna_tabela\\"+plik,false);
      for(int i=0;i<ekstensja.size();i++)
        {
          Akcja_temp ak=ekstensja.get(i);
          fw.write(""+ak.czas+";"+ak.nazwa+";"+ak.kurs_biezacy+";"+'\n');
        }
    fw.close();
  }catch(Exception ex){System.out.println(""+ex);}
}
```

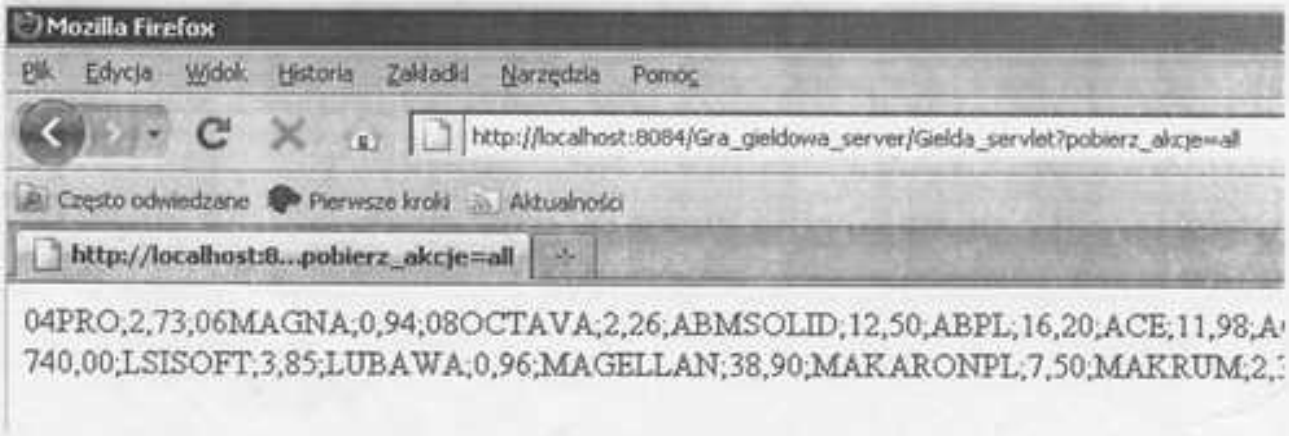
" Metoda zapisz_tabele_do_pliku() klasy Akcja_temp.

Zastosowanie ekstensji Akcja_temp pozwala w łatwy sposób manipulować zapisywanymi danymi. Wystarczy odwołać się do odpowiedniego pola akcji.

Obsługiwanie zapytań klienta i zwracanie odpowiedzi.

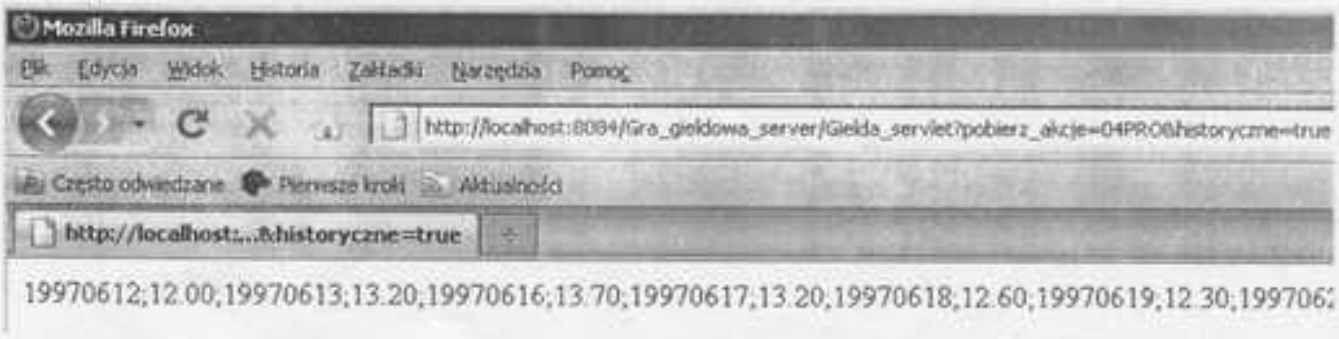
Zapytania wysyłane do serwletu obsługiwane są przez metodę doGet(). Metoda sprawdza wartości dwóch parametrów przesłanych do serwletu. Nazwy tych parametrów to pobierz_akcje oraz historyczne.

W przypadku wysłania parametru pobierz_akcje o wartości all serwlet zwróci listę wszystkich akcji (Rys. 18).



Rys. 18. Efekt działania serwletu widoczny z przeglądarki Mozilla.

Parametr pobierz_akcje może zawierać nazwę konkretnej spółki. W takim przypadku brany pod uwagę jest także drugi parametr o nazwie historyczne który może przyjąć wartość true albo false (Rys. 19).



Rys. 19. Odpowiedź serwletu na żądanie „?pobierz_akcje=04PRO&historyczne=true" widoczna z przeglądarki Mozilla.

„ *String parametr="pobierz_akcje";*

String parametr2="historyczne";

String wartosc_parametru = request.getParameter(parametr);

String wartosc_parametru_2 = request.getParameter(parametr2); " Fragment metody doGet().

Wartości parametrów pobrane zostają z obiektu request metody doGet.

Odpowiedź jest zwracana za pomocą obiektu response wykorzystując obiekt PrintWriter.
PrintWriter out = response.getWriter().

Do obiektu PrintWriter wysyłane zostają odpowiedzi w postaci typu String.

```
.. ..... }else if(wartosc_parametru.equalsIgnoreCase("all"))
    {
      wczytaj_aktualna_tabele_z_dysku();
      out.println(""+Akcja_system.zwroc_string());
    } .....
```

" Fragment metody doGet(), który zwraca aktualną listę akcji.

Wczytywanie danych z dysku i rzutowanie do systemowych obiektów Akcja_system.

Wczytywanie danych z dysku i rzutowanie do systemowej ekstensji przebiega bardzo podobnie do procesu pobierania danych z internetu. Serwlet korzysta z obiektów klas Parser_system oraz Akcja_system (Rys. 16). Dane są analogicznie rzutowane do ekstensji Akcja_system. Główną różnicą między klasą Parser a Parser_system jest sposób pobierania danych. Źródłem danych są pliki na dysku. Ponadto klasa Parser_system posiada pomocniczą ArrayList'ę arraylista_pojedyncza_spolka która pobiera dane dla jednej spółki potrzebne do wykresu. Dane do wykresu są zwracane za pomocą metody zwroc_stringa_spolki() obiektu klasy Parser_system. Dane aktualnej tabeli wszystkich akcji zwracane są przez metodę zwroc_string() obiektu klasy Akcja_system.

Rozbudowa systemu w przyszłości.

W obecnej wersji system jest bardzo prosty. Wykorzystuje dane opóźnione i przechowuje tylko najbardziej istotne informacje jak nazwy spółek i ich kursy. Dlatego ciekawym rozwiązaniem byłoby wykorzystanie bardziej szczegółowych danych czasu rzeczywistego i ich składowanie w bazie danych z zastosowaniem np. Hibernate.
Do strony funkcjonalnej klienta należy jeszcze wprowadzić zapis i odczyt do RMS. Należałoby także poprawić rysowanie wykresów tak aby były automatycznie skalowane. Można by usprawnić GUI po stronie klienta – dodać np. możliwość sortowania tabeli, wykorzystać więcej parametrów akcji.

Literatura

1. Basham Bryan, „Head First Servlets & JSP", O'Reilly, 2004
2. Feng Yu, "Network Programming with J2ME Wireless Devices", „http://www.wirelessdevnet.com/channels/java/features/j2me_http.phtml"
3. Giguere Eric, „Understanding J2ME Application Models", "http://developers.sun.com/mobility/midp/articles/models/index.html", 2002
4. Hall Marty, „Java Servlet i Java Server Pages", Helion, 2002
5. Java Community Process, „Mobile Information Device Profile for Java 2 Micro Edition version 2.0" (JSR118)
6. Muchow John, „Persistent Storage in J2ME and MIDP", „http://www.ibm.com/developerworks/library/j-j2me3/", 2003
7. Topley Kim, „J2ME in a Nutshell.", O'Reilly, 2002
8. „Java ME Platform Overview", „http://java.sun.com/javame/technology/index.jsp"
9. http://pl.wikipedia.org/wiki/J2ME
10. http://www.gpw.pl/gpw.asp?cel=ogieldzie&k=2&i=/historia/historia&sky=1
11. http://www.gpw.pl/gpw.asp?cel=informacje_gieldowe&k=6&i=/serwisy/serwisy&sky=1
12. http://pl.wikipedia.org/wiki/Serwlet
13. http://pl.wikipedia.org/wiki/Klient-serwer
14. http://pl.wikipedia.org/wiki/Architektura_tr%C3%B3jwarstwowa