AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI



DOKUMENTACJA TECHNICZNA

**NARZĘDZIE DO WIZUALIZACJI SIATEK TRÓJWYMIAROWYCH**

WOJCIECH DYMEK

KATARZYNA GŁĄB

KATARZYNA KONIECZNA

EWA MARCZEWSKA

OPIEKUN:

Dr inż. Tomasz Jurczyk

Spis treści

[Dziedzina problemu 4](#_Toc469856543)

[Siatka trójwymiarowa 4](#_Toc469856544)

[Siatka objętościowa 4](#_Toc469856545)

[Stos technologiczny 5](#_Toc469856546)

[OpenGL 5](#_Toc469856547)

[GLU 5](#_Toc469856548)

[GLUT 5](#_Toc469856549)

[Protocol Buffers 5](#_Toc469856550)

[Architektura aplikacji 5](#_Toc469856551)

[Opis modułów 6](#_Toc469856552)

[Moduł komunikacji CORE 6](#_Toc469856553)

[Moduł struktur CORE 7](#_Toc469856554)

[Moduł konfiguracji użytkownika 8](#_Toc469856555)

[Moduł filtracji CORE 8](#_Toc469856556)

[Moduł wizualizacji CORE 10](#_Toc469856557)

[Moduł importu / eksportu 10](#_Toc469856558)

[Smeshalist Manager 10](#_Toc469856559)

[Opis API klienckiego 12](#_Toc469856560)

[Komunikacja wewnątrz sytemu 13](#_Toc469856561)

[Wiadomości przesyłane z API klienckiego 13](#_Toc469856562)

[Struktury 13](#_Toc469856563)

[Pakiety danych 14](#_Toc469856564)

[Nagłówek danych 14](#_Toc469856565)

[Własności 15](#_Toc469856566)

[Wiadomości przesyłane z i do Smeshalist Manager’a 16](#_Toc469856567)

[Komunikacja moduł komunikacji -> Smeshalist Manager 16](#_Toc469856568)

[Komunikacja Smeshalist Manager -> moduł komunikacji 17](#_Toc469856569)

[Spis ilustracji 19](#_Toc469856570)

# Dziedzina problemu

## Siatka trójwymiarowa

Zestaw punktów połączonych krawędziami tworzących trójkątne ściany. Wykorzystując siatki trójwymiarowe można dokonać przybliżenia zarówno prostych brył, jak i bardzo skomplikowanych kształtów.

## Siatka objętościowa

Zbiór brył połączonych krawędziami lub ścianami za pomocą, których można reprezentować wnętrze obiektów.

# Stos technologiczny

## OpenGL

Specyfikacja opisująca standardy tworzenia grafiki trójwymiarowej. Implementowana jest przez różne języki programowania i dostępna na różnych platformach. Pozwala na tworzenie animacji z wykorzystaniem obiektów trójwymiarowych.

## GLU

Biblioteka dostarczająca interfejs wyższego poziomu dla funkcjonalności OpenGL.

## GLUT

Niezależny zbiór narzędzi implementujący interfejs programistyczny OpenGL'a do dostarczania aplikacji okienkowych.

## Protocol Buffers

Narzędzie umożliwiające szybką i prostą serializację ustrukturyzowanych danych, którego zaletą jest niezależność od języka programowania i platformy systemowej.

# Architektura aplikacji

Struktury.png

Rysunek 1 Architektura aplikacji

Narzędzie Smeshalist charakteryzuje się rozproszoną architekturą. Jego głównymi składowymi są:

* API klienckie
* CORE narzędzia, który podzielić można na następujące moduły:
  + moduł komunikacji
  + moduł drzewa struktur
  + moduł wizualizacji
  + moduł konfiguracji użytkownika
  + moduł filtracji
  + moduł importu /eksportu
* Smeshalist Manager

# Opis modułów

## Moduł komunikacji CORE

Moduł zapewniający komunikację pomiędzy wszystkimi modułami systemu.

Do jego głównych zadań należą:

* odbiór wiadomości oraz przekształcanie ich na wewnętrzne obiekty reprezentujące elementy przechowywane w głównej strukturze danych
* przesyłanie statystyk dotyczących ilości przesłanych oraz widocznych elementów do modułu Smeshalist Manager
* pośredniczenie w przekazywaniu informacji pomiędzy modułem filtracji a modułem Smeshalist Manager
* obsługa wiadomości związanych z interakcją pomiędzy systemem a użytkownikiem

Komunikacja - diagram klas.png

Rysunek 2 Moduł komunikacji - diagram klas

Najważniejsza część logiki modułu znajduje się w klasie *AbstractServer*. Odpowiada ona za utrzymywanie połączenia z aplikacją kliencką, jak i oknem *Smashalist Manager’a*. Zawiera również logikę potrzebną do deserializacji wiadomości na obiekty wewnętrzne, zapewnia odpowiednią sekwencję wywołań metod odpowiedzialnych za filtrowanie oraz aktualizację statystyk.

Klasy dziedziczące z *AbstractServer*, tj. *LinuxServer* oraz *WindowsServer*, dostarczają implementacji metod służących do przesyłania danych poprzez gniazda systemowe na odpowiednich platformach. Takie rozwiązanie było konieczne ze względu na znaczne różnice w implementacji oraz API gniazd dla obsługiwanych systemów.  
Zastosowanie klasy abstrakcyjnej pozwoliło na wyniesienie całej logiki obsługi wiadomości do wspólnej klasy.

## Moduł struktur CORE

Moduł odpowiedzialny za przechowywanie poszczególnych elementów składających się na siatkę. Dostarcza hierarchiczną strukturę umożliwiającą optymalną wizualizację, filtrację i odpowiednie dodawanie obiektów według rodzajów i grup. Ponadto generuje statystyki odnośnie przechowywanych i widocznych elementów.

Struktury - diagram klas.png

Rysunek 3 Moduł struktur - diagram klas

Podstawową logikę dostarcza klasa *Data*. Zapewnia ona interfejs pozwalający na dodawanie pojedynczych struktur, bądź ich kolekcji oraz umożliwia pobieranie statystyk.

Ważnym rozszerzeniem klasy *Data* jest klasa abstrakcyjna *AbstractDataTree*. Synchronizuje ona dostęp do instancji drzewa struktur oraz zarządza jej kolekcją. Udostępnia również bardziej rozbudowane API, pozwalające na przefiltrowanie danych we wszystkich drzewach struktur.  
Klasy *LinuxDataTree* oraz *WindowsDataTree* dostarczają konkretne implementacje metod *LOCK* oraz *UNLOCK*.

## Moduł konfiguracji użytkownika

Moduł, którego zadaniem jest odczyt konfiguracji użytkownika dostarczonej w pliku *user.config.xml*. W przypadku braku któregokolwiek parametru konfiguracji ustawiana jest wartość domyślna.

Konfiguracja użytkownika - diagram klas.png

Rysunek 4 Moduł konfiguracji użytkownika - diagram klas

Odczyt konfiguracji następuje tylko raz, w momencie tworzenia nowego obiektu *UserPreferencesManager* (tj. przy pierwszym wywołaniu metody *getInstance*). Obiekt ten zapewnia wygodne API do pobierania określonych parametrów oraz odpowiada za przechowywania, bądź wyliczanie, wartości domyślnej.  
Logika parsera zawarta jest w klasie *XMLParser*.

## Moduł filtracji CORE

Odpowiada za przechowywanie oraz zarządzanie zestawami filtrów jak również za filtrację.

Obsługiwane rodzaje filtrów:

* typ geometrii (Vertex, Edge, Face, Block)
* numer grupy
* wartość własności *quality*
* współrzędne

Filtry - diagram klas.png

Rysunek 5 Moduł filtracji - diagram klas

Główne klasy modułu (*CoordinatesFilter*, *GroupsFilter*, *QualityFilter*, *TypesFilter*) zawierają logikę pozwalającą na optymalne przechodzenie drzewa struktur. Wymaga to odpowiedniej sekwencji wywołań poszczególnych rodzajów filtrów, za które odpowiada moduł komunikacji CORE. Każda z tych klas zawiera kolekcję prostych filtrów zdefiniowanych przez użytkownika. Ze względu na charakter filtru, bądź też wybór użytkownika, filtry te są łączone spójnikiem *OR* lub *AND*.

Klasa *CoordinatesFilter* zawiera dodatkowo logikę pozwalającą na wyrysowanie poszczególnych filtrów na współrzędne zdefiniowanych przez użytkownika. Ze względu na stosunkowo duży stopień skomplikowania, część logiki została wydzielona do klasy *CMathUtils*.

## Moduł wizualizacji CORE

Odpowiada za stworzenie instancji okna, obsługę zdarzeń, głównej pętli programu i sterowanie położeniem kamery.

Ze względu na swój charakter, moduł zawiera również logikę inicjalizującą i uruchamiającą wszystkie pozostałe moduły.

## Moduł importu / eksportu

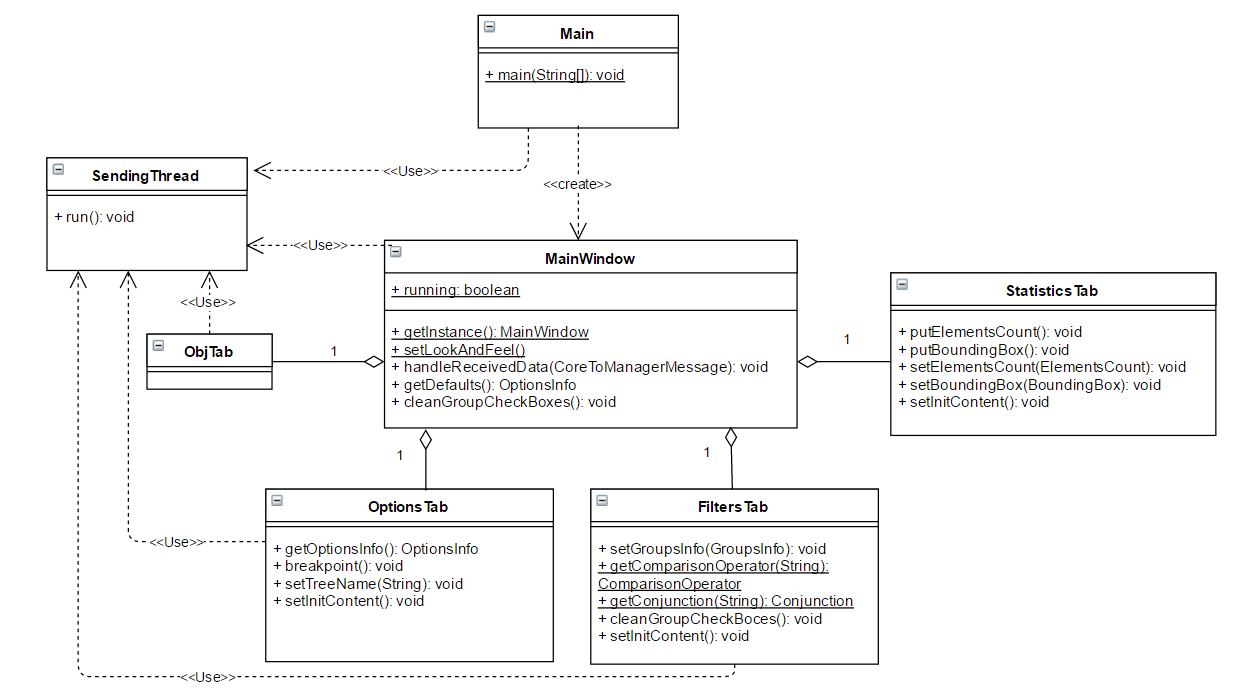
Moduł udostępnia funkcjonalność eksportowania aktywnego drzewa struktur oraz importu struktur z formatu OBJ.

Import Eksport - diagram klas.png

Rysunek 6 Moduł importu / eksportu - diagram klas

## Smeshalist Manager

Moduł odpowiadający za interakcję z użytkownikiem. Realizowany jest w postaci okna zaimplementowanego z wykorzystaniem biblioteki Swing języka Java. Umożliwia on użytkownikowi zarządzanie opcjami wyświetlania struktur w module wizualizacji, jak również ustawianie filtrów i prezentowanie statystyk.



Rysunek 7 Smeshalist Manager diagram klas

Główna logika modułu zawarta jest w klasie *MainWindow* . Klasa odpowiada za stworzenie odpowiednich zakładek (*ObjTab, OptionsTab, FiltersTab, StatisticsTab*) okna oraz za komunikację z modułem komunikacji CORE. Metoda *setLookAndFeel* jest odpowiedzialna za ustawienie wyglądu okna odpowiadającego standardom platformy systemowej, na której aplikacja jest uruchomiona.

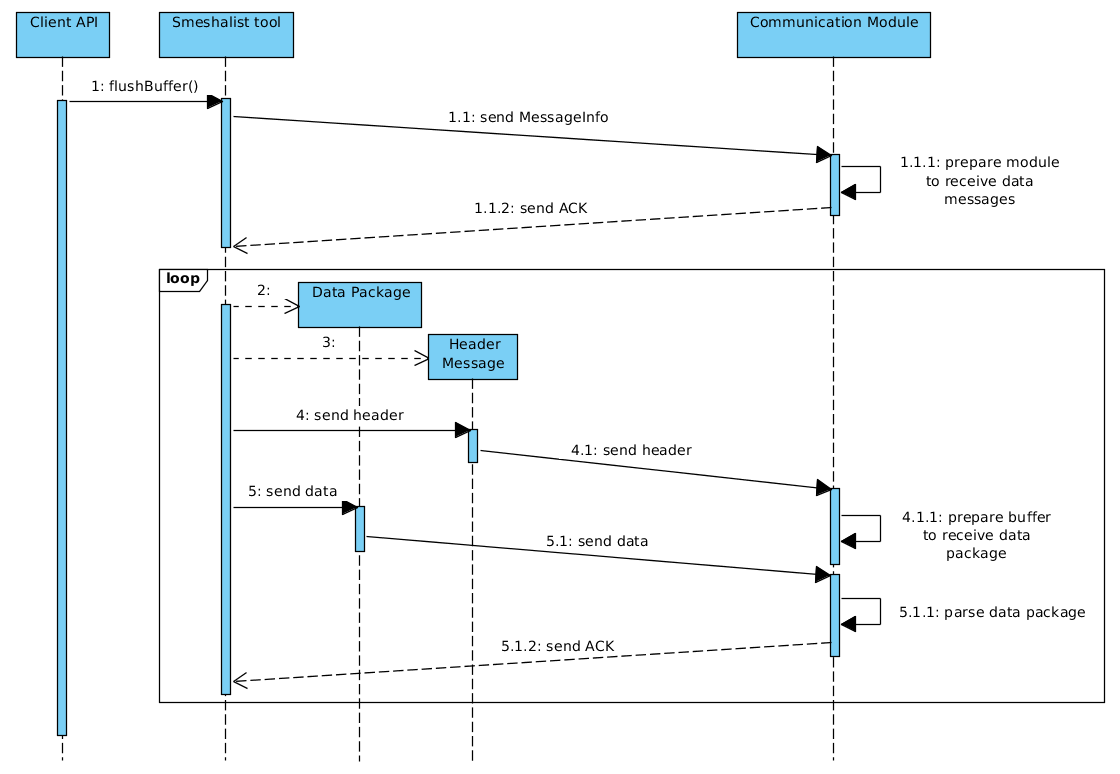
Zadaniem każdej z klas zakładek jest stworzenie widoku okna oraz zbudowanie odpowiedniej wiadomości, a następnie wywołanie metody klasy *SendingThread,* która zajmuje się przesłaniem jej do modułu komunikacji CORE.



Rysunek 8 Smeshalist Manager diagram klas - filtry

Klasa *FiltersTab*  agreguje obiekty zakładek każdego typu filtrów: *QualityTab, TypesTab, CoordinatesTab* oraz *GroupsTab*. Posiada ona metody statyczne odpowiedzialne za działanie filtrowania po jakości oraz współrzędnych. Każda z klas zakładek filtru danego typu posiada metody do zainicjalizowania wyglądu okna oraz budowania wiadomości przesyłanych do modułu komunikacji.

# Opis API klienckiego



Rysunek 9 Diagram sekwencji opisujący algorytm przesyłania paczek z danymi do modułu komunikacji CORE

Po stworzeniu zestawu struktur do wizualizacji użytkownik wywołuje metodę *flushBuffer()*. Narzędzie Smeshalist wysyła wówczas wiadomość o rozpoczęciu wymiany danych do modułu komunikacji, który potwierdza jej otrzymanie. Następnie dane organizowane są w pakiety i wysyłane do modułu komunikacji. Każdy pakiet poprzedzany jest nagłówkiem z polem opisującym wielkość danych oraz informacją czy pakiet jest ostatnim pakietem z serii. Po otrzymaniu tego pakietu moduł komunikacji wysyła potwierdzenie otrzymania zestawu danych.

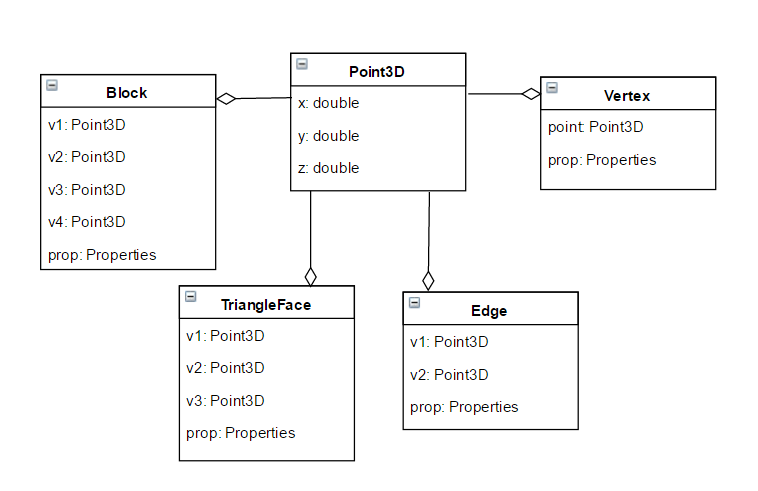
# Komunikacja wewnątrz sytemu

Poniżej zaprezentowane zostaną diagramy opisujące strukturę wiadomości Protocol Buffers przesyłanych w systemie, które służą do sterowania działaniem całego narzędzia.

## Wiadomości przesyłane z API klienckiego

### Struktury

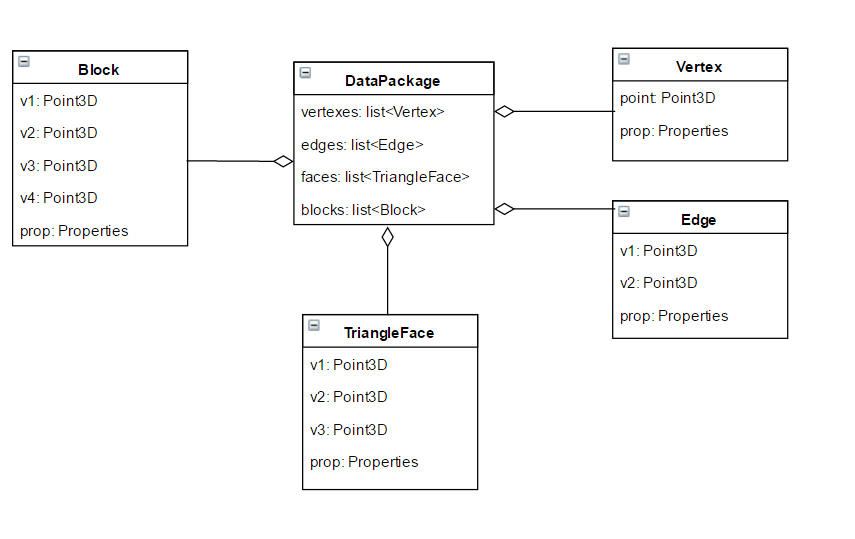
Wiadomość zawierająca informacje o przesyłanych elementach danego typu



Rysunek 10 Diagram przesyłanych w komunikacji struktur

Element każdego typu budowany jest z wykorzystaniem podstawowej klasy *Point3D*  oraz wiadomości typu *Properties* opisanej poniżej.

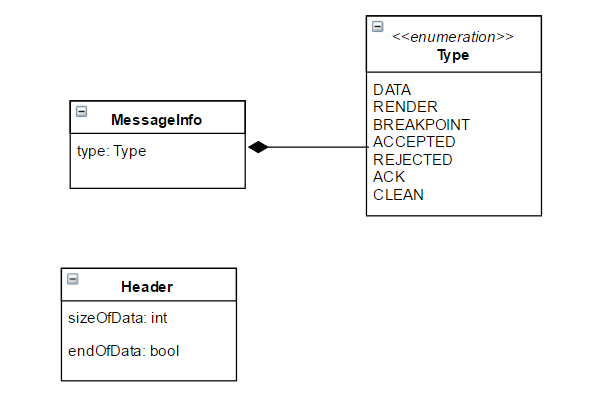
### Pakiety danych



Rysunek 11 Diagram przedstawiający strukturę wysyłanych pakietów danych

Wiadomość *DataPackage* zawiera listy odpowiednich typów elementów tj. *Vertex, Edge, TriangleFace, Block*, które zostały opisane powyżej.

### Nagłówek danych



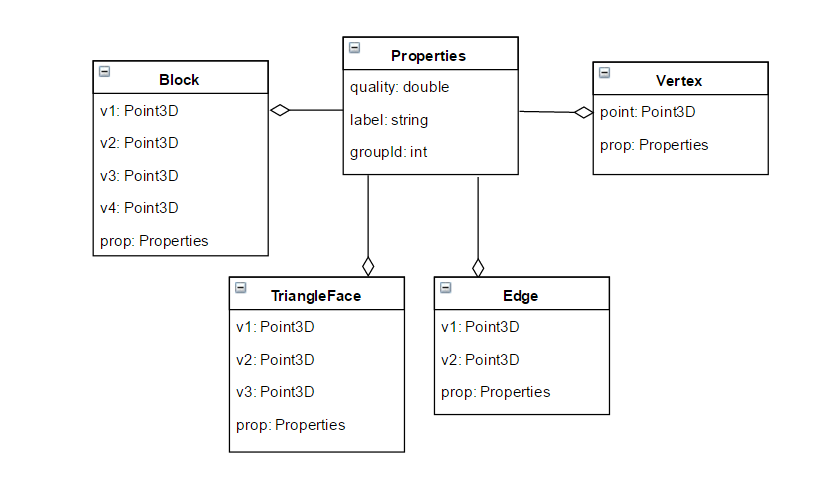
Rysunek 12 Diagram przedstawiający strukturę wiadomości nagłówka oraz pomocniczą wiadomość MessageInfo

Wiadomość typu *Header* zawiera informacje o wielkości przesyłanego pakietu danych oraz flagę czy przesyłana paczka danych jest ostatnia.

Pomocnicza wiadomość *MessageInfo* zawiera informację o typie pomocniczej wiadomości tzn. *Data, Render, Breakpoint, Accepted, Rejected, Ack, Clean*

### Własności

Wiadomość zawierająca informacje o cechach danego elementu



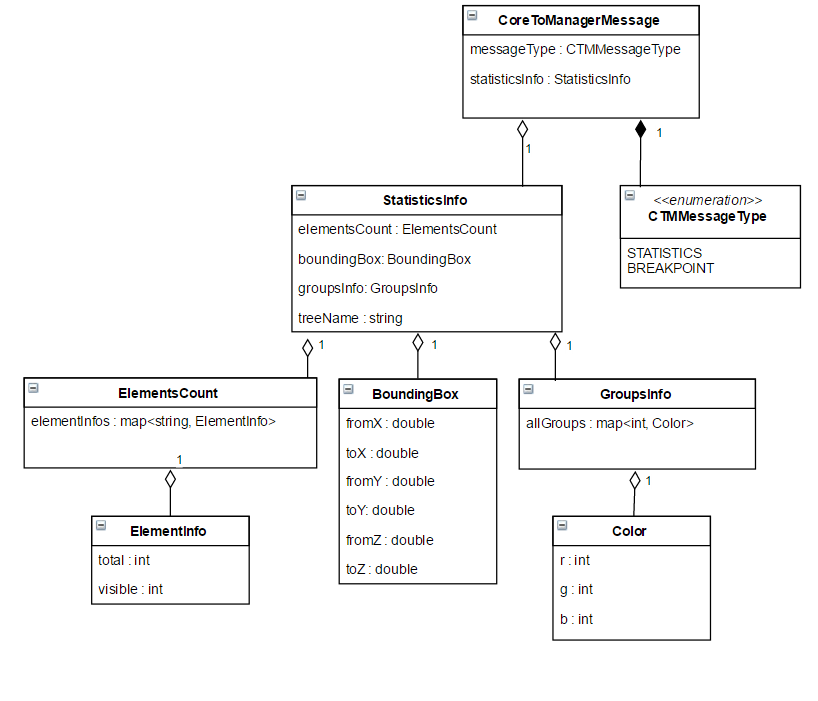
Rysunek 13 Diagram klas własności elementów API

Każda z wiadomości struktur tj. *Block, TriangleFace, Edge, Vertex* może zawierać obiekt *Properties*, która przechowuje informacje o własnościach elementu danego typu, czyli jego jakość, etykietę oraz numer grupy, do której należy.

## Wiadomości przesyłane z i do Smeshalist Manager’a

Poniższe diagramy przedstawiają strukturę wiadomości służących do komunikacji pomiędzy Smeshalist Manager’em oraz modułem komunikacji CORE

### Komunikacja moduł komunikacji -> Smeshalist Manager

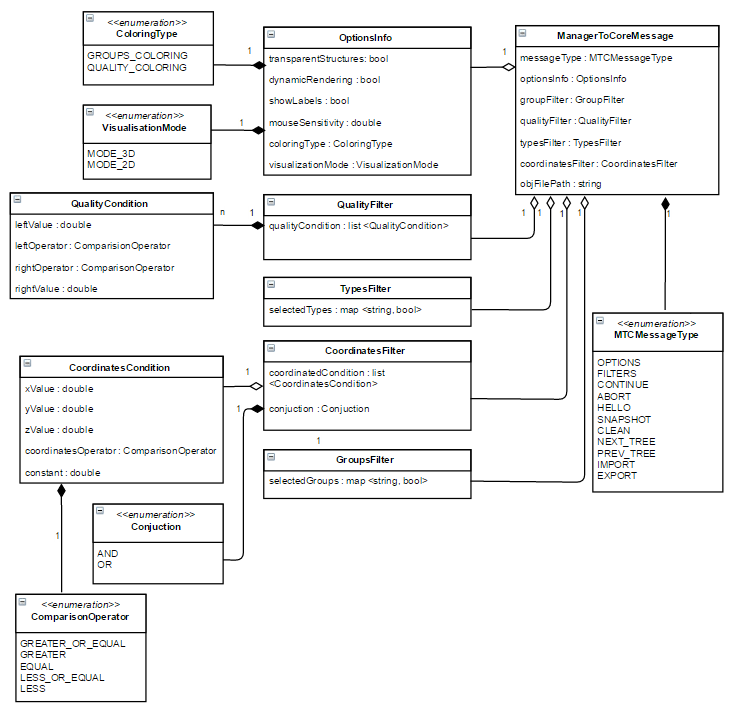


Rysunek 14 Diagram struktury wiadomości moduł komunikacji CORE do Smeshalist Manager’a

Wiadomość *CoreToManagerMessage* zawiera informacje o typie przesyłanej wiadomości (*Statistics, Breakpoint)* oraz przesyłane statystyki. *StatisticsInfo* posiada informacje o ilości wszystkich oraz wyświetlanych elementów danego typu (*ElementsCount)*, współrzędne prostopadłościanu ograniczającego (*BoundingBox)* , listę grup wraz z przypisanych do nich kolorami (*GroupsInfo)* oraz nazwę wyświetlanego drzewa struktur.

### Komunikacja Smeshalist Manager -> moduł komunikacji

Poniższe diagramy przedstawiają strukturę wiadomości przesyłanych podczas komunikacji Smeshalist Manager’a z modułem komunikacji CORE



Rysunek 15 Diagram struktury wiadomości przesyłanych od Smeshalist Manager’a do modułu komunikacji CORE

Wiadomość typu *ManagerToCoreMessage* zawiera informacje o:

* typie przesyłanej wiadomości (*Options, Filters, Continue, Abort, Hello, Snapshot, Clean, Next\_Tree, Prev\_Tree, Import, Export)*
* ustawionych w okienku opcjach:
  + *transparentStructures*
  + *dynamicRendering*
  + *showLabels*
  + *mouseSensitivity*
  + *coloringType –* kolorowanie po numerze grupy lub jakości
  + *visualizationMode –* wizualizacja typu 2D lub 3D
* filtrach
* ścieżce pliku do importu/eksportu

# Spis ilustracji

[Rysunek 1 Architektura aplikacji 6](#_Toc470208224)

[Rysunek 2 Moduł komunikacji - diagram klas 7](#_Toc470208225)

[Rysunek 3 Moduł struktur - diagram klas 8](#_Toc470208226)

[Rysunek 4 Moduł konfiguracji użytkownika - diagram klas 9](#_Toc470208227)

[Rysunek 5 Moduł filtracji - diagram klas 10](#_Toc470208228)

[Rysunek 6 Moduł importu / eksportu - diagram klas 11](#_Toc470208229)

[Rysunek 7 Smeshalist Manager diagram klas 11](#_Toc470208230)

[Rysunek 8 Smeshalist Manager diagram klas - filtry 12](#_Toc470208231)

[Rysunek 9 Diagram sekwencji opisujący algorytm przesyłania paczek z danymi do modułu komunikacji CORE 13](#_Toc470208232)

[Rysunek 10 Diagram przesyłanych w komunikacji struktur 14](#_Toc470208233)

[Rysunek 11 Diagram przedstawiający strukturę wysyłanych pakietów danych 15](#_Toc470208234)

[Rysunek 12 Diagram przedstawiający strukturę wiadomości nagłówka oraz pomocniczą wiadomość MessageInfo 15](#_Toc470208235)

[Rysunek 13 Diagram klas własności elementów API 16](#_Toc470208236)

[Rysunek 14 Diagram struktury wiadomości moduł komunikacji CORE do Smeshalist Manager’a 17](#_Toc470208237)

[Rysunek 15 Diagram struktury wiadomości przesyłanych od Smeshalist Manager’a do modułu komunikacji CORE 18](#_Toc470208238)