JAVA & VRML

Zastosowania techniki programistycznej Java w rzeczywistości wirtualnej światów VRML

Michał Tomaszewski

Spis treści:

Spis ilustracji:

1. Wprowadzenie

Celem niniejszej pracy dyplomowej było stworzenie modelu wirtualnego świata oraz klientów, którzy poruszaliby się wewnątrz niego. Dokonano tego wykorzystując język skryptów VRML (ang. Virtual Reality Modeling Language) i język programowania Java. Na przykładach prezentowane są możliwości drzemiące w połączeniu wyżej wymienionych narzędzi. Język wysokiego poziomu, jakim jest Java, obsługuje komunikację pomiędzy komputerem klienta a serwerem wykorzystując funkcje, które oferuje w swojej składni. Natomiast język VRML odpowiada za oprawę graficzną całości projektu.

Rozdział „Java wczoraj i dziś” porusza tematykę powstawania języka Java i problemy z tym związane. Ponadto w rozdziale tym opisane są elementy bibliotek java.awt i java.net .

Rozdział „Historia VRML” przybliża niektóre problemy napotykane przez twórców języka VRML oraz jego zastosowania - w przeszłości i dziś.

W rozdziale „Aplikacje VRML” przedstawione są mechanizmy funkcjonowania przeglądarek VRML, sposoby nawigacji oraz programy, którymi posłużyłem się podczas tworzenia wirtualnego świata, w którym będą poruszali się klienci.

Pracę kończy rozdział „VRML i Java”, w którym prezentowany jest proces powstawania aplikacji końcowej, wraz z problemami, które napotkano w trakcie jej tworzenia.

2. Historia rozwoju języka Java

2.1. Cele i rezultaty projektu „The Green”

W roku 1990 Patrick Naughton, Mike Sheridan i James Gosling rozpoczęli prace nad finansowanym przez firmę SUN ściśle tajnym projektem o kryptonimie „The Green”. Celem było zbadanie rynku komputerowego i określenie trendów, na które w przyszłości firma powinna zwrócić szczególną uwagę. Dość szybko specjaliści z „Green Team” przeprowadzili niezbędne badania. Z uzyskanych danych wynikało, iż w niedalekiej przyszłości istotną częścią rynku będzie segment cyfrowo sterowanych urządzeń powszechnego użytku. Na nim więc skoncentrowały się wysiłki inżynierów. [6]

Zgodnie z deklaracjami twórców tego programu ważne były dwie rzeczy: po pierwsze model biznesowy i produkt końcowy; po drugie technologia wykonania. Między innymi z tego powodu liczba osób biorących udział w projekcie „The Green” stale rosła. I tak w kwietniu roku 1991 do zespołu dołączyło trzy nowe osoby: Chris Warth, Ed Frank oraz Craig Forest.

Efektem prac było urządzenie nazwane Star7, które było wyposażone w :

- pięciocalowy, kolorowy, ciekłokrystaliczny, dotykowy wyświetlacz
- bezprzewodowe złącze sieciowe pracujące na częstotliwości 900 MHz
- interfejs PCMCIA (ang *Personal Computer Memory Card International Association*)
- multimedialny kodek audio

Rys. 1 Urządzenia Star7

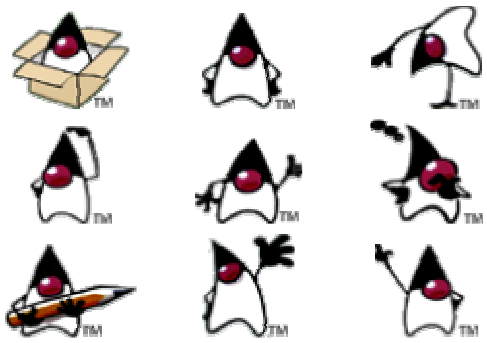
Latem 1992 prace nad tym urządzeniem zostały zakończone i 3 sierpnia ich efekt zaprezentowano kierownictwu firmy. Osoby biorące udział w prezentacji były zachwycone zarówno wyglądem, jak i funkcjonalnością produktu. Jednak pomimo bardzo pochlebnych opinii do dnia dzisiejszego nie znalazła się firma chętna do produkcji urządzeń Star7.

2.2. Oak, Java i Duke

Aby zapewnić elastyczność i umożliwić komunikację pomiędzy dowolnym komputerem a Star7 zaistniała potrzeba stworzenia wieloplatformowego systemu.

Zadanie napisania nowego systemu operacyjnego, mogącego wspierać wszystkie funkcje Star7, zlecono James'owi Gosling'owi. Przystąpił on do pracy korzystając z języka C++. Dość szybko jednak zdał sobie sprawę, iż język ten, nawet z dużą ilością dodatkowych klas, nie pozwoli na osiągnięcie zaplanowanego celu. Dlatego w czerwcu 1991 roku zdecydował się rozpocząć prace nad własnym językiem programowania. Roboczo został on nazwany Oak (ang. *dąb*) - drzewo tego gatunku rosło przed domem Goslinga. Bardzo szybko, bo już 1 sierpnia został uruchomiony pierwszy program napisany w Oak, a 19 dni później zademonstrowano działający interfejs użytkownika Star7.

Ponieważ Star7 ciągle nie miał swojego logo, dlatego Joe Palrang wymyślił i narysował dla niego maskotkę nazwaną „Duke”. Zespołowi maskotka spodobała się i została wykorzystana jako przewodnik po nowym systemie oraz pomocnik użytkownika.



Rys. 2 Maskotka „Duke”

Ponieważ, jak już wspomniano, nie znaleziono firmy, która byłaby zainteresowana produkowaniem jednostek Star7, kierownictwo SUN postanowiło odzyskać chociażby część funduszy zainwestowanych w projekt „The Green”. Postanowiono więc, iż elementy technologii wykorzystanych przy budowie Star7 zostaną użyte w Internecie. Decyzja ta zapadła w czasie, kiedy ogólnoświatowa sieć stawała się coraz bardziej popularna i coraz częściej na stronach HTML (ang. Hyper Text Markup Language) wyświetlano filmy oraz grafikę. Ponieważ większość z tych technik stanowiło integralną część *Oak*, na tym właśnie produkcie skupiły się wysiłki pracowników SUN. Ze względów marketingowych nazwa nowego języka uległa zmianie na Java.

2.3. Dziś i jutro języka Java

Powodem powszechnej akceptacji Javy jest zapewne jej podobieństwo do C++. Z C++ zapożyczono większość składni. Zachowano hermetyzację, dziedziczenie i polimorfizm typ orzecznikowy i wyjątki. Z wielu rzeczy jednak zrezygnowano. Nie ma w Javie wskaźników ani złożonych konwersji. Zrezygnowano z dyrektyw, struktur, unii, wyliczeń, wielodziedziczenia oraz z możliwości definiowania operatorów. Inna jest koncepcja zarządzania tablicami, zaś obiekty są identyfikowane wyłącznie przez odnośniki

Zapewniono przenośność programów między dowolnymi platformami sprzętowymi i systemowymi, umożliwiono zdalne wywoływanie metod, wbudowano w język mechanizmy współbieżności i odzyskiwania pamięci, oraz uniemożliwiono wyrządzanie szkód przez aplety pochodzące z niepewnych źródeł

Dzisiaj Java zyskuje ogromną popularność – głównie dzięki swojej niezależności od platformy sprzętowej i systemu operacyjnego. Pliki źródłowe są kompilowane do postaci beta-kodu, identycznego dla każdej platformy, który można uruchomić na dowolnej maszynie wirtualnej.

Obecnie z wykorzystaniem tego języka pisane są gry, programy użytkowe dla komputerów i oprogramowanie dla urządzeń powszechnego użytku. W niektórych krajach już przyjęto Javę jako podstawowy język w nauczaniu informatyki, zastępując nią stosowany do tej pory C++. Są to między innymi Stany Zjednoczone i Kanada, przyjęcie tego rozwiązania rozważa też wiele uniwersytetów w krajach europejskich.

Pojawia się coraz więcej kompilatorów i coraz to nowe produkty, dzięki którym Java jest językiem żywym i zyskującym sobie rzesze użytkowników.

Twórcy Javy ciągle rozszerzają możliwości języka. Wciąż powstają nowe klasy oraz rozwijane są te już istniejące. Dowodem na to może być najnowsza wersja Java 2 Platform, która została oficjalnie zaprezentowana w lutym roku 1999. Innym przykładem może być klasa Java3D, która jest realizacją języka VRML w kodzie Java.

Rozwój technik Javy i postępująca miniaturyzacja pozwoliły na stworzenie urządzeń na tyle małych, aby zmieściły się w przedmiotach codziennego użytku. Na uwagę zasługują np. pierścionki z wbudowaną maszyną wirtualną, dzięki której można będzie przeprowadzić transakcję w banku, podjąć gotówkę z bankomatu lub zapisać w jej pamięci listę zakupów. Chipy ze sprzętowo realizowaną maszyną wirtualną są wykorzystywane w robotach i urządzeniach AGD.

2.4. Wybrane klasy java.net.* i java.io.*

Klasę można zdefiniować jako opis pewnej rodziny obiektów. Struktura tych obiektów jest określana przez pola klasy, zaś możliwe do wykonania na nich operacje są określone przez jej konstruktory i metody. Składowymi klasy są jej konstruktory i funkcje.

2.4.1 java.net.URL

Pod pojęciem zasobów sieciowych można rozumieć zarówno zdalne pliki czy katalogi, jak też i odnośniki do bardziej skomplikowanych struktur, takich jak zapytania w bazach danych czy wyszukiwarkach (tzw. *search engines*). Położenie konkretnego zasobu określa jego lokalizator. W ogólnym przypadku lokalizator ma postać

```
protocol:/host:port/~user/file#ref
```

w której

- ***protocol*** jest nazwą użytego protokołu komunikacyjnego, rozumianego jako zestaw instrukcji sterujących umożliwiających nawiązanie połączenia i transmisję danych

  np. `file, http, ftp, telnet`

- ***host*** jest nazwą komputera-gospodarza

  np. `www.ncsa.uiuc.edu`
- ***port*** jest numerem portu (dla http domyślnie 80)
- ***user*** jest nazwą użytkownika

  np. `tomasz`
- ***file*** jest nazwą pliku albo katalogu

  `demoweb/url-primer.html`
- ***ref*** jest odnośnikiem do miejsca w pliku HTML

  np. `Chapter3`

Nazwę katalogu odróżnia w lokalizatorze od nazwy pliku to, że kończy ją skośnik (/).

Do zarządzania lokalizatorami służy klasa java.net.URL. Jej konstruktory akceptują zarówno kompletny lokalizator, jak i umożliwiają utworzenie lokalizatora na podstawie protokołu, gospodarza, portu i pliku.

```
new URL("http://www.ncsa.uiuc.edu/Index.html")
new URL("http", " www.ncsa.uiuc.edu ", "Index.html")
new URL("http", " www.ncsa.uiuc.edu ", 80, "Index.html")
```

Lokalizatory są dostarczane przez metody: *getDocumentBase* i *getCodeBase*. Pierwsza z nich dostarcza lokalizator pliku HTML z opisem apletu, a druga lokalizator katalogu użytego w parametrze *codebase*.

Konstruktory klasy java.net.URL :

Konstruktor jest metodą klasy, której zadaniem jest inicjowanie obiektów. Konstruktory nie zwracają żadnych wartości. Za każdym razem, gdy aplet tworzy zmienną typu pewnej klasy, Java wywołuje jej konstruktor, chyba że funkcja taka nie istnieje. Konstruktory maja taką samą nazwę jak klasa, w której występują [5].

- `URL(String)`

  tworzy obiekt typu URL w oparciu o zmienną typu *String*
- `URL(String, String,int, String)`

  tworzy obiekt typu URL wykorzystujący określony przez argumenty wejściowe protokół, gospodarza i numer portu oraz pliku;
- `URL(String, String, String)`

  tworzy pełny obiekt typu URL na podstawie podanych nazw: protokołu, gospodarza oraz pliku
- `URL(URL, String)`

  tworzy obiekt typu URL analizując specyfikację ze scharakteryzowanym kontekstem

Metody klasy java.net.URL :

Metody są to jedne ze składowych klasy. Określają one operacje, które można wykonywać na obiektach tej klasy.

- `equals(Object)` porównuje dwa obiekty typu URL
- `getContent()` zwraca zawartość lokalizatora

- `getFile()` zwraca nazwę pliku z lokalizatora
- `getHost()` zwraca nazwę gospodarza z lokalizatora
- `getPort()` zwraca numer portu z lokalizatora
- `getProtocol()` zwraca nazwę protokołu z lokalizatora
- `getRef()` zwraca odnośnik do obiektu typu URL
- `hashCode()` tworzy odpowiednią reprezentację w oparciu o tabelę indeksowania
- `openConnection()` zwraca obiekt typu *URLConnection*, który reprezentuje połączenie do obiektu zdalnego powiązanego z danym lokalizatorem
- `openStream()` otwiera połączenie z lokalizatorem i zwraca strumień danych „*InputStream*” dla czytania z tego połączenia
- `sameFile(URL)` porównuje dwa lokalizatory, pomijając pole „*ref*”
- `set(String, String, int, String, String)` ustawia wartości pól w lokalizatorze
- `setURLStreamHandlerFactory(URLStreamHandlerFactory)` ustawia aplikacje *URLStreamHandlerFactory*
- `toExternalForm()` tworzy ciąg znaków, który odpowiada lokalizatorowi
- `toString()` tworzy ciąg znaków, który odpowiada lokalizatorowi

2.4.2 java.net.Socket

Najczęściej spotykanym w Internecie sposobem komunikacji jest układ klient-serwer. Klientem jest komputer lub proces wysyłający zapytania lub żądania obsługi. Komputer (lub proces) obsługujący klienta jest nazywany serwerem lub hostem. Ponieważ wszystkie komputery połączone w sieć Internet są równorzędne to, który z nich będzie spełniał rolę serwera, a który klienta zależy od konkretnego przypadku.

W prezentowanym projekcie funkcje serwera spełnia komputer, na którym zostanie uruchomiony program serwera, realizujący m.in. zadanie rozsyłania do wszystkich użytkowników informacje o położeniu postaci reprezentujących pozostałe osoby korzystające z systemu. Może to być jednak dowolny komputer, w szczególności również komputer jednego z użytkowników – wówczas pełni on rolę jednocześnie serwera i klienta.

Punktem komunikacyjnym komputera jest tzw. gniazdo (ang. *socket*). W programie rozproszonym, jakim jest ten projekt, serwer tworzy gniazdo serwera i związuje je z ustalonym portem. Poprzez ten port oczekuje się na zgłoszenia klientów. Klienci dowiadują się numeru portu i wysyłają do niego zgłoszenia z żądaniem połączenia z serwerem. Jeśli w danej chwili trwa obsługa takiego zdarzenia, wówczas przychodzące zgłoszenie jest ustawiane w kolejce. Serwer rozładowuje kolejkę tworząc oddzielne gniazda dedykowane poszczególnym klientom. Dalsza komunikacja odbywa się poprzez te gniazda, zaś gniazdo pierwotne pozostaje otwarte i służy do przyjmowania nowych zgłoszeń.

Do implementowania gniazd w języku Java służy klasa java.net.Socket. Klasa ta tworzy obiekt typu gniazdo. Jest on wykorzystywany przez konstruktory klas java.io.DataInputStream oraz java.io.DataOutputStream. Obiekty utworzone przez te

konstruktory używają metod odpowiednio *readInt()* oraz *writeInt(int)* celem odbierania i wysyłania danych.

Konstruktory klasy java.net.Socket :

- `Socket (InetAddress address, int port, InetAddress localAddr, int localPort)`

  tworzy gniazdo o numerze podanym jako wartość argumentu „*localPort*” i łączy go z portem (*int port*) w komputerze o adresie określonym w polu „*address*”
- `Socket(String host, int port, boolean stream)`

  tworzy strumień danych dla gniazda i łączy go z portem w komputerze, którego nazwa zawarta jest w zmiennej „*host*”
- `Socket(InetAddress host, int port, boolean stream)`

  tworzy gnizado w komputerze klienta i łączy go z wyspecyfikowanym portem w komputerze o określonym adresie IP
- `Socket(String host, int port)`

  tworzy strumień portu i łączy go z portem o określonym numerze na komputerze którego nazwa jest zawarta w zmiennej „host”
- `Socket(InetAddress address, int port)`

  tworzy strumień danych gniazda i łączy go z portem o określonym numerze w komputerze opisanym podanym adresem IP

Metody klasy java.net.Socket:

- `close()` zamyka gniazdo
- `getInetAdress()` zwraca adres, do którego jest podłączone gniazdo
- `getInputStream()` zwraca strumień wejściowy dla portu
- `getLoaclAddress()` pobiera adres urządzenia na którym jest utworzony port
- `getLocalPort()` zwraca numer portu
- `getOutputStream()` zwraca strumień wyjściowy dla portu
- `getPort()` zwraca numer zdalnego portu przez który jest połączony
- `getSoLinger()` zwraca wartość parametru *SO_LINGER* (określa on czas oczekiwania portu na dane)
- `getSoTimeout()` zwraca wartość parametru *SO_TIMEOUT* (określa on *timeout* w milisekundach)
- `getTcpNoDelay()` sprawdza, czy parametr *TCP_NODELAY* jest włączony
- `setSoLinger(boolean, int)` włącza lub wyłącza parametr *SO_LINGER*
- `setSoTimeout(int)` włącza lub wyłącza parametr *SO_TIMEOUT*
- `setTcpNoDelay(boolean)` włącza lub wyłącza parametr TCP_NODELAY
- `toString()` przekształca zmienną typu „*Socket*” w zmienną typu „*String*”

2.4.3 java.net.ServerSocket

Klasa ta, podobnie jak wyżej opisana java.net.Socket) służy do implementacji gniazd. Efektem jej działania jest utworzenie wspomnianego wcześniej gniazda serwera, na którym oczekuje on zgłoszeń przychodzących z sieci.

Konstruktory klasy java.net.ServerSocket:

- `ServerSocket(int)`

  tworzy port serwera nadając mu numer określony przez zmienną typu *int*
- `ServerSocket(int, int)`

  tworzy port serwera nadając mu numer (określony przez zmienną typu *int*) a następnie wiąże go z portem lokalnym o numerze określonym drugą zmienną typu *int*

Metody klasy java.net.ServerSocket:

- `accept()` oczekuje na stworzenie połączenia i aprobuje je
- `close()` zamyka dany port
- `getInetAddress()` zwraca lokalny adres portu
- `getLocalPort()` zwraca numer portu, na którym serwer oczekuje na zgłoszenia klientów
- `getSoTimeout()` zwraca wartość parametru *SO_TIMEOUT*
- `setSoTimeour(int)` włączenie lub wyłączenie parametru *SO_TIMEOUT*, z określeniem limitu czasowego podawanego w milisekundach
- `toString()` zwraca implementację adresu i implementacji portu jako *String*

2.4.4 java.io.DataInputStream

Klasy java.net.* odpowiadają za implementację i obsługę portów. Jednakże sam port jest jedynie strukturą umożliwiającą komunikację. Potrzebne są jeszcze narzędzia pozwalające obsługiwać i nadzorować samą transmisję. W języku Java służą do tego klasy java.io.* obsługujące strumienie wejściowe i wyjściowe.

Według najbardziej podstawowej interpretacji strumień jest to ciąg bajtów. Strumieniem wejściowym jest nazywana sekwencja bajtów **pochodząca z** pliku, pamięci operacyjnej lub urządzenia. Strumieniem wyjściowym jest sekwencja danych **wysyłana do** pliku, pamięci operacyjnej lub urządzenia. Za pomocą odpowiednich klas ciągi bajtów mogą być przekształcane w ciągi znaków.

Strumień wejściowy “*data input stream*” pozwala aplikacji czytać (pobierać) podstawowe typy danych. Do zapisu (wysyłania) danych aplikacja wykorzystuje strumień „*data output stream*”. Dane mogą być później ponownie odczytane przez strumień „*data input stream*”.

W Javie powyższe strumienie są reprezentowane przez ciągi znakowe Unicode, w nieznacznie zmodyfikowanym formacie UTF-8.

Wszystkie znaki z zakresu od ‘\u0001’ do ‘\u007F’ są reprezentowane przez pojedynczy bajt:

0 bity 0-7

Znak zera ‘\u0000’ oraz znaki z zakresu od ‘\u0080’ do ‘\u07FF’ są reprezentowane przez pary bajtów:

1 1 0 bity 6-10

1 0 bity 0-5

Znaki z zakresu od ‘\u0800’ do ‘\uFFFF’ są reprezentowane przez trójki bajtów:

1 1 1 0 bity 12-15

1 0 bity 6-11

1 0 bity 0-5

Konstruktory klasy java.io.DataInputStream:

- `DataInputStream(InputStream)`
  tworzy strumień „data input” ze strumienia wejściowego.

Metody klasy java.io.DataInputStream:

- `read(byte[])` wczytuje dane do tablicy typu “*byte*”
- `read(byte[], int, int)` wczytuje określoną liczbę danych do tablicy typu „*byte*”
- `readBoolean()` wczytuje wartość typu „*boolean*”
- `readByte()` wczytuje 8-bitową wartość
- `readChar()` wczytuje literę *Unicode*
- `readDouble()` wczytuje wartość typu *double*
- `readFloat()` wczytuje wartość typu *float*
- `readFully(byte[])` wczytuje bajty do tablicy „*byte[]*”
- `readFully(byte[], int, int)` wczytuje dokładną ilość bajtów do tablicy „*byte[]*”
- `readInt()` wczytuje 32-bitową wartość typu *int*
- `readLine()` wczytuje całą linię tekstu
- `readLong()` wczytuje 64-bitową wartość typu *int*
- `readShort()` wczytuje 16-bitową wartość typu *int*

2.4.5 java.io.DataOutputStream

Zasada działania tej klasy została podana w ogólnym opisie klas java.io.*.

Konstruktor klasy java.io.DataOutputStream:

- DataOutputStream(OutputStream) – tworzy nowy strumień wyjściowy

Metody klasy java.io.DataOutputStream:

- `size()` zwraca ilość bajtów zapisanych do strumienia wyjściowego
- `write(byte[], int, int)` zapisuje określoną liczbę danych z tablic typu „*byte*”
- `writeBoolean(boolean)` zapisuje wartość typu „*boolean*” do strumienia
- `writeByte(int)` zapisuje 1-bajtową wartość
- `writeChar(int)` zapisuje literę jako 2-bajtową wartość
- `writeChars(String)` zapisuje wartość typu *String* jako sekwencje wartości typu char
- `writeDouble(double)` konwertuje wartość *double* na wartość *long* wykorzystując metodę klasy *Double* „*doubleToLongBits*”, następnie zapisuje wartości *long* jako 8-bajtową liczbę
- `writeFloat(float)` konwertuje wartość *float* na wartość *int* wykorzystując metodę klasy *Float* „*floatToIntBits*” i zapisuje wartość *int* jako 4-bajtową liczbę
- `writeInt(int)` zapisuje wartość *int* do strumienia jako 4 bity
- `writeLong(long)` zapisuje wartość *long* jako 8 bitów
- `writeShort(int)` zapisuje wartość *short* jako 2 bity

3. VRML

3.1. Historia powstania VRML

Idea stworzenia języka, który pozwoliłby na komputerowe modelowanie rzeczywistego świata powstała wiosną 1994 roku podczas konferencji „Word Wide Web Conference” w Genevie. Istniały wprawdzie programy typu CAD (ang. *Computer Aided Design*), pozwalające na wizualizację realizowanych w nich projektów, lecz miały one dwie zasadnicze wady. Po pierwsze niemożliwe były jakiekolwiek interakcje z użytkownikiem, a ewentualne poruszanie się po takim „świecie” byłoby możliwe jedynie według zaplanowanej przez jego twórcę ścieżki. Po drugie zaś generowane przez takie programy pliki miały duże rozmiary, co powodowało, iż przesyłanie ich w Internecie na dużą skalę było wyjątkowo utrudnione i kosztowne.

Koncepcja rzeczywistości wirtualnej (ang. virtual reality) opierała się zaś na pomyśle stworzenia w pamięci komputera świata w pełni interaktywnego. W najbardziej zaawansowanych systemach; dzięki zastosowaniu odpowiednich urządzeń wejścia / wyjścia, takich jak hełmy z ekranami wypełniającymi całe pole widzenia użytkownika oraz skafandry z czujnikami informującymi program aplikacyjny o położeniu jego ciała, miałby on możliwość interaktywnego oddziaływania na obiekty wirtualnego świata. Użytkownik mógłby w nim nie tylko oglądać przedmioty ze wszystkich stron, pod dowolnym kątem, ale również zmieniać ich położenie, kształt itp. Miałby złudzenie całkowitego „zanurzenia się” w sztucznie wykreowanym świecie.

Panowie Tim Berners-Lee i Dave Ragett zorganizowali sesję, której celem miało być uzgodnienie jednolitego wyglądu interfejsu dla użytkowników wirtualnej rzeczywistości. Dyskusja była niezwykle ożywiona, lecz zbyt duże różnice zdań nie pozwoliły na osiągnięcie kompromisu i jakichkolwiek ustaleń.

Natomiast niedługo po nieudanej sesji powstała internetowa lista dyskusyjna. Grono jej uczestników w bardzo krótkim czasie osiągnęło liczbę ok. 1000 osób. Dzięki wspólnemu wysiłkowi tej listy na kolejnej konferencji WWW przedstawiono propozycję specyfikacji. Była to pierwsza wersja nowego języka – VRML (ang. *Virtual Reality Modeling Language*, czyli język do modelowania rzeczywistości wirtualnej)

Były trzy główne wymagania, jakie postawiono nowemu językowi:

- niezależność platformowa,
- wprowadzanie rozszerzeń,
- możliwość pracy na łączach o niskiej przepustowości.

Wyżej wymienione postulaty nie były życzeniami twórców, lecz ograniczeniami nałożonymi na nich przez rynek i istniejącą technologię. Od samego początku uwzględniano możliwości stosowania nowego języka w Internecie. Musiał więc działać na wielu różnych, obecnych w sieci platformach systemowych i tolerować małą przepustowość ówczesnych łącz. Łatwośc wprowadzania nowych rozszerzeń miała zaś zapewnić możliwość szybkiego rozwoju nowego języka.

3.2. Tworzenie struktur VRML.

Istnieją dwa sposoby tworzenia pliku VRML. Pierwszy z nich to bezpośrednie wpisanie kodu w jakimkolwiek edytorze tekstowym. Jeżeli autor dobrze zna składnię języka i ma rozwinięty zmysł przestrzenny, nie powinien mieć żadnych problemów z przelaniem myśli na kod. Proces twórczy można wówczas określić jako „pisz i patrz”, tj. najpierw napisz kod, a później zobacz, co osiągnąłeś.

Diametralnie odmiennym sposobem tworzenia światów jest używanie edytorów graficznych typu WYSIWYG (ang. *What You See Is What You Get*). Często są one wyposażone w pomocnicze procedury zwane kreatorami, prowadzące użytkownika „za rękę” przez proces tworzenia obiektów. Z tego względu metodologia ta jest polecana zwłaszcza początkującym użytkownikom. Podczas pracy cały czas widać efekt, jaki osiągnięto do tej pory. Dlatego też ten sposób określono jako „patrz i porównaj”, tj. najpierw coś zrób, a później porównaj to ze swoimi zamiarami.

Standardowy plik VRML składa się z czterech komponentów:

- nagłówka
- opisu graficznego sceny
- prototypów
- kierowania zdarzeniami.

Nagłówek jest nieodzowną częścią każdego z plików. Dzięki niemu interpreter jest w stanie określić, jakim pakietem komend należy się posłużyć. Zgodnie ze standardem ISO/IEC 14772-1:1997 nagłówek powinien wyglądać następująco:

```
#VRML V2.0 <encoding type> [optional comment] <line terminator>
```

Składa się on z 3 części: identyfikującej wersję, określającą sposób kodowania oraz opcjonalnego komentarza.

Opis graficzny sceny zawiera węzły (ang. *node*), które opisują obiekty (ang. *object*) i ich właściwości. Tekst jest zorganizowany hierarchicznie w strukturę drzewiastą, tak aby w prosty sposób zaprezentować węzły, ich właściwości i relacje pomiędzy nimi.

Prototypy pozwalają na wprowadzenie modyfikacji w węzłach VRML. Mogą one być zaimplementowane w pliku, w którym są one używane lub też zdefiniowane poza nim.

Kierowanie zdarzeniami jest mechanizmem, który określa, co stanie się z danym węzłem, jeżeli zaistnieje zdefiniowana sytuacja. Zarówno definicje zdarzenia wejściowego, jak i efekt działania, mogą być różne w zależności od zmian, jakich dokona autor. Takie procesy mogą wpływać na właściwości węzłów, generować dodatkowe zdarzenia, lub dokonywać zmian w prezentowanej scenie.

3.2.1 Bryły i ich kształty

Każdy węzeł VRML składa się z pól. Wszystkie pola mają taką samą postać ogólną:

```
<rodzaj> <typ> <nazwa> <wartość> #opcjonalny_komentarz
```

Parametr <*nazwa*> jest nazwą pola typu określonego przez parametr <*typ*>. Zależnie od typu pole może przyjmować pewną <*wartość*> z listy wartości dozwolonych dla tego typu.

Aby opisać różne bryły w języku VRML zastosowano węzły typu *Kształt* (ang. *Shape node*). Węzły te są zbudowane z pól dwóch typów: *appearance* i *geometry*.

Struktura węzła *shape* przedstawia się następująco:

```
Shape {
```

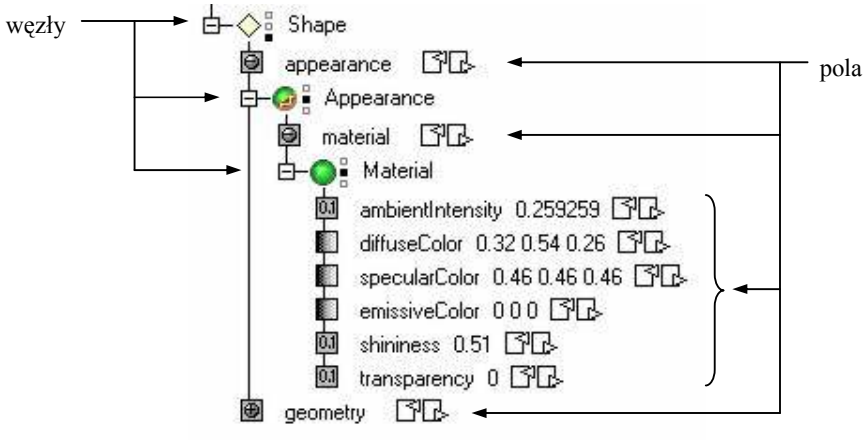
```
    exposedField SFNode appearance NULL
    exposedField SFNode geometry NULL
}
```

Rodzaj *ExposedField* lub *Field* informuje, że dana struktura jest polem, nie zaś węzłem.

Wartością pola *appearance* jest węzeł typu *Appearance*, którego zadaniem jest określenie właściwości wyglądu bryły. Jego budowa przedstawiona jest poniżej.

```
Appearance {
exposedField SFNode material NULL
exposedField SFNode texture NULL
exposedField SFNode textureTransform NULL
}
```

W węźle tym wyróżniamy trzy pola: *material, texture* i *textureTransform*. Podobnie jak pole *appearance* zawiera węzeł *Appearance*, również wyżej wymienione pola zawierają węzły o takich samych jak te pola nazwach. Opisana struktura przedstawiona jest na rysunku 3:



Rys. 3 Fragment hierarchii węzłów i pól w języku VRML

Węzeł *Material* określa właściwości materiału, z jakiego będzie zbudowana bryła. Jest również używany w procesie modelowania, dla określenia właściwości światła odbitego od danego obiektu. Składa się on z następujących pól:

```
Material {
exposedField SFFloat ambientIntensity 0.2      # [0,1]
exposedField SFColor diffuseColor 0.8 0.8 0.8  # [0,1]
exposedField SFColor emissiveColor 0 0 0  # [0,1]
exposedField SFFloat shininess 0.2        # [0,1]
exposedField SFColor specularColor 0 0 0  # [0,1]
exposedField SFFloat transparency 0 # [0,1]
}
```

Wszystkie te pola przyjmują wartości z zakresu [0,1]. Ich znaczenie jest następujące:

a) *ambientIntensity* – określa jasność obiektu. Dotyczy tylko światła typu *ambient*, czyli równomiernie rozproszonego, nie mającego wyraźnego kierunku padania. Określa on, jaka część tego światła zostanie odbita od powierzchni bryły. Kolor, jaki zostanie uzyskany wskutek odbicia, jest wyliczany jako iloczyn wartości pól *ambientIntensity* i *diffuseColor*.

b) *diffuseColor* ma wpływ na każdy rodzaj światła z uwzględnieniem jego typu i kąta padania. Określa główny kolor bryły a także, czy jej powierzchnia będzie wyglądała na gładką, czy też bardziej chropowatą.
c) *EmissiveColor* – określa rodzaj i natężenie światła emanującego z obiektu, niezależnie od liczby zewnętrznych źródeł oświetlenia istniejących w scenie. Można w ten sposób uzyskać np. efekt obiektu jarzącego się w ciemności. Parametr ten jest szczególnie użyteczny podczas tworzenia modeli, gdzie energia światła jest szczególnie wyraźna lub podczas reprezentacji modeli naukowych.
d) *specularColor* - w przypadku, gdy kąt pomiędzy źródłem światła a powierzchnią odbijającą jest bliski kątowi pomiędzy tą powierzchnią a osobą patrzącą, ma wpływ na tworzenie efektu połysku gładkiej powierzchni oraz na stopień rozproszenia koloru.
e) *shininess* – określa stopień wygładzenia powierzchni oraz stopień jej połysku / matowości.
f) *transparency* odpowiada za przejrzystość. Wartość 1 oznacza całkowitą przezroczystość obiektu, a zerowa kompletny jej brak.

Kolejnym polem w węźle *Appearance* jest *texture*. Pole to może, w zależności od życzenia projektanta, zawierać węzeł przynależący do jednego z trzech następujących typów:

- *ImageTexture* - opisuje teksturę zachowaną w pliku zapisaną w formacie: JPG, PNG, CGM lub GIF,
- *MovieTexture* - opisuje teksturę, w której występuje zależność czasowa (zmienia się ona wraz z upływem czasu),
- *PixelTexture* - posługuje się teksturą zawartą w tablicy.

Ostatnim polem węzła *Appearance* jest *textureTransform*. Jego zadaniem jest opisanie położenia tekstury w przestrzeni 2D. Możliwe jest np. użycie jako tekstury jedynie fragmentu większej całości, bez dokonywania modyfikacji pliku ją zawierającego.

```
TextureTransform {
  exposedField SFVec2f center      0 0
  exposedField SFFloat rotation    0
  exposedField SFVec2f scale       1 1
  exposedField SFVec2f translation 0 0
}
```

Wszystkie wymienione pola mogą przyjmować wartości z zakresu $(-\infty; +\infty)$.

Znaczenie poszczególnych pól jest następujące:

a) *center* – określa punk rzutowania tekstury
b) *rotation* – określa kąt obrotu tekstury
c) *scale* – określa stopień przeskalowanie tekstury
d) *translation* – wektor przesunięcia tekstury

Równorzędne w stosunku do *appearance* pole *geometry* określa typ żądanej bryły, np. sześcian, kula, walec...

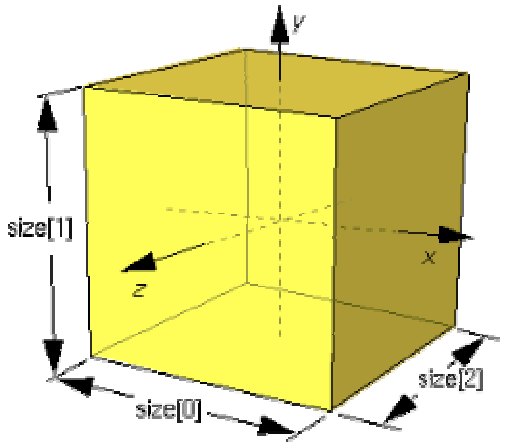
Ponieważ w języku VRML zostało zdefiniowane kilka różnych typów brył, ich budowa i wygląd są przedstawione w oddzielnych podrozdziałach.

3.2.1.1 Węzeł typu „sześcian”

Węzeł typu „sześcian” (ang. *box*) jest definiowany przez następującą formułę:

```
Box {
    field SFVec3f size 2 2 2 # (0, ∞)
}
```

Kolejne wartości pola *size* odpowiadają osiom współrzędnych przestrzennych: X, Y oraz Z.



Rys. 4 Interpretacja pola węzła typu sześcian

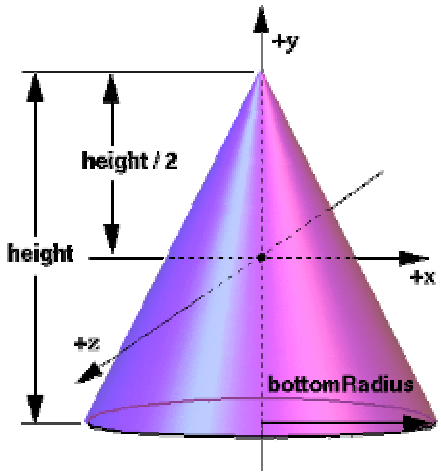
3.2.1.2 Węzeł typu „stożek”

Węzłowi typu „stożek” (ang. *cone)* odpowiada następująca formuła:

```
Cone {
    field SFFloat bottomRadius 1 # (0,∞)
    field SFFloat height 2 # (0,∞)
    field SFBool side TRUE
    field SFBool bottom TRUE
}
```

Kolejne pola tego węzła odpowiadają:

- *bottomRadius* - promieniowi podstawy
- *height* – wysokości stożka
- *side* - powierzchni bocznej bryły
- *bottom* - podstawie bryły



Rys. 5 Interpretacja pola węzła typu stożek

W bryle typu „sześcian” jedynymi parametrami były wymiary bryły. W przypadku stożka zostały wprowadzone dwa dodatkowe pola przyjmujące wartości logiczne *TRUE* lub *FALSE*. Są to pola:

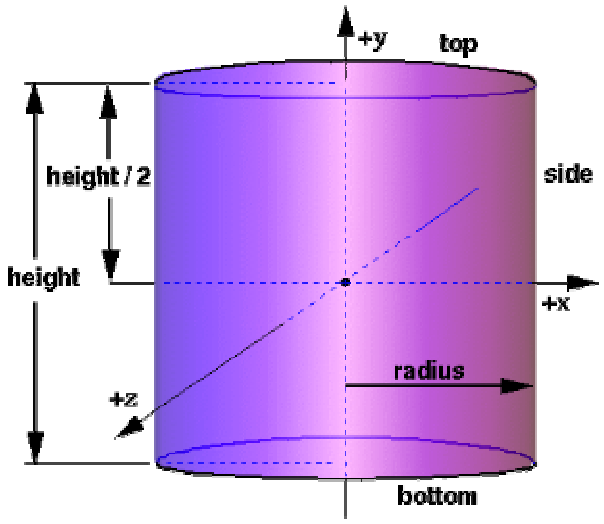
- *side*, za pomocą którego ustalane jest, czy ma być wyświetlana powierzchnia boczna bryły
- *bottom*, którego wartość przesądza o wyświetlaniu powierzchni podstawy

3.2.1.3 Węzeł typu „walec”

Kolejnym węzłem, który zostanie omówiony jest „walec” (ang. *cylinder*). Jego opis został zamieszczony poniżej.

```
Cylinder {
    field SFBool bottom TRUE
    field SFFloat height 2 # (0,∞)
    field SFFloat radius 1 # (0,∞)
    field SFBool side TRUE
    field SFBool top TRUE
}
```

Występują tu omówione już pola „*bottom*” i „*side*”, oraz nowe pole „*top*”. Definiuje ono wyświetlanie powierzchni górnej podstawy. Pola „*height*” i „*radius*”, podobnie jak w obu przedstawionych wcześniej przypadkach, określają odpowiednio wysokość i promień podstawy bryły.



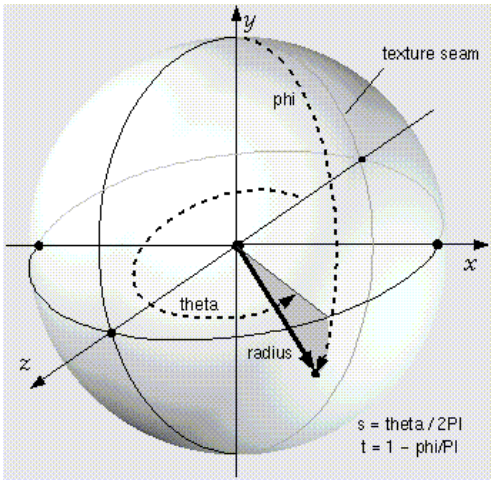
Rys. 6 Interpretacja pola węzła typu walec

3.2.1.4 Węzeł typu „kula”

Kolejny węzeł to „kula” (ang. *sphere*). Budowa tej bryły jest niezwykle prosta:

```
Sphere {
    field SFFloat radius 1 # (0,∞)
}
```

Za wyjątkiem pola, które określa promień bryły, nie ma żadnych innych parametrów definiowanych przez użytkownika.

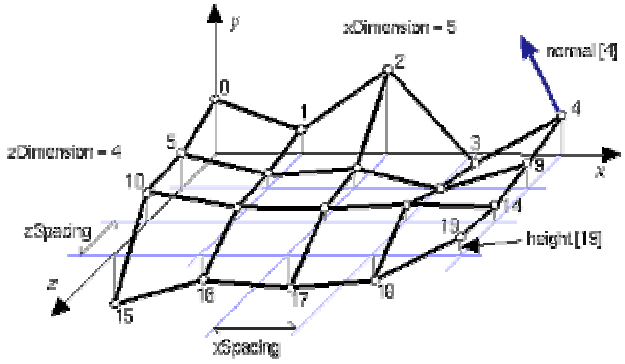


Rys. 7 Interpretacja pola węzła typu sfera

3.2.1.5 Pozostałe typy brył

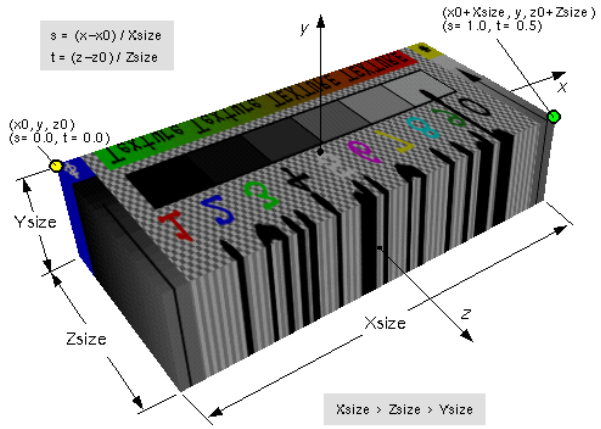
Do zbioru brył zostały zaliczone również takie obiekty jak:

- *ElevationGrid* (siatka czworoboków, powstała z połączenia punktów rozmieszczonych w trójwymiarowej przestrzeni)



Rys. 8 Interpretacja pola węzła typu ElevationGrid

- *Extrusion* (opisuje dwuwymiarowy przekrój figury)
- *IndexedFaceSet* (dodaje trzeci wymiar do płaskiej figury)



Rys. 9 Interpretacja pola węzła typu IndexedFaceSet

- *IndexedLineSet*
- *PointSet*
- *Text*

Szczegółowa składnia tych węzłow została przedstawiona w Załączniku 1

3.2.2 Łączenie i grupowanie węzłów

Jak już wspomniano w poprzednim rozdziale, tekst w pliku VRML jest zapisany w postaci drzewa. W przypadku scen zawierających dużą liczbę węzłów taki zapis staje się nieczytelny, zaistniała więc konieczność wprowadzenia węzłów nadrzędnych i podrzędnych.

Jeżeli któryś z węzłów grupujących zawiera w swojej budowie inne węzły grupujące wówczas nazywamy go ojcem. Z kolei węzły zawarte w nim noszą nazwę jego dzieci.

Ponieważ do stworzenia nawet najprostszego obiektu potrzebne jest określenie jego położenia w przestrzeni trójwymiarowej, omówienie tej grupy obiektów zacznę od węzła *Transform*.

3.2.2.1 Węzeł typu „Transform”

```
Transform {
    eventIn MFNode addChildren
    eventIn MFNode removeChildren
    exposedField SFVec3f center 0 0 0 # (-∞,∞)
    exposedField MFNode children []
    exposedField SFRotation rotation 0 0 1 0 # [-1,1],(- ∞,∞)
    exposedField SFVec3f scale 1 1 1 # (0, ∞)
    exposedField SFRotation scaleOrientation 0 0 1 0 # [-
    1,1],(- ∞,∞)
    exposedField SFVec3f translation 0 0 0 # (-∞,∞)
    field SFVec3f bboxCenter 0 0 0 # (-∞,∞)
    field SFVec3f bboxSize -1 -1 -1 # (0, ∞) or -1,-1,-1
}
```

Jednym z ważniejszych pól w tym węźle jest „*translation*”. Określa ono wektor, o jaki należy przesunąć względem węzła nadrzędnego wszystkie „dzieci” zawarte w polu *children*.

Jednak określenie dokładnego położenia bryły w przestrzeni nie wystarczy do określenia, czy jest ona obrócona, a jeżeli tak, to o ile stopni. Wprowadzenie pola *rotation* jest rozwiązało wyżej wymieniony problem. Wyjątkowo interesująca jest jego budowa. Składa się ono z czterech liczb. Pierwsze trzy wartości odpowiadają za oś i kierunek obrotu bryły. Przykładowo wartości 0 0 –1 określają, iż obrót nastąpi względem osi Z (jedynka na pozycji odpowiadającej tej osi); zgodnie z ruchem wskazówek zegara (znak minus przy jedynce wyznaczającej oś obrotu). Czwarta liczba określa kąt obrotu wyrażony w radianach.

Pole *center* definiuje środek, względem którego dokonywany jest obrót. Jest ono niezwykle przydatne podczas animowania obiektów. Przykładem mogą być drzwi, które obracają się w miejscu zawieszenia tj. na zawiasach. Jeżeli więc środek zawiasów zostanie ustawiony jako środek obrotu wówczas otwierane drzwi będą sprawiały realistyczne wrażenie.

Ponieważ może zaistnieć konieczność utworzenia lub usunięcia obiektu dopiero po zajściu jakiegoś zdarzenia, węzeł *Transform* zawiera na taką okoliczność dwa zdarzenia: *addChildren* i *removeChildren*.

Węzły *Transform* pozwalały w pełni określić przestrzenne usytuowanie obiektu dzięki polom *translation* i *rotation*. VRML oferuje również inny, podobnie działający węzeł, lecz pozbawiony możliwości definiowania tych dwóch parametrów.

3.2.2.2 Węzeł typu „Group”

```
Group {
    eventIn MFNode addChildren
```

```
    eventIn MFNode removeChildren
    exposedField MFNode children []
    field SFVec3f bboxCenter 0 0 0 # (-∞,∞)
    field SFVec3f bboxSize -1 -1 -1 # (0, ∞) or -1,-1,-1
}
```

Praktycznym efektem zastosowania tego węzła jest uporządkowanie struktury pliku wynikowego i ułatwienie orientacji w kodzie. Wykorzystywany jest także do grupowania pewnej liczby obiektów, wówczas powoływanie się na nazwę tej grupy powoduje stworzenie jej klonu. Operacja taka zmniejsza ilość kodu i wpływa na szybkość działania świata.

3.2.2.3 Węzeł typu „Inline”

Jest to przykład kolejnego węzła grupującego. Pozwala on na importowanie z zewnątrz obiektów VRML, dzięki czemu raz stworzona bryła może być wielokrotnie wykorzystywana w różnych scenach. Funkcjonuje na zasadzie kontenera wypełnianego gotowymi klockami.

```
Inline {
    exposedField MFString url []
    field SFVec3f bboxCenter 0 0 0 # (-∞,∞)
    field SFVec3f bboxSize -1 -1 -1 # (0,∞) or -1,-1,-1
}
```

W swojej strukturze zawiera on pole *url*. W tym miejscu specyfikujemy adres URL obiektu lub sceny, których będziemy chcieli użyć. Węzeł ten jest szczególnie przydatny, gdy twórca nie dysponuje potężną maszyną graficzną, a chce stworzyć pomieszczenie wymagające dużych nakładów obliczeniowych. W takim przypadku może on tworzyć małe elementy swojego projektu, aby w końcowej fazie połączyć je.

3.2.2.4 Węzeł typu „Level Of Details”

Ponieważ celem VRML jest prezentacja danych szerokiemu gronu użytkowników sieci Internet, twórcy muszą uwzględniać fakt, iż nie każdy, kto będzie oglądać ich prace, będzie dysponował sprzętem o odpowiednio dużej mocy obliczeniowej. W standardzie uwzględniono ten problem i stworzono węzeł „*LOD*”. Pozwala on na wyświetlanie oddalonych od obserwatora obiektów z pominięciem drobnych szczegółów, które z dużej odległości i tak nie są dostrzegane. Dzięki temu zmniejszają się nakłady obliczeniowe wymagane do stworzenia danej sceny.

```
LOD {
    exposedField MFNode level []
    field SFVec3f center 0 0 0 # (-∞,∞)
    field MFFloat range [] # (0, ∞)
}
```

„*LOD*” lub „*Level Of Details*”, co można przetłumaczyć jako poziom szczegółowości, jest węzłem, który w swojej strukturze definiuje dwa parametry:

- odległość od środka węzła (ang. *range*)
- numer poziomu odpowiadający tej odległości (ang. *level*)

Jeżeli wartość X zostanie ustalona jako progowa dla poziomu 1, a odległość użytkownika od obiektu będzie większa niż Y = X+*range*, wówczas zostanie on wyświetlony z liczbą szczegółów przewidzianą dla poziomu 2. W przypadku, gdy poziom 2 nie jest określony tj. nie zawiera żadnych obiektów, wyświetlanie obiektów z poziomu pierwszego zostanie zaprzestane.

3.2.2.5 Węzeł typu „Switch”

```
Switch {
    exposedField MFNode choice []
    exposedField SFInt32 whichChoice -1 # [-1,∞)
}
```

Efektem działania tego pola jest wybór któregoś z węzłów zawartych w polu *choice*. Wybór ten jest dokonywany na podstawie wartości, jaka zostanie przypisana polu *whichChoice* przez zdarzenie (mechanizmy działania zdarzeń są opisane w dalszej części pracy).

Powyżej zostały opisane najważniejsze węzły grupujące. Oprócz nich istnieją jeszcze typy „*Anchor*”, „*Billboard*” i „*Collision*”. Są one jednak rzadko wykorzystywane, więc dokładny opis ich struktury został zawarty jedynie w Załączniku 2.

Wszystkie prezentowane węzły są w stanie łączyć lub grupować węzły następujących typów:

- *Anchor*
- *Background*
- *Billboard*
- *Collision*
- *ColorInterpolator*
- *CoordinateInterpolator*
- *CylinderSensor*
- *DirectionalLight*
- *Fog*
- *Group*
- *Inline*
- *LOD*
- *NavigationInfo*
- *NormalInterpolator*
- *OrientationInterpolator*
- *PlaneSensor*
- *PointLight*
- *PositionInterpolator*
- *ProximitySensor*
- *ScalarInterpolator*
- *Script*
- *Shape*
- *Sound*
- *SpotLight*

- *SphereSensor*
- *Switch*
- *TimeSensor*
- *TouchSensor*
- *Transform*
- *Viewpoint*
- *VisibilitySensor*
- *WorldInfo*

Oprócz wymienionych węzłów, które mogą podlegać łączeniu lub grupowaniu, istnieje również grupa węzłów, które nie mają takiej możliwości. Są to węzły:

- *Appearance*
- *AudioClip*
- *Box*
- *Color*
- *Cone*
- *Coordinate*
- *Cylinder*
- *ElevationGrid*
- *Extrusion*
- *ImageTexture*
- *IndexedFaceSet*
- *IndexedLineSet*
- *Material*
- *MovieTexture*
- *Normal*
- *PointSet*
- *Sphere*
- *Text*
- *TextureCoordinate*
- *TextureTransform*

3.2.3 Oświetlenie

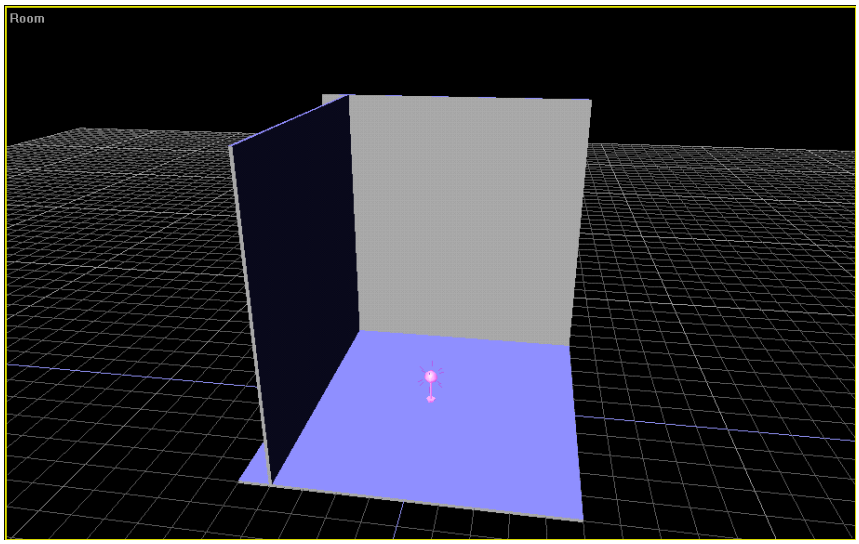
W składni języka VRML zdefiniowane zostały trzy rodzaje źródeł światła.

3.2.3.1 Directional Light

Pierwsze z nich to światło ukierunkowane (ang. *DirectionalLight*).

```
DirectionalLight {
    exposedField SFFloat ambientIntensity 0 # [0,1]
    exposedField SFColor color 1 1 1 # [0,1]
    exposedField SFVec3f direction 0 0 -1 # (-∞,∞)
    exposedField SFFloat intensity 1 # [0,1]
    exposedField SFBool on TRUE
}
```

Źródło to będzie emitować światło zgodnie z kierunkiem wektora określonego w polu *direction*. Wszystkie obiekty, na które będzie padać to światło, zostaną oświetlone kolorem, jaki zostanie ustawiony w polu *color*.



Rys. 10 Światło ukierunkowane (ang. Directional Light)

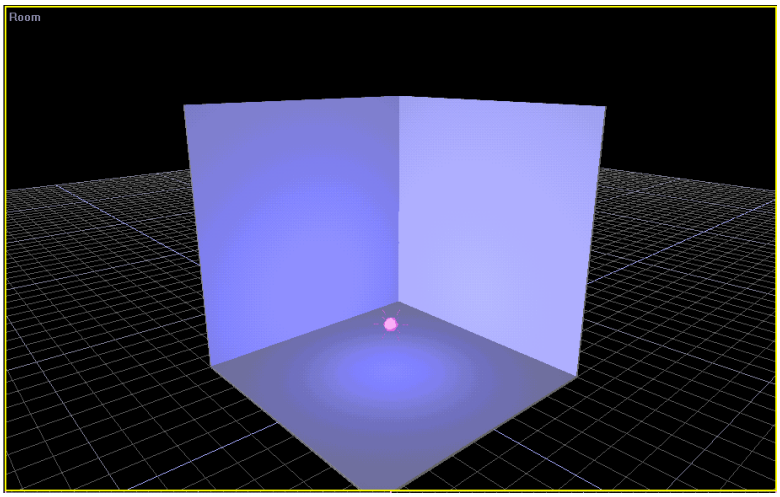
Na rys.10 pokazano, jak światło pochodzące ze źródła oświetla nie tylko podłoże, ale i górne krawędzie ścian prostopadłościanów. Pomimo ścian, jakie są umieszczone w tej scenie widzimy, iż światło przez nie przechodzi. Tak więc niezależnie od miejsca, w którym zostanie umieszczone źródło, efekt jest taki sam.

3.2.3.2 Point Light

Drugim dostępnym typem jest światło punktowe (ang. *PointLihgt*). Światło z tego źródła rozchodzi się równomiernie we wszystkich kierunkach, nie jest możliwe zdefiniowanie wektora, zgodnie z którym będzie świeciło. W związku z tym należy określić zarówno dokładną pozycję źródła, jak i jego zasięg.

```
PointLight {
    exposedField SFFloat ambientIntensity 0 # [0,1]
    exposedField SFVec3f attenuation 1 0 0 # [0,∞)
    exposedField SFColor color 1 1 1 # [0,1]
    exposedField SFFloat intensity 1 # [0,1]
    exposedField SFVec3f location 0 0 0 # (-∞,∞)
    exposedField SFBool on TRUE
    exposedField SFFloat radius 100 # [0, ∞)
}
```

Patrząc na budowę tego węzła można zauważyć, iż pole *location*, będzie odpowiedzialne za położenie źródła. Natomiast w polu *radius* określany jest zasięg światła. Przykład dzialania światła tego typu przedstawiono na rys.11.



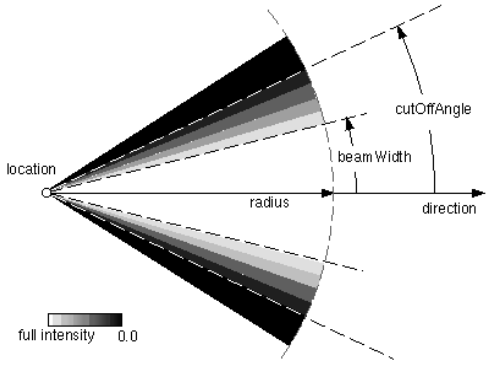
Rys. 11 Światło punktowe (ang. PointLihgt)

3.2.3.3 Spot Light

Ostatnim rodzajem światła jest światło reflektorowe (ang. *Spot Light*).

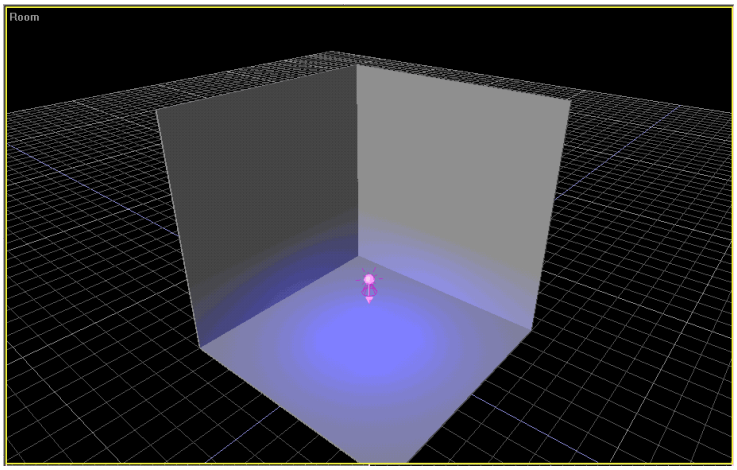
```
SpotLight {
    exposedField SFFloat ambientIntensity 0 # [0,1]
    exposedField SFVec3f attenuation 1 0 0 # [0,)
    exposedField SFFloat beamWidth 1.570796 # (0,/2]
    exposedField SFColor color 1 1 1 # [0,1]
    exposedField SFFloat cutOffAngle 0.785398 # (0,/2]
    exposedField SFVec3f direction 0 0 -1 # (-,)
    exposedField SFFloat intensity 1 # [0,1]
    exposedField SFVec3f location 0 0 0 # (-,)
    exposedField SFBool on TRUE
    exposedField SFFloat radius 100 # [0,)
}
```

Pola nie występujące w poprzednich strukturach to *cutOffAngle*, które określa kąt powyżej którego światło jest nie widoczne; oraz pole *beamWidth*, które określa kąt w którym światło będzie świecić swoją czystą barwą. Wpływ wartości pól na sposób generowania światła został przedstawiony na rys.12.



Rys. 12 Parametry pola „światło reflektorowe”

Poniżej przedstawiono realizację praktyczną, dzięki której można zaobserwować, jak w rzeczywistości działa powyższy węzeł.



Rys. 13 Światło reflektorowe (ang. Spot Light)

3.2.4 Czujniki

Czujniki to grupa węzłów, które odpowiadają za pobieranie i obsługę zdarzeń. Istnieje dziewięć rodzajów czujników. Są to:

- *Anchor*
- *Collision*
- *CylinderSensor*
- *PlaneSensor*
- *ProximitySensor*
- *SphereSensor*
- *TimeSensor*
- *TouchSensor*
- *VisibilitySensor*

3.2.4.1 Czujnik typu „Anchor”

Jego struktura przedstawia się nastepująco:

```
Anchor {
    eventIn MFNode addChildren
    eventIn MFNode removeChildren
    exposedField MFNode children []
    exposedField SFString description ""
    exposedField MFString parameter []
    exposedField MFString url []
    field SFVec3f bboxCenter 0 0 0 # (-∞,∞)
    field SFVec3f bboxSize -1 -1 -1 # (0,∞) or -1,-1,-1
}
```

Czujnik ten zawiera trzy pola węzłów grupujących:

- *addChildren*
- *RemoveChildren*
- *Children*

Pojawiły się też nowe pola, niespotykane we wcześniej opisanych obiektach.

Główną funkcją węzła *Anchor* jest odwołanie się do lokalizacji w sieci Internet. Adres internetowy określany jest w polu „*url*”. Pole *description* zawiera nazwę okna, w którym ma zostać umieszczona zawartość wyspecyfikowanej lokalizacji.

Ponieważ większość przeglądarek internetowych zawiera komendy wewnętrzne, twórcy VRML umożliwili ich wykorzystanie poprzez wpisanie do pola *parameter*.

3.2.4.2 Czujnik typu „Collision”

Rolą tego czujnika jest wykrywanie kolizji obiektów i podjęcie działań przewidzianych jako reakcja na dane zdarzenie. Przykładem takiej sytuacji może być symulacja gry w kręgle. Po wykryciu, iż rzucona kula uderzyła w kręgle, powinny się one przewrócić. Za podjęcie takiej decyzji odpowiada właśnie czujnik typu *Collision*.

```
Collision {
    eventIn MFNode addChildren
    eventIn MFNode removeChildren
    exposedField MFNode children []
    exposedField SFBool collide TRUE
    field SFVec3f bboxCenter 0 0 0 # (-∞,∞)
    field SFVec3f bboxSize -1 -1 -1 # (0, ∞) or -1,-1,-1
    field SFNode proxy NULL
    eventOut SFTime collideTime
}
```

W węźle *Collision*, podobnie jak w węźle *Anchor*, występuje pole węzłów grupujących *proxy*.

Jedynym nie opisanym dotąd polem jest *collideTime*. Pełni ono funkcję informacyjną - staje się aktywne, tj. pojawia się w nim jakaś wartość, tylko podczas kontaktu użytkownika z węzłami znajdującymi się w polu *children*.

3.2.4.3 Czujnik typu „Proximity Sensor”

Jego celem jest śledzenie położenia „osoby” poruszającej się w świecie VRML.

```
ProximitySensor {
    exposedField SFVec3f center 0 0 0 # (-∞,∞)
    exposedField SFVec3f size 0 0 0 # [0, ∞)
    exposedField SFBool enabled TRUE
    eventOut SFBool isActive
    eventOut SFVec3f position_changed
    eventOut SFRotation orientation_changed
    eventOut SFTime enterTime
    eventOut SFTime exitTime
}
```

Czujnik ten jest reprezentowany jako prostopadłościan, którego środek leży w punkcie określonym w polu *center*. Natomiast wielkość jego ścian jest definiowana przez wartości w polu *size*.

O fakcie wejścia bądź też opuszczenia strefy zasięgu czujnika informują pola *enterTime* i *exitTime*. Natomiast podczas obecności obiektu w tej strefie, pola *position_changed* i *orientation_changed* informują o zajściu zmiany jego położenia lub obrotu.

3.2.4.4 Czujnik typu „Touch Sensor”

```
TouchSensor {
    exposedField SFBool enabled TRUE
    eventOut SFVec3f hitNormal_changed
    eventOut SFVec3f hitPoint_changed
    eventOut SFVec2f hitTexCoord_changed
    eventOut SFBool isActive
    eventOut SFBool isOver
    eventOut SFTime touchTime
}
```

Funkcją tego czujnika jest informowanie o fakcie kliknięcia na jakimś określonym węźle bądź też grupie węzłów.

Dokładny czas zajścia zdarzenia rejestruje pole *touchTime*, natomiast pola *hitNormal_changed*, *hitPoint_changed* oraz *hitTexCoord_changed* powiadamiają o zmianach pomiędzy kolejnymi kliknięciami.

3.2.4.5 Czujnik typu „Time Sensor”

```
TimeSensor {
    exposedField SFTime cycleInterval 1 # (0,)
    exposedField SFBool enabled TRUE
    exposedField SFBool loop FALSE
    exposedField SFTime startTime 0 # (-,)
    exposedField SFTime stopTime 0 # (-,)
    eventOut SFTime cycleTime
    eventOut SFFloat fraction_changed # [0, 1]
    eventOut SFBool isActive
    eventOut SFTime time
}
```

Powyższy węzeł jest wprawdzie formalnie klasyfikowany jako czujnik, jednak nie pobiera on żadnych danych z otaczających go obiektów. Może natomiast przyjmować dane z innych czujników, dlatego też wykorzystywany jest przede wszystkim podczas tworzenia animacji.

Pole *startTime* jest wykorzystywane do zainicjowania ruchu obiektu, natomiast *stopTime* służy do zatrzymania zmian.

Dzięki polu *loop* istnieje możliwość zapętlenia zmian, tzn. po tym, jak zakończy się cykl o długości określonej w polu *cycleTime*, nastąpi przejście znów do pierwszej wartości.

3.2.5 Interpolatory

Węzły interpolujące, inaczej nazywane interpolatorami, zostały zaprojektowane w celu umożliwienia realizacji animacji liniowych. Dokonuje się ich na podstawie generowania stanów pomiędzy dwoma określonymi punktami. Budowa ogólna węzła jest następująca:

```
eventIn SFFloat set_fraction
exposedField MFFloat key [...]
exposedField MF<type> keyValue [...]
eventOut [S|M]F<type> value_changed
```

Pole *key* zawiera wartości określające moment w czasie, natomiast pole *keyValue* zawiera wartość, jaka ma być przyjęta w tym momencie. Informacje o wartości pośredniej są zapisywane w polu *value_changed*. Całość interpolacji jest realizowana na podstawie następującego wzoru:

$$f(t)=\begin{cases} v_0 & if\ \ t \le t_0 \\ linterp(t, v_i, v_{i+1}) & if\ t_i <= t <= t_{i+1}, \\ v_{n-1} & if\ t \ge t_{n-1} \end{cases} \qquad (1)$$

gdzie*f(t)* jest funkcją interpolującą, wartości $t_0, t_1, ..., t_{n-1}$ są kolejnymi wartościami z pól „*key*”, zaś $v_0, v_1, ..., v_{n-1}$ to wartości zawarte w polu *keyValue.* Natomiast *linterp(t,x,y)* to interpolator linearny zawierający się w przedziale *{0,1,..., n-2}*.

W składni języka VRML dostępne jest 6 różnego typu interpolatorów:

- *ColorInterpolator*
- *CoordinateInterpolator*
- *NormalInterpolator*
- *OrientationInterpolator*
- *PositionInterpolator*
- *ScalarInterpolator*

W zależności od żądanego efektu, każdy z wyżej wymienionych węzłów generuje wartości pośrednie innego typu. Zależność pomiędzy rodzajem węzła i jego wynikiem jego interpolacji jest przedstawiona w Tabeli 1

Tab.1 Parametry interpolatorów

Nazwa węzła	Typ wyjścia interpolatora	Wykorzystanie przy interpolacji :
ColorInterpolator	SFColor (3 składowe RGB)	kolorów pośrednich RGB
CoordinateInterpolator	MFVec3f (tablica trzech składowych typu *float*)	
NormalInterpolator	MFVec3f (tablica trzech składowych typu *float*)	
OrientationInterpolator	SFRotation (cztery wartości, trzy z zakresu <-1, 1>, czwarta typu *float*)	kątów pośrednich przy obrocie np. bryły
PositionInterpolator	SFVec3f (3 składowe typu *float*)	współrzędnych pośrednich w przestrzeni opisanej przez X,Y,Z
ScalarInterpolator	SFFloat (pojedyncza wartość typu *float*)	

4. Projekt

4.1. Założenia projektowe

Celem projektu było stworzenie modelu wirtualnego świata oraz klientów, którzy poruszaliby się wewnątrz niego i mieli możliwość obserwowania poczynań pozostałych użytkowników obecnych w tym samym pomieszczeniu. Przyjęto język VRML jako narzędzie do opracowania strony graficznej projektu, czyli modelu sali multimedialnej PJWSTK. Jako język programowania, potrzebny do stworzenia serwera obsługującego komunikację między klientami, wybrano Javę.

Założono, iż stworzony świat wirtualny powinien być dostępny w Internecie lub typowej sieci lokalnej. Ponieważ nie wiadomo, jakie komputery mogą znajdować się w sieci, sam model powinien być na tyle elastyczny, aby można go było obejrzeć (przynajmniej fragmentarycznie) również na maszynach o słabszej konfiguracji. Pomimo tak postawionego problemu, muszą być postawione pewne warunki natury programowej i sprzętowej.

Wymagania programowe:

- graficzny system operacyjny (Windows, OS/2, X-Windows, MacOS ...)
- obecna w systemie przeglądarka VRML
- system musi obsługiwać protokół TCP\IP

Wymagania sprzętowe:

- karta sieciowa
- przynajmniej 32 MB pamięci operacyjnej

4.1.1 Problemy powstałe podczas tworzenia obiektów w VRML.

Pracownia multimedialna, znajdująca się w Polsko-Japońskiej Wyższej Szkole Technik Komputerowych, została podzielona na dwie części – jedna z nich wyposażona jest w sprzęt do obróbki materiałów audio, druga zaś - wideo. Z powodu znacznej liczby urządzeń w laboratorium oraz dużej złożoności obliczeniowej ich modeli, niemożliwe okazało się tworzenie i obróbka poszczególnych obiektów w już istniejącym pomieszczeniu. Przetwarzanie pojedynczego obiektu powodowało konieczność ponownego odświeżenia wszystkich elementów sceny, co przekraczało możliwości obliczeniowe komputera, którym dysponowano. Z tych samych powodów niepowodzeniem zakończyły się próby skupienia ich w jedynie dwóch grupach – osobno urządzeń do obróbki audio i wideo.

W zaistniałej sytuacji jedynym sensownym sposobem postępowania okazało się oddzielne tworzenie każdego modelu. Następnie wszystkie części stopniowo były łączone w coraz większe całości takie jak: „szafka nr1 – część audio” → „prawa część - audio” → „audio”.

Takie rozwiązanie posiadało wiele zalet. Jedną z najważniejszych była łatwość aktualizacji modelu. Jeżeli dokonano zmiany w pliku zawierającym „obiekt podstawowy”

np.: magnetowid, wówczas zmiany te były widoczne we wszystkich modelach, które wykorzystywały ten obiekt, bez żadnych dodatkowych ingerencji ze strony twórcy. Zaoszczędzało to wiele pracy związanej z modyfikacją już istniejących modeli.

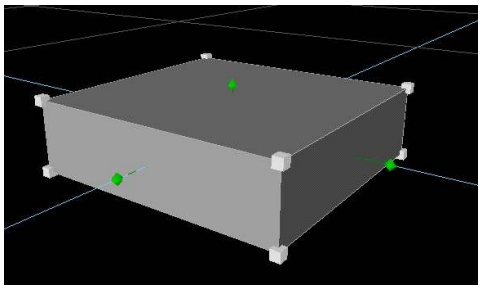
Również konstruowanie nowych scen od pewnego momentu zaczęło przypominać składanie gotowych modułów – duża liczba dostępnych, zdefiniowanych obiektów pozwalała na stosowanie ich w różnych konfiguracjach, bez straty czasu na tworzenie wszystkiego od nowa.

Jednak, pomimo niewątpliwych zalet, rozwiązanie to posiadało również pewne wady. Jedną z nich była duża ilość plików, jaka powstała w procesie tworzenia modelu. Jednak odpowiednie zarządzanie nimi i stworzenie przemyślanej struktury katalogów pozwoliło w znacznym stopniu wyeliminować tę niedogodność.

Ponieważ celem projektu było stworzenie pomieszczenia, które wiernie oddawałoby rzeczywistość, w związku z tym twórca był zmuszony do bardzo precyzyjnego określenia wymiarów wszystkich sprzętów znajdujących się w sali multimedialnej.

Pomimo to, aby w pełni zobrazować możliwości niektórych obiektów nie wystarczyły podstawowe parametry, takie jak wysokość, szerokość i długość. Przykładem może być syntezator marki Korg. Do oddania natury tego instrumentu, tj. możliwości grania na nim, należało precyzyjnie zamodelować wszystkie klawisze z osobna oraz wiernie oddać wszystkie detale konstrukcji.

Również w przypadku prostszych form, jak monitory czy drukarki, podstawowe parametry nie zawsze wystarczały. Po zgromadzeniu wszystkich wymiarów, przystąpiono do tworzenia obiektów. Bryłę będącą podstawą dla modelu magnetowidu przedstawiono na rysunku 14.



Rys. 14 Model wideo

Zakończenie modelowania kształtów obiektów pozwoliło na przejście do drugiej fazy tworzenia pracowni - nadawania tekstur.

W tej części projektu należało sporządzić fotograficzną dokumentację wyglądu wszystkich urządzeń, które znalazły się w modelu. Podczas tej części pracy problemy pojawiały się na każdym niemal kroku. Niezależnie od zastosowanej techniki fotografowania ostrość i nasycenie kolorów nie zawsze były idealne.

Problemy sprawiało też ustawienie aparatu dokładnie równolegle do fotografowanej powierzchni, co powodowało skróty perspektywiczne zniekształcające uzyskane obrazy.

Niezadowalające efekty zastosowania takich zdjęć powodowały konieczność wielokrotnego powtarzania fotografii.

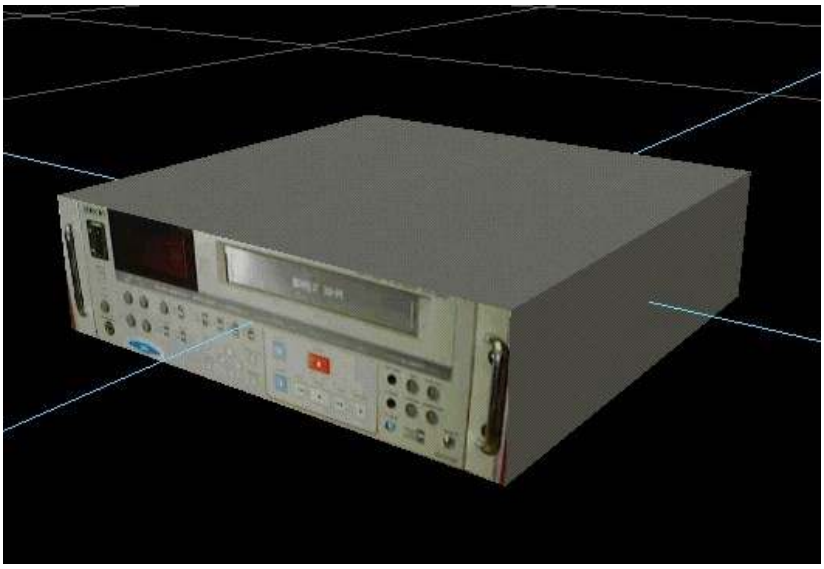
Rys. 15 Fotografia sekcji wideo

Gdy została zgromadzona odpowiednia liczba zdjęć, dzięki programowi Corel Photo-Paint wyselekcjonowano, a później wyizolowano potrzebne obrazy, które następnie posłużyły jako tekstury. Na rysunkach 15 i 16 widać odpowiednio fotografię całej sekcji wideo, oraz wycięty z niej fragment, który posłużył jako tekstura dla obiektu *wideo*.

Rys. 16 Obraz uzyskany po obróbce

Ostatnim elementem tej fazy był proces „naciągania tekstur”, dzięki któremu krawędzie każdego z modelowanych sprzętów pokryły się z krawędziami fotografii. Naciaganie tekstury zostało pokazane na rysunku 17, zaś ostateczny efekt operacji, czyli gotowy model magnetowidu, na rysunku 18

Rys. 17 Naciąganie tekstury na obiekt



Rys. 18 Efekt końcowy modelowania urządzenia

Ostatnią fazą było połączenie wszystkich elementów w jedną całość. Jak przewidywano, operacja ta nie spowodowała żadnych problemów.

4.2. Problemy związane z serwerem Java.

Podczas tworzenia szkieletu oprogramowania nie było problemów z utworzeniem serwera oraz podstawowych protokołów komunikacji. Kłopoty zaczęły się jednak pojawiać w momencie, gdy kod klienta Javy połączony został z kodem VRML.

4.2.1 Współpraca serwera różnymi platformami.

Język Java gwarantuje, że niezależnie od tego, w jakim systemie operacyjnym zostanie uruchomiony serwer, będzie on działał poprawnie i nie będzie powodował problemów. Kiedy korzysta się z systemu operacyjnego posiadającego graficzny interfejs użytkownika (ang. *Graphic User Interface*), takiego jak Windows95, WindowsNT, OS/2 lub X-Windows, serwer działa poprawnie. Jednak podczas próby uruchomienia go w środowisku tekstowym powstaje problem wyświetlania okien i w efekcie błędne działanie aplikacji.

4.2.2 Czas propagacji danych.

Jeżeli założymy, iż do serwera podłączone jest 10 osób i każda z nich wyśle do portu tylko jedną nową pozycję, to pozycja ta musi zostać rozesłana do 9 innych klientów. Każdy klient będzie oczekiwał maksymalnie 50 ms na potwierdzenie gotowości odbierania i 250 ms na jedną daną (przy czym należy pamiętać, iż zawsze wysyłamy 3 dane). Przyjęte czasy oczekiwania zostały określone na podstawie przeprowadzonych badań i zdrowego rozsądku.

Efektem tego rozważania jest poniższy wzór, określający najdłuższy dopuszczalny czas trwania pojedynczego cyklu przesyłania danych:

$$t_i = n \cdot ((n-1) \cdot ((250 \cdot 3) + 50)) \qquad (2)$$

gdzie: n-liczba użytkowników

dla danych przyjętych jako przykład jest to :

$$t_i = 10 \cdot 9 \cdot 800 = 72000 \ \ ms \ \ czyli \ \ 1 min \ \ 12 s$$

Powyższy wzór nie uwzględnia opóźnienia, jakie powstanie w sieci lokalnej bądź Internecie.

4.3. Problemy związane z klientem

Zadaniem klienta powinno być wysyłanie pozycji obserwatora do serwera, który powinien przekazać tę informacje do pozostałych klientów.

4.3.1 Problem przeładowanego portu.

Podczas, gdy użytkownik porusza się po jakimś pomieszczeniu w świecie VRML, każda zmiana jego pozycji może być zarejestrowana. Tak więc, jeżeli przebył on drogę z punktu A o współrzędnych [0,0,0] do punktu B o współrzędnych [0,0,1], to pomiędzy tymi dwoma punktami jest nieskończenie wiele punktów pośrednich, które znajdują się na linii ruchu. Ponieważ procesor nie jest taktowany nieskończoną częstotliwością, więc rzeczywiście zarejestrowana liczba punktów jest znacznie ograniczona przez skończony, niezerowy czas trwania cyklu zegarowego. Jednak, pomimo tego ograniczenia, w dalszym ciągu do dyspozycji pozostaje znacznie większa ilość danych, niż serwer jest w stanie odebrać i efektywnie przetworzyć. Jeśli informacje o zmianie położenia nie będą dodatkowo

selekcjonowane, to w bardzo krótkim czasie dojdzie do zerwania połączenia wskutek przeładowania portu.

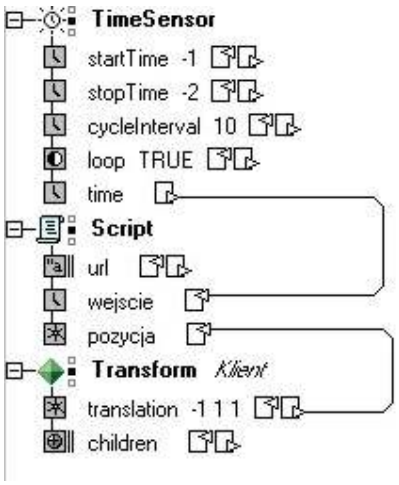
4.3.2 Wyodrębnienie stanów końcowych (postoju)

Podczas analizy problemu przeładowania portu, najbardziej logicznym rozwiązaniem wydawało się być przesyłanie jak najmniejszej ilości danych o położeniu klienta. Wymaga to dokonania wyboru pewnych charakterystycznych punktów trajektorii, na podstawie których można by skutecznie i z małym błędem interpolować jej przebieg, i przesyłania współrzędnych jedynie tych punktów. Zadanie odtworzenia przebiegu ruchu spoczywa wówczas na komputerze klienta, co pozwala zmniejszyć zarówno obciążenie serwera, jak i łączy sieciowych. Mogą być wykorzystywane nawet łącza o bardzo małej przepustowości, praktycznie nieużyteczne w sytuacji, gdy przesyłany jest pełen komplet danych.

Pozostaje problem wyboru punktów, których współrzędne będą przesyłane.

Mogłyby to być punkty postoju klienta, wówczas można by wysłać tylko jeden komplet danych, a mechanizmy interpolacji wygenerowałyby przebieg drogi klienta z punktu A do punktu B. Jednak to rozwiązanie, choć najefektywniejsze, jest praktycznie niemożliwe do zrealizowania. Dzieje się tak dlatego, iż czujnik ruchu (ang. *ProximitySensor*) zwraca informacje tylko o **zmianach** w położeniu klienta. Gdy pozostaje on w bezruchu, czujnik ten jest praktycznie bezużyteczny.

Kwestię tę można również rozwiązać inaczej, mianowicie tworząc programową pętlę VRML wykorzystującą czujnik czasu (ang. *TimeSensor*). W skład takiej pętli wchodzi wspomniany czujnik czasu oraz klasa *.java. Jej struktura przedstawiona jest na rysunku 17



Rys. 19 Pętla w języku VRML

W takim układzie *TimeSensor* co określony czas wysyłałby żądanie określenia aktualnych współrzędnych pozycji klienta. Klasa Javy pobierałaby wartość pola *translation* zawierającego żądane dane. Jeżeli byłaby ona taka sama jak poprzednia, oznaczałoby to, iż klient się nie porusza. Należałoby wówczas rozesłać tę pozycję do pozostałych klientów, jako informacje o punkcie końcowym dla algorytmu interpolacji.

Jednak postępowanie takie rodzi kolejne pytania: jak długo powinien klient przebywać w jednym miejscu, aby zostało uznane to za postój? Czy taka realizacja nie odbierze interaktywności całemu przedsięwzięciu?

Gdyby problem rozwiązano tak jak zostało to opisane powyżej, wówczas klient mógłby bez ustanku chodzić po pomieszczeniu, a pozostali klienci byliby przekonani ze stoi on w miejscu, gdyż nie odbieraliby informacji o położeniu końcowym i niemożliwe byłoby rozpoczęcie interpolacji trajektorii. Ponadto niezależnie od stopnia zawiłości rzeczywistej trasy klienta, pozostali obserwatorzy widzieliby, że porusza się on wyłącznie wzdłuż linii prostych.

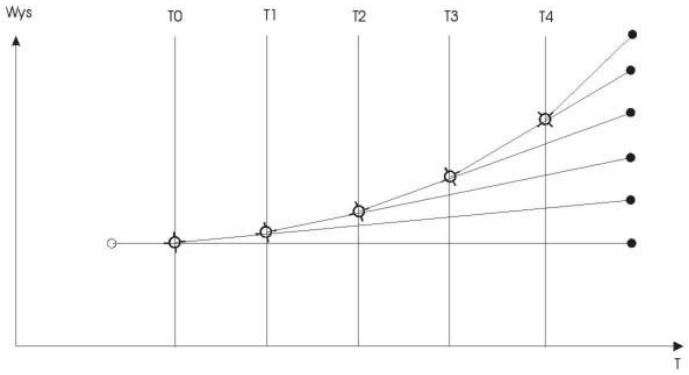
4.3.3 Interpolacja stanów pośrednich

Kłopot z interpolacją stanów pośrednich jest ściśle związany z wcześniej przedstawionym problemem przeładowania portu. Jak już wykazano, nie ma fizycznej możliwości przesłania informacji o wszystkich stanach pośrednich ruchu pomiędzy dwoma punktami przestrzeni. Pojawia się problem skoku pomiędzy dowolnymi punktami A i B oraz rozsądnego wyboru jego długości, tak by z jednej strony nie wygenerować zbyt wielu danych, z drugiej – by zapewnić wrażenie płynnego ruchu obiektu reprezentującego postać klienta.

Jak już wspomniano, język VRML zawiera pewną ilość interpolatorów. Jedne z nich służą do interpolowania stanów pośrednich pomiędzy dwoma kolorami powodując płynne zmiany zabarwienia obiektów, inne zaś pozwalają animować obiekty.

Zasada użycia takiego interpolatora jest bardzo prosta. Należy podać stan początkowy A, oraz stan końcowy B (pomiędzy tymi stanami mogą być zadane pożądane stany pośrednie). Każdemu ze stanów przypisujemy czas, w jakim obiekt ma go osiągnąć. W jednym z pól węzła interpolującego wpisujemy czas trwania jednego cyklu. Kiedy wszystkie te dane zostaną zebrane, węzeł interpoluje stany pośrednie i w odpowiednim momencie uruchamia całą sekwencję. W tym projekcie nie można skorzystać z takiego mechanizmu, gdyż nie wiadomo, jaka będzie następna pozycja klienta.

Dlatego należy odpowiednio zmodyfikować algorytm tak, aby po otrzymaniu danych o nowym położeniu obiekt odwzorowujący klienta rozpoczął ruch. W momencie przyjścia kolejnych współrzędnych należy zatrzymać sekwencję interpolatora, pobrać pozycję, w której znajduje się klient, a następnie wygenerować nową sekwencję ruchu dla nowej zmiennej końcowej. Proces ten można prześledzić na schemacie przedstawionym na rysunku 20.



Rys. 20 Przyjęty sposób symulacji ruchu

4.3.4 Informacje o kliencie

Podczas testowania jednej z pierwszych wersji programu zauważono, iż klient, który zakończył procedurę logowania nie staje się tym samym widoczny dla pozostałych użytkowników „obecnych” w tym samym pomieszczeniu. Spowodowane było to pewną niedoskonałością użytego algorytmu wizualizacji postaci klienta. Otóż algorytm ten opierał się na wykrywaniu **zmian** położenia klienta, przez co reagował dopiero na pierwszy jego ruch. Aby rozwiązać powstały problem stworzono dodatkową zmienną typu *boolean*, która określała fakt zmiany pozycji. Gdy zmiana nastąpiła zmienna ta przyjmowała wartość *true*. W trakcie normalnej pracy programu wartość zmiennej ustawiana była przez węzeł typu *PositionInterpolator*, zależnie od zachowania klienta. Natomiast przyjęto, że podczas logowania zmienna ta będzie odgórnie ustawiana wartością *true*, dzięki czemu interpolator odbierze informację, że klient wykonał ruch, i dokona aktualizacji jego pozycji. Dzięki temu obecnie postać klienta staje się widoczna natychmiast po dokonaniu logowania.

4.3.5 Wnioski

Pomimo faktu, iż również w Polsce nastąpił gwałtowny rozwój sieci Internet, to jakość oferowanych usług nadal jest niezwykle niska. Łącza o przepustowości sięgającej 2 Mbit/s w godzinach największego natężenia ruchu są niemal całkowicie zablokowane, co sprawia, że przesłanie pliku wielkości zaledwie kilkuset kilobajtów potrafi zająć kilka godzin. Często zdarza się również utrata połączenia pomiędzy serwerem a klientem i przerwanie transmisji.

W takiej sytuacji możliwość ciągłego wysyłania informacji o aktualnym położeniu klientów w świecie VRML jest bardzo utrudniona. Jednak, ponieważ informacja o pojedynczym ruchu klienta to zaledwie kilkadziesiąt bajtów, mieści się ona w pojedynczym pakiecie TCP/IP. Podczas przeprowadzonych testów takie pojedyncze pakiety w dużym procencie przypadków były przesyłane pomyślnie, nawet przy stosunkowo dużym obciążeniu łącz. Utrata małej części pakietów jest praktycznie niezauważalna dla klienta, następuje wówczas jedynie zafałszowanie trajektorii ruchu pozostałych widzianych przez niego osób znajdujących się w tym samym pomieszczeniu.

Przeciwieństwem sieci globalnej jest sieć lokalna. Przedstawiony program został przetestowany w wewnętrznej sieci komputerowej w budynku Polsko-Japońskiej Wyższej Szkoły Technik Komputerowych. Jest to typowa sieć lokalna wykorzystująca karty o przepustowości 10/100 [MB/s], okablowana skrętką ekranowaną. W opisanym środowisku program funkcjonował w zasadzie bez zarzutu. Przy maksymalnym obciążeniu testowym, tj. 10 klientach, wizualizacja ich ruchu była płynna i nie wystąpiły żadne poważne problemy związane z komunikacją pomiędzy serwerem a klientami.

Podsumowując, w obecnym czasie niestety nie ma możliwości powszechnego zastosowania tego typu aplikacji. Wiąże się to z niemożliwością udostępnienia efektów takiej pracy szerokim masom, jednak niewątpliwie przyszłość pokaże, czy będzie możliwe jej upowszechnienie.

5. Bibliografia

5.1. Książki :

[1] Rodger Lea, Kouichi Matsuda, Ken Miyashita , *„ Java for 3D and VRML Worlds”* (1996); New Riders; ISBN 1-56205-689-1

[2] Jan Bielecki, „*Java od podstaw*”; wyd. Intersoftland 1997; ISBN: 83-86861-24-X

[3] Jan Bielecki, „*Java 2*” ; wyd. Intersoftland 1997; ISBN: 83-86861-40-1

[4] Jan Bielecki, „*Java 3 RMI*”; wyd. Helion 1999; ISBN: 83-7197-169-9

[5] Kris Jamsa “Java”; wyd. Mikom 1996; ISBN:83-7158-042-8

5.2. Materiały inne:

[6] „*VRML 97*”, ISO/IEC 14772-1:1 (1997)

[7] „*A Brief History of the Green Project*”
http://www.javasoft.com/people/jag/green/index.html

[8] *„Java: The inside story”*
http://www.sunworld.com/swol-07-1995/swol-07-java.html

[9] Janson English „*The story of the Java platform*”,
http://www.javasoft.com/nav/whatis/storyofjava.html

[10] Jon Byous „*Happy 3-rd birthday*”
http://www.javasoft.com/features/1998/05/birthday.html

[11] David Bank „*The Java Saga*”
http://www.wired.com/wired/archive/3.12/java.saga.html

[12] „*Living Worlds*”
http://www.vrml.org/WorkingGroups/living-worlds/draft_2/lw_ideas.htm

[13] *„The History of VRML*”
http://home.hiwaay.net/~crispen/vrmlworks/history.html

[14] „*VRML History Aproposal for a Virtual Reality Modeling Language History Extension*”,
http://www-winfo.uni-siegen.de/vrmlHistory/docs/index.html

[15] Hartmut Luttermann & Manfred Grauer „*VRML History: Storing And Browsering Temporal 3D-Words*”

[16] „Software narzędzia programy sieci”, Nr.12/98 (48) ISSN 1233-5886 *„Wielowątkowy server w Javie”* Mani Malarvannan

6. Załączniki

6.1. Załącznik 1

Zawiera fragmenty kodu VRML obrazujące budowę następujących węzłów:

ElevationGrid

```
ElevationGrid {
eventIn MFFloat set_height
exposedField SFNode color NULL
exposedField SFNode normal NULL
exposedField SFNode texCoord NULL
field MFFloat height [ ] # (-,∞)
field SFBool ccw TRUE
field SFBool colorPerVertex TRUE
field SFFloat creaseAngle 0 # [0, ∞]
field SFBool normalPerVertex TRUE
field SFBool solid TRUE
field SFInt32 xDimension 0 # [0, ∞)
field SFFloat xSpacing 1.0 # (0, ∞)
field SFInt32 zDimension 0 # [0, ∞)
field SFFloat zSpacing 1.0 # (0, ∞)
}
```

Extrusion

```
Extrusion {
eventIn MFVec2f set_crossSection
eventIn MFRotation set_orientation
eventIn MFVec2f set_scale
eventIn MFVec3f set_spine
field SFBool beginCap TRUE
field SFBool ccw TRUE
field SFBool convex TRUE
field SFFloat creaseAngle 0 # [0, ∞)
field MFVec2f crossSection [ 1 1, 1 -1, -1 -1,
 -1 1, 1 1 ] # (-∞,∞)
field SFBool endCap TRUE
field MFRotation orientation 0 0 1 0 # [-1,1],(- ∞,∞)
field MFVec2f scale 1 1 1 # (0, ∞)
field SFBool solid TRUE
```

```
field MFVec3f spine [ 0 0 0, 0 1 0 ] # (-,∞)
}
```

IndexedFaceSet

```
IndexedFaceSet {
eventIn MFInt32 set_colorIndex
eventIn MFInt32 set_coordIndex
eventIn MFInt32 set_normalIndex
eventIn MFInt32 set_texCoordIndex
exposedField SFNode color NULL
exposedField SFNode coord NULL
exposedField SFNode normal NULL
exposedField SFNode texCoord NULL
field SFBool ccw TRUE
field MFInt32 colorIndex [ ] # [-1, ∞)
field SFBool colorPerVertex TRUE
field SFBool convex TRUE
field MFInt32 coordIndex [ ] # [-1, ∞)
field SFFloat creaseAngle 0 # [ 0, ∞)
field MFInt32 normalIndex [ ] # [-1, ∞)
field SFBool normalPerVertex TRUE
field SFBool solid TRUE
field MFInt32 texCoordIndex [ ] # [-1, ∞)
}
```

IndexedLineSet

```
IndexedLineSet {
eventIn MFInt32 set_colorIndex
eventIn MFInt32 set_coordIndex
exposedField SFNode color NULL
exposedField SFNode coord NULL
field MFInt32 colorIndex [ ] # [-1, ∞)
field SFBool colorPerVertex TRUE
field MFInt32 coordIndex [ ] # [-1, ∞)
}

PointSet {
exposedField SFNode color NULL
exposedField SFNode coord NULL
}
```

Text

```
Text {
exposedField MFString string [ ]
exposedField SFNode fontStyle NULL
exposedField MFFloat length [ ] # [0, ∞)
exposedField SFFloat maxExtent 0.0 # [0, ∞)
}
```

6.2. Załącznik 2

Zawiera fragmenty kodu VRML obrazujące budowę następujących węzłów grupujących:

Anchor

```
Anchor {
    eventIn      MFNode   addChildren
    eventIn      MFNode   removeChildren
    exposedField MFNode   children        []
    exposedField SFString description     ""
    exposedField MFString parameter       []
    exposedField MFString url             []
    field        SFVec3f  bboxCenter      0 0 0     # (-∝,∝)
    field        SFVec3f  bboxSize        -1 -1 -1  # (0, ∝)
or -1,-1,-1
}
```

Billboard

```
Billboard {
    eventIn      MFNode   addChildren
    eventIn      MFNode   removeChildren
    exposedField SFVec3f  axisOfRotation 0 1 0     # (-∝,∝)
    exposedField MFNode   children       []
    field        SFVec3f  bboxCenter     0 0 0     # (-∝,∝)
    field        SFVec3f  bboxSize       -1 -1 -1  # (0, ∝)
or -1,-1,-1
}
```

Collision

```
Collision {
    eventIn      MFNode   addChildren
    eventIn      MFNode   removeChildren
    exposedField MFNode   children        []
    exposedField SFBool   collide         TRUE
    field        SFVec3f  bboxCenter      0 0 0      # (-∝,∝)
    field        SFVec3f  bboxSize        -1 -1 -1   # (0, ∝)
or -1,-1,-1
    field        SFNode   proxy           NULL
    eventOut     SFTime   collideTime
}
```

6.3. Załącznik 3 - klasy serwera

6.3.1 Klasa TomServ.java

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Toolkit.*;

class TomServ {

    static int PORT[][] = new int[11][2];
    static ServerHelper sh[]=new ServerHelper[11];
    static int clientNumer=1;
    static Frame f = new MyFrame("Server VRML");
    static String str[] = new String[11];


    public static void main(String[] args){

// Graficzna reprezenatcja servera
        f.setLayout(null);
        f.addNotify();
        f.resize(380,400);
        f.show();
        // Okreslenie przycisku
        Button b = new MyButton("Close");
        f.add(b);
        b.reshape(10,340,355,25);

        //Okreslenie napisu Server
        Label label1 = new Label("Server :");
        label1.setFont(new Font("Dialog",Font.BOLD,16));
        f.add(label1);
        label1.reshape(20,35,70,20);

        //Okreslenie nezwy servera
        TextField edit1 = new TextField(20);
        edit1.disable();
        f.add(edit1);
        edit1.reshape(90,35,168,20);
        Graphics gDC = f.getGraphics();

        //Panel komunikatow servera
        TextField edit2 = new TextField(20);
        edit2.disable();
        f.add(edit2);
        edit2.reshape(10,370,355,25);
// Koniec reprezentacji graficznej servera
```

```
// Inicjalizacja portów i tablicy;

        for(int i=0;i<11;i++){
            PORT[i][0]=(4130+i);PORT[i][1]=0;
            sh[i]=new ServerHelper();
            Label lab = new Label("Port :"+PORT[i][0]);
            f.add(lab);
            lab.reshape(30,60+(25*i),70,20);
            sh[i].graph(f);
        }

// Przypisanie PORTU 4130 Serverowi
        try{
            sh[0].start(PORT[0][0]);
            PORT[0][1]=1;
        }catch(Exception ex){
           System.out.println(ex);
        }
        try{
            edit1.setText(""+InetAddress.getLocalHost());
        }catch(Exception ex){
        }

// Przydzielenie i obsluga klientow
        while(true){
            //znalesc wolny port
            int i=0;

            while(PORT[i][1]!=0){
                i++;
            }

            edit2.setText("Znaleziono wolny port
"+PORT[i][0]);
//            System.out.println("Poszukuje nowego klienta
:\n");

            //Sprawdza czy nie ma nowego klienta

            if(sh[0].newClient(PORT[i][0])){
                str[i] = sh[0].login;

                sh[0].login = "";
                sh[0].pass = "";

                System.out.println("Login :"+str[i]);
                System.out.println("Uruchomic port
"+PORT[i][0]+" dla klienta sh"+i);
                PORT[i][1]=1;
                sh[i].start(PORT[i][0]);
                System.out.println("Rozpoczac polaczenie
wsteczne");
                sh[i].backConnect();

                for(int c=1;c<11;c++){
```

```
                    if(PORT[c][1]>1 &&
PORT[i][0]!=PORT[c][0]){

sh[i].sendCorrection(PORT[c][0],1,sh[c].X,sh[c].Y,sh[c].Z,str
[c]);
                    }
                }

            }else{
                edit2.setText("Nie znaleziono nowych
klientow");
            }

            edit2.setText("Sprawdzanie czy klienci którz juz
sa zalogowani nie chca czegos przeslac");

            for(i=1;i<11;i++){
                edit2.setText("Sprawdzam klieta na porcie
"+PORT[i][0]);
                gDC.setColor(Color.red);
                gDC.fillOval(100,65+(25*(PORT[i][0]-
4130)),10,10);

                if(PORT[i][1]!=0){
                    try{
                        if(!sh[i].checkClient(PORT[i][0])){
                            PORT[i][1]=0;
                            gDC.setColor(Color.black);

gDC.fillOval(100,65+(25*(PORT[i][0]-4130)),10,10);
                            for(int j=1;j<11;j++){
                                if(PORT[j][1] != 0 &&
PORT[i][0] != PORT[j][0]){

sh[j].sendDisconnection(PORT[i][0]);
                                }
                            }
                            sh[i].backDisconnect();
                            str[i] = "";
                        }else{
                            gDC.setColor(Color.green);

gDC.fillOval(100,65+(25*(PORT[i][0]-4130)),10,10);
                        }

                    }catch(Exception ex){
                        System.out.println(i+" "+ex);
                    }

                    if(sh[i].news == true){
                        System.out.println("dane uległy
zmianie");
                        for(int j=1;j<11;j++){
                            if(PORT[j][1] != 0 && PORT[i][0]
!= PORT[j][0]){
```

```
                                    System.out.println("wyslac
korekte polarzenia klientu "+PORT[j][0]);

sh[j].sendCorrection(PORT[i][0],PORT[i][1],sh[i].X,sh[i].Y,sh
[i].Z,str[i]);
                                }
                            }
                            if(PORT[i][1]==1)
                                PORT[i][1]++;
                            sh[i].news = false;
                        }

                    }else{
                        gDC.clearRect(100,65+(25*(PORT[i][0]-
4130)),10,10);
                    }
                }



            }

        }

}
//
**********************************************************
**
class MyFrame extends Frame {
    public MyFrame(String caption)
    {
        super(caption);
        addWindowListener(
            new WindowAdapter() {
                public void windowClosing(WindowEvent evt){
                    hide();
                    dispose();
                    System.exit(0);
                }
            }
        );
    }
}
//
**********************************************************
**
class MyButton extends Button{
    public MyButton(String caption){
        super(caption);
        addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt){
                System.exit(0);
            }
        }
        );
    }
```

```
}
```

6.3.2 Klasa Tomreceiver.java

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

import java.net.*;
import java.io.*;

public class Tomreceiver extends Script {
    private MFNode Inne_obiekty;    // eventOut

    static int PORT=0;
    TomCli1 tc;
    Socket socket=null;
    DataInputStream in2=null;
    DataOutputStream out2=null;
    boolean connect=false;

    int object=0;
    float X,Y,Z;
    float tab[][] = new float[11][5];

    public void initialize() {
        try{
            socket.setSoTimeout(250);
        }catch(Exception ex){
        }
        Inne_obiekty = (MFNode)getEventOut("Inne_obiekty");
        for(int i=0;i<11;i++){
            for(int j=0;i<5;j++){
                tab[i][j] = 0.0f;
            }
        }
    }

    public void processEvent (Event event) {
        if(!connect){
            try{
                socket = new Socket("1.1.1.2",tc.PORT+10);
                in2 = new
DataInputStream(socket.getInputStream());
                out2 = new
DataOutputStream(socket.getOutputStream());
                socket.setSoTimeout(50);
                connect = true;
            }catch(Exception ex){
                System.out.println("Sorry cos nie tak z
portem");
            }
        }else{
```

```
            try{
                if(in2.readBoolean()){
                    socket.setSoTimeout(250);
                    out2.writeBoolean(true);

                    object=in2.readInt();
                    X=in2.readFloat();
                    Y=in2.readFloat();
                    Z=in2.readFloat();
                    socket.setSoTimeout(50);



                String str1 =
                    "DEF "+object+" Transform {" +
                    "    translation ";
                String str2 =
                    " "+X+" "+Y+" "+Z+" " +
                    "    children[" +
                    "    Shape{geometry Sphere{}" +
                    "          appearance Appearance{" +
                    "              material
Material{diffuseColor ";
                String str3 =
                    "              }" +
                    "          }" +
                    "    }" +
                    "    ]" +
                    "}";
                    Browser browser = getBrowser();
                    BaseNode baseNodes[];
                    baseNodes =
browser.createVrmlFromString(str1+str2+1+" "+0+" "+0+str3);
                    if(baseNodes!= null)
                        Inne_obiekty.setValue(baseNodes);
                }
            }catch(Exception ex){
            }
        }
    }

    public void shutdown() {
        try{
            in2.close();
            out2.close();
            socket.close();
        }catch(Exception ex){
        }
    }
}
```

6.3.3 Klasa Serverhelper.java

```
import java.io.*;
```

```
import java.util.*;
import java.net.*;
import java.awt.*;

class ServerHelper {
        ServerSocket ss = null,ss1=null;
        Socket cs =null,cs1=null;
        DataInputStream in = null,in1=null;
        DataOutputStream out = null,out1=null;
        boolean initPORT=false;
        static int PORT = 0;


        float X,Y,Z;
        boolean news = false;
        String login="";
        String pass="";

// Dla obsługi graficznej

        Frame f;
        Graphics gDC;

// Koniec obslugi graficznej

    public void graph(Frame f){
        this.f=f;
        gDC = f.getGraphics();
    }
    public void start(int PORT){
        try{
            gDC.setColor(Color.green);
            gDC.drawOval(100,65+(25*(PORT-4130)),10,10);
            ss = new ServerSocket(PORT);
            gDC.fillOval(100,65+(25*(PORT-4130)),10,10);
            System.out.println("Utworzono port Servera;
"+PORT);
        }catch(IOException ex){
            System.out.println("1"+ex);
        }
    }


    public boolean newClient(int PORT){
        try{
            ss.setSoTimeout(5);
            cs = ss.accept();
            out = new DataOutputStream(cs.getOutputStream());
            in = new DataInputStream(cs.getInputStream());
            System.out.println("Nawiazano kontakt z nowym
klientem na porcie 4130");
            int a = in.readInt();
            for(int b=0;b<a;b++){
                login = login + in.readChar();
            }
            System.out.println(login);
```

```
            a = in.readInt();
            for(int b=0;b<a;b++){
                pass = pass + in.readChar();
            }
            System.out.println(pass);

//************************************************************
***************
//  Weryfikacja hasla
//
//  Lista możliwych klientow
//************************************************************
***************

boolean correct = false;
            try{
                FileReader inpR = new
FileReader("C:/HKI/pass.log");
                BufferedReader inp = new
BufferedReader(inpR);
                int c=0;
                boolean ok=false;
                while(inp.ready()){

                    char litera = (char) inp.read();
                    System.out.print(litera);

                    if(!ok){
                        if(litera == login.charAt(c)){
                            c++;
                            if(login.length()==c){
//                                System.out.println(" Login
correct");
                                c=0;
                                ok=true;
                                inp.read();
                            }
                        }else{
//                              System.out.println(" Login
incorrect");
                            inp.readLine();
                            c=0;
                            correct = false;
                        }
                    }else{
                        if(litera == pass.charAt(c)){
                            c++;
                            if(pass.length()==c){
//                                System.out.println(" Pass
correct");
                                correct = true;
                                break;
                            }
                        }else{
```

```
//                              System.out.println(" Pass
incorrect");
                                inp.readLine();
                                c=0;
                                ok=false;
                                correct = false;
                            }
                        }
                    }
                }catch(Exception ex){
                    System.out.println(ex);
                }

                System.out.println("Odpowiedz weryfikacji :
"+correct);

                if(correct){
                    out.writeInt(PORT);
                    if(in.readBoolean()==true){
                        this.PORT=PORT;
                        System.out.println("Przydzielanie
klientowi numeru portu:"+this.PORT);
                        return true;
                    }
                }else{
                    System.out.println("Nielegalna proba
dostepu");
                    out.writeInt(5000);
                }

                login = "";
                pass = "";

                return false;
            } catch (IOException ex){
                return false;
            }
        }

        public boolean checkClient(int PORT){
            if (initPORT){
                try{

                    if(!in.readBoolean()){
                        X=in.readFloat();
                        Y=in.readFloat();
                        Z=in.readFloat();
                        System.out.println(X+'\t'+Y+'\t'+Z);
                        System.out.println(Y);
                        System.out.println(Z);
                        news=true;
                    }else{
                          in.close();
                          out.close();
                          cs.close();
                          ss.close();
```

```
                        initPORT=false;
                        return false;
                    }

                }
                catch (InterruptedIOException ex){
                }
                catch(Exception ex){
                    System.out.println(ex);
                }
            }else{
                try{
                    System.out.println(ss);
                    cs = ss.accept();
                    cs.setSoTimeout(190);
                    in = new
DataInputStream(cs.getInputStream());
                    out = new
DataOutputStream(cs.getOutputStream());
                    initPORT=true;
                    X= (in.readFloat());
                    Y= (in.readFloat());
                    Z= (in.readFloat());
                    System.out.println(X);
                    System.out.println(Y);
                    System.out.println(Z);
                    return true;
                }catch(Exception ex){
                    System.out.println(ex);
                }
            }
            return true;
        }

        public void backConnect(){
            try{
                ss1 = new ServerSocket(PORT+10);
                cs1 = ss1.accept();
                in1 = new DataInputStream(cs1.getInputStream());
                out1 = new
DataOutputStream(cs1.getOutputStream());
                System.out.println("Polaczenie wsteczne
zakonczone pomyslnie");
             }catch(Exception ex){
             }
        }



       public void sendCorrection(int PORT,int first,float
X,float Y,float Z,String str){
            try{
                out1.writeBoolean(true);
                if(in1.readBoolean()){
                    System.out.println("Wysylam na port "+PORT+"
\t"+X+'\t'+Y+'\t'+Z);
```

```
                out1.writeInt(PORT);
                out1.writeFloat(X);
                out1.writeFloat(Y);
                out1.writeFloat(Z);
                if(first == 1){
                    out1.writeInt(str.length());
                    for(int c=0;c<str.length();c++){
                        out1.writeChar(str.charAt(c));
                    }
                }
            }
        }catch(Exception ex){
        }
    }

    public void sendDisconnection(int PORT){
        try{
            System.out.println("Wysylam sygnal zniszczenia");
            out1.writeBoolean(false);
            out1.writeInt(PORT);
        }catch(Exception ex){
            System.out.println(ex);
        }
    }

    public void backDisconnect(){
     try{
         in1.close();
         out1.close();
         cs1.close();
         ss1.close();
     }catch(Exception ex){
     }
    }
}
```

6.4. Załącznik 4 - klasy klienta

6.4.1 Klasa odbieranie.java

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

import java.net.*;
import java.io.*;

import Okno.*;

public class odbieranie extends Script {
```

```
    private MFNode o4140del;    // eventOut
    private MFNode o4139del;    // eventOut
    private MFNode o4138del;    // eventOut
    private MFNode o4137del;    // eventOut
    private MFNode o4136del;    // eventOut
    private MFNode o4135del;    // eventOut
    private MFNode o4134del;    // eventOut
    private MFNode o4133del;    // eventOut
    private MFNode o4132del;    // eventOut
    private MFNode o4131del;    // eventOut
    private MFNode o4140cr;    // eventOut
    private MFNode o4139cr;    // eventOut
    private MFNode o4138cr;    // eventOut
    private MFNode o4137cr;    // eventOut
    private MFNode o4136cr;    // eventOut
    private MFNode o4135cr;    // eventOut
    private MFNode o4134cr;    // eventOut
    private MFNode o4133cr;    // eventOut
    private MFNode o4132cr;    // eventOut
    private MFNode o4131cr;    // eventOut
    private SFNode o4140n;    // field
    private SFNode o4139n;    // field
    private SFNode o4138n;    // field
    private SFNode o4137n;    // field
    private SFNode o4136n;    // field
    private SFNode o4135n;    // field
    private SFNode o4134n;    // field
    private SFNode o4133n;    // field
    private SFNode o4132n;    // field
    private SFNode o4131n;    // field

    Node
node1,node2,node3,node4,node5,node6,node7,node8,node9,node10;
    BaseNode
baseNodes1[],baseNodes2[],baseNodes3[],baseNodes4[],baseNodes
5[],

baseNodes6[],baseNodes7[],baseNodes8[],baseNodes9[],baseNodes
10[];
    SFVec3f pozycja;


            String str1 =
                "Transform { "+
                "  children    Shape { "+
                "    appearance     Appearance { "+
                "      material     Material { "+
                "      } "+
                "      texture ImageTexture { "+
                "   repeatS    TRUE "+
                "   repeatT    TRUE ";

            String str2 =
                "   url   \"C:/HKI/";

            String str3 =
```

```
            "      } "+
            "    } "+
            "    geometry  Box { "+
            "      size    1 1 1 "+
            "    } "+
            "  } "+
            "  translation 0 1 0"+
            "} ";

        String strl ="";
        String strf =".gif\" ";



static int PORT=0;
wysylanie tc;
Socket socket=null;
DataInputStream in2=null;
DataOutputStream out2=null;
boolean connect=false;
int object=0;
float X,Y,Z;
float tab[][] = new float[11][5];
static String url;
static String log;
static String pass;

public void initialize() {
    System.out.println("Jestem w Init");

     o4131cr  = (MFNode)getEventOut("o4131cr");
     o4132cr  = (MFNode)getEventOut("o4132cr");
     o4133cr  = (MFNode)getEventOut("o4133cr");
     o4134cr  = (MFNode)getEventOut("o4134cr");
     o4135cr  = (MFNode)getEventOut("o4135cr");
     o4136cr  = (MFNode)getEventOut("o4136cr");
     o4137cr  = (MFNode)getEventOut("o4137cr");
     o4138cr  = (MFNode)getEventOut("o4138cr");
     o4139cr  = (MFNode)getEventOut("o4139cr");
     o4140cr  = (MFNode)getEventOut("o4140cr");

     o4131del = (MFNode)getEventOut("o4131del");
     o4132del = (MFNode)getEventOut("o4132del");
     o4133del = (MFNode)getEventOut("o4133del");
     o4134del = (MFNode)getEventOut("o4134del");
     o4135del = (MFNode)getEventOut("o4135del");
     o4136del = (MFNode)getEventOut("o4136del");
     o4137del = (MFNode)getEventOut("o4137del");
     o4138del = (MFNode)getEventOut("o4138del");
     o4139del = (MFNode)getEventOut("o4139del");
     o4140del = (MFNode)getEventOut("o4140del");

    try{
        socket.setSoTimeout(1250);
    }catch(Exception ex){
    }
```

```
        try{
            o4131n = (SFNode) getField("o4131n");
            o4132n = (SFNode) getField("o4132n");
            o4133n = (SFNode) getField("o4133n");
            o4134n = (SFNode) getField("o4134n");
            o4135n = (SFNode) getField("o4135n");
            o4136n = (SFNode) getField("o4136n");
            o4137n = (SFNode) getField("o4137n");
            o4138n = (SFNode) getField("o4138n");
            o4139n = (SFNode) getField("o4139n");
            o4140n = (SFNode) getField("o4140n");

            node1 = (Node)(o4131n.getValue());
            node2 = (Node)(o4132n.getValue());
            node3 = (Node)(o4133n.getValue());
            node4 = (Node)(o4134n.getValue());
            node5 = (Node)(o4135n.getValue());
            node6 = (Node)(o4136n.getValue());
            node7 = (Node)(o4137n.getValue());
            node8 = (Node)(o4138n.getValue());
            node9 = (Node)(o4139n.getValue());
            node10 = (Node)(o4140n.getValue());
        }catch(Exception ex){
            System.out.println("Cos sie stalo :"+ex);
        }
//*********************************************************
*********
// Proces logowania
//*********************************************************
*********
//url = "1.1.1.2";
//log = "Tomasz";
//pass = "ncc1701";

        int bool = 0;
            while(bool <3){

                if(bool==0){
                    Okno.display();
                    bool++;
                }

                if(bool==1 && Okno.getZamkniecie() ){
                    bool++;
                }

                if(bool==2){
                    url = Okno.getURL();
                    log = Okno.getLogin();
                    pass = Okno.getPass();

                    System.out.println("URL : "+url+"\tLogin
: "+log+"\tPass : "+pass);
                    bool++;
                }
```

```
            }
            bool=0;

//*************************************************************
*********
    }

    public void processEvent (Event event) {

        if(!connect){
            try{
                socket = new Socket(url,tc.PORT+10);
                in2 = new
DataInputStream(socket.getInputStream());
                out2 = new
DataOutputStream(socket.getOutputStream());
                socket.setSoTimeout(50);
                connect = true;
            }catch(Exception ex){
                System.out.println("Sorry cos nie tak z
portem");
            }
        }else{

            try{
                if(in2.readBoolean()){
                    socket.setSoTimeout(250);
                    out2.writeBoolean(true);

                    object=in2.readInt();

                    X=in2.readFloat();
                    Y=in2.readFloat();
                    Z=in2.readFloat();
                    System.out.println("dane otrzymane: "+X+"
"+Y+" "+Z);

                    socket.setSoTimeout(50);

                    System.out.println("Obiekt jest :
"+object);


                    Browser browser = getBrowser();
                    System.out.println("Object :"+object);
                    switch(object){
                        case 4131:
                            if(baseNodes1 == null){
                                int c = in2.readInt();
                                while(c>0){
                                    strl = strl +
in2.readChar();
                                    c--;
                                }
                                System.out.println(strl+"
"+c);
```

```
                                baseNodes1 =
browser.createVrmlFromString(str1+str2+strl+strf+str3);
                                o4131cr.setValue(baseNodes1);
                            }
                            try{
                                System.out.println("Wykonuje
try dla 4131");
                                pozycja =(SFVec3f)
node1.getExposedField("translation");
                                pozycja.setValue(X,Y,Z);
                            }catch(Exception ex){
                                System.out.println("Exception
:"+ex);
                            }
                            System.out.println("Powinno byc
ok 4131\n");
                            break;

                        case 4132:
                            if(baseNodes2 == null){
                                int c = in2.readInt();
                                while(c>0){
                                    strl = strl +
in2.readChar();
                                    c--;
                                }
                                System.out.println(strl+"
"+c);

                                baseNodes2 =
browser.createVrmlFromString(str1+str2+strl+strf+str3);
                                o4132cr.setValue(baseNodes2);
                            }
                            try{
                                System.out.println("Wykonuje
try dla 4132");
                                pozycja =(SFVec3f)
node2.getExposedField("translation");
                                pozycja.setValue(X,Y,Z);
                            }catch(Exception ex){
                                System.out.println("Exception
:"+ex);
                            }
                            System.out.print("Powinno byc ok
4132");
                            break;

                        case 4133:
                            if(baseNodes3 == null){

System.out.println(in2.readInt());

                                baseNodes3 =
browser.createVrmlFromString(str1+str2+strl+strf+str3);
                                o4133cr.setValue(baseNodes3);
```

```
                                }
                                try{
                                    System.out.println("Wykonuje
try dla 4133");
                                    pozycja =(SFVec3f)
node3.getExposedField("translation");
                                    pozycja.setValue(X,Y,Z);
                                }catch(Exception ex){
                                    System.out.println("Exception
:"+ex);
                                }
                                System.out.print("Powinno byc ok
4133");
                                break;
                        }


                    }else{
                        switch(in2.readInt()){
                            case 4131:
                                System.out.println("Zniszczenie
obiektu ");
                                o4131del.setValue(baseNodes1);
                                baseNodes1=null;
                                break;
                            case 4132:
                                System.out.println("Zniszczenie
obiektu ");
                                o4132del.setValue(baseNodes2);
                                baseNodes2=null;
                                break;
                            case 4133:
                                System.out.println("Zniszczenie
obiektu ");
                                o4133del.setValue(baseNodes3);
                                baseNodes3=null;
                                break;
                        }

                    }

                }catch(Exception ex){
                }
            }


        }
        public void shutdown() {
            try{
                in2.close();
                out2.close();
                socket.close();
            }catch(Exception ex){
            }
        }
    }
```

6.4.2 Klasa wysyłanie.java

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

import java.net.*;
import java.io.*;
import java.util.*;

public class wysylanie extends Script {

    Socket socket = null;
    DataInputStream in = null;
    DataOutputStream out=null;
    static int PORT;
    float yp[] = new float[3];
    int ilosc=0;
    odbieranie odb;

    public void initialize() {

        System.out.println("IN init"+"URL: "+odb.url);
        try{
            socket = new Socket(odb.url, 4130);
            System.out.println("Socket OK");
            in = new
DataInputStream(socket.getInputStream());
            out = new
DataOutputStream(socket.getOutputStream());
            System.out.println("in out OK");

            out.writeInt(odb.log.length());
            for(int a=0;a<odb.log.length();a++){
                out.writeChar(odb.log.charAt(a));
            }

            out.writeInt(odb.pass.length());
            for(int a=0;a<odb.pass.length();a++){
                out.writeChar(odb.pass.charAt(a));
            }

            PORT = in.readInt();

            System.out.println("Otrzymalem port: "+PORT);
            out.writeBoolean(true);
            in.close();
            out.close();
            socket.close();
        }catch (Exception ex){
            System.out.println(ex);
        }

        try{
            if(PORT !=5000){
```

```
                socket = new Socket(odb.url,PORT);
                out = new
DataOutputStream(socket.getOutputStream());
                in = new
DataInputStream(socket.getInputStream());
                System.out.println("Klient nawiazal kontakt z
serverem "+PORT);
            }else{
            }
        }catch(Exception ex){
            System.out.println(ex);
        }

    }
    public void processEvent (Event event) {
        if(event.getName().equals("obecna_pozycja") == true
&& (ilosc++%3==0)){

            ((ConstSFVec3f)event.getValue()).getValue(yp);

            float X = yp[0];
            float Y = yp[1];
            float Z = yp[2];

            try{
                out.writeBoolean(false);
                System.out.println("Pozycja observatora "+X+"
"+Y+" "+Z);
                System.out.println("Wysyłam X");
                out.writeFloat(X);
                System.out.println("wyslalem X, wysylam Y");
                out.writeFloat(Y);
                System.out.println("wyslalem Y, wysylam Z");
                out.writeFloat(Z);
                System.out.println("wyslalem Z");
            } catch (Exception ex){
                Browser browser = getBrowser();
                browser.setDescription("Wystapil blad w
komunikacji");
            }

        }
    }
    public void shutdown() {
        System.out.println("shout down sequence initieted");
        try{
            out.writeBoolean(true);
            out.close();
            in.close();
            socket.close();
        } catch (Exception ex){
        }
    }
}
```

6.4.3 Klasa okno.java

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

import java.net.*;
import java.io.*;
import java.util.*;

public class wysylanie extends Script {

    Socket socket = null;
    DataInputStream in = null;
    DataOutputStream out=null;
    static int PORT;
    float yp[] = new float[3];
    int ilosc=0;
    odbieranie odb;

    public void initialize() {

        System.out.println("IN init"+"URL: "+odb.url);
        try{
            socket = new Socket(odb.url, 4130);
            System.out.println("Socket OK");
            in = new
DataInputStream(socket.getInputStream());
            out = new
DataOutputStream(socket.getOutputStream());
            System.out.println("in out OK");

            out.writeInt(odb.log.length());
            for(int a=0;a<odb.log.length();a++){
                out.writeChar(odb.log.charAt(a));
            }

            out.writeInt(odb.pass.length());
            for(int a=0;a<odb.pass.length();a++){
                out.writeChar(odb.pass.charAt(a));
            }

            PORT = in.readInt();

            System.out.println("Otrzymalem port: "+PORT);
            out.writeBoolean(true);
            in.close();
            out.close();
            socket.close();
        }catch (Exception ex){
            System.out.println(ex);
        }

        try{
            if(PORT !=5000){
```

```
                socket = new Socket(odb.url,PORT);
                out = new
DataOutputStream(socket.getOutputStream());
                in = new
DataInputStream(socket.getInputStream());
                System.out.println("Klient nawiazal kontakt z
serverem "+PORT);
            }else{
            }
        }catch(Exception ex){
            System.out.println(ex);
        }

    }
    public void processEvent (Event event) {
        if(event.getName().equals("obecna_pozycja") == true
&& (ilosc++%3==0)){

            ((ConstSFVec3f)event.getValue()).getValue(yp);

            float X = yp[0];
            float Y = yp[1];
            float Z = yp[2];

            try{
                out.writeBoolean(false);
                System.out.println("Pozycja observatora "+X+"
"+Y+" "+Z);
                System.out.println("Wysyłam X");
                out.writeFloat(X);
                System.out.println("wyslalem X, wysylam Y");
                out.writeFloat(Y);
                System.out.println("wyslalem Y, wysylam Z");
                out.writeFloat(Z);
                System.out.println("wyslalem Z");
            } catch (Exception ex){
                Browser browser = getBrowser();
                browser.setDescription("Wystapil blad w
komunikacji");
            }

        }
    }
    public void shutdown() {
        System.out.println("shout down sequence initieted");
        try{
            out.writeBoolean(true);
            out.close();
            in.close();
            socket.close();
        } catch (Exception ex){
        }
    }
}
```