Internetowa wersja Maszyny W

1 Skład sekcji

- Flakus Józef
- Musiał Sebastian
- Poloczek Dawid
- Rakoczy Michał
- Rzepka Tomasz

2 Serwer

Kod serwera napisany został w języku Java. Wykorzystano mechanizmy wprowadzone w wersji 8 tego języka, dlatego jest to minimalna wymagana wersja. Całość kodu umieszczona została w pakietach **pl.polsl.***. Poniżej została opisana zawartość poszczególnych pakietów.

2.1 pl.polsl.architecture

Tutaj znajduje się całość kodu odpowiedzialnego za logiczną reprezentację Maszyny W. Są to klasy komponentów oraz sygnałów. Klasy zostały zaprojektowane tak, aby działająca instancja Maszyny W mogła być traktowana jako zbiór komponentów połączonych sygnałami, które sterują przepływem danych pomiędzy tymi komponentami.

Klasa WMachine reprezentuje działającą Maszynę W. Przed użyciem musi ona zostać zbudowana aby zawierała wszystkie niezbędne komponenty i sygnały. WMachineFactory umożliwia stworzenie pełnej architektury Maszyny W. WMachineBuilder zapewnia, że proces budowania jest spójny, dlatego powinno się używać tej klasy zamiast ręcznie dodawać komponenty.

2.1.1 pl.polsl.architecture.components

W tym pakiecie zostały zdefiniowane interfejsy komponentów. Najbardziej podstawowe to DataSource oraz DataTarget. Są one odpowiednio źródłem oraz miejscem docelowym dla danych podczas sterowania przepływem danych. Dalej jest NonVolatileDataStorage (pamięć, rejestr), VolatileDataStorage (bufor, magistrala). VolatileDataStorage traci wartość w nim zapisaną wraz z rozpoczęciem nowego taktu.

2.1.2 pl.polsl.architecture.components.finalized

Klasy komponentów: rejestrów, buforów, magistrali, pamięci, komórki pamięci, jednostki arytmetyczno-logicznej.

2.1.3 pl.polsl.architecture.data

Klasy używane do przechowywania danych w komponentach. Odpowiadają słowom reprezentującym adres, rozkaz lub wartość rejestru flag.

2.1.4 pl.polsl.architecture.signals

Sygnały DataFlowSignal sterują przepływem danych. Każdy z tych sygnałów posiada źródło danych oraz miejsce docelowe i ma on za zadanie przepisać wartość ze źródła do celu. ScriptDataFlowSignal przed zapisem wykonuje na wartości operację zdefiniowaną w skrypcie.

2.2 pl.polsl.dao

Ten pakiet zawiera obiekty dostępu do danych (Data Access Object – DAO). Wykonują one operacje na bazie danych za pośrednictwem Hibernate. Klasa Dao udostępnia generyczne operacje:

get, delete, save. Poszczególne klasy umożliwiają operacje na konkretnych klasach zdefiniowanych w pakiecie pl.polsl.storage.

2.3 pl.polsl.database

Klasy w tym pakiecie obsługują połączenie z bazą danych w Javie bez użycia dodatkowych bibliotek. Wykorzystują one bezpośrednio zapytania języka SQL. Ta część aplikacji jest używana po stronie forum.

2.4 pl.polsl.forum

Tutaj znajduje się obsługa forum. Wykorzystane zostały tutaj klasy znajdujące się w pakiecie pl.polsl.database.

2.5 pl.polsl.hibernate

Pakiet zawiera klasę obsługującą sesję Hibernate.

2.6 pl.polsl.i18n

Klasy odpowiadające za język znajdują się w tym pakiecie. LanguageAccessor umożlwia komunikację pomiędzy serwerem i klientem. Klasa Language reprezentuje używany język. Udostępnia metody umożliwiające ustawienie odpowiednich tłumaczeń.

2.6.1 pl.polsl.i18n.keywords

Reprezentacja języka słów kluczowych w rozkazach Maszyny W.

2.7 pl.polsl.parser

Parser rozkazów został zdefiniowany w tym pakiecie. Do parsowania pojedynczej linii wykorzystane zostały wyrażenia regularne. Są one tworzone w klasie CommandLineParser. Każda linia jest zastępowana przez odpowiedni obiekt z pakietu pl.polsl.parser.line. Następnie zestaw takich obiektów zamieniany jest na kolekcję obiektów Tact z pakietu pl.polsl.runner.tact, która jest rozumiana przez Maszynę W i może zostać wykonana.

CommandListAccessor obsługuje komunikację z klientem i żądania wykonania operacji na rozkazach i listach rozkazów: utworzenie nowej listy, dodanie/usunięcie rozkazu.

2.7.1 pl.polsl.parser.line

Pakiet zawiera klasy reprezentujące pojedynczą linię rozkazu. Zawierają one dane wyłuskane z poszczególnych linii. CommandLine jest interfejsem, który implementuje każda z tych klas.

2.8 pl.polsl.runner

Runner odpowiada za wykonanie programu zapisanego w Maszynie W na dowolnym poziomie śledzenia: program (aż do aktywacji sygnału STOP), rozkaz, takt. Umożliwia również wykonanie pojedynczego taktu podczas sterowania ręcznego.

RunnerAccessor udostępnia interfejs dla klienta, dzięki któremu można wywołać uruchamianie programu. CommandList zawiera zestaw rozkazów. Instancja klasy WMachine posiada ustawiony obiekt tej klasy i korzysta z tych rozkazów. Domyślnie jest to obiekt klasy DefaultCommandList. Jeżeli baza danych jest podłączona domyślna lista to lista o ID 1. W przeciwnym wypadku domyśla lista jest budowana wewnątrz tej klasy.

2.8.1 pl.polsl.runner.command

Pakiet zawiera definicję klasy rozkazu. Znajdują się w niej wszystkie informacje dotyczące rozkazu: nazwa, opis, liczba argumentów, lista taktów. Umożliwia uruchomienie rozkazu na dowolnej

instancji Maszyny W. Wspiera wykonywanie rozkazu takt po takcie lub w całości. Rozkaz obsługuje rozgałęzienia.

2.8.2 pl.polsl.runner.tact

Klasa Tact reprezentuje zbiór sygnałów które mają być aktywowane w danym takcie. Umożliwia wykonanie danego taktu. Sygnały są aktywowane według ich identyfikatorów w kolejności rosnącej. Identyfikatory sygnałów są zdefiniowane w pl.polsl.servler.ArchitectureInfo.AvailableSignals.

2.8.2.1 pl.polsl.runner.tact.branch

Pakiet zawiera klasy reprezentujące rozgałęzienia. Rozgałęzienia same w sobie nic nie robią, jedynie informacje jakie zawierają mogą zostać wykorzystane przez klasę Command.

2.9 pl.polsl.servlet

W tym pakiecie znajduje się większość klas odpowiadających za komunikację sieciową. Obsługiwany protokół to HTTP, metody GET i POST.

ArchitectureInfo umożliwia odczyt dostępnej architektury Maszyny W, tj. wszystkich rejestrów, i sygnałów. Oprócz tego ArchitectureInfo.AvailableSignals definiuje kolejność aktywacji sygnałów w takcie poprzez numer sygnału. Najpierw aktywowane są sygnały o najniższych numerach. Kolejność została ustalona tak aby sygnały były aktywowane według rodzaju w następującej kolejności: odczyt z pamięci, wyjścia z rejestrów, połączenie międzymagistralowe, inkrementacja licznika rozkazów, inkrementacja/dekrementacja wskaźnika stosu, wejście JAL, operacje w JAL, wejście akumulatora, zapis do pamięci, stop.

MemoryAccessor, RegisterAccessor, SignalAccessor, SettingsAccessor umożliwia odczyt i zamianę stanu odpowiednio: pamięci, rejestru, sygnału, ustawień (są one przechowywane aby uniknąć ich utraty po odświeżeniu strony). WMachineState umożliwia odczyt stanu całej Maszyny W i jego odtworzenie na stronie.

WMachineServletBase jest klasą bazową dla wszystkich wyżej wymienionych klas. Umożliwia odczyt z sesji aktualnie używaną instancję Maszyny W, użytkownika oraz Runnera.

WMachineServletContextListener zamyka połączenie z bazą danych oraz tworzy je w przypadku klas z pakietu pl.polsl.database.

2.10 pl.polsl.settings

Zawiera klasy reprezentujące ustawienia możliwe do zmiany w kliencie. Umożliwia ich zgrupowanie i wygodne przechowywanie w sesji oraz łatwy dostęp.

2.11 pl.polsl.storage

Klasy z tego pakietu stanową obiektową reprezentację tabel bazy danych.

2.12 pl.polsl.utils

WMachineSerializer serializuje stan Maszyny W do JSON-a.

Primitive jest klasą opakowania dla typów podstawowych. Dzięki jej wykorzystaniu można je traktować jak typy referencyjne, dzięki czemu nie trzeba w każdym miejscu aktualizować danej wartości. Klasa jest wykorzystana w klasach Memory, AddressWord oraz CommandWord do przechowywania liczby bitów adresu i kodu rozkazu.

RegisterChangeListener jest wykorzystywany w FlagWord do zmiany stanu flag po zmianie zawartości akumulatora.

3 Interfejs WWW

Do realizacji strony zostały użyte technologie takie jak:

- bootstrap – głównie przy realizacji forum,
- jQuery – komunikacja z serwerem oraz manipulacja elementami DOM,
- raphaelJS – do rysowania elementów Maszyny W w formacie SVG,
- jwertyJS – do aktywowania skrótów klawiszowych.

W celu instalacji wszystkich komponentów należy z poziomu konsoli przejść do folderu "web/assets" i wykonać komendę bower install. Wszystkie style zostały napisane w dynamicznym języku arkuszy stylów SASS. Aby dokonać kompilacji kodu, w celu otrzymania wynikowego pliku CSS należy posłużyć się komendą: sass --watch sass/style.scss:css/style.css --style compressed. Podobnie jak w przypadku instalacji wszystkich pakietów, komendę należy wykonać z poziomu konsoli po przejściu do folderu "web/assets".

Każdy komponent (sygnał, rejestr, komórka pamięci) posiada odpowiadający mu obiekt umożliwiający wykonywanie na nim abstrakcyjnych operacji, np. ustawienie wartości, aktywacja/deaktywacja. Operacje te są zbiorem przekształceń dokumentu HTML oraz komunikacji z serwerem. Obiekty te są zdefiniowane jako funkcje w odpowiednich plikach z folderu interaction. Poniżej znajduje się lista plików JavaScript wraz z ich krótkim opisem.

3.1 _mappings.js

W tym pliku zostały zdefiniowane zmienne zawierające odwołania do elementów DOM wykorzystywanych w pozostałych plikach. Dzięki temu skrypty nie są zależne od nazw znaczników HTML czy struktury dokumentu.

3.2 _ready.js

W tym pliku znajduje się kod wykonywany po załadowaniu strony. Następuje tu odtworzenie stanu Maszyny W na postawie danych przechowywanych w sesji.

3.3 _view.js

W tym pliku tworzona jest graficzna reprezentacja Maszyny W z pomocą biblioteki RaphaelJS. Oprócz tego znajdują się tu funkcje do obsługi okna wiadomości.

3.4 _registration.js

Plik zawiera kod obsługi rejestracji oraz logowania. Znajduje się tutaj funkcja walidująca adres e-mail, hasło oraz sprawdzająca czy login jest wolny.

3.5 _commands.js

W tym pliku znajduje się obsługa zakładki „Rozkazy”. Zakładka dla każdego rozkazu jest generowana w tym miejscu. W tym pliku następuje komunikacja z klasą CommandListAccessor.

3.6 interaction/_ interaction.js

W tym pliku inicjalizowane są edytowalne kontrolki oraz funkcja odpowiadająca za wczytanie i przywrócenie stanu Maszyny W na podstawie danych zapisanych w sesji.

3.7 interaction/_ memory.js

W tym pliku generowane są komórki pamięci oraz znajduje się tutaj kod obsługi tych kontrolek. Komórki pamięci są generowane w tle. Komórki pamięci zostają również opakowane w obiekty pośredniczące.

3.8 interaction/_ registers.js

Plik zawiera kod obsługi rejestrów oraz są one opakowane w obiekty pośredniczące.

3.9 interaction/_ runner.js

Plik zawiera kod obsługi przycisków „Uruchom program”, „Uruchom rozkaz”, „Uruchom takt” oraz checkbox’a „Sterowanie ręczne”.

3.10 interaction/_ settings.js

Tutaj znajduje się kod obsługujący zakładkę „Ustawienia”.

3.11 interaction/_ signals.js

W tym pliku obiekty utworzone za pomocą biblioteki RaphaelJS w pliku _view.js zostają opakowane w obiekty pośredniczące, podobnie jak rejesty i komórki pamięci.

4 Hibernate

4.1 hibernate.cfg.xml

W tym pliku przechowywana jest konfiguracja Hibernate’a. Do konfiguracji należą:

- dane opisujące połączenie z bazą,
- mapowanie.

W celu konfiguracji połączenia z bazą należy ustawić:

- adres bazy (*jdbc:mysql://adres:port/nazwa_bazy_danych*),
- nazwa użytkownika,
- hasło.

W pliku wygląda to następująco:

```
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://mysql3.hekko.net.pl:3306/sssebuss_bd</property>
        <property name="hibernate.connection.username">sssebuss_bd</property>
        <property name="hibernate.connection.password">maszynaW2015</property>
```

W projekcie powinna zostać wykorzystana baza MySQL, ale można implementować inny typ. Może to jednak wymagać zmian w skrypcie tworzącym bazę oraz w klasach związanych z usługą Hibernate.

4.2 Zasada działania

Dostęp do bazy jest zapewniony poprzez użycie klas typu Data Access Object (DAO). Klasy te implementują cztery metody (save, saveOrUpdate, delete, get, merge, getAll), które odpowiadają instrukcją SQL (insert, insert lub update, delete, select by ID, insert, select all). Każda tabela jest reprezentowana klasą tego typu. Klasy te muszą rozszerzać klasę *Dao*.

5 Database

5.1 Informacje ogólne

Model bazy danych jest przedstawiony w pliku databaseModel.ea (projekt utworzony w programie Enterprise Architect).

5.2 Tabele

5.2.1 Users

W tej tabeli przechowywane są dane na temat zarejestrowanych użytkowników. Kolumna hasło musi zawierać zaszyfrowane hasło. W tym celu wykorzystuje się szyfrowanie SHA-256.

5.2.2 Topics

W tej tabeli przechowywane są dane o tematach dodanych przez użytkoników, jednak treść tematu jest zawarty w tabeli Reply.

5.2.3 Categories

Tabela niewykorzystywana obecnie, zawiera informację na temat kategorii w forum.

5.2.4 Reply

Tabela zawiera wszystkie odpowiedzi w tematach.

5.2.5 Commands

Tabela przechowuje dane rozkazów, które zostały utworzone przez użytkoników oraz domyślnie dostępne rozkazy.

5.2.6 CommandsLists

Tabela przechowuje informacje o listach rozkazów utworzonych przez użytkowników.

5.2.7 Programs

Tabela przechowuje dane na temat programów stworzonych przez użytkowników.

5.2.8 ProgramsLibrary

Tabela jest złączenie tabeli Users z tabelą Library (relacja wiele do wielu).

Connects users with libraries (M to N relation).

5.2.9 ProgramsMMLibrary

Tabela jest złączenie tabeli Programs z tabelą Library (relacja wiele do wielu).

6 Informacje dodatkowe

W folderze database znajduje się model bazy danych oraz skrypty tworzące tabele i wypełniające je domyślną listą rozkazów oraz przykładowymi danymi.

language.json zawiera definicję języków.

build.xml jest plikiem opisującym proces budowania aplikacji za pomocą narzędzia Apache Ant.

db_manager.xml umożliwia przeładowanie bazy danych za pomocą Ant-a.