Złożenie pracy online:
2014-06-10 20:12:52
Kod pracy:
11424
Kod załącznika:
11414

Dariusz Hudziak
(nr albumu: 19623*INF/INŻ)

Praca inżynierska

Menedżer Map Myśli

Mind Map Manager

Wydział: Nauk Społecznych i Informatyki
Kierunek: Informatyka
Specjalność: inżynieria oprogramowania
Promotor: dr inż. Bogdan Batko

Nowy Sącz, 2014

Streszczenie

Praca stanowi opis procesu projektowania i budowy systemu wspomagajacego tworzenie map myśli. Projekt programu wprowadza funkcje stanowiące wkład w rozwój programów tego typu. Wyróżnia go wsparcie opisu gałęzi map myśli z wykorzystaniem słownika wyrazów bliskoznacznych oraz mozliwość edycji w środowisku rozproszonym. Podczas projektowania systemu wykorzystano modelowanie w języku UML. Stworzony menedżer jest przydatnym kompaktowym narzędziem do nielinearnej notacji i klasyfikacji informacji oraz zarządzania projektami.

Pierwszy rozdział pracy charakteryzuje cel pracy oraz opsiuje zawartośc pozostałych rozdziałów. Rozdział drugi opisuje skrótowo ogólną teorię i zastosowanie map myśli. Rozdział trzeci dokonuje przeglądu istniejących darmowych rozwiązań oraz wylicza proponowane funkcje. Rozdział czwarty charakteryzuje po krótce programy wykorzystane podczas tworzenia pracy. Opisuje również wykorzystane technologie motywując ich użycie Rozdział piąty opisuje cały proces projektowania aplikacji oraz ważne elementy jej implementacji Rozdział szósty podsumowuje realizację celu pracy oraz opisuje perspektywę rozwoju aplikajcji.

Słowa kluczowe

uml, java, javafx, wizualizacja, mapa myśli, synonimy, słownik, zarządzanie

Abstract

Thesis describes the process of designing and implementing a system supporting mind map creation. System design introduces new functions that provide development in this kind of programs Among distinguishing enchancements are the support for branch annotation using thezarus and ability to modify mind maps in distributed environment. The design process was conducted with usage of UML modeling language. Created manager is a usefull,compact tool for nonlinear notation, infromation classification and project management.

Fist section presents the goal of thesis and describes the content of other sections.Second section discusses general theory of mind mapping and it's possible applications. Third section looks through popular free mind mapping programs and proposes new functionality. Forth section presents programs that enabled development od the system. Technologies and language are also discussed providing motivations for their usage. Fifth sesction highlights key concepts of the design and explains crucial fragments of the code. Sixth section sums up realization degree of the goal and presents perspective for aplication evolution

Keywords

uml, java, javafx, wizualizatoin, mind map, synonyms, dictionary, management

Serdecznie dziękuje za wszelką pomoc otrzymaną w trakcie tworzenia pracy.

Spis treści

1 Wstęp

Zamierzeniem autora jest budowa systemu (menedżera) wspomagania wizualizacji (odwzorowania) map myśli, zapewniającego edycję graficzną oraz tekstową danych powiązanych w środowisku rozproszonym. Zrealizowane w pracy zagadnienia stanowią wkład w rozwój programów do tworzenia map myśli w aspekcie transparentności tworzonych map dla odbiorcy nie będącego autorem mapy. System powstanie z wykorzystaniem technologii JavaFX zapewniającej wieloplatformowość oraz nowoczesny i dynamiczny interfejs użytkownika. System wyróżnia wsparcie opisu gałęzi map myśli (słowa klucze) z wykorzystaniem słownika wyrazów bliskoznacznych programu OpenOffice. Realizacja projektu będzie wykorzystywać modelowanie UML. Stworzony menedżer będzie przydatnym kompaktowym narzędziem do nielinearnej notacji i klasyfikacji informacji oraz zarządzania projektami. W przyszłości możliwa jest rozbudowa menedżera o elementy analizy semantycznej z pełnym wsparciem wizualizacji 3D i 4D.

Rozdział drugi opisuje skrótowo ogólną teorię i zastosowanie map myśli. Rozdział trzeci dokonuje przeglądu istniejących darmowych rozwiązań oraz stawia cele do realizacji. Rozdział czwarty charakteryzuje po krótce programy wykorzystane podczas tworzenia pracy. Opisuje również wykorzystane technologie motywując ich użycie. Rozdział piąty opisuje cały proces projektowania aplikacji oraz ważne elementy jej implementacji. Rozdział szósty podsumowuje efekty niniejszej pracy oraz przedstawia perspektywę dalszego rozwoju aplikacji.

2 Mind Mapping – nowa jakość notatek

Mind Mapping /czyli mapowanie myśli/ to szczególny system notacji nieliniowej który aktywizuje i daje efekt synergiczny współpracy obu półkul mózgowych. Za twórców tej metody uważa się dwóch brytyjskich uczonych: Tony'ego i Barry'ego Buzana, choć są również zwolennicy teorii, że ojcem tej metody jest Leonardo da Vinci.

Tworzenie map podnosi efektywność procesów uczenia się, zapamiętywania, tworzenia baz wiedzy a przede wszystkim procesów twórczych.

Kompilacja lewej półkuli mózgu /myślenie logiczne, linearność, analiza, słowa, liczby/ z prawą półkulą /wyobraźnia, rytm, postrzeganie przestrzenne, kolory, GESTALT (obraz całości)/ daje niesamowity efekt synergicznej współpracy obu półkul [1],[3].

Mankamentem tak tworzonych map jest to, że są one zrozumiałe jedynie dla autora, są więc silnie zindywidualizowane wręcz zaszyfrowane, nic w tym dziwnego ponieważ mapują myśli autora.

2.1 Zasady tworzenia map myśli

Twórcy metody opracowali technikę tworzenia map oraz zasady które sprowadzają się do 10 przykazań:

1. W centrum umieszczaj rysunek składający się z minimum trzech kolorów.
2. Twórz obrazy i symbole na całej mapie.
3. Najważniejsze słowa – klucze powinny być najsilniej oznaczone.
4. Na jednej linii może się znajdować tylko jedno słowo lub rysunek.
5. Linie powinny być takiej długości jak słowa.
6. Używaj litery, linie i obrazki różnej wielkości.
7. Używaj wielu różnych kolorów.
8. Przestrzegaj hierarchii pojęć-wykorzystuj porządek numeryczny.
9. Wykreuj swój własny styl tworzenia Map Myśli.
10. Umieszczaj na Mapie Myśli nie tylko same fakty, ale także problemy, skojarzenia, puste linie itp. [1],[3],[4]

2.2 Potencjalne zastosowania

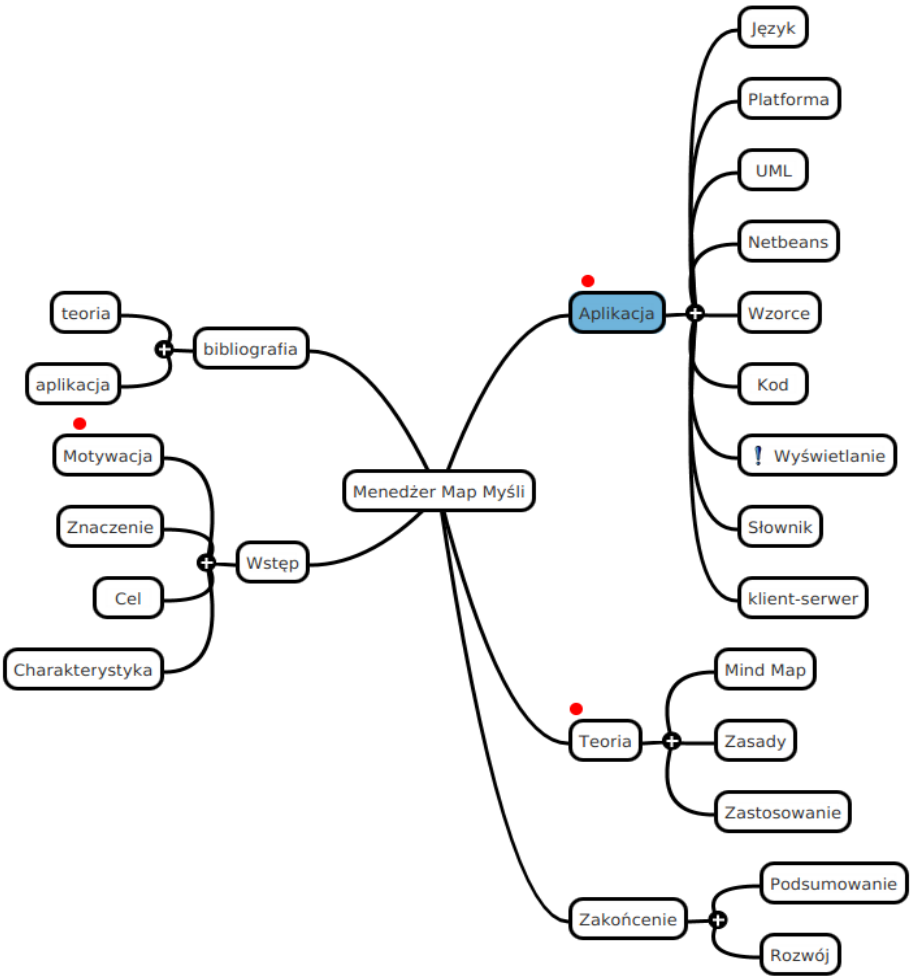
Mind Maping to narzędzie wspomagające umiejętności zawodowe, w tym zarządzanie czasem pracy, rozmowami telefonicznymi, pocztą elektroniczną i robieniem notatek. Pozwala

zorganizować proces negocjacji , prezentację firmową i co najważniejsze, pomóc w skutecznym zarządzaniu nawet największymi projektami.

Trudno się obyć bez map w zarzadzaniu strategicznym organizacjami, w szczególności w procesach innowacyjnych i wynalazczości [2],[5].

Poniżej zaprezentowano mapę myśli niniejszej pracy zrealizowaną w napisanej aplikacji.

Rys 1. Mapa myśli pracy



Źródło: Własne

3 Ocena dostępnych programów

Tworząc mapy myśli ręcznie uzyskujemy najlepszy efekt, pobudzamy obie półkule mózgu, ale w ten sposób powstają silnie zindywidualizowane projekty czytelne tylko dla autora lub współautorów, a więc do zastosowań nieformalnych.

Zastosowanie map myśli do organizacji pracy zespołu, współtworzenie map, dystrybucja map dla członków zespołu, tworzenie map zrozumiałych dla członków zespołu, szczególnie w warstwie słów kluczy narzuca konieczność stosowania odpowiednich programów komputerowych do tworzenia map myśli. Są to zastosowania formalne. Większość programów daje możliwości odręcznego tworzenia map, wypełniania ich rysunkami, kolorami, ikonami i innymi treściami, które można swobodnie zmieniać i uzupełniać. Szczególnie cenne jest linkowanie stron internetowych oraz plików różnych typów. Daje to duże możliwości tworzenia repozytorium dokumentów w rozwiązaniach intranetowych czy też archiwów w ramach Internetu.

Wirtualna mapa myśli daje nieograniczone możliwości co do wielkości i treści ale z zachowaniem poziomu szczegółowości ograniczonego jedynie ekranem urządzenia wykorzystywanego do pracy.

3.1 Przegląd darmowych rozwiązań

Poniżej zaprezentowano krótką charakterystykę popularnych darmowych rozwiązań w dziedzinie tworzenia map myśli.

- **FreeMind** – informacje dotyczące programu oraz jego wersje instalacyjne są dostępne pod adresem [17] Program jest dostępny na zasadzie licencji GPL. Dostępną wersją stabilną jest wersja 1.0.1. W tej wersji plik instalacyjny ma rozmiar 35,9 MB, co klasyfikuje go pośrodku pod względem wielkość. Program do poprawnego działania wymaga środowiska uruchomieniowego Java. Program dzięki wykorzystaniu technologii Java jest przenośny pomiędzy różnymi platformami sprzętowymi oraz programowymi. Nie mniej łatwość jego przenoszenia wraz z użytkownikiem jest ograniczona przez konieczność instalacji. Aplikacja posiada wiele skrótów klawiaturowych, które zaawansowanym użytkownikom mogą pozwolić na sprawniejsze używanie programu. Tworzenie mapy jest proste, choć domyślnie zastosowane skróty wydały się autorowi pracy nieintuicyjne. Program wspiera wiele dodatkowych treści takich jak: kolory ikony, odnośniki, notatki, załączniki itp.

Dołączanie tych treści do mapy nie stanowi problemu. Do budowy relacji pomiędzy gałęziami służą strzałki oraz chmury. Duże korzyści daje możliwość zwijania mapy, gdy pracujemy na dużej mapie a chcemy ograniczyć obszar pracy do pewnego fragmentu.

- **Blumind** – informacje dotyczące programu oraz plik instalacyjny są dostępne pod adresem [18]. Dostępną stabilną wersją jest wersja 3.1. W tej wersji plik instalacyjny zajmuje tylko 872,754 KB. Strona główna nie dostarcza informacji o licencji na jakiej program jest udostępniany. Informacje zawarte w pomocy programu zaznaczają że program jest darmowy dla każdego. Program do poprawnego działania wymaga środowiska .NET Framework 2.0 dostarczanego z większością obecnie dostępnych wersji systemu Windows. Interfejs jest wzorowany na tym znanym z MS Office 2010. Intuicyjny interfejs, który łatwo jest dostosować do potrzeb użytkownika, sprawia że tworzenie map jest bardzo proste. Do dyspozycji użytkownika są różne motywy graficzne uatrakcyjniające aplikację wizualnie. Aplikacja skupia swoje możliwości między innymi na szerokich możliwościach definiowania kolorów dla elementów mapy. Może to prowadzić do tworzenia nieczytelnych map myśli. Aplikacja posiada rozbudowany eksport danych. Z ciekawostek można wymienić minutnik – timer, służący do odmierzania wyznaczonego czasu.
- **XMind** – informacje dotyczącego programu oraz plik instalacyjny można pobrać ze strony [19]. Aplikacja jest dostępna pod licencją GPL oraz EPL. Ponieważ jednak program ma również wersję komercyjną dostęp do wielu jego funkcji jest uzależniony od wykupienia wersji rozszerzonej. Wersją stabilną jest wersja o numerze 3.4.1. W tej wersji wielkość pliku instalacyjnego programu jest największa pośród omawianych programów i wynosi 95,5 MB. Interfejs programu jest zbliżony do programu Bluemind, jest jednak znacznie bardziej rozbudowany, co może utrudniać odnalezienie się w nim początkującym użytkownikom. Program obsługuje formaty zapisu innych aplikacji tego typu, między innymi format programu Freemind. Posiada również rozbudowany import oraz eksport. Program jest wyposażony w ciekawe symbole graficzne. Ciekawą, wyróżniającą ten program funkcją jest możliwość publikacji mapy myśli w Internecie.

3.2 Propozycje funkcji nowego narzędzia

Analiza prezentowanych powyżej rozwiązań wskazuje, że aplikacje skupiają na podstawowych funkcjach edycji map myśli oraz uatrakcyjnienia ich wyglądu. Nierzadko odbywa się to kosztem łatwości użycia programu oraz ociężałością aplikacji. Autor pracy zdecydował się opracować inne ciekawe funkcjonalności dla tego typu aplikacji:

- implementacja słownika wyrazów bliskoznacznych jako repozytorium dla słów kluczy używanych do opisu gałęzi mapy. Docelowo wyposażenie aplikacji w mechanizmy analizy semantycznej tekstów umożliwi półautomatyczne generowanie map jako bazy wiedzy zawartej w analizowanych obiektach. Implementacja słownika sprawia że mapy stają się transparentne dla odbiorców w zakresie treści przekazywanych przez słowa klucze,
- obsługa pracy zespołowej w sieci lokalnej w architekturze klient-serwer. Wiele tworzonych map myśli zyskuje na treści, kiedy mapa jest tworzona przez klika osób. Mapa może być znacznie bardziej dogłębna dzięki połączeniu wiedzy wielu ludzi. Możliwość współtworzenia map w sieci lokalnej poprzez odpowiednio przystosowany program otwiera nowy zakres funkcjonalny dla programów do edycji map myśli,
- intuicyjny system sygnalizacji za pomocą kolorów rodzaju dołączonych informacji dodatkowych. Ponieważ mapy myśli mogą pełnić funkcję repozytorium danych, warto dostarczyć użytkownikowi prosty sposób zorientowania się jakich danych może się spodziewać w danych częściach mapy. Niektóre programy próbują to robić stosując ikony kojarzące się z rodzajem treści. Jednak z teorii map myśli wynika, że łatwiejsze do odbioru dla ludzkiego umysłu są proste sygnały, najlepiej kolory, ponieważ pobudzają wyobraźnie,
- zachowanie formy programu portable. Przedstawione w analizie aplikacje mają wspólną wadę , którą jest konieczność ich instalacji. Ponieważ notacja za pomocą map myśli powinna również służyć do tworzenia podręcznych notatek, użytkownik musi mieć łatwy dostęp do programu oraz możliwość bezproblemowego zabrania aplikacji ze sobą. Aplikacja dostarczona w jednym pliku która będzie działać bez rozpakowania jest znacznie łatwiejsza do przenoszenia.

Te właśnie propozycje udoskonaleń stały się przedmiotem niniejszej pracy.

4 Wykorzystane technologie i oprogramowanie

Rozdział ten ma na celu krótkie przybliżenie technologii w oparciu o które stworzono aplikacje będącą przedmiotem pracy, jak również oprogramowanie które przysłużyło się w fazie projektowania oraz implementacji.

4.1 Język programowania

Do stworzenia oprogramowania będącego przedmiotem niniejszej pracy wykorzystano język programowania Java. Wybór ten podyktowany był dopasowaniem cech tego języka do potrzeb projektu, oraz wysoką produktywnością jaką osiągają programiści w tej technologii. Dodatkowo aby poznać nowinki ze świata Javy, do wykonania projektu posłużono się wersją beta środowiska JRE (Java Runtime Envinorment), które w toku rozwoju pracy zostało udostępnione jako Java 8.

Język Java został stworzony w odpowiedzi na gwałtowny rozwój Internetu oraz zmieniające się w związku z tym wymagania stawiane aplikacjom [9]. Celem jaki postawiono temu językowi było ułatwienie dostarczania aplikacji do pracy w rozproszonym środowisku sieciowym, zapewnienie ich niezawodności i bezpieczeństwa. Do głównych cech Javy należą:

- **wieloplatformowość** – łatwość przenoszenia skompilowanych, gotowych programów pomiędzy różnymi platformami sprzętowymi oraz systemowymi, jest możliwa, ponieważ Java jest jednocześnie językiem kompilowanym oraz interpretowanym. Oznacza to, że proces kompilacji programu nie produkuje kodu maszynowego, a jedynie ustalony *bytecode* ,który musi być interpretowany w trakcie wykonania. Rolę interpretera pełni wirtualna maszyna Javy. Maszyna wirtualna oraz *bytecode* stanowią warstwę abstrakcji od platformy na jakiej oprogramowanie ma działać, dając możliwość uruchomienia programu na dowolnej platformie na którą dostępna jest implementacja maszyny wirtualnej,
- **niezawodnośc** – twórcy języka doszli do wniosku że, najczęstsze błędy programistyczne powstają na skutek ręcznego zarządzania pamięcią oraz nieprawidłowego użycia wskaźników oraz arytmetyki na nich. Tego rodzaju błędy są trudne do wykrycia ponieważ objawiają się dopiero gdy odwołanie naruszy ochronę pamięci, a wtedy informacje o źródle tego problemu są nie do ustalenia. Dlatego Java korzysta z automatycznego zarządzania pamięcią z odśmiecaniem. Dzięki temu programista nie musi pamiętać o zwalnianiu pamięci, co ogranicza możliwość jej

wycieku. Użycie odśmiecania pamięci implikuje wykluczenie użycia arytmetyki wskaźników, dostępne są jedynie referencje które są w istocie zabezpieczonymi wskaźnikami. Dzięki temu nie jest możliwie wykonywanie niekontrolowanych odwołań do pamięci, które mogłyby powodować niestabilnie działanie programu. Drugim aspektem niezawodności Javy jest wsparcie dla systemu wyjątków, które jest wbudowane w maszynę wirtualną. Dzięki temu każdy nawet najpoważniejszy błąd w kodzie, taki jak brak pamięci lub odwołanie pod niezainicjalizowaną referencję, może zostać odpowiednio obsłużone przez programistę. Podczas zgłoszenia takiego błędu programista dostaje szczegółową informację w którym miejscu w kodzie błąd ten powstał,

- **obiektowość** – ułatwianie wytwarzania coraz bardziej skomplikowanego oprogramowania wymagało zmiany dotychczasowego podejścia do programowania. Potrzebna była metodologia która była by bardziej intuicyjna dla człowieka. W tym czasie popularność zyskał paradygmat programowania obiektowego, którego ideą jest podział i pogrupowanie kodu w tzw. obiekty czyli elementy funkcjonalne łączące dane z operacjami które można wykonywać na tych danych. Dzięki obiektom łatwiej jest podzielić kod na autonomiczne bloki o jasno określonej funkcjonalności, które łatwo się testuje w testach jednostkowych Pozwala to również łatwiej zauważyć zależności między większymi fragmentami kodu. Takie podejście do programowania rozszerzone o koncepcję dziedziczenia pozwala na opis rzeczywistości przyjaźniejszy dla umysłu człowieka. Zalety programowania obiektowego został dostrzeżone i wykorzystane w nowym języku. Java została od podstaw zaprojektowana jako język obiektowo orientowany. Klasy opisujące obiekty stały się podstawową jednostką organizacji kodu. Obsługa polimorfizmu oraz RTI (Runtime Type Identification) zostały wbudowane w maszynę wirtualną, dając podstawy stworzenia języka całkowicie obiektowego,
- **produktywność** – ułatwienie dostarczenia oprogramowania wymaga od języka wsparcia dla wygodnego i szybkiego przekładania koncepcji na działający kod. Zapożyczenie znacznej części składni z popularnego języka C++, ułatwia programistom oswojenie się z nowym językiem. Przyspiesza to okres nauki języka i pozwala szybciej zacząć go używać jako wygodnego narzędzia. Najpoważniejszym jednak czynnikiem przyspieszającym programowanie w Javie jest jej pokaźna biblioteka, która jest znacznie szersza niż w konkurencyjnych języka kompilowanych.

Twórcy mogli sobie pozwolić na dostarczenie tak szerokiej biblioteki ze względu na uzależnienie programów w Javie od obecności środowiska uruchomieniowego, które dostarcza nie tylko maszynę wirtualną, ale także zestaw klas przydatnych podczas programowania. Dzięki temu programista ma dostęp do wielu gotowych rozwiązań przy zachowaniu małego rozmiaru swojego programu.

To właśnie te cechy pozwoliły stwierdzić iż język Java pasuje do wymagań niniejszej pracy.

Dodatkowym atutem Javy stał się jej rozwój i nowe możliwości które zostały wprowadzone w 8 już wersji środowiska oraz języka . Jedną z najciekawszych zmian jakie wprowadzono, było poszerzenie składni języka o obsługę **wyrażeń lambda**. Wyrażenia lambda są składniowym mechanizmem do przekazywania kodu jako parametru dla innego kodu. Dotychczas takie możliwości Java uzyskiwała pośrednio poprzez interfejsy oraz anonimowe klasy. Wyrażenia lambda są wygodniejszym mechanizmem który okazuje się bardzo przydatny przy programowaniu zdarzeniowym często spotykanym przy tworzeniu interfejsów użytkownika.

4.2 Platforma JavaFX

Platforma uruchomieniowa języka Java (JRE) od początku oferowała programistom istotną pomoc w szybkim rozpoczęciu pracy nad aplikacją. Jednym z ważniejszych elementów umożliwiających szybki start jest dostęp do biblioteki graficznej. Konkurencyjne języki jak chociażby C++ nie oferują bibliotek graficznych w standardzie. Programista musi na własną rękę poszukać odpowiedniej biblioteki, która może być mniej lub bardziej dopracowana. Pojawiają się też problemy przy przenoszeniu takiej aplikacji na inne platformy, biblioteki graficzne potrafią działać różnie pod kontrolą różnych systemów. Java znacznie ułatwia sprawę programistom dostarczając proste obiektowe API, do budowy graficznego interfejsu użytkownika.

Sytuacja uległa drastycznej zmianie kiedy na rynku pojawiły się takie technologie jak Flash oraz Silverlight. Technologie te umożliwiają szybkie i wygodne budowanie interaktywnych graficznych aplikacji działających na dowolnej platformie sprzętowej. W porównaniu do tych technologii biblioteka graficzna Javy nie oferowała porównywalnych możliwości. Wygląd podstawowych kontrolek był mało atrakcyjny, jego zmiana była trudna lub niemożliwa. Brakowało też wsparcia dla animacji. Utworzenie efektów które z łatwością udaje się osiągać we Flashu, wymagało sporego nakładu pracy po stronie Javy.

Zaczęły się więc prace nad stworzeniem nowej technologii która dawałaby

porównywalne z Flashem możliwości. Podążając za konkurencją zaprojektowano nowy język skryptowy, który miał usprawnić i ułatwić budowę aplikacji. Nowy język nazwano JavaFx Script, natomiast środowisko wykonawcze nazwano JavaFx. Technologia JavaFx doczekała się kliku wydań JavaFx 1.1,JavaFx 1.2, JavaFx 1.3. Jednak gdy projekt został przejęty przez Oracle, zaprzestano rozwoju JavaFx Script. Istniejące środowisko JavaFx zostało przekształcone w nową bibliotekę graficzną dla Javy. Przemiana ta dopełniła się całkowicie, kiedy postanowiono dołączyć bibliotekę JavaFx do podstawowego wydania środowiska uruchomieniowego Javy – stało się to w wydaniu JRE 7u6 [10] dołączono wtedy JavaFx w wersji 2.2.

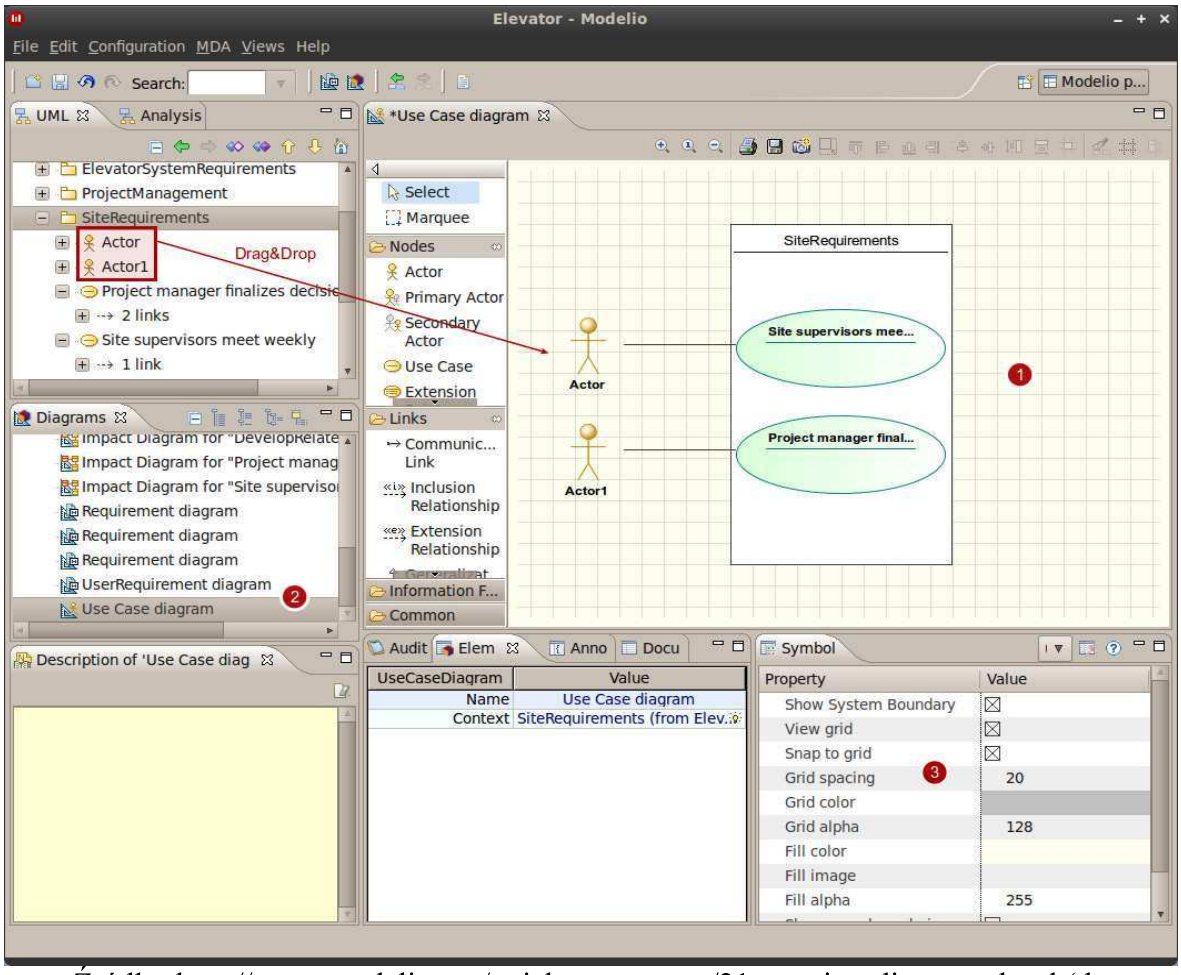
W taki sposób Java odzyskała konkurencyjność w zakresie tworzenia rozbudowanych interaktywnych interfejsów użytkownika. Przedstawione powyżej fakty skłoniły autora do wykorzystania tej technologii.

4.3 Edytor UML – Modelio

Projektowanie aplikacji będącej przedmiotem pracy odbywało się z wykorzystaniem notacji UML. Do stworzenia potrzebnych diagramów wykorzystano otwarte środowisko modelowania „Modeilo”. Główne cechy jakie zaważyły na użyciu tego programu to jego dostępność na zasadzie licencji GPLv3 oraz wieloplatformowość, uzyskana dzięki językowi Java.

Program posiada szerokie możliwości w zakresie modelowania. Umożliwia wygodne tworzenie diagramów w zgodzie ze standardem UML 2.0. Posiada funkcje kontroli poprawności modelu, prowadzi użytkownika w kierunku budowy diagramów, które są zgodne ze standardami, ale nie wymusza tych standardów bezwzględnie. Umożliwia generowanie kodu w języku Java dla przygotowanych diagramów klas. Dzięki wsparciu systemu wtyczek środowisko Modelio jest gotowe wspierać dowolny język bądź metodologie modelowania, jeżeli wymagana wtyczka nie istnieje, można ją napisać.

Rys 2. Interfejs programu Modelio



Źródło: http://www.modelio.org/quick-start-pages/21-creating-diagrams.html (data weryfikacji 30.05.2014)

Program okazał się być przydatnym narzędziem podczas tworzenia diagramów. Po zapoznaniu się z jego filozofią, staje się intuicyjny. Pomimo posiadania rozlicznych funkcji program działa sprawnie i płynnie.

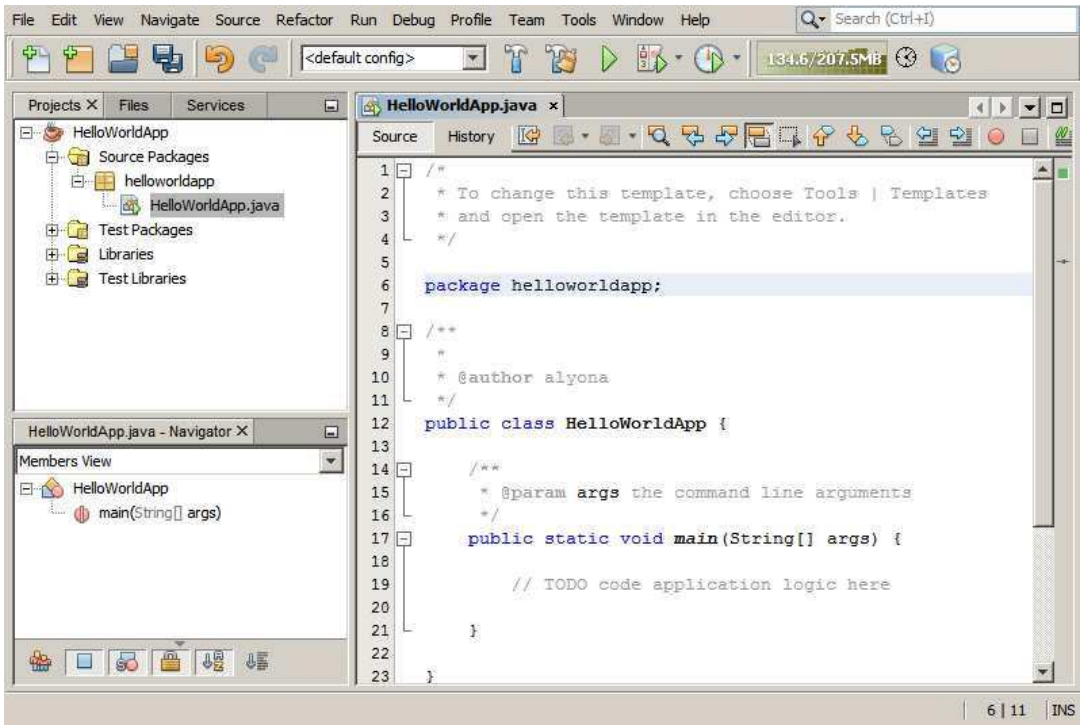
4.4 Środowisko programistyczne Netbeans 7.4 IDE

Zintegrowane środowisko programistyczne (ang. IDE) jest obecnie standardem w dziedzinie programowania. Jego zastosowanie przyspiesza tworzenie oprogramowania oraz pomaga zwiększyć niezawodność programów. Środowisko Netbeans jest jednym z głównych konkurentów w dziedzinie środowiska do programowania w Javie. Jest równie dobrze dostosowane do programowania w Javie jak Microsoft Visual Studio do programowania w C#.

Netbeans posiada liczne generatory typowo występującego kodu, jak chociażby generator konstruktorów dla klas czy typowych funkcji. Asystent wprowadzania kodu potrafi

przeszukiwać i podpowiadać nie tylko w ramach samodzielnie napisanego kodu, ale przeszukuje również podpięte biblioteki oraz umożliwia przeszukiwanie biblioteki standardowej. Udostępnia również wgląd w implementacje klas bibliotecznych języka Java, dzięki źródłom które są dostarczane wraz ze środowiskiem Java. Inteligenta wyszukiwarka potrafi wyszukiwać wszystkie miejsca użycia danej zmiennej klasy czy metody, co bardzo ułatwia analizę zależności między klasami. Przydaje się również kiedy chcemy pozbyć się starych fragmentów kodu, kiedy nie ma pewności czy nie są gdzieś jeszcze wykorzystywane. Posiada rozbudowany system powiadamiania o błędach programistycznych, jak również potrafi ostrzegać o potencjalnych błędach programistycznych. Umożliwi również programowanie w wielu innych językach jak chociaż PHP czy C++. Posiada rozbudowany system wtyczek które umożliwiają rozszerzanie funkcjonalności środowiska. Dla przykładu dostępna jest wtyczka która umożliwi tworzenie aplikacji dla Androida.

Rys 3. Interfejs programu Netbeans



Źródło: https://netbeans.org/images_www/articles/72/java/quickstart/proj-opened.png (data weryfikacji 30.05.2014)

Wraz z wydaniem wersji 7.4 wprowadzone zostało testowe wsparcie dla nowych funkcji języka Java udostępnionych wraz z wersją 8. Dzięki temu program potrafi sugerować, w których miejscach kodu można użyć nowej składni. Dzięki temu programista znacznie łatwiej może się zapoznać z nowymi funkcjami i nauczyć się ich stosowania.

5 Projektowanie i implementacja

Rozdział ma na celu prezentację toku rozumowania podczas projektowania oraz implementacji aplikacji jak również zastosowanych technik modelowania.

5.1 Wzorce projektowe

Wzorzec projektowy jest opisem pewnego powtarzalnego problemu lub dylematu pojawiającego się podczas projektowania interakcji pomiędzy klasami systemu, jak również dostarcza koncepcji możliwego rozwiązania problemu w maksymalnie abstrakcyjny sposób [15]. Zastosowanie wzorców projektowych umożliwia usprawnienie procesu projektowania jak również poprawia jakość projektu utrudniając popełnienie typowych błędów projektowych. Wzorce dzielą się na trzy kategorie:

- kreacyjne – grupa ta zajmuje się opisem problematyki alokacji obiektów w systemie. Dostarczają metod efektywnej inicjalizacji i konfiguracji złożonych obiektów,
- strukturalne – zajmują się opisem powiazań między obiektami systemu oraz sprawnym zarządzaniem dostępnymi interfejsami,
- behawioralne – zajmują się sprawnym przetwarzaniem obiektów oraz zarządzaniem zakresem odpowiedzialności obiektów wchodzących w relacje.

Rozwój graficznych interfejsów użytkownika oraz ich wzrastająca popularność dostarczyła nowego rodzaju problemów projektowych. Doprowadziło to do rozwoju nowego rodzaju wzorców – architektonicznych wzorców prezentacji. Wzorce tego rodzaju zajmują się problemem organizacji kodu wyświetlającego graficzny interfejs. Zajmują się logicznym podziałem kodu na części które były by niezależne od siebie, a przez to łatwe do utrzymania i testowania. Ogólnie rzecz przedstawiając, wzorce te proponują oddzielenie kodu obsługującego logikę biznesową programu, od kodu zajmującego się prezentowaniem danych na ekranie. Ze względu na fakt, iż biblioteka JavaFX w dość istotny sposób wspiera stosowanie takich wzorców projektowych, projekt niniejszej aplikacji oparto o wzorzec **MVP** (Model View Presenter). Koncepcja **MVP** wprowadza podział kodu na trzy części:

- **View** – to jest cześć kodu najbliższa użytkownikowi służąca do wyświetlania gotowych danych. Zajmuje się również przechwytywaniem zdarzeń i ich przekazywaniem do warstwy wyższej,
- **Presenter -** jest to warstwa pośrednicząca. Jej zadaniem jest przygotowanie danych do

prezentacji przez widoki oraz przetwarzanie logiki. Zarządza stanem warstwy najniższej,

- **Model** – najniższa warstwa kodu, służąca do przechowywania danych, które są prezentowane użytkownikowi. Zajmuje się zapisaniem danych do pliku lub bazy.

Wzorzec ten został wybrany do realizacji projektu, ze względu na dobrą separację kodu oraz łatwość jego zastosowania w środowisku JavaFX, pomimo przywiązania nazewnictwa niektórych rozwiązań do **MVC**(Model View Controller). Podczas klasyfikacji kodu zastosowano dodatkowo własną logikę podziału pomiędzy komponentami **Model** oraz **Presenter**. Za kod należący do modelu uznaje się kod przetwarzający takie dane, które muszą zostać utrwalone, zapisane do pliku lub w innej formie. Dane chwilowe reprezentujące stan interfejsu przechowywane są przez kod komponentu **Presenter**.

Podczas projektowania aplikacji w zgodzie z opisanym powyżej wzorcem przydatne okazało się wykorzystanie kilku pomniejszych wzorców:

- **Singleton** – wzorzec charakteryzujący problem zapewnienia istnienia tylko jednej instancji danej klasy. Rozwiązaniem jakie wzorzec proponuje jest uczynienie konstruktora klasy prywatnym lub chronionym. W zamian udostępnia się klasową, publiczną funkcję która będzie pobierać jedyną instancję klasy. Funkcja ta stosuje technikę leniwej inicjalizacji (ang. lazy initialization) – tworzy obiekt przy pierwszym wywołaniu,
- **Builder** – wzorzec rozdziela sposób tworzenia obiektu od jego reprezentacji. Elementami wzorca są: **klasa nadzorcy** której używa klient do wytworzenia obiektów, interfejs **budowniczy** dostarczający metod do budowy obiektów oraz interfejs **Produkt** definiujący wspólne cechy budowanych obiektów. Implementacje interfejsu **budowniczy** dostarczają niezależnych od siebie implementacji sposobów tworzenia obiektów. Dzięki temu możliwa jest elastyczna zmiana sposobu tworzenia obiektu. Wzorzec ten wykorzystano w projekcie w zdegenerowanej formie skupiając się bardziej na oddzieleniu zbierania danych do stworzenia obiektu od momentu jego utworzenia,
- **Abstract factory** – celem wzorca jest umożliwienie wytwarzania różnych obiektów wypełniających ten sam interfejs w taki sam sposób. Uzyskuje się abstrakcje od konkretnego typu obiektów jakie powstają. Wzorzec zakłada istnienie **abstrakcyjnej fabryki**, oraz **abstrakcyjnego produktu**, które dostarczają cech wspólnych.

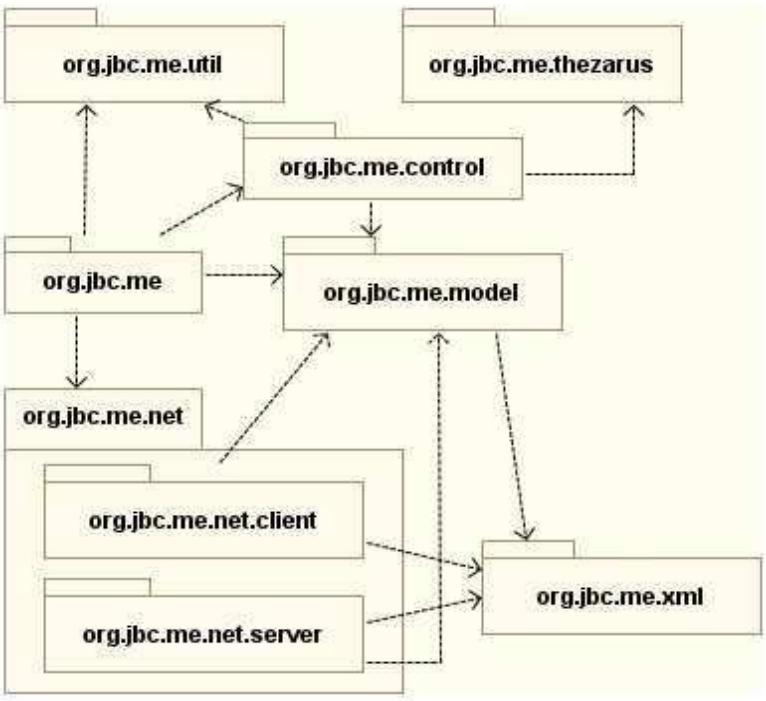
Konkretne fabryki będą dostarczały konkretnych produktów, dzięki czemu można swobodnie dodawać nowe typy pochodne po produkcie. Dopisane odpowiadającej fabryki umożliwi użycie tych typów w istniejącym kodzie,

- **Observer** – wzorzec dostarcza mechanizmu pozwalającego na śledzenie zmian wprowadzanych w obiektach. Wprowadza dwa interfejsy. Pierwszy z nich **Observable** służy do identyfikacji tych klas, które są wstanie dostarczać powiadomień o zmianach. Drugi z nich to **Observer** identyfikujący klasy zainteresowane odbieraniem powiadomień od obiektów **Observable.** Klasa implementująca **Observable** może powiadamiać wielu klas **Observer.** Tak samo klasa **Observer** może monitorować wiele klas **Observable.**

5.2 Organizacja kodu

Wykorzystując możliwości organizacji kodu jakich dostarcza język Java, implementacja została podzielona na pakiety. Podział następował głównie ze względu na grupy klas realizujące wspólnie daną funkcjonalność. Podział ma na celu ułatwienie orientacji oraz zarządzania kodem. Organizacją klas wraz z zależnościami jakie występują pomiędzy pakietami znajduje się na poniższym diagramie:

Rys 4. Diagram pakietów



Źródło: Opracowanie własne

Zastosowane nazwy pakietów wymagają wyjaśnienia. Zaleceniem dla programistów Java jest, by nazwy pakietów były odwróconą nazwą DNS witryny firmy, która wytarza dane oprogramowanie [11]. Dlatego też nazwa pakietów została skonstruowana w taki sposób. Aplikację zakorzeniono w domenie .org ponieważ aplikacja nie osiągnęła poziomu rozwoju wystarczającego do komercjalizacji. Znaczenie poszczególnych pakietów dla aplikacji przedstawia się następująco:

- **org.jbc.me** – jest to główny pakiet aplikacji. Oprócz bycia kontenerem pozostałych pakietów zwiera podstawowe klasy budujące aplikację. Miedzy innymi klasę pochodną po **javafx.application.Application** pełniącą funkcję punktu wejścia do programu oraz **Presenter** głównego okna programu,
- **org.jbc.me.model –** pakiet ten składa się z interfejsu i implementacji modelu danych dla pojedynczego wpisu mapy myśli oraz obiektu będącego źródłem danych dla mapy myśli. Ze względu na umieszczenie kodu związanego z zapisem i odczytem map myśli do plików xml, ten pakiet używa pakietu xml. Klasy te zostały wydzielone aby zwrócić uwagę na zastosowanie wzorca projektowego MVP wymagającego obecności modelu,
- **org.jbc.me.xml** – zawiera klasy odpowiedzialne za parsowanie danych zapisanych w formacie xml. Klasy te są wydzielone do osobnego pakietu ponieważ każda z tych klas odpowiada za obsługę jednego dokumentu xml. Oddzielenie tych klas umożliwia szybkie ustalenie obsługiwanych dokumentów xml,
- **org.jbc.me.control** – pakiet stanowi serce aplikacji. Zawiera klasy niezbędne do implementacji komponentu renderującego mapę myśli,
- **org.jbc.me.thezarus –** pakiet zawiera klasy zapewniające obsługę słownika wyrazów bliskoznacznych,
- **org.jbc.me.util** – pakiet zawiera klasy narzędziowe, służące do rozwiązywania problemów implementacyjnych bądź upraszczania kodu,
- **org.jbc.me.net.server –** pakiet zawiera implementację serwerowej części komunikacji sieciowej aplikacji. Powiązanie z pakietem **xml** jest konieczne ze względu na potrzebę odczytywania zaktualizowanej gałęzi przesłanej w xml. Powiazanie z pakietem **model** wynika z użycia modelu mapy do zapisu całej mapy do xml podczas jej wysyłania do klienta,

- **org.jbc.me.net.client** – pakiet zawiera implementację komunikacji sieciowej od strony klienta jak również implementację okienka dialogowego wykorzystanego podczas inicjowania trybu edycji rozproszonej. Powiązanie z pakietem **xml** jest konieczne ze względu na konieczność odebrania przesłanej mapy myśli.

5.3 Wyświetlanie mapy myśli

Komponent MindMap jest sercem programu. Jest elementem odpowiedzialnym za wyświetlanie mapy myśli oraz możliwości jej edycji strukturalnej czyli dodawania, usuwania gałęzi oraz przeciągania gałęzi z jednego miejsca w drugie.

Biblioteki wspierające budowę graficznych interfejsów użytkownika dostarczają swoich możliwości w formie gotowych „klocków", za pomocą których można budować złożone interfejsy użytkownika. Takie elementy różnie są nazywane w różnych bibliotekach. Framework JavaFX nazywa te komponenty kontrolkami. Nazwa wywodzi się z tego, iż taki komponent to klasa pochodna po **javafx.scene.control.Control.** Zestaw dostępnych kontrolek jest na tyle bogaty iż odpowiednie ich zestawienie i złożenie pozwala zrealizować wiele mniej lub bardziej skomplikowanych interfejsów.

Problem wyświetlania mapy myśli nie jest trywialny w rozwiązaniu. Należy zachować maksymalną elastyczność wyświetlania, aby pozwolić użytkownikowi swobodnie odwzorować swoje myśli. Ponieważ mapa myśli jest w swojej istocie strukturą drzewiastą, jej rysowanie można uprościć sprowadzając je do rysowania zwykłego drzewa. W związku z tym jednym z pomysłów na realizację tej części programu, było wykorzystanie dostarczonych przez JavaFX kontrolek w szczególności kontrolki **TreeView,** która potrafi rysować drzewo. Taki pomysł miał szansę powodzenia, ze względu na możliwości jakie pozostawili twórcy tej kontrolki.

TreeView jest zbudowane z mniejszych kontrolek nazywanych komórkami (**TreeCell**). Każda gałąź z drzewa jest wyświetlana przez jedną komórkę. Ponieważ jednak wykorzystano wzorzec projektowy **Abstract factory**, pozostawiona została możliwość dostarczania własnych klas pochodnych po **TreeCell**. Wzorzec jest zastosowany w nieco zmodyfikowanej formie ze względu na nowe możliwości języka Java. Otóż Fabryka Abstrakcyjna (**abstract factory**) jest interfejsem funkcjonalnym, który może być implementowany poprzez podpięcie referencji do funkcji o odpowiedniej sygnaturze. Dlatego fabryka jest de facto zwykłą funkcją której można dostarczyć do wytwarzania komórek w drzewie. Jednak samo modyfikowanie komórek nie byłoby wystraczające, ponieważ

istotnym elementem mapy myśli są połączenia pomiędzy gałęziami. Połączenia takie musiały by się znaleźć w przestrzeni pomiędzy komórkami, która jest zarządzana przez kontrolkę drzewka bez możliwości ingerencji. Dodatkowo układ komórek jaki stosuje kontrolka **TreeView** nie podlega zmianom. Przez to byłoby trudne czy wręcz nie możliwe uzyskać układ w którym gałęzie otaczają idee główną. Na tej podstawie narodził się pomysł aby napisać własną kontrolkę której zadaniem będzie wyświetlanie mapy myśli. Dodatkowym atutem który przemawiał za tym rozwiązaniem była hermetyzacja kodu i możliwość wykorzystanie gotowego komponentu w innych przyszłych projektach.

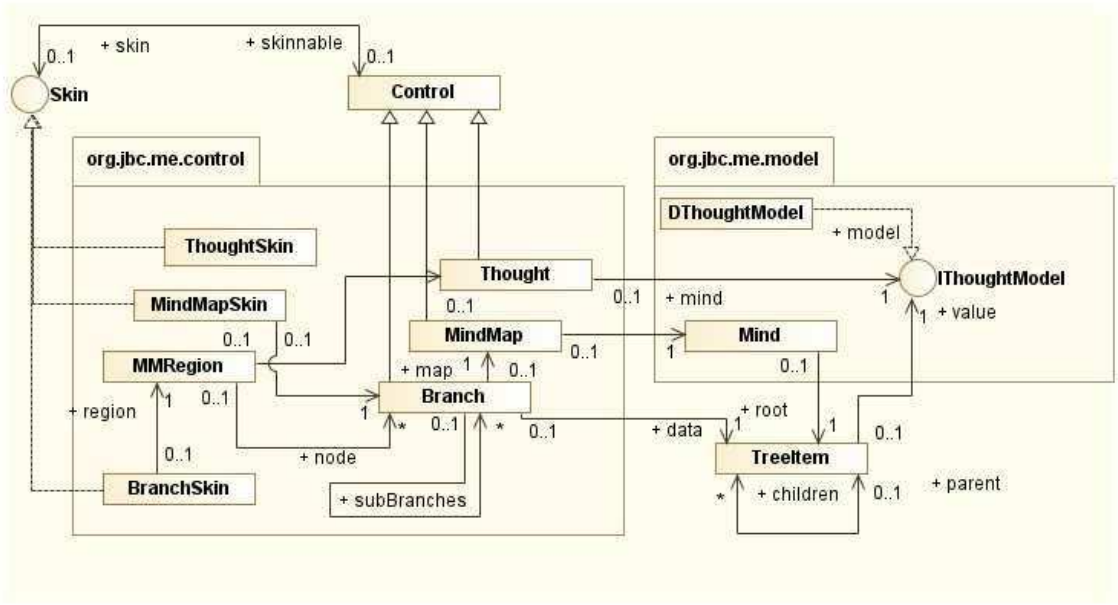
Interfejs programistyczny przewidziany do tworzenia własnych kontrolek jest zaprojektowany z myślą o podziale kodu ze względu na zadania. Można tutaj zauważyć podział na model, którego zadaniem jest przechowywanie wyświetlanych danych oraz przetwarzanie ewentualnej logiki programu, oraz widoku którego zadaniem jest wyświetlenie danych za pomocą dostępnych prymitywów geometrycznych (geometric primitives). W związku z czym implementacja własnej kontrolki musi się składać z co najmniej dwóch klas. Pierwsza z nich to oczywiście pochodna po **javafx.scene.control.Control,** które pełni funkcje komponentu **Model**. Drugą z nich jest klasa implementująca interfejs **javafx.scene.control.Skin** (czyli skórka) która pełni funkcje widoku.

Widać tutaj wyraźny potencjał do zastosowania wspominanego wzorca projektowego **MVP**. Uściślając jeżeli przyjrzymy się implementacjom kontrolek JavaFX możemy zauważyć dopełnienie podziału na 3 komponenty wymagane przez **MVP**. Implementacje te posługują się koncepcją **Zachowania** (ang. Behavior, tak nazywa się wykorzystywany interfejs). **Zachowanie** przejmuje obsługę zdarzeń z widoków i przekazuje je do modelu. Dzięki oddzieleniu tej części kodu jako niezależnego komponentu, kontrolki JavaFX zbliżają się do wzorca MVP. Nie mniej jednak koncepcja **Zachowania** nie została udostępniona do użytku programistom, nie można było wykorzystać jej w pracy. Ponieważ jednak zdecydowano że, aplikacja będzie utrzymana w zgodzie ze wzorcem MVP należało wyeksponować trzy komponenty kodu. Dlatego też koniecznie stało się rozdzielenie komponentu **Model** na dwie częsci. Aby tego dokonać komponent **Model** zostaje przemianowany na **Presenter** a cześć jego odpowiedzialność zostaje wydzielona do nowego komponentu **Model.** Nowy **Model** ma na celu przechowywanie danych które mają podlegać trwałemu przechowywaniu. Te dane które program musi zapisywać do pliku. Nie zaliczają się do nich chociażby informacje o tym czy dana gałąź jest zaznaczona czy nie. Tego rodzaju tymczasowe informacje pozostają odpowiedzialnością komponentu **Presenter**. Dodatkowo cześć bardziej wysoko poziomego

reagowania na działania użytkowania zostanie przeniesiona z poziomu komponentu **View** do komponentu **Presenter**, aby poziom odpowiedzialności poszczególnych komponentów był w miarę równy.

Elementem który dodatkowo wzięto pod uwagę projektując kontrolkę była technika wyświetlania danych stosowana przez bibliotekę JavaFX. Wszystkie komponenty graficzne są ułożone w strukturę drzewiastą która jest później przetwarzana na bitmapę przez wątek renderujący. Przeprowadzanie prezentacji mapy mysli przy użyciu jednej kontrolki byłoby niepotrzebnie obciążające. Koniecznością stało by się spłaszczenie drzewiastej struktury mapy do pojedynczego komponentu w drzewie kontrolek JavaFX. Wymagało by to implementacji własnego mechanizmu przetwarzania, który powodowałby wspominanie niepotrzebnie obciążenie aplikacji. Dlatego tez postanowiono wykorzystać istniejący system przetwarzania elementów wizualnych zamiast implementować własny. W tym celu należało podzielić zadanie jakie ten komponent miałby realizować na mniejsze części, które będą realizowane przez stosowne kontrolki. Dodatkową zaletą takiego rozwiązania jest możliwość pozostawania większej elastyczności kodu. Zastosowanie wzorca projektowego **Abstract factory** umożliwia dostarczanie różnych podklas realizujących wspomniane części komponentu **MindMap** w analogiczny sposób jak analizowana kontrolka **ListView** posługuje się mniejszymi komponentami **TreeCell.** Ostatecznie kod który powstał w celu wyświetlania mapy myśli wygląda tak jak to przedstawiono na poniższym diagramie klas:

Rys 5. Diagram klas realizujących komponent mapy myśli



Źródło: Opracowanie własne

Na powyższym diagramie przedstawiono zestaw klas zaplanowanych do realizacji kontrolki renderującej mapę myśli oraz relacje jakie występują między tymi klasami. Dodatkowo aby obraz zależności jakie występują pomiędzy klasami był przejrzysty na tym samym diagramie przestawiono klasy i interfejsy będące częścią biblioteki JavaFX. Aby łatwo było się zorientować które klasy należą do biblioteki zastosowano następującą koncepcję. Klasy zaimplementowane przez autora niniejszej pracy zostały przedstawione wewnątrz odpowiadających im pakietów. Klasy znajdujące się poza pakietami należą do biblioteki.

Główną klasą jest **MindMap,** tutaj znajdują się referencje do modelu mapy myśli, informacja o tym, która z gałęzi została zaznaczona, oraz dwie fabryki. Pierwsza z nich produkuje implementacje interfejsu **IThoughtModel,** druga natomiast służy do produkcji kontrolek zdolnych do prezentacji tego modelu. Modelem mapy myśli jest klasa **Mind.** Zasadniczo obecność tej klasy nie jest potrzebna, ponieważ cały kod później korzysta ze zbudowanej na klasie **TreeItem** struktury drzewiastej. Technicznie nawet bez tej klasy separacja pomiędzy komponentami wzorca **MVP** jest zapewniona, nie mniej jej zastosowanie tej klasy uwypukla komponent **Model**. Model dla mapy myśli stanowi kontener na kod związany z przetwarzaniem tej mapy. Chodzi o odczytywanie mapy z pliku i zapis do pliku. Komponentem **View** dla dwóch poprzednich klas jest klasa **MindMapSkin.** Ze względu na zastosowany podział odpowiedzialności komponent ten zasadniczo deleguje zadanie rysowania mapy do klasy **Branch,** sam odpowiada jedynie za pozycjonowanie całości. Te trzy klasy stanowią pierwszy najwyższy poziom przetwarzania mapy myśli.

Kontrolka **Branch** jest elementem kluczowym dla działania całości. Jej odpowiedzialnością wraz z widokiem jest zarządzanie pojedynczą gałęzią w mapie, ale z uwzględnieniem swoich podgałęzi. Klasa **Branch** jest Prezenterem tej warstwy. Jej głównym zadaniem jest utrzymywanie synchronizacji pomiędzy gałązkami jakie znajdują się modelu, a listą pod gałęzi widocznych na ekranie. Klasa ta nie posiada własnej klasy modelu gdyż nie było takiej konieczności biblioteczna klasa **TreeItem** idealnie nadaje się na model. Synchronizacja o której wspomniano odbywa się za pomocą mechanizmów JavaFX. Dzięki specjalnym kolekcjom danych które obsługują wzorzec **Observer** możliwe jest wygodne reagowanie na zmiany w modelu klasy **Branch**, dodatkowo dzięki wprowadzeniu referencji do funkcji kod synchronizacji może być umieszczony w czytelnym miejscu czyli jako metoda klasy **Branch.** Mechanizm powiadomień dostarcza informacji o tym które obiekty zostały usunięte a które dodane, dzięki temu funkcja synchronizacji działa

efektywnie. **Presenter** tej warstwy zajmuje się również obsługą mechanizmu przeciągania. Są tutaj zadeklarowane odpowiednie metody, które następnie są podpięte do odpowiednich zdarzeń w komponencie **View** gałęzi

Jak można wyczytać z diagramu klasa **Branch** jest powiązana z **MindMap**. To powiązanie jest potrzebne aby dostarczać wspólnych dla wszystkich gałęzi informacji Powiązanie z mapą myśli jest wykorzystane podczas implementacji mechanizmu przeciągania. Typowo przeciąganie jest implementowane jako przekazywanie danych przez schowek systemowy. Niestety schowek systemowy umożliwia transfer jedynie wybranych typów danych, takich jak tekst czy obrazek. Przeciąganie gałęzi w mapie myśli wymaga transferu fragmentu modelu czyli wybranego poddrzewa. Ze względu na to wykorzystano własny mechanizm przekazywania danych podczas przeciągania **Presenter MindMap** służy do przechowania referencji do poddrzewa które jest przenoszone oraz do przechowania gałęzi do której ma zostać wykonane przeniesienie. Referencje te są wypełnianie przez kolejne zdarzenia związane z przeciąganiem. Po zakończeniu przeciągania zostaje wykonane faktyczne przeniesienie danych. Zmiana wprowadzana jest jedynie w modelu danych, jej odzwierciedlenie na ekranie odbywa się poprzez mechanizmy powiadamiania w tym poprzez wspominaną już funkcję synchronizacji.

Posiadanie referencji do klasy **MindMap** zapewnia dostęp do fabryk. Użycie wzorca **abstract factory** uelastycznia połączenie gałęzi z komponentem najniższego poziomu. Komponentem najniższego poziomu jest edytor pojedynczego wpisu mapy myśli który jest reprezentowany przez **Presenter Thought**. Zadaniem tego komponentu jest graficzne zaprezentowanie tych danych zgromadzonych we wpisie które są przeznaczone do prezentacji na mapie. Dostęp do modelu danych tego komponentu odbywa się poprzez interfejs **IThoughtModel.** Zastosowanie tutaj interfejsu uniezależnia kod od wyboru sposobu przechowywania danych oraz sposobu realizacji mechanizmu powiadamiania o zmianach. Interfejs ten narzuca jednie jaki zestaw danych ma być dostępny oraz wymusza implementacje mechanizmu obserwacji zmian tych danych. Dostarczono jedną implementację wykonaną w najprostszy sposób z wykorzystaniem klas bibliotecznych. Najważniejszym elementem tych danych jest opis gałęzi, który ma być wyświetlany na mapie. Zadaniem kontrolki jest umożliwienie jego edycji. Dodatkowo wyświetlane są ikony których identyfikatory zostały zapisane w modelu. Kontrolka ta odpowiada również za realizację jednej z ważnych zaplanowanych dla programu funkcji jaką jest intuicyjna sygnalizacja informacji dodatkowych. Jej realizacja odbywa się poprzez zestaw wizualnych

oznaczeń o kolorach przypisanych do rodzaju treści oraz o ustalonej pozycji. Zastosowanie kolorów do przekazywania informacji na mapie myśli, jest zalecane przez jej twórców. [1]

Implementacja gałęzi korzysta z referencji do **MindMap** w celu pobrania fabryk. Pierwszą z nich jest fabryka tworząca implementacje **IThoughtModel.** Wytworzenie obiektów tej klasy jest konieczne w miejscach w których gałąź musi obsługiwać wstawienie nowej gałęzi. Druga z fabryk służy do dostarczania instancji kontrolki, ze względu na jej zastosowanie można zmusić gałąź aby korzystała z nowo napisanej klasy pochodnej a nie tylko klasy bazowej. Fabryka ta znajduje swoje zastosowanie głównie podczas obsługi synchronizacji danych wewnętrznych z modelem danych oraz podczas inicjalizacji kontrolki kiedy to reprezentacja graficzna wszystkich danych musi zostać utworzona.

Komponent **View** zajmujący się wyświetlaniem gałęzi w mapie to **BranchSkin.** Zadaniem klasy jest odpowiednie rozmieszczenie elementów wizualnych składających się na gałąź. Klasa ta zajmuje się również odwzorowaniem informacji o tym czy dana gałąź ma być zwinięta czy nie. Wizualna reprezentacja składa się z elementu tytułowego czyli kontrolki która ma przedstawiać treść gałęzi (tą funkcje spełnia kontrolka **Thought),** z elementu służącego do manipulacji stanem zwinięcia bądź rozwinięcia, oraz podgałęzi. Zadanie które ma spełniać **BranchSkin** w całości zostało jednak oddelegowane do dedykowanej klasy narzędziowej **MMRegion.** Klasa ta pełni funkcję nowego panelu który układa kontrolki w kształt zaprojektowany dla mapy myśli. Została stworzona aby oddzielić samą idee układania komponentów w formę gałęzi mapy myśli od reszty implementacji. Może to w przyszłości pozwolić na zbudowanie programu którego interfejs będzie wyglądał jak jedna wielka mapa myśli, przy dobrym projekcie mogło by to znacznie poprawić intuicyjność programu. Nowy panel poza utrzymywaniem położenia swoich komponentów dodatkowo musi zarządzać rysowaniem połączeń pomiędzy nimi. W celu uzyskania wyglądu zbliżonego do konkurencyjnych rozwiązań do rysowania wykorzystano biblioteczną klasę **QubicCurve,** która potrafi rysować na ekranie krzywą Beziera stopnia trzeciego.

Typowy wygląd mapy myśli składa się z tematu głównego umieszczonego na środku, od którego następnie wychodzą gałęzie. Analizowane programy pozwoliły dojść do wniosku że, w stopniu wystraczającym udaje się wypełnić przestrzeń dookoła tematu głównego, gałęziami które rozciągają się poziomo. Przygotowany panel w związku z powyższym musi obsługiwać aż trzy różne układy komponentów. Pierwszy to sytuacja

w której podgałęzie są wysunięte najbardziej na lewo, a gałąź nadrzędna znajduje się po ich prawej stronie wyśrodkowana względem wysokości. Drugi przypadek jest lustrzanym odbiciem pierwszego, nadrzędna gałąź jest po lewej stronie a podrzędne po prawej. Trzeci przypadek to układ który jest używany tylko raz na każdej mapie myśli jest to układ centralny. W tym układzie gałąź nadrzędna, która w tym przypadku jest korzeniem mapy, czyli ideą główną, znajduje się na środku całego obszaru mapy natomiast jej podgałęzie muszą zostać podzielone na 2 grupy. Grupa pierwsza będzie wyświetlana za pomocą kontrolek **Branch** z narzuconym kierunkiem układania na lewo, druga z kierunkiem narzuconym na prawo. Zastosowany algorytm podziału gałęzi jest bardzo prosty. Lista dostępnych gałęzi jest dzielona na połowę. Pierwsza połowa jest kierowana na lewo, druga zaś na prawo. Dzięki temu użytkownik odniesie wrażenie że przestrzeń wokoło głównej idei jest wypełniana przeciwnie do ruchu wskazówek zegara.

Z punktu widzenia implementacji, rozwiązanie to ma jedną wadę – dodawanie gałęzi do idei głównej będzie powodować zmianę punktu podziału gałęzi na prawe oraz lewe, a co za tym idzie pojawi się konieczność odwracania gałęzi. Z tego właśnie powodu nie zdecydowano się na implementację każdego z przewidywanych rozkładów jako osobnych podklas które w związku z powyższym trzeba by często zmieniać tworząc nowe obiekty. Obsługę różnych układów zapewniono w postaci osobnych funkcji realizujących rozmieszczanie komponentów. Wybór układu odbywa się poprzez zmianę wartości jednej zmiennej. Ponieważ jednak wartość tej zmiennej brana jest pod uwagę jedynie podczas układania komponentów na ekranie (to zadanie jest okresowo wykonywane przez wątek przetwarzania JavaFx) aby zmiana kierunku układu mogła mieć miejsce trzeba wymusić odświeżenie ekranu. Ponieważ jednak zaobserwowano, iż do zmiany kierunku panelu może dojść jedynie podczas dodawania nowych gałęzi do mapy myśli, które to i tak musi się zakończyć odświeżeniem zawartości ekranu, podczas zmiany zmiennej kontrolującej kierunek układu nie ma potrzeby wymuszać odświeżania.

Panel wyświetlający komponenty w formie gałęzi mapy myśli poza układaniem komponentów, oraz rysowaniem połączeń pomiędzy nimi musi dostarczyć jeszcze jednego elementu – potrzeba w jakiś sposób zwizualizować możliwość zwijania gałęzi użytkownikowi. W rozwiązaniu tego problemu zasugerowano się typowymi rozwiązaniami. Element umożliwiający zwijanie znajduje się zawsze pomiędzy gałęzią nadrzędną a podrzędnymi, ale tylko wtedy gdy istnieje przynajmniej jedna podgałąź. Jeżeli chodzi o sam wygląd, najbardziej intuicyjny oraz dobrze się komponujący jest znak plusa. Biblioteka

standardowa nie dostarcza klasy która umożliwiała by narysowanie takiego kształtu. Konieczna stała się implementacja takiego pomocnika.

Dostępne dla programisty są klasy rysujące koło oraz linie – te są szczególnie przydatne w celu budowy znaku plusa. Jednak pozostaje kilka pytań jak dokładnie ten znak powinien wyglądać. Pozostawiając maksymalnie dużą elastyczność na wygląd, powstała klasa z całkiem sporo ilością parametrów konstruktora. Jej użycie w kodzie byłoby nie praktyczne, ponadto większość z tych parametrów może mieć wartości domyślne nie wymagające zmiany. W takiej sytuacji rozsądne okazało się wykorzystanie wzorca projektowego **Builder**. Zastosowano uproszczoną jego wersję. Do dyspozycji jest klasa **PlusBuilder,** której celem jest zbieranie parametrów niezbędnych do utworzenia klasy rysującej plusa. Implementacja rysowania plusa została zahermetyzowana wewnątrz klasy **PlusBuilder**. Wszystkie parametry jakie klasa rysująca plusa obsługuje zostały udostępnione użytkownikowi w formie właściwości zgodnych ze standardem proponowanym do definiowania właściwości w komponentach Java Beans [12]. Dokonano jednak małego odstępstwa od tego standardu, aby umożliwić programiście łatwiejsze korzystanie z tej klasy. Bardzo wygodnym mechanizmem jest możliwość łańcuchowego wykonywania metod. (ang. method chaning),która umożliwia pominięcie tworzenia pomocniczych zmiennych tymczasowych. Aby umożliwić tą technikę wszystkie metody klasy **PlusBuilder,** które zajmują się zmianą stanu obiektu zwracają obiekt na rzecz którego zostały wywołane. Metody te nie muszą niczego zwracać, ale zwrócenie modyfikowanego obiektu umożliwia stworzenia łańcucha wywołań. Dzięki temu zabiegowi zbudowana klasa jest zgodna z techniką projektowania interfejsów programistycznych znaną jako Fluent API [13]. Poniżej przedstawiono fragment kodu odpowiedzialny na stworzenie domyślnego wyglądu dla plusa. Kod ten znajduje się w klasie **MMRegion**

Rys 6. Wykorzystanie klasy PlusBuilder

```
private Node createDefaultKnot()
{
  return  new PlusBuilder().
            setColor(Color.WHITE).
            setBackgroundColor(Color.BLACK).
            setShape(PlusBuilder.Type.CIRCLE).
            setBackgroundSize(15).
            setHsize(2).
            setVsize(2).
            setVlenght(7).
            setHlenght(7).
            create();
}
```

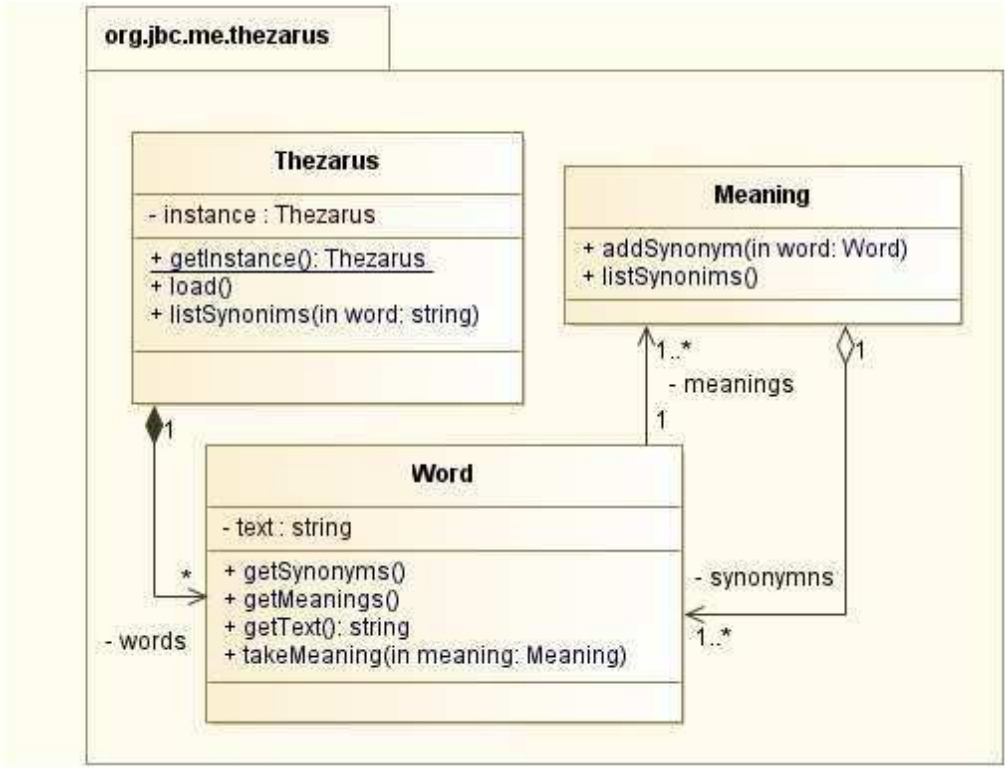
Źródło: Opracowanie własne

Łatwo jest zauważyć jak czytelny staje się kod programu dzięki łańcuchowi wywołania metod.

5.3.1 Słownik synonimów

Tworzenie mapy myśli to proces twórczy. Stworzenie dobrej mapy myśli wymaga od autora właściwego doboru opisów gałęzi. Znacznym ułatwieniem dla użytkownika byłaby sytuacja w której program pomagał by podczas dobru najlepiej pasującego określenia. Aby wyjść naprzeciw tym oczekiwaniom projektowana aplikacja została poszerzona o możliwość przeszukiwania słownika wyrazów bliskoznacznych w poszukiwana słów lepiej ujmujących to co użytkownik pragnie opisać. Podstawą poprawnego działania tej funkcji jest przygotowana wcześniej baza słów bliskoznacznych. Na szczęście samodzielne tworzenie takiej bazy nie było konieczne, ze względu na obecność darmowego narzędzia OpenOffice którego słowniki w tym słownik synonimów są dostępne w Internecie [14]. Baza jest zapisana w formie pliku tekstowego o czytelnym, formacie. Po zapoznaniu się z formatem pliku, należało zaprojektować odpowiednie klasy które pozwolą przechować informacje o synonimach oraz efektywnie je wyszukiwać. Powstałe klasy zostały przedstawione na poniższym diagramie:

Rys 7. Diagram klas – słownik synonimów



Źródło: Opracowanie własne

Sercem tego pakietu jest klasa **Thezarus**, umożliwia ona korzystanie z bazy synonimów. Została zaprojektowana z wykorzystaniem wzorca projektowego **Singleton**, ponieważ informacje zawarte w słowniku mogą być dość obszerne a przez to zajmować dużo pamięci, a nie ma żadnej potrzeby aby były wczytywane w więcej niż jednej instancji. Procedura ładowania słownika nie została ściśle związana z tworzeniem obiektu tej klasy, aby pozostawić większą władzę nad tym w którym momencie te informacje zostaną załadowane, jako że dane z pliku wymagają przetworzenia. Przeprowadzone testy wykazały ,że załadowanie pliku podczas normalniej procedury uruchamiania programu w wątku głównym nie powoduje zauważalnego opóźnienia.

Głównym problemem podczas projektowania tych klas była wieloznaczność słów. Gdyby każde słowo miało tylko jedno znaczenie i należało tylko do jednej grupy synonimów nie potrzebne by były klasy pomocnicze, ponieważ wtedy do przechowania informacji wystarczyła by zwykła tablica wielowymiarowa wsparta funkcją haszującej do szybkiego wyszukiwania. Sytuacja jednak jest inna i słowa potrafią należeć do wielu grup. W celu odwzorowania relacji w jakie wchodzą słowa stworzono klasę opisującą pojedynczy wyraz oraz klasę opisującą pojedyncze znaczenie, czy też grupę synonimiczną.

Schemat użycia słownika polega na wyszukaniu danego słowa w bazie a następnie przedstawienia wszystkich słów z grupy znaczeniowej do której należy. Logicznym więc wydało się ze klasa **Thezarus** musi posiadać listę wszystkich znanych słów, która dodatkowo będzie oferować szybkie wyszukiwanie. Aby zapewnić szybkie wyszukiwanie wykorzystano dostępną w bibliotece Javy klasę **HashMap,** która służy do zapamiętywania asocjacji pomiędzy kluczem a wartością. W tym przypadku kluczami są słowa w formie ciągu znaków, wartościami są obiekty klasy **Word.** Asocjacje są przechowywane z wykorzystaniem technik haszowania. Dla ułatwienia korzystania ze słownika zdefiniowano metodę **Thezarus.listSynonims()** która ma za zadanie zwrócić listę obiektów **Word** które są synonimami wskazanego słowa. Funkcja ta może jednak zwrócić wartość tylko wtedy gdy słowo należy do jednej grupy znaczeniowej – w przeciwnym wypadku zgłaszany jest wyjątek, ponieważ nie ma możliwości stwierdzenia, według której grupy znaczeniowej zwrócić słowa.

Słownik jest podpięty do kontrolki **Thought**, ponieważ to w tym miejscu użytkownik będzie chciał go użyć. Jest on dostępny po wprowadzeniu kontrolki w tryb edycji. Jest prezentowany jako menu podręczne. Kiedy użytkownik wpisze pożądany opis gałęzi, może sprawdzić czy istnieją dla niego jakieś synonimy. Wybranie jakiejkolwiek pozycji z tego menu wstawia wybrane słowo jako opis gałęzi zamiast dotychczasowej treści. Menu prezentuje wyniki zarówno wtedy gdy słowo należy do jednej czy też do wielu grup znaczeniowych. Grupy znaczeniowe są oddzielane w menu separatorami. Wyrazy które są wieloznaczne są wyróżniane wytłuszczeniem czcionki.

5.4 Zdalna Edycja

Jedną z kluczowych funkcji, która ma wyróżniać projektowaną aplikację jest możliwość zdalnego edytowania map myśli tworzonych przez innych ludzi. Ideą jest dać możliwość wielu użytkownikom do pracy nad jedną wspólną mapą myśli i to bez konieczności przebywania w jednym pomieszczeniu. Taka funkcja może poszerzyć obszary zastosowania notacji za pomocą map myśli. Można sobie wyobrazić zarządzanie dużym przedsięwzięciem w formie mapy myśli, która miała by za zadanie zebrać razem wszystkie aspekty projektu które wymagają uwagi. Taką mapę myśli może być ciężko stworzyć jednej osobie. Zakres tematyczny może wykraczać poza wiedzie pojedynczego człowieka. Zbudowanie takiej mapy mogło by być łatwiejsze w grupie gdzie, projekt jest inicjowany przez jedną osobę która nakreśla wstępną mapę myśli wyznaczając kierunki rozwoju, a następnie osoby które czują się najbardziej kompetentne wybierają jeden z zaprojektowanych celów/idei i pracują nad ich szczegółami.

5.4.1 Projekt komunikacji

Zadanie jakie stawia przed programem funkcja zdalnej edycji sprowadza się do synchronizacji jednostronnej dwóch modeli mapy myśli. Ponieważ jednak musi istnieć możliwość równoczesnej edycji jednej mapy przez kilku użytkowników należy rozwiązać problem jednoczesnej edycji. Ponieważ problem jednoczesnej edycji jest trudny, rozwiązano go poprzez likwidacje możliwość wprowadzania kolidujących zmian do mapy. Rozwiązanie to ma na celu przyspieszenie procesu powstawania aplikacji oraz osiągnięcie podstawowej użyteczność. Dalszy rozwój aplikacji będzie wymagał bardziej wyrafinowanego rozwiązania problemu. Aby wykluczyć możliwość edycji tych samych danych przez różnych użytkowników aplikacja stosuje metodę blokowania gałęzi. Kiedy użytkownik będzie chciał przeprowadzać zdalną edycję, będzie musiał wybrać pewien jej wycinek wybierając jedną z gałęzi jako tematu głównego swojej edycji. Nikt nie będzie mógł wybrać tej gałęzi dopóki nie zostanie zwolniona. Aplikacja musi udostępnić zdalnym użytkownikom wgląd w dostępne mapy. Nie można pozwolić na całkiem swobodny dostęp do map nad którymi pracuje użytkownik. W celu zapewniania pewnego poziomu bezpieczeństwa aplikacja wprowadza koncepcję publikacji mapy myśli. Autor danej mapy musi ją upublicznić aby zdalni użytkownicy mogli wprowadzać w niej zmiany. Przesłanie wybranej jako korzeń edycji gałęzi rodzi problem sposobu identyfikacji gałęzi. Aplikacja proponuje proste i skuteczne rozwiązanie nazwane roboczo „ścieżką indeksów". Ideą tego rozwiązania jest zapisanie ścieżki jaką należy pokonać od korzenia drzewa do wybranej gałęzi. Ponieważ model przechowuje podgałęzie na uporządkowanej lisice obsługującej indeksy, najprostszym wyborem staje się podawanie indeksów pod którymi znajdują się poszczególne gałęzie w swojej gałęzi nadrzędnej. Opierając się na tych rozważaniach, sesja zdalnej edycji powinna mieć trzy etapy:

1. **Wybór mapy myśli** – ustalanie do której mapy myśli dana sesja będzie się odnosić. Nie potrzeba tutaj żadnej skomplikowanej komunikacji wystarczy proste przesłanie upublicznionych map w formie tablicy ich tytułów. Klient odpowie indeksem z tej tablicy wskazującym której mapy chce użyć.
2. **Wybór gałęzi głównej** – na komunikację w tej części składać się będzie wysłanie wybranej mapy do klienta. Klient odpowie przesłaniem ścieżki indeksowej która zidentyfikuje wybraną gałąź.
3. **Transfer zdarzeń modyfikacji** – w tej części sesji serwer staje się pasywny, nasłuchuje przesłanych zdarzeń modyfikacji, oraz wprowadza je we wskazanej mapie

myśli. Klient przysyła polecenia, które odzwierciedlają zmiany jakie zaszły w mapie po jego stronie.

Zdecydowano aby komunikacja pomiędzy klientem a serwerem odbywała się tekstowo. Ponieważ ułatwia to przetestowanie każdej ze stron za pomocą prostego polecenia systemowego telnet**.** Ponad to aby bardziej ułatwić proces komunikacji przesyłanie danych odbywa się z wykorzystaniem formatu xml. Ten sam kod który służy do zapisu danych mapy w pliku xml, jest wykorzystywany w komunikacji sieciowej aby uniknąć pisania drugi raz bardzo podobnego kodu. Zaletą tego podejścia jest łatwość poszerzenia zestawu danych przechowywanych w mapie. Wystarczy zmienić tylko jeden zestaw funkcji odczytującej i zapisującej aby wszystko działało poprawnie. Omówienia wymaga protokół zastosowany w trzecim etapie sesji. Wyróżnić można trzy zdarzenia modyfikacji:

- **delete** – pierwsza linia powinna zwierać słówko „delete", natomiast kolejna ścieżkę indeksów do gałęzi która została usunięta. Kiedy serwer odbierze to polecenie usuwa wskazaną gałąź z edytowanej mapy,
- **insert** – pierwsza linia powinna zawierać słówko „insert", druga linia powinna zawierać ścieżkę indeksów do gałęzi do której dodano nowy wpis. Natomiast w trzeciej linii powinien się znajdować indeks pod który wstawiono nowy wpis. W przypadku dodania nowego wpisu na sam koniec listy indeks wstawiania będzie ostaniem dostępnym indeksem,
- **update –** pierwsza linia powinna zawierać słówko „update", w drugiej linii powinna się znaleźć ścieżka indeksów wskazująca gałąź której dotyczy aktualizacja, natomiast w dalszych liniach znajduje się dokument xml zawierający aktualny opis gałęzi – bez przesyłania podgałęzi. Serwer po odczytaniu przesłanego dokumentu xml ustawia stan wskazanej gałęzi na taki jaki znajdował się w dokumencie.

Dzięki takim komunikatom serwer jest w stanie utrzymać synchronizacje pomiędzy danymi na serwerze a danymi u klienta.

5.4.2 Implementacja serwera

Implementacja części serwerowej zawiera się w dwóch klasach pakietu **org.jbc.me.net.server.** Odpowiedzialność zarządzania udostępnianiem map myśli spoczywa na klasie **Publisher.** Zadaniem tej klasy jest zarządzanie listą dostępnych map oraz przyjmowanie połączeń od klientów. Klasa ta również opiera się o wzorzec **Singelton,** ponieważ posiada wewnątrz implementacje osobnego wątku nasłuchującego połączeń

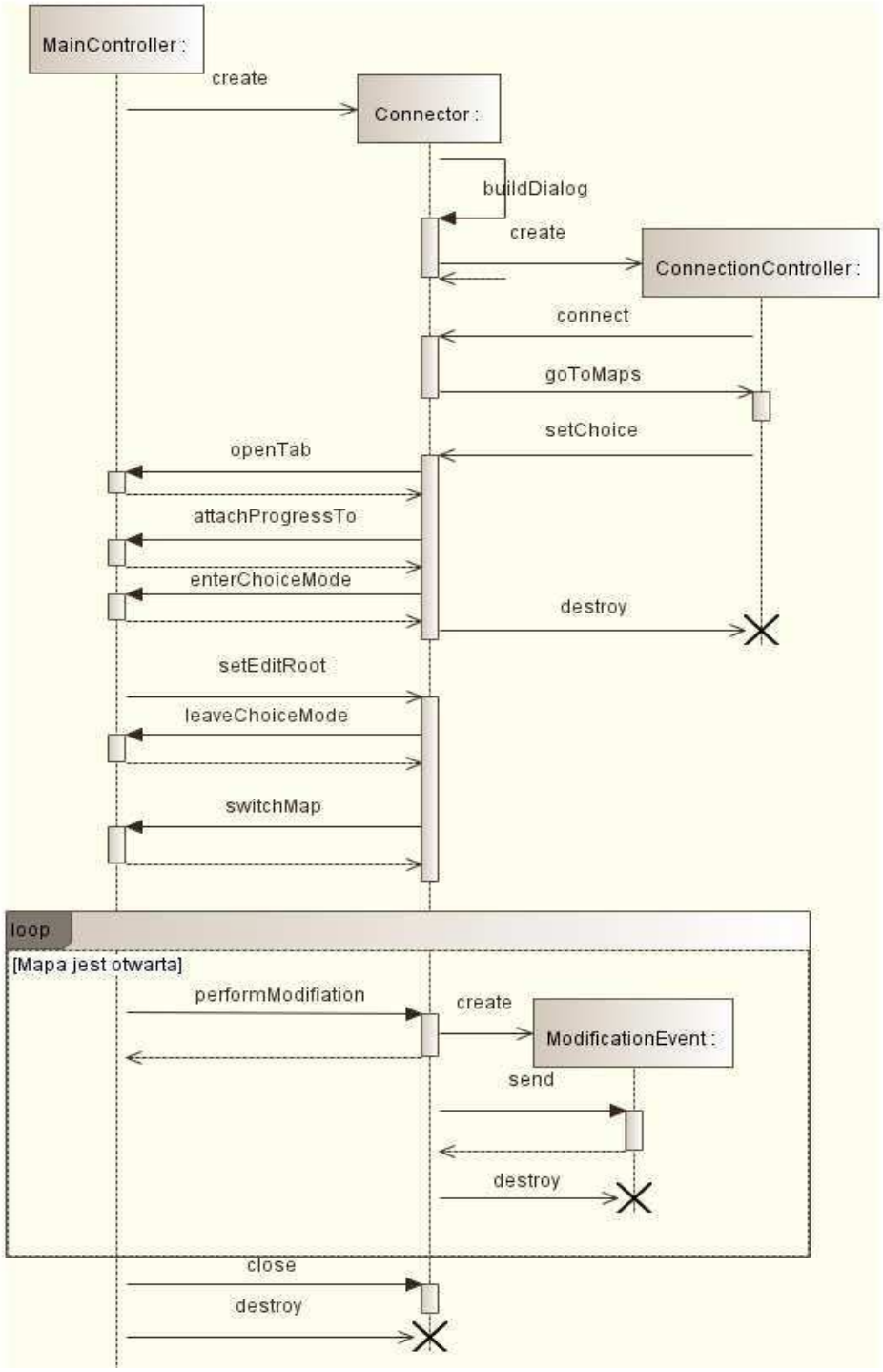
sieciowych. Ten wątek również powinien istnieć tylko w jednym egzemplarzu. Użycie tej klasy sprowadza się do przekazania modelu mapy którą się udostępnia wraz z tekstowym tytułem, który ma służyć użytkownikowi do identyfikacji o którą mapę chodzi. Najrozsądniejszym wyborem na tą nazwę wydała się nazwa pliku w którym tę mapę zapisano. Klasa **Publisher** automatycznie zarządza uruchomieniem wątku nasłuchiwania połączeń sieciowych. Zarejestrowanie pierwszej mapy myśli powoduje uruchomienie tego wątku i gotowość programu do przyjmowania zdalnych sesji. Analogicznie kiedy zakończymy publikację ostatniej mapy myśli nasłuchiwanie połączeń sieciowych traci sens, wtedy tez wątek nasłuchu zostaje zatrzymany. Po przyjęciu zdalnego połączenia do głosu dochodzi druga klasa **SSCThread.** Jej zadaniem jest obsługa pojedynczej sesji edycji. Ponieważ aplikacja chce obsługiwać więcej niż jedną sesje naraz, klasa ta została zaimplementowana jako osobny wątek przetwarzania, dzięki temu nie dochodzi do zablokowania wątku nasłuchu, którego jedynym zadaniem jest tworzenie obiektów **SSCThread** w odpowiedzi na nawiązanie połączenia przez klienta. Klasa **SSCThread** po odebraniu od klienta decyzji o wyborze mapy myśli rejestruje siebie w klasie **Publisher** aby w przypadku przerwania udostępniania przez autora, mieć możliwość zerwania otwartych połączeń oraz usunięcia sesji.

5.4.3 Implementacja klienta

Implementacja części klienckiej komunikacji zawiera się w trzech klasach z pakietu **org.jbc.me.net.client.** Głównym jej elementem jest klasa **Connector,** która obejmuje całość komunikacji z serwerem. Klasa ta również została zaprojektowana jako osobny wątek. W przypadku klienta, powiadomienia o zmianach w modelu są generowane i przysyłane przez wątek główny aplikacji JavaFx, metody obsługi tych zdarzeń nie mogą trwać zbyt długo ponieważ powstrzymało by to wątek główny od obsługi innych zdarzeń i mogło by doprowadzić do wrażenia że program się zawiesił. Przeniesienie odpowiedzialności za przesyłanie powiadomienia do osobnego wątku daje możliwość ich połączenia w większe grupy lub pominięcia powtarzających się powiadomień aby nie generować niepotrzebnego ruchu sieciowego. Zainicjowanie edycji zdalnej wymaga pobrania od użytkownika informacji takiej jak adres serwera, wybrana mapa myśli oraz gałąź główna. Najczytelniejszym sposobem na zrobienie tego jest przedstawienie użytkownikowi okna dialogowego które umożliwi, bądź podanie wymaganych danych lub przerwanie procesu łączenia się. Okienko to jest specyficzne dlatego też należało je samodzielnie zaimplementować. Interfejs okienka dialogowego został zapisany w pliku fxml

zaproponowanym przez JavaFx formacie do opisu graficznego interfejsu użytkownika. Prezenterem dla okienka dialogowego jest klasa **ConnectionController**, która zajmuje się przekazywaniem danych od klienta do klasy **Connector.** Zaprojektowane okienko umożliwia jedynie wskazanie adresu serwera oraz po pobraniu listy, umożliwi wybór mapy którą chcemy edytować. Aby umożliwić klientowi przeglądanie szczegółowej zawartości mapy myśli zanim dokona wyboru którą gałąź będzie edytował, zdecydowano aby pokazać całą otrzymaną z serwera mapę myśli w oknie głównym programu. Wyboru dokonuje się poprzez przeciągnięcie pożądanej gałęzi w przeznaczony do tego obszar po czym widok całej mapy myśli zostaje zastąpiony częścią która znajdowała się w wybranej gałęzi. Szczegóły komunikacji komponentów po stronie serwera przedstawia diagram sekwencji:

Rys 8. Diagram sekwencji – edycja zdalna



Źródło: Opracowanie własne

Diagram pokazuje w jakie relacje wchodzą poszczególnej komponenty programu, które są istotne podczas przeprowadzania zdalnej edycji. Klasa **MainController** jest komponentem **Presenter** głównego okna programu. Wykorzystanie tego diagramu pomogło ustalić które fragmenty kodu w klasie **Connector** powinny być objęte synchronizacją. Można zauważyć ze najwięcej synchronizacji jest wymagane kiedy użytkownik korzysta z okna dialogowego do momentu przesłania komunikatu *setChoice* pokazanego na powyższym diagramie. Obsługa przesyłania modyfikacji w klasie **Connector** została zrealizowana za pomocą kolejki zdarzeń modyfikacji. Kiedy użytkownik korzysta z mapy wątek przesyłający modyfikacje przez większość czasu jest uśpiony. Kiedy użytkownik wprowadzi jakąś zmianę wątek zostaje o tym powiadomiony poprzez pojawienie się w jego kolejce modyfikacji nowego zlecenia. Diagram wskazuje do którego momentu wątek klasy **Connector** oczekuje na zlecenia. W momencie zamknięcia zakładki wątek zostaje przerwany, a edytowana gałąź zostaje zwolniona.

6 Podsumowanie

Wytyczne do pracy przedstawione w rozdziale trzecim zostały zrealizowane, choć nie bez trudności.

Implementacja słownika wyrazów bliskoznacznych została zrealizowana w trybie tylko do odczytu. Dodawanie nowych synonimów do słownika jest nie możliwe, ze względu na inne założenie pracy dotyczące kompaktowości aplikacji. Aplikacja zawarta jest w jednym spakowanym pliku z którego można jej używać. Plik słownika znajduje się wewnątrz tej paczki co znacznie utrudnia możliwość edycji tego pliku. Nie udało się znaleźć eleganckiego sposobu na edycje zawartości spakowanego pliku. To niedociągnięcie jest jednak rekompensowane przez obszerność dostarczonego słownika. Rozwiązanie problemu z zapisem do słownika, lub zmiana jego lokalizacji otworzy aplikacji drogę do rozwoju słownika skojarzeń który znacznie bardziej będzie mógł wspierać proces tworzenia map myśli. Program obsługuje odczytywanie plików map wytwarzanych przez program FreeMind. Nie ma żadnych przeszkód aby aplikacja rozwinęła się w przyszłości i dostarczyła pełne wsparcie dla innych popularnych formatów.

Założenie dotyczące obsługi pracy zespołowej w środowisku sieciowym w architekturze klient – serwer zostało zrealizowane. Innowacyjność tej funkcji spowodowała powstanie licznych problemów podczas implementacji. Większość z tych problemów została rozwiązania w najprostszy sposób. Aplikacja prezentuje swoiste pierwsze podejście do problemu .To pozostawia aplikacji szerokie pole do rozwoju w zakresie tej funkcji. W przyszłości możliwe jest zastosowanie serwera pośredniczącego który umożliwiłby komunikację nie tylko w intranecie ale także w Internecie.

Aplikacja zrealizowała koncepcję sygnalizacji obecności dodatkowych informacji zawartych w mapie. Pomimo wąskiego zakresu dostępnych danych dodatkowych, funkcja ta jest pomocna przy określeniu w których miejscach mapy dodatkowe informacje są dostępne.

Aplikacja z sukcesem zrealizowała założenie zachowania formy programu portable. Dzięki zastosowaniu języka Java, aplikacja uzyskała przenośność pomiędzy różnymi platformami sprzętowymi oraz systemowymi. Udało się uzyskać program który jest jednym plikiem wykonywalnym spakowanym dodatkowo popularnym algorytmem pakowania zip. Dzięki temu jego całkowity rozmiar jest mniejszy niż 1 MB.

7 Literatura

[1] Buzan T., Mapy twoich myśli, wyd 2, Ravi, Łódź 2003

[2] Buzan T.,Griffiths C., Mapy Myśli w Biznesie, wyd 2, Aha BBC ACTIVE, Łódź 2010

[3] Buzan T., Mapy Myśli, Aha BBC ACTIVE, Łódź 2007

[4] Roam Dan, Narysuj swoje myśli, Helion, Gliwice 2010

[5] Jaworska-Jamruszkiewicz J., Kurs doskonalenia pamięci, wyd I, Videograf II Chorzów 2008

[6] Specyfikacja notacji UML - http://www.omg.org/spec/UML/ (data weryfikacji 30.05.2014)

[7] JavaFx - http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html (data weryfikacji 30.05.2014)

[8] Java 8 - http://www.oracle.com/technetwork/java/javase/downloads/index.html (data weryfikacji 30.05.2014)

[9] Cele projektowe języka Java - http://www.oracle.com/technetwork/java/intro-141325.html (data weryfikacji 30.05.2014)

[10] JavaFx - http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html (data weryfikacji 30.05.2014)

[11] Konwencja nazewnictwa pakietów języka Java - http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html (data weryfikacji 30.05.2014)

[12] Definiowanie właściwości w komponentach JavaBeans - http://docs.oracle.com/javase/tutorial/javabeans/writing/properties.html (data weryfikacji 30.05.2014)

[13] Interfejs Fluent - http://en.wikipedia.org/wiki/Fluent_interface (data weryfikacji 30.05.2014)

[14] Słownik synonimów programu OpenOffice - http://sourceforge.net/projects/synonimy/ (data weryfikacji 30.05.2014)

[15] Inżyniera oprogramowania – wzorce projektowe http://wazniak.mimuw.edu.pl/images/e/ed/Zpo-5-wyk.pdf (data weryfikacji 30.05.2014)

[16] Strona główna programu Modelio - http://www.modelio.org/ (data weryfikacji 30.05.2014)

[17] Strona główna programu FreeMind http://freemind.sourceforge.net/wiki/index.php/Main_Page (data weryfikacji 30.05.2014)

[18] Strona główna programu Bluemind - http://blumind.pl/ (data weryfikacji 30.05.2014)

[19] Strona główna programu Xmind - http://www.xmind.net/ (data weryfikacji 30.05.2014)

8 Wykaz rysunków