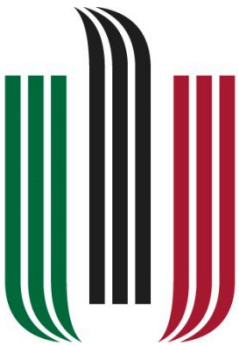


AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

DOKUMENTACJA TECHNICZNA

Implementacja narzędzia modelowania brył elastycznych

Tool for modelling flexible objects

Autor: *Piotr Baran, Michał Kasprzyk*
Kierunek studiów: informatyka
Opiekun pracy: *doktor inżynier Paweł Topa*

Kraków, 2015

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór
w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia
27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

.....

Spis treści

1. Wykorzystane technologie

W projekcie wykorzystano następujące technologie:

- **Java** - język w którym powstawała niemalże cała aplikacja;

- **JavaFX** - wykorzystany do stworzenia graficznego interfejsu użytkownika i połączenia go z logiką aplikacji;

- **JavaFX 3D** - wykorzystany do zwizualizowania układu;

- **JAXB** - Java Architecture for XML Binding - wykorzystywany do zapisu oraz wczytywania układu z pliku *.xml*;

- **JCSG** - biblioteka pozwalająca na połączenie kilku brył w jedną i eksport do pliku *.obj*.

Do kontroli wersji wykorzystany został system Git. Repozytorium zawierające kod aplikacji znajduje się w serwisie GitHub.

Projekt powstawał w zintegrowanym środowisku programistycznym IntelliJ IDEA.

2. Wizja architektury projektu

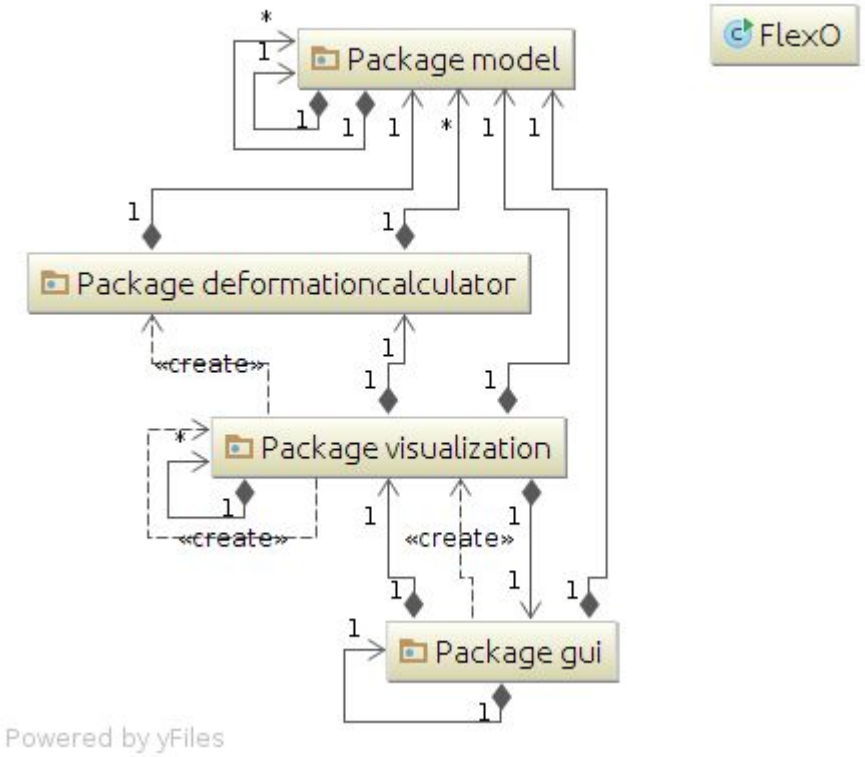
Do stworzenia podstawowej architektury został wybrany wzorzec *Model-Widok-Kontroler*. Powodów takiej decyzji było kilka. Przede wszystkim, MVC jest pierwszym wyborem, ku któremu skłaniamy się, mając do stworzenia aplikację z graficznym interfejsem użytkownika, która ma na celu pewną interakcję z użytkownikiem. Jest to klasyczny schemat umożliwiający logiczne rozdzielenie części aplikacji zgodnie z ich funkcjonalnością, zapewniając uporządkowany przepływ danych w aplikacji. Kolejnym istotnym powodem wykorzystania wzorca był fakt, że zdecydowaliśmy się wykorzystać technologię JavaFX do stworzenia GUI. Technologia ta, poza wprowadzeniem wygodnego formatu definiowania elementów interfejsu w postaci plików *.xml*, w naturalny sposób wprowadza do aplikacji kontrolery umożliwiające komunikację między modelem a widokiem.

Model aplikacji stanowi reprezentację dziedziny problemu, odzwierciedla stan konkretnego przypadku, który będzie rozwiązywany przez program. Zawiera informacje na temat położenia każdego z węzłów, a także o ich połączeniach z innymi węzłami, jak również parametry tych połączeń.

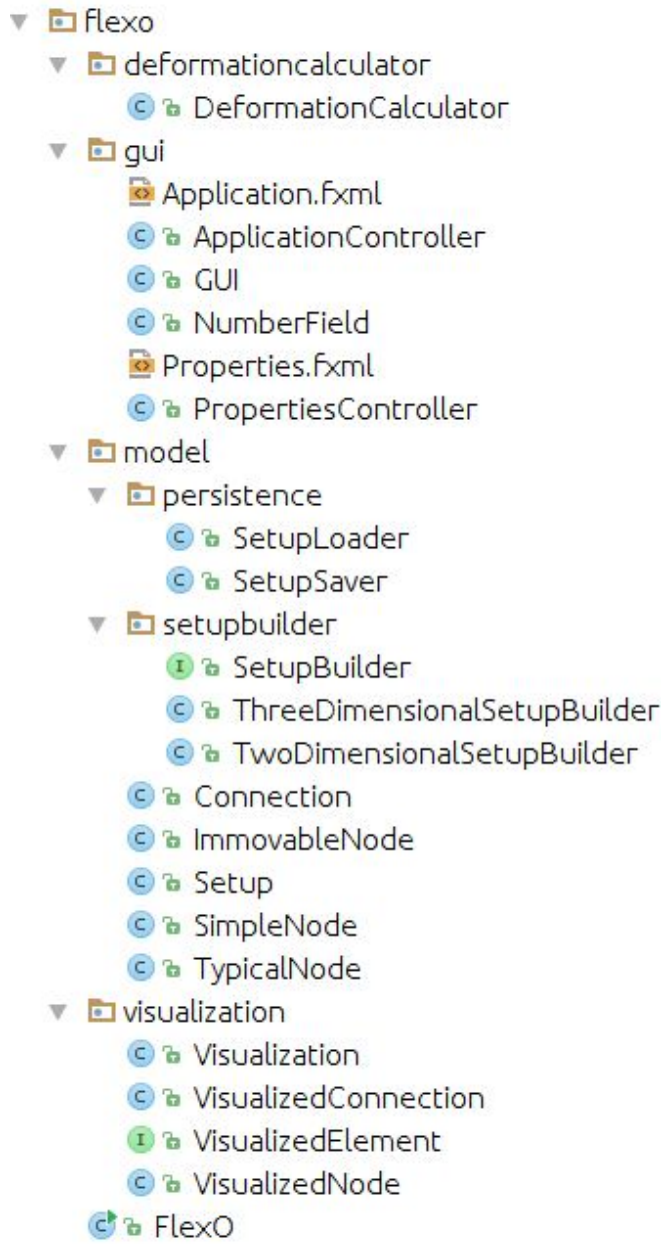
Widok to trójwymiarowa wizualizacja siatki oparta na danych zawartych w *Modelu*, a także elementy graficznego interfejsu użytkownika pozwalające na interakcję użytkownika aplikacji z warstwą odpowiadającą za wykonywanie obliczeń.

Kontroler to część, której zadaniem jest przyjmowanie danych od użytkownika, odpowiednia modyfikacja modelu i odświeżenie widoku, w zależności od tej modyfikacji.

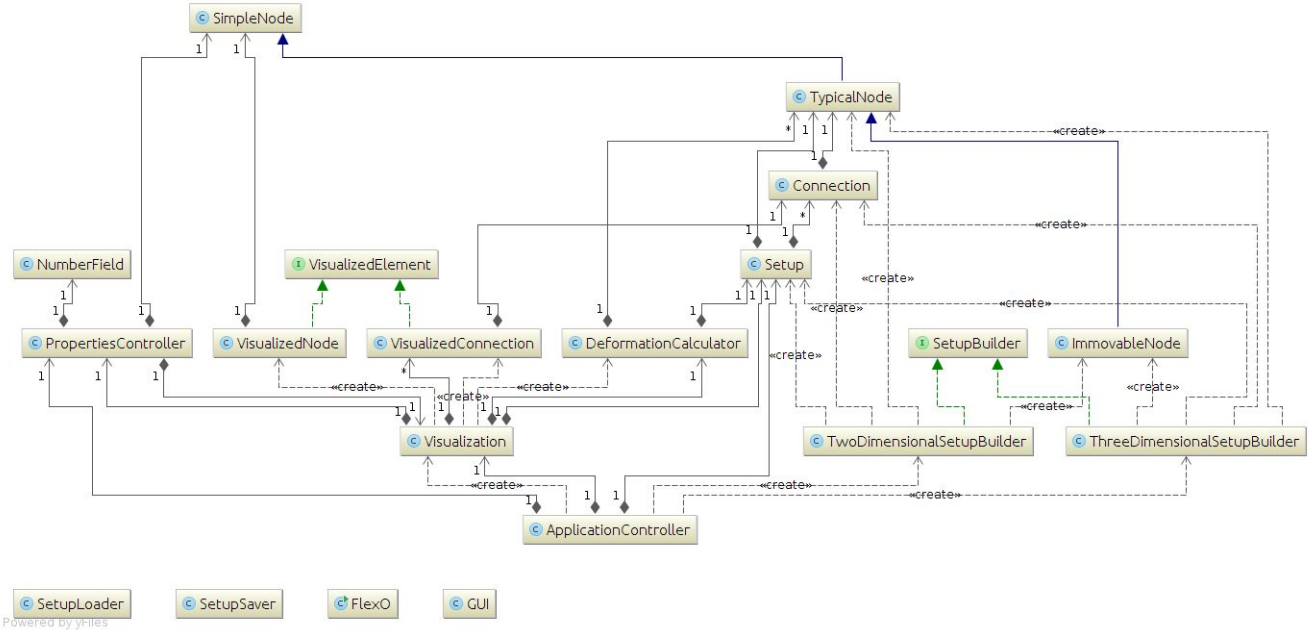
3. Struktura projektu



Rys. 1: Diagram pakietów



Rys. 2: Hierarchia plików



Rys. 3: Diagram klas

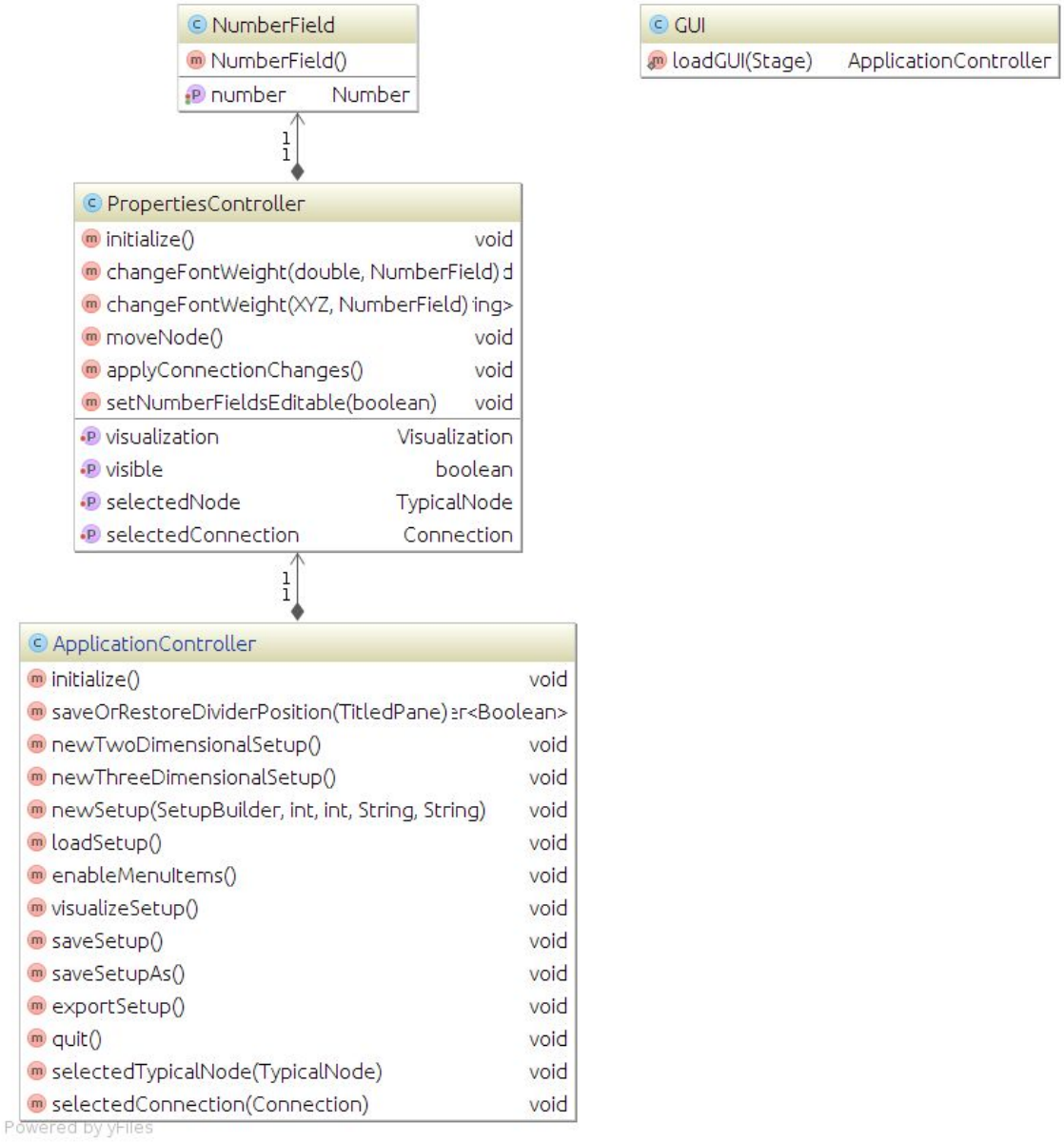
4. Opis modułów

4.1. Moduł głównego okna aplikacji

Rozkład elementów w głównym oknie aplikacji definiowany jest w pliku *Aplication.fxml*. Klasą zarządzającą oknem jest *ApplicationController*. Ponadto wykorzystany jest dodatkowy plik *Properties.fxml* odpowiadający za wygląd właściwości elementów. Jest on zarządzany przez klasę *PropertiesController*.

Wykorzystanie plików *.fxml*, będących plikami typu *.xml*, które zawierają informacje o ułożeniu elementów graficznego interfejsu użytkownika, pozwala na wygodną i szybką edycję układu okna. Ponadto pozwala w łatwy sposób oddzielić logikę aplikacji z warstwą definicji wyglądu okna. Rozwiązanie to wynika z wyboru technologii JavaFX.

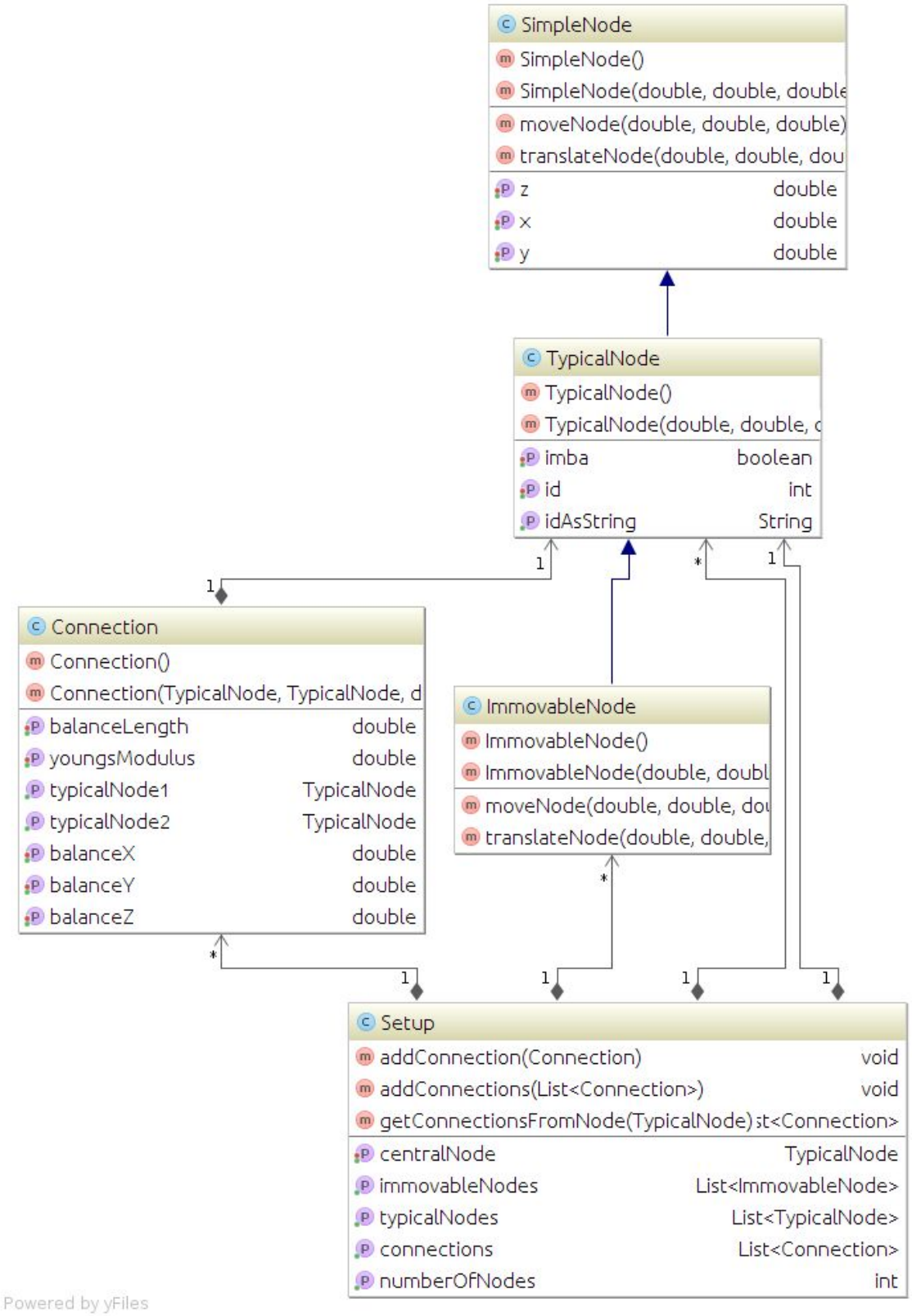
Ponadto w celu zwizualizowania elementów tworzony jest nowy *SubScene*. Pozwala to między innymi na wykorzystanie dodatkowej kamery, która może być innego typu - odpowiednia do wyświetlania elementów trójwymiarowych. Ponadto można nią zupełnie oddzielnie zarządzać - przesuwać, czy obracać.



Rys. 4: Diagram klas okna głównego

4.2. Model danych aplikacji

Model danych przechowuje informacje o stanie bryły w aktualnym momencie. Skonstruowany jest hierarchicznie - klasy o logicznie wyższym poziomie zawierają w sobie klasy niższego poziomu. Podstawową jednostką w modelu jest węzeł. Reprezentowany jest w kilku różnych klasach odpowiadających parę rodzajów węzłów (zwykłe węzły, węzeł centralny, nieruchome węzły na krawędzi siatki) rozszerzających podstawową klasę *SimpleNode*. Klasa ta zawiera informacje o położeniu węzła w trzech wymiarach, oraz metody pozwalające w łatwy sposób zmieniać położenie węzła. Zwykłe węzły, których zasadniczo najwięcej jest w układzie są reprezentowane przez klasę *TypicalNode*. Klasa ta, oprócz informacji z klasy po której dziedziczy, zawiera również numer identyfikacyjny węzła oraz flagę informującą o tym, czy dany węzeł znajduje się w stanie równowagi. Węzły nieruchome znajdują się na krańcach układu i stanowią ograniczenie dla pozostałych węzłów co do możliwości przemieszczenia układu. Węzły te reprezentowane są przez obiekty klasy *ImmovableNode*. Pary węzłów przechowywane są w postaci połączeń w obiektach klasy *Connection*. Połączenia zawierają informację o węzłach, które łączą, jak również inne dane niezbędne do obliczeń. Można wśród nich wyróżnić moduł Younga wpływający na sztywność danego połączenia oraz długość równowagi danego połączenia. Połączenie to będzie dążyć do odzyskania takiej długości, niezależnie od tego, czy zostanie rozciągnięte, czy ściśnięte. Wszystkie połączenia zebrane są w układzie, czyli klasie *Setup*. Klasa ta zawiera listę wszystkich połączeń w bryle. Zawiera również metody, pozwalające łatwo otrzymać obiekty niższego poziomu, jak na przykład połączenia wychodzące z danego węzła, czy listę wszystkich zwykłych węzłów układu. Model ten jest odwzorowaniem jednego ze sposobów przechowywania grafów - za pomocą listy połączeń. W nomenklaturze aplikacji bryła jest zaś właśnie grafem ważonym z punktami o pewnych współrzędnych oraz krawędziami o pewnych wagach.



Rys. 5: Diagram klas modelu danych

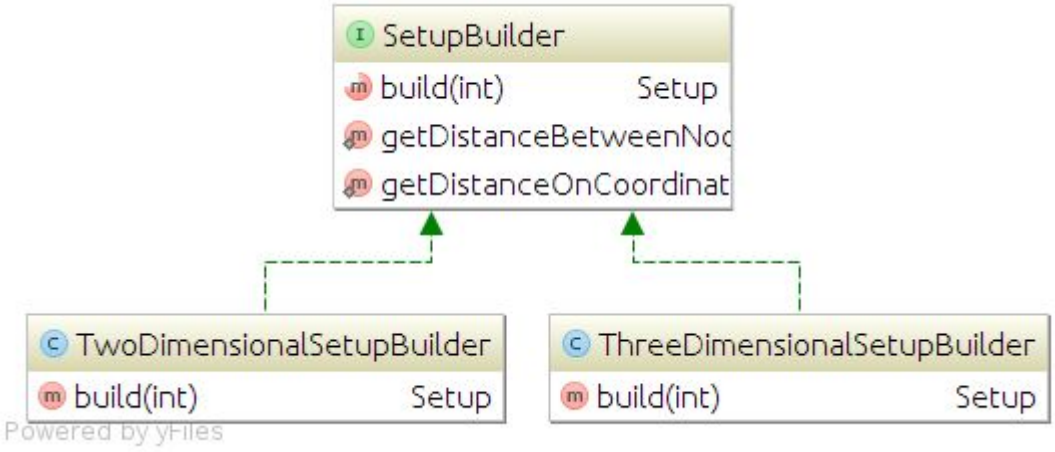
4.3. Moduł tworzenia układu

Po wybraniu przez użytkownika odpowiedniego typu układu jest on generowany. W praktyce sprowadza się to do stworzenia odpowiedniej listy połączeń oraz elementu centralnego z którym połączone są pozostałe. Każde połączenie składa się z węzła początkowego i końcowego. Węzły mogą oczywiście być wspólne dla kilku połączeń.

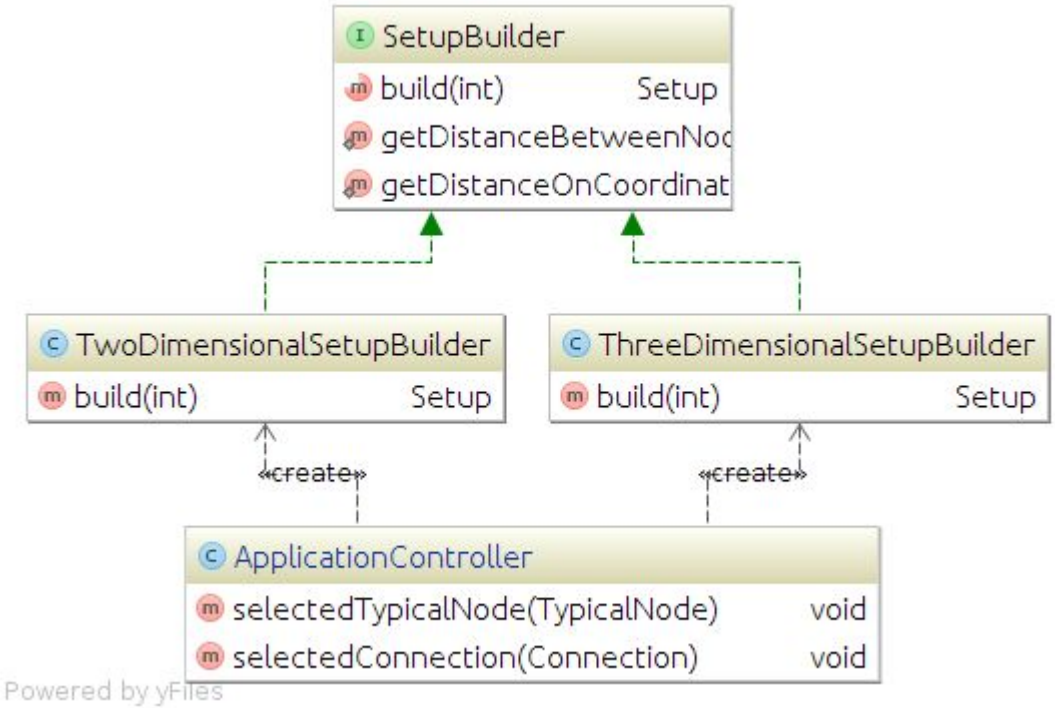
W wypadku układu dwuwymiarowego połączenia tworzą początkowo zwykłą linię. Mimo, że układ jest dwuwymiarowy węzły są tego samego typu jak w wypadku trójwymiarowym - po prostu trzecia współrzędna jest stała.

Układ trójwymiarowy początkowo generowany jest jako półsfera. Jej wielkość zależna jest od wartości węzłów w podstawie, która możliwa jest do zdefiniowania przez użytkownika.

W celu ułatwienia rozszerzalności projektu klasy generujące układy implementują wspólny interface *SetupBuilder*. Dzięki temu dodanie nowego układu początkowego do projektu wymaga jedynie stworzenia nowej klasy rozszerzającej ten interface, dodania odpowiedniego wpisu w *Application.fxml* oraz prostej, analogicznej do pozostałych tego typu, metody w *ApplicationController*.



Rys. 6: Diagram klas modułu tworzenia układu v1



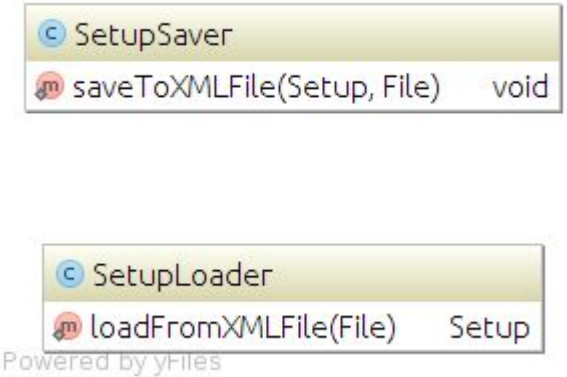
Rys. 7: Diagram klas modułu tworzenia układu v2

4.4. Moduł zapisu i wczytywanie układu z pliku

Zapis układu do pliku sprowadza się do przedstawienia układu, czyli obiektu klasy *Setup*, wraz z listą połączeń i każdym z węzłów w postaci odpowiednich tagów pliku *.xml*. W tym celu wykorzystana jest technologia *JAXB - Java Architecture for XML Binding*. Za pomocą odpowiednich adnotacji - takich jak między innymi: *@XmlRootElement*, czy *@XmlElementWrapper* - pozwala ona w stosunkowo prosty sposób zdefiniować które pola powinny zostać zapisane i w jaki sposób. Bardzo istotnymi adnotacjami są *@XmlID* oraz *@XmlIDREF*. Wskazują one które elementy w docelowym pliku *.xml* mają zostać przedstawione jako referencje obiektów, których dokładna definicja znajduje się w innym miejscu. Przykładowo, gdyby adnotacje te nie zostały zastosowane, każdy obiekt klasy *Connection* posiadałby swoje obiekty klasy *TypicalNode*, mimo że powienien je współdzielić z innymi połączeniami.

Ostatecznie, w celu zapisania układu do pliku należy wykonać tak zwany *marshalling*. Wymaga to wskazania odpowiedniej klasy - w tym wypadku *Setup* - i oczywiście pliku do którego układ ma zostać zapisany. *JAXB* nie wymaga na tym etapie żadnych bardziej skomplikowanych kroków. Działa automatycznie wykorzystując adnotacje, które zawiera klasa *Setup*.

Wczytywanie układu działa w analogiczny sposób. Należy wykonać *unmarshalling*, wymagający wskazania klasy, której obiekt został zapisany do pliku.



Rys. 8: Diagram klas modułu wczytywania i zapisu

4.5. Moduł wizualizacji

Moduł wizualizacji odpowiada przede wszystkim za interpretację listy połączeń tworzących układ. Na ich podstawie tworzone są odpowiednie sfery, reprezentujące węzły, oraz cylindry, reprezentujące połączenia. Wizualizacja węzłów jako sfer nie stanowi wielkiego problemu, jest to jednak bardziej skomplikowane w wypadku połączeń. Wynika to z faktu, że zazwyczaj cylindry nie są definiowane przez podanie im punktu początkowego i końcowego, a raczej punktu środkowego i ewentualnego kąta nachylenia względem danej osi. W praktyce zatem, aby zwizualizować połączenie należy:

1. znaleźć punkt środkowy między węzłami;

2. przemieścić cylinder do znalezionego punktu;

3. znaleźć wektor przemieszczenia między węzłami;

4. poznać kąt nachylenia między znalezionym wektorem, a osią ‘y’;

5. obliczyć oś względem której cylinder powinien zostać obrócony;

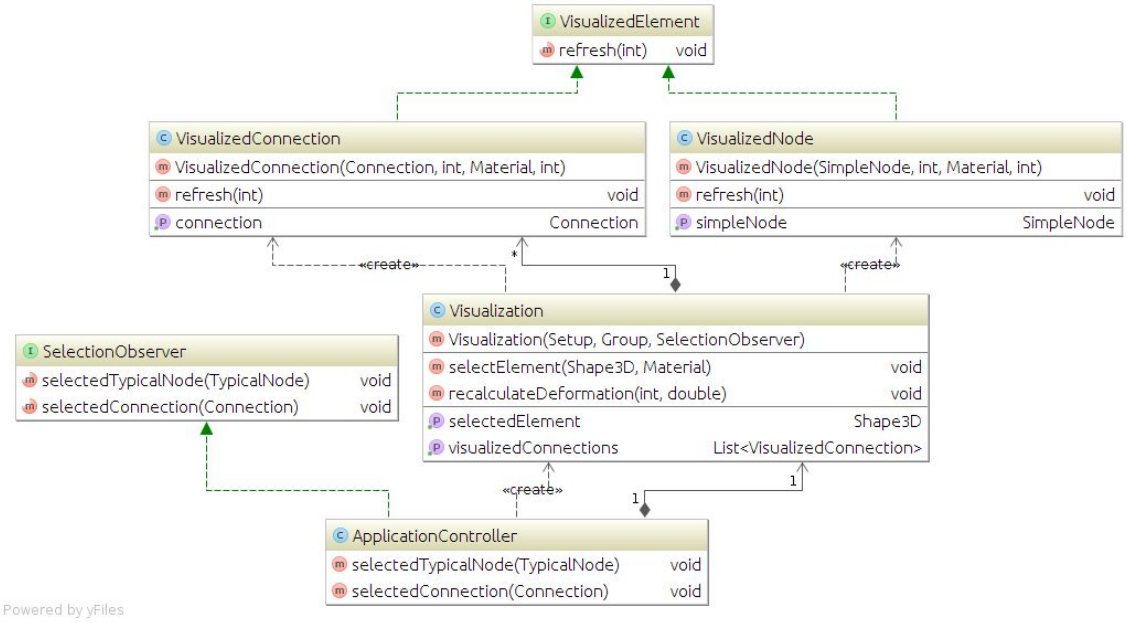
6. obrócić cylinder;

7. obliczyć odległość między węzłami;

8. ustalić wysokość cylindra na wyliczoną wartość.

Ponadto moduł odpowiada za zapewnienie poprawnej obsługi zaznaczania elementów. Wiąże się to z zaktualizowaniem panelu ustawień i listy połączeń. W tym celu w konstruktorze klasy *Visualization* przekazywany jest obiekt klasy implementującej *SelectionObserver*, który informowany jest o zmianie zaznaczenia przez wywołanie odpowiednich metod. W ten sposób informacje tego typu przekazywane są dalej.

Po zmianie odpowiednich parametrów układ powinien zostać uaktualniony. Analogicznie, zmiany powinny zostać przedstawione w wizualizacji. Realizowane jest to poprzez odświeżenie każdego zwizualizowanego elementu. W celu usprawnienia tego procesy powstały klasy *VisualizedNode* oraz *VisualizedConnection*, rozszerzające odpowiednia *Sphere* i *Cylinder*, implementujące przy tym *VisualizedElement*. Obiekty tych klas przechowują informacje o tym jaki konkretnie obiekt układu odpowiadają i posiadają metodę *refresh* pozwalającą w prosty sposób odświeżyć wyświetlane elementy.



Rys. 9: Diagram klas modułu wizualizacji

4.6. Moduł obliczający odkształcenie bryły

Obliczaniem odkształcenia bryły zajmuje się zachłanny algorytm iteracyjny. Działanie algorytmu jest następujące :

1. oblicz wektor wypadkowy sił działających na dowolny węzeł;

2. przesuń węzeł zgodnie z działającą na niego wypadkową siłą;

3. wykonaj kroki 1 i 2 dla wszystkich pozostałych węzłów;

4. sprawdź, czy układ jest w równowadze, to jest, czy działające na każdy węzeł siły równoważą się;

5. jeżeli nie, powtarzaj poprzednie kroki;

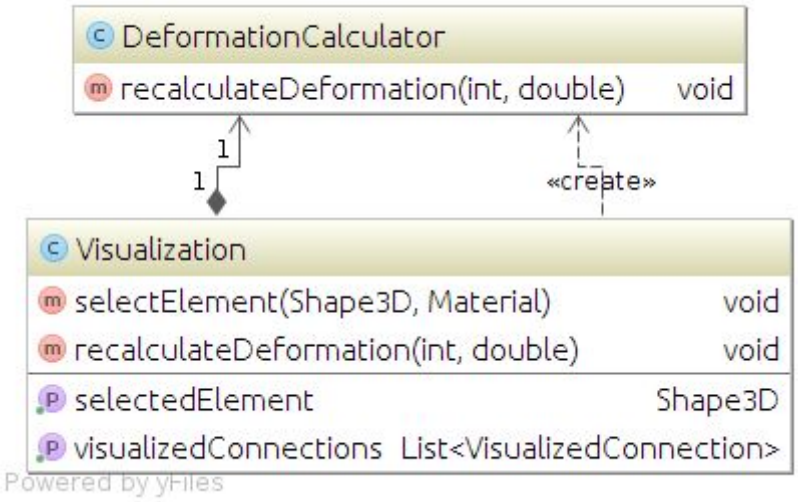
6. jeżeli tak, obliczenia są zakończone.

Obliczanie wektora wypadkowego jest wykonywane w ten sposób, że dla każdego połączenia obliczana jest siła, z jaką to połączenie oddziałuje na węzeł. Siła ta obliczana jest na podstawie prawa Hooke’a uzależniającego

bezwzględne wydłużenie rozciąganego obiektu od siły na niego działającej.

Warunkiem stopu algorytmu jest otrzymanie bryły w stanie równowagi. Może się jednak okazać, że dla danego układu rozwiązanie nie istnieje. W takiej sytuacji, po pewnej liczbie iteracji bez znalezienia rozwiązania uruchamiana jest metoda sprawdzająca zbieżność procesu obliczeń. Zapisuje siły obliczane podczas działania algorytmu i sprawdza, czy suma sił w układzie zmniejsza się w kolejnych iteracjach. Jeżeli przez pewną liczbę iteracji, sytuacja nie ulega poprawie, uznaje się, że dany układ jest niemożliwy do zrównoważenia. Sytuacje takie mogą występować, kiedy użytkownik poda aplikacji zupełnie nierzeczywiste parametry, dające zbyt małe możliwości przesuwania węzłów, lub umieszczając centralny punkt w miejscu, w którym oddziałuje na pozostałe w sposób, który powoduje, że nie jest możliwe znalezienie ustawienia równowagi. Dla ustawień, dla których da się jednak obliczyć wynik, algorytm ten zwraca poprawny wynik.

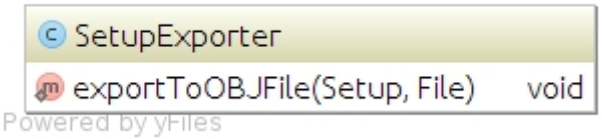
Fakt, że obliczenia wykonywane są iteracyjnie pozwala między innymi na śledzenie kolejnych etapów działania algorytmu oraz obserwowanie zmieniających się naprężeń występujących w siatce.



Rys. 10: Diagram klas modułu obliczającego odkształcenia

4.7. Moduł eksportu układu do pliku *.obj*

W celu wyeksportowania układu do pliku *.obj* wykorzystano bibliotekę *JCSG*. Nie pozwala ona jednak na eksport obiektów klas udostępnionych przez *JavaFX*. Wymagane zatem było wykorzystanie analogicznych obiektów biblioteki JCSG. Po stworzeniu każdego takiego obiektu na jego podstawie tworzony był obiekt klasy *CSG* - wykorzystywany w technice *constructive solid geometry*. Pozwalało ta na połączenie elementów układu w jeden. W ten sposób powstał obiekt reprezentujący cały układ, który następnie mógł zostać zapisany do pliku *.obj*.



Rys. 11: Diagram klas modułu eksportu do *.obj*

Spis rysunków